GENERAL AND FINE-GRAINED VIDEO UNDERSTANDING USING MACHINE LEARNING & STANDARDISED NEURAL NETWORK ARCHITECTURES

A thesis presented for the degree of
Doctor of Philosophy

by
Hayden James Faulkner

THE UNIVERSITY of ADELAIDE

The Australian Institute for Machine Learning
School of Computer Science
The University of Adelaide
Australia

Submitted March 7, 2021

Abstract

Recently, the broad adoption of the internet coupled with connected smart devices has seen a significant increase in the production, collection, and sharing of data. One of the biggest technical challenges in this new information age is how to effectively use machines to process and extract useful information from this data. Interpreting video data is of particular importance for many applications including surveillance, cataloguing, and robotics, however it is also particularly difficult due to video's natural sparseness - *for lots of data there is small amounts of useful information*. This thesis examines and extends a number of Machine Learning models in a number of video understanding problem domains including *captioning*, *detection* and *classification*.

Captioning videos with human like sentences can be considered a good indication of how well a machine can interpret and distill the contents of a video. Captioning generally requires knowledge of the scene, objects, actions, relationships and temporal dynamics. Current approaches break this problem into three stages with most works focusing on visual feature filtering techniques for supporting a caption generation module. Current approaches however still struggle to associate ideas described in captions with their visual components in the video. We find that captioning models tend to generate shorter more succinct captions, with overfitted training models performing significantly better than human annotators on the current evaluation metrics. After taking a closer look at the model and human generated captions we highlight that the main challenge for captioning models is to correctly identify and generate specific nouns and verbs, particularly rare concepts. With this in mind we experimentally analyse a handful of different concept grounding techniques, showing some to be promising in increasing captioning performance, particularly when concepts are identified correctly by the grounding mechanism.

To strengthen visual interpretations, recent captioning approaches utilise object detections to attain more salient and detailed visual information. Currently, these detections are generated by an image based detector processing only a single video frame, however it's desirable to capture the temporal dynamics of objects across an entire video. We take an efficient image object detection framework, and carry out an extensive exploration into the effects of a number of network modifications towards improving the model's ability to perform on video data. We find a number of promising directions which improve upon the single frame baseline. Furthermore, to increase concept coverage for object detection in video we combine datasets from both the image and video domains. We then perform an in-depth analysis on the coverage of the combined detection dataset with the concepts found in captions from video captioning datasets.

While the bulk of this thesis centres around general video understanding - *random videos from the internet* - it's also useful to determine the performance of these Machine Learning techniques on a more fine-grained problem. We therefore introduce a new Tennis dataset, which includes broadcast video for five tennis matches with detailed annotations for match events and commentary style captions. We evaluate a number of modern Machine Learning techniques for performing shot classification, as a stand-alone and a precursor process for commentary generation, finding that current models are similarly effective for this fine-grained problem.

Contents

List of Figures

List of Tables

Code & Commands

Acronyms

AMT Amazon Mechanical Turk.

ANN Artificial Neural Network.

BCE binary cross-entropy.

BN batch normalisation.

CNN Convolutional Neural Network.

DT Dense Trajectories.

DVS Descriptive Video Services.

FPN Feature Pyramid Network.

GRU Gated Recurrent Unit.

GT ground truth.

HoF Histograms of Oriented Flow.

HoG Histograms of Oriented Gradients.

IoU intersection over union.

LSTM Long Short Term Memory.

mAP mean Average Precision.

MbH Motion Boundary Histograms.

ML machine learning.

MLP Multi-Layer Perceptron.

MSE mean squared error.

NAS Neural Architecture Search.

NMS Non-Maximal Suppression.

NN neural network.

OF optical flow.

POS Parts-of-Speech.

RCN Recurrent Convolutional Network.

ReLU Rectified Linear Unit.

RGB red-green-blue.

RNN Recurrent Neural Network.

ROI region of interest.

RPN Region Proposal Network.

SGD stochastic gradient descent.

SIFT Scale-Invariant Feature Transform.

SSD Single-Shot Multi-box Detector.

SVM Support Vector Machine.

SVO subject-verb-object.

TN Transformer Network.

YOLO You Only Look Once.

Symbols

A The number of anchors per grid cell.

C The number of classes or categories (in Chapter 2).

C The number of concepts (in Chapter 1).

D The number of channels of a feature tensor.

F The number of visual features.

H The number of heads in a multi-head attention mechanism.

I An image or video dataset.

$K^{(\mathbf{S})}$ The spatial dimensions of a kernel K.

$K^{(\mathbf{T})}$ The temporal dimensions of a kernel K.

L The number of layer repetitions of some neural network module.

M The number of frame region features per frame.

N The maximum permissible length of a caption.

R The number of region features per time-step.

S The spatial size (width and height) of an image or feature.

T The number of frames in a video.

W The temporal window size.

α A scalar attention weight.

$\bar{d}$ The maximum displacement value for a feature correlation operation.

$\check{\mathbf{x}}$ The input to an RNN unit.

$\hat{c}$ A candidate (predicted) caption usually related to the i^{th} sample in the dataset I.

$\mathbf{E}$ The word embedding matrix for all words in the captioning vocabulary.

$\mathbf{F}$ The unfiltered set of visual features.

$\mathbf{K}$ The keys matrix for a Transformer Network.

$\mathbf{M}$ A binary mask matrix utilised in training Transformer Networks.

$\mathbf{V}$ The values matrix for a Transformer Network.

$\mathbf{W}$ A weight matrix of a neural network.

$\mathbf{X}$ A video input.

$\tilde{\mathbf{F}}$ The filtered set of visual features.

$\tilde{\mathbf{e}}$ A single filtered word embedding feature.

$\tilde{\mathbf{f}}$ A single filtered visual feature.

$\tilde{\mathbf{s}}$ A sentence feature that has been passed through a filtering technique.

$\mathbf{b}$ A bias vector of a neural network.

$\mathbf{c}^{(\alpha)}$ A context vector used in the calculation of the attention weight α.

$\mathbf{e}$ A word embedding vector for a word in the captioning vocabulary.

$\mathbf{f}$ A visual feature vector representing a particular frame.

$\mathbf{h}$ The hidden state of an RNN.

$\mathbf{q}$ A query vector for a Transformer Network.

$\mathbf{r}$ A region visual feature vector representing a particular frame region.

$\mathbf{x}$ A video frame.

$\mathbf{y}$ A one-hot encoding of a word in a caption.

$\mathcal{C}$ The set of ground truth captions usually related to the i^{th} sample in the dataset I.

$\mathcal{D}$ The word vocabulary dictionary for the captions.

$\mathring{\mathbf{e}}$ A concept word embedding.

$\mathring{\mathcal{D}}$ The word vocabulary dictionary for the concepts.

$\mathring{\rho}$ The confidence probability score across the set of words in the concept vocabulary.

$\mathring{y}$ A binary label for the existence of a particular concept in a particular caption.

$\rho^{(\mathbf{a})}$ The confidence probability score across a set of attributes.

ρ The confidence probability score across the set of words in the captioning vocabulary.

W A tensor representing a temporal window of tensors.

Z' A DarkNet-53 output feature.

Z A YOLO output feature.

$\mathbf{N}$ The n-gram length (used in caption evaluation metrics).

c A ground truth (reference) caption usually related to the i^{th} sample in the dataset I.

$d^{(\alpha)}$ The attention size for an additive attention mechanism.

$d^{(\mathring{\mathbf{e}})}$ The dimension of the concept word embeddings.

$d^{(\mathbf{F})}$ The embedded dimension of the unfiltered features.

$d^{(\mathbf{b})}$ The dimension of the unfiltered box coordinates.

$d^{(\mathbf{c})}$ The dimension of the unfiltered category feature (one-hot).

$d^{(\mathbf{e})}$ The dimension of the embedding space matrix and vectors.

$d^{(\mathbf{f})}$ The dimension of the unfiltered frame feature.

$d^{(\mathbf{h})}$ The dimension of the hidden state $\mathbf{h}$ (for RNN) or hidden layers of some other network architecture.

$d^{(\mathbf{i})}$ The dimension of the inner fully connected layer of a Transformer Network.

$d^{(\mathbf{m})}$ The dimension of the unfiltered motion feature.

$d^{(\mathbf{r})}$ The dimension of the unfiltered region features.

$d^{(\tilde{\mathbf{f}})}$ The dimension of an encoded visual feature.

$k^{(\mathbf{b})}$ The beam size for the beam search algorithm.

p Precision.

r Recall.

s The temporal stride between sampled frames.

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

13/7/2021

Signature *Date*

Acknowledgements

I'd like to acknowledge that I am incredibly lucky and grateful for the opportunity to undertake and complete such a prestigious degree, particularly as part of the University of Adelaide and within the Australian Institute for Machine Learning.

To my supervisor Professor Anthony Dick, it's definitely been a long journey from the start of my honours degree to the completion of my PhD, but I appreciate your support and guidance related to my studies over the past 7 years. Thanks for sticking by and seeing everything through to the end with me despite some ups and downs along the way. I'm grateful for the time and effort you have put towards my studies, and for your insightful advice that helped grow me as a researcher and critical thinker.

To Dr Zygmunt Szpak, although you won't receive the formal recognition you deserve, your contribution towards my studies, and personal well-being, especially in the final weeks of my PhD cannot be overstated. The time you gave to help me, when you were short on time yourself, knowing you'd get nothing in return is truly unselfish and a hallmark of the kind of person you are. I am forever grateful for everything you have done for me professionally and personally.

To my desk neighbour and partner in crime, Dr Gabriel Maicas Suso, despite our better judgements we just can't seem to leave each other's side. Your comments and insights related to both my work and my mindset have been invaluable during our many years together. I believe I have attained a lifelong friend in you, and even when we do eventually go our separate ways, I will always be here to help you with your romantic poetry, Australian slang, and comedy routines.

To the one and only, Mr Ergnoor Shehu or Lali, thanks for being my number one fan during our time together as PhD students. We may share many different views, but I do believe you have indeed carved a little bit of 'European' into me, and conversely I hope I have aligned you closer to 'original' Aussie Aussie. May I continue to entertain you to death, even if it is at my expense. Musha, musha.

To Dr Rafael Felix Alves, thanks for the personal and professional support you have provided me, and that you give to everyone around you. Your care and desire to help those around you can be a lesson to many, including myself. To Dr Adrian Johnston, thanks for your discussions throughout our time studying together. To Dr Damien Teney, thanks for giving me your time and insights about what good research is. To my other colleagues at the AIML, as well as those in the Computer and Mathematical Sciences department, I thank you for any and every ounce of

support you offered me. I'd also like to thank the thesis examiners for their time and comments, which have no doubt helped form a stronger thesis.

I'd like to acknowledge the Data to Decisions CRC for their financial and professional support throughout my PhD. The D2D CRC conferences and workshops are still strong highlights of my time as a PhD student, and no doubt helped my professional career outcomes. I'd like to particularly thank those involved, including Ms Megan Prideaux, Ms Amy Rangel, and Ms Regine Richelle for their care and friendliness towards the students. Also, thanks to the cohort of D2D CRC students, particularly Caitlin, Adrian, Peter, Dennis, George, Max and Alex, who made Canberra trips a pretty fun time.

To my dearest friends past and present - Adam, Alanah, Amy, Ash, Clare, Ella, Gemma, Haylie, Jacob, James, James, James, Jono, Kira, Laura, Lauren, Lindsay, Mary, Matt, Mel, Rachel, Samantha, Sher Li, Tom and Vanessa. You have all provided me with joy and support some time over the course of my PhD which allowed me to continue to pursue it, thanks for being there and being considerate of my studies.

Lastly and most importantly, to my family - my mum, dad and sister. You are the reason I had the opportunity to pursue such a level of education, your constant support and love throughout my PhD and every moment of my life is truly special, and I am truly lucky for you all. If nothing else, I hope I have been able to make you all proud.

This is one for the future generation. You can ignore this if you want.

Who am I to tell you what to do and what not to do?
But don't do what I did. Do what I didn't do. Make the choices I didn't make.

Support your community. Start with your neighbor and walk outward.
Introduce yourself by name. Why not go for that coffee you always end the conversation on?

Spend time with your friends and family.
Call them, text them. Write them a letter if you're feeling nostalgic.
It might be hard, but tell them you love them. They won't be around forever.

Respect the ground you walk on.
Your kids are gonna walk on it someday too. And their kids after that.
And you don't want your grandchildren only seeing elephants in picture books.

Get up every day. Make the bed all nice and fling open the curtains. Feel the sun on your face.

Love your body. Look into the mirror and think "Wow, I'm amazing".

Don't dwell on Instagram. There far better things to do than that.

But if I have to tell you one thing to change.
Smile more.
Just smile more.

Put down your phone. See the world through your own eyes. Create memories, not JPEGs.

Be creative. Draw, learn a language. Dance even if it feels really awkward.

Why worry about something now, that you won't worry about a year from now?
Remember, no matter how difficult things get, the sun will always rise again tomorrow.

Pay someone a compliment. The checkout assistant, the postman.
You'll probably make their day. I know it would make mine.

Challenge yourself. Challenge others. Ask more questions.
Bring something out of someone that even they didn't know was in there.

Look, no one knows where they're going. You don't have to follow a path.
You don't have to follow anything. Just do what feels right.

But if I had to tell you just one thing to change.
Just smile more.

HONNE
James Hatcher & Andrew Clutterbuck

Preface

What is Video Understanding?

The main focus of this work is to improve video understanding methods, but **what do we mean by understanding?** In general the goal is to enable a computer to be able to extract any necessary information from a video for a particular application or scenario. Imagine asking a computer the fundamental "what?", "where?", "when?", "who?", "why?" questions about a video. How can we enable a computer to be able to answer such questions?

Why should Computers Understand Video?

There is an increased need for computer based automated understanding of all forms of data, video being one of the more important data representations. With most people having a phone camera in their pockets, coupled with the rise of online video streaming and sharing platforms such as YouTube, the amount of video data in existence is growing exponentially. In 2019, over 500 hours of video were uploaded to YouTube every minute[1], and **it is expected by 2022 that online videos will account for more than 82% of all consumer internet traffic**[2]. Furthermore, there is an increasing amount of surveillance footage from governments, businesses, and home owners. The amount of data has become too excessive for people to process, catalogue, annotate and summarise, so computer automation becomes necessary.

An Introduction to Computer Vision & Machine Learning

Computer Vision is a field of computer science that focuses on understanding, or extracting information from imagery, or processing image-like data from one form to another. The field has existed for many decades, but has recently grown in notoriety due to rapid advancements in what computer vision technologies are able to accomplish. Such rapid advancement stemmed

[1] https://www.tubefilter.com/2019/05/07/number-hours-video-uploaded-to-youtube-per-minute/

[2] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

from the advancements in machine learning techniques that are increasingly being adopted for computer vision problems.

Machine learning is a sub-field of artificial intelligence, that is based on enabling machines to learn from data or experience to solve problems, rather than being explicit programmed to do so. Generally machine learning models are very large and computationally expensive to run, and also require vast amounts of data to learn from. Although we have vast amounts of data for many problems, the data needs to be arranged and annotated specifically for each problem, which can often be very difficult and time consuming.

General video understanding is a *long-term* goal, with the field as a whole many years away from any form of general understanding abilities. Instead of tackling understanding *in general*, there are a group of specific sub-problems that are addressed individually, and for the most part separately from one another. These problems are roughly categorised as follows:

- **Image/Video Classification:** Given an image/video, classify its contents into a single category or label;
- **Object Localisation in Images/Videos:** Given an image/video, specify a single location (bounding box or pixel-level segmentation) and a label for the object;
- **Object Detection in Images/Videos:** Given an image/video, detect all objects of interest by specifying a location (bounding box) and label per object;
- **Object Segmentation in Images/Videos:** Given an image/video, detect all objects of interest by specifying a location (pixel-level segmentation) and label per object;
- **Temporal Event Localisation in Videos:** Given a video, specify the temporal location (via start and end times) and label for an event;
- **Temporal Event Detection in Videos:** Given a video, detect all temporal events of interest by specifying temporal locations (start and end times) and labels per event;
- **Spatio-Temporal Event Detection in Videos:** Given a video, detect all events of interest both spatially and temporally by specifying spatial locations (bounding box) and temporal locations (start and end times) and labels per event;
- **Object Tracking in Videos:** Given a video, detect, track and associate all objects of interest by specifying a location, a label and an ID to associate the same object instance across time. This differs from object detection in videos as we want to associate specific object instances across time;
- **Image/Video Captioning:** Given an image/video, describe its contents with a natural language caption (can be a single sentence or an entire paragraph);
- **Dense Image/Video Captioning:** Regions of an image/video have individual captions;
- **Image/Video Visual Question Answering (VQA):** Given an image/video and a question, answer the question with natural language (a word, sentence, or paragraph).

Introduction

The broad scope of this thesis is in relation to the task of **automated video understanding with the utilisation of machine learning techniques**. The concept of *understanding* can be difficult to define and evaluate, particularly in the context of computers. One way of determining understanding is with *description*. If you are able describe something, then at least in some form and to some extent, you have some understanding of it. In the case of video understanding, the problem is posed as - **given a video can the computer describe what happens in the video in the same way that a human would**?

The problem of describing videos is one of the main areas of research in machine learning, called **video captioning**. Algorithms that address the video captioning problem need to be able to perform three key procedures. Firstly, they need to be able to **interpret** the video in some logical way, extracting all of the necessary information. Secondly, they need to be able to **reason** about the relevance of all of the pieces of visual information. Lastly, they need to be able to **express** their interpretation and reasoning in an accurate and compelling human-like way.

Video captioning research has followed many trends from its preceding problem, **image captioning**, where the goal is to describe a single still image. However, it is important to point out that working with video data in comparison to image data is much more difficult for the following reasons:

- **Video data is much larger** - videos take up more storage space, take longer to transfer, take longer to process with algorithms, and require significantly more resources. Many researchers are unlikely to have the resources available to perform large-scale video based analysis and experimentation;
- **Video data is more sparsely informative** - despite their size, videos contain a lot of redundant information, especially between directly adjacent frames, so the majority of the extra data is uninformative. It is especially challenging to extract the important information from the much larger data space occurring in videos;
- **Video imagery is naturally more complex** - the temporal nature allows for object movement resulting in more object occlusions and rarer object poses. Furthermore, object and camera motion leads to artifacts such as blur and rolling shutter, resulting in noisier imagery.

This thesis is split into three chapters, which are discussed in greater detail throughout this introduction:

- **Chapter 1 : Captioning with Concepts**
  We investigate the video captioning process, performing a thorough analysis of both computer and human generated captions. We focus on captioning with the aid of concept prediction to act as an intermediate step between the visual information and caption generation. We experiment with several captioning, concept prediction, and integration techniques to determine means of improved performance.
- **Chapter 2 : Concept Detection & Localisation**
  We perform an extensive experimental analysis of numerous techniques for extending an efficient image object detection pipeline to be suitable for video object detection, as well as constructing a new dataset combining a number of image datasets with a video dataset.
- **Chapter 3 : Fine-Grained Understanding**
  We develop a fine-grained dataset centred around the sport of tennis, and investigate some standard machine learning models' abilities to perform temporal event detection and to generate commentary style captions based on the detected events.

Furthermore, we also include a thorough Appendix which covers:

- **Background** - general theoretical background knowledge to aid in the understanding of concepts, models, and processes.
- **Code** - the implementation details about the software utilised within and written for this thesis.
- **Datasets** - information about the datasets mentioned and used within this thesis.
- **Extended Tables** - extended tables that are summarised in the main thesis text.

Chapter 1 : Captioning with Concepts

One of the key problems in the field of Computer Vision is video captioning, as it provides a relatively useful measure of video understanding. In recent years, the problem has been addressed with the utilisation of machine learning approaches trained on human annotated datasets. See [Chen et al., 2019] for a comprehensive review of approaches up to 2019.

Past approaches address the video captioning problem in a three-stage approach:

- *Stage one* involves the extraction of a set of visual features that represent the video in a more condensed and focused representation. These features are generated by processing the video through a single, or set of, models which are pre-trained on different problem domains such as image [He et al., 2016, Szegedy et al., 2017] and video classification [Karpathy et al., 2014, Xie et al., 2017], or object detection [Ren et al., 2015, Redmon and Farhadi, 2017];
- *Stage two*, which is the main focus of most research in video captioning, involves filtering or transforming the visual features as to best extract key information useful for captioning. Past approaches focus on converging different features over time with mean pooling [Venugopalan et al., 2015b], temporal attention [Yao et al., 2015], or temporal encoders [Venugopalan et al., 2015a, Pan et al., 2016a, Ballas et al., 2016, Zhou et al., 2018]. More recently, approaches focus on spatial region attention [Ma et al., 2018] or the utilisation of memory modules [Wang et al., 2018b, Pei et al., 2019] for better concept recollection. Also, focusing on strengthening the language and visual connection some works utilised semantic embeddings [Pan et al., 2016b, Pan et al., 2017] or Parts-of-Speech (POS) tagging [Wang et al., 2019, Zheng et al., 2020].
- *Stage three*, is the captioning module, which uses the filtered or transformed visual features to generate human like captions word-by-word. Almost all works utilise a Long Short Term Memory (LSTM) Recurrent Neural Network (RNN) model architecture [Venugopalan et al., 2015b] for performing caption generation, with a few also utilising Transformer Networks (TNs) [Zhou et al., 2018, Yu et al., 2019].

Overview

In Section 1.1 we describe the video captioning process in greater detail, taking a comprehensive look at the recent neural network based approaches in relation to the three stages. Using a simple example we look to present the effects of each of the previous works in an easy to understand and accessible manner. We also discuss and analyse the datasets utilised for training and testing these models, and give thorough explanations of the standard evaluation metrics used for evaluating captioning datasets.

In Section 1.2 we implement a set of captioning models which utilise attention over the input features, investigating the proficiency of various network architectures. We also carry out an

in-depth analysis of both the model generated and human annotated captions, identifying the importance of accurate concept detection and identification.

In Section 1.3 we implement and experimentally analyse a number of different concept prediction modules to act as a grounder for the captioning module. We find that with an effective concept detector the captioning performance can be greatly improved, however learning an effective concept detector is challenging, with many attempts distracting the captioning module leading to mixed captioning performance.

Contributions

In Chapter 1 the following contributions are made:

- The design, implementation and investigation of various network architectures for video captioning;
- An in-depth analysis of both model and human generated captions with regards to multiple video captioning datasets;
- The design, implementation and investigation of various novel concept prediction modules for grounding the captioning model.

Chapter 2 : Concept Detection & Localisation

Following on from investigations carried out in Chapter 1, where it is clear that a key to successful captioning and understanding is the correct identification of specific objects, Chapter 2 investigates the problem of object detection in videos. Object detection is a well established problem in both the image and video domains, with video focused models relying on their image based counterparts.

Video based object detector works either focus on performing post-detection box linking or on feature refinement for more accurate box prediction. Focusing on the latter of these, video based detectors utilise image based object detectors such as Faster R-CNN [Ren et al., 2015], R-FCN [Dai et al., 2016], SSD [Liu et al., 2016] and YOLO [Redmon and Farhadi, 2018] at their core. Despite input frames being relatively similar, outputs and representative features in these networks differ greatly across the span of a video, resulting in inconsistent and inaccurate detections. Utilising means such as optical flow [Zhu et al., 2017a, Zhu et al., 2017c, Zhu et al., 2018, Wang et al., 2018c], correlation filters [Feichtenhofer et al., 2017], RNNs [Li et al., 2018, Ning et al., 2017], spatio-temporal hierarchies [Chen et al., 2018a], deconvolutional operations [Bertasius et al., 2018] or relation networks [Deng et al., 2019b], works look to combine features across time to address the problem of inconsistent and deteriorated features. More recently works have utilised memory networks [Deng et al., 2019a, Chen et al., 2020] for greater temporal span while maintaining processing efficiency.

Overview

In Section 2.1 we describe the key methodologies and datasets, as well as give an overview of the evaluation protocols, used for the object detection problem for the still image domain. After performing an experimental analysis of the performance and efficiency of three main model architectures, we chose one and investigate its ability to detect objects in video compared to images.

In Section 2.2 we provide an overview of a number of video domain detection approaches before implementing and analysing a number of modifications to an image based detector towards making it more suited for video processing. We find mostly positive results depending on the particular methodology implemented, with some attaining relatively significant performance gains.

In Section 2.3 we highlight the lacking concept diversity of current video object detection datasets and hence design a new hierarchical class dataset which combines both video and image detection datasets. We train an initial framewise model on this dataset, as well as investigate its concept coverage with video captioning datasets.

Contributions

In Chapter 2 the following contributions are made:

- The design, implementation and investigation of various custom designed network modules for the extension of an image based object detection framework to be utilised video object detection;
- The formation of a large object detection dataset consisting of the combination and organisation of three image based and one video based object detection datasets.

Chapter 3 : Fine-Grained Understanding

While generalised video understanding and captioning is a popular and interesting area of research, it can sometimes be difficult to comprehend its real world application. With the growing interest in machine learning technologies from outside of the computer science and research communities, it is interesting and important to understand the applicability and usability of modern techniques on real world challenges.

Automated sports analytics is an area that has been steadily gaining commercial interest over the past decade, and video understanding techniques applied to the sports domain are particularly interesting. While being human centric and existing in the real world, sports are generally very restricted environments with restricted practices and constrained outcomes. The sport of tennis is a good example of this - two players stand on a marked out court, either side of a net,

and hit a ball with racquets competing to win points. The specialised domain focus is termed *fine-grained* in the research community, and permits more directed research and applicable insights.

There has been a range of more fine-grained video understanding works with relation to cooking [Rohrbach et al., 2012a, Rohrbach et al., 2012b, Rohrbach et al., 2016, Regneri et al., 2013, Rohrbach et al., 2014], shopping [Singh et al., 2016], and other sports such as basketball [Ramanathan et al., 2016]. Our focus is on a new domain - the sport of tennis.

Overview

In Section 3.1 we briefly describe some pre-existing fine-grained datasets related to temporal event detection and captioning. We also briefly discuss some of the approaches implemented to address each of the fine-grained problems.

In Section 3.2 we introduce our new tennis dataset, which is the first dataset to contain both temporal event annotations and coinciding captions. We implement a temporal annotation tool which is released publicly for free and is generalised enough to be used for other problem domains.

In Section 3.3 we implement and investigate the performance of a number of modern temporal event detection and classification architectures, as well as a video captioning pipeline. Although the models are general and not specifically tailored to the tennis domain, other than being trained on the tennis dataset, they are able to provide relatively reliable baselines for future works.

Contributions

In Chapter 3 the following contributions are made:

- The development and public release of a fine-grained video event detection and captioning dataset based on the sport of tennis, as well as a GUI based annotation tool;
- The design, implementation and investigation of a number of network architectures for performance evaluation on the tennis dataset for the problems of hit recognition, and point-level commentary generation.

1 Captioning with Concepts

This chapter investigates several techniques for performing video captioning. Specifically, when provided with a video clip the goal is to **generate a descriptive caption for that clip which describes the contents of the video**. Figure 1.1 presents an example of the video captioning problem.

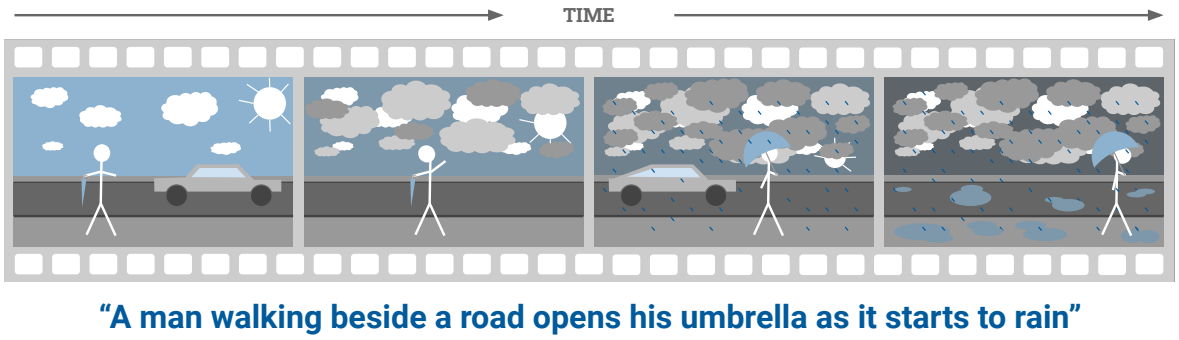


Figure 1.1: An example of the video captioning problem. Provided a video portraying a scene, the goal is to generate a caption that describes the scene.

Generating such a caption is not trivial. If you consider your own thought process in your attempt to caption the video depicted above, you start to realise how much you need to interpret and infer. You firstly need to identify the scene, with the road, the man and the umbrella. You might only be able to discern it's a road due to the cars. You also need to determine that the man is moving, and most likely walking, beside the road. You also take into consideration your prior knowledge, that people normally walk beside roads. You also identify the accumulation of the clouds, the rainfall, and then the formation of the puddles to determine that it's raining. Your prior knowledge, along with the rain, gives further supportive context to your identification of the umbrella which may not have been clearly identifiable earlier in the video.

Captioning was initially presented as a **template matching task**, or based around **retrieval** of the best caption from a pre-defined set. In recent years the problem has shifted more into a **generation task** where individual words are generated one-by-one conditioned on both the imagery and previously generated words. Captioning was initially focused on still images, however it is also becoming prevalent in video. Furthermore, **dense captioning** is a related problem where localised regions of space, and or time, are captioned individually. [Aafaq et al., 2019b] presents a good survey of video captioning methods, including those that existed prior to the wide adoption of neural networks.

1.1 The Video Captioning Process

There are three main stages in all modern video captioning methodologies (Figure 1.2):

1. **Visual Interpretation** - involves transforming the raw video information into a more compact representation, which still maintains all of the important information that may be necessary for generating a caption;
2. **Interpretation Filtering** - involves further filtering and transformation of the representation towards containing information useful for generating a caption; and
3. **Caption Generation** - involves generating a human-like caption based on the filtered representation, and can also provide feedback to the filtering process.

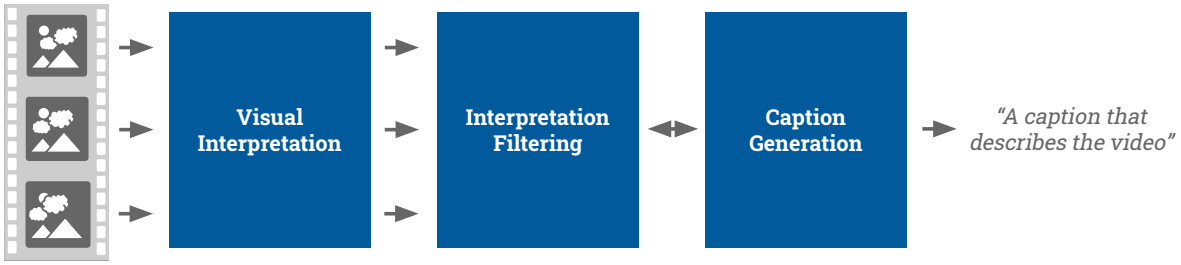


Figure 1.2: The three stages of the video captioning process. The video, consisting of a set of frames, is first passed through a visual interpretation process which encodes the frames into features, which are then further filtered and encoded by a filtering process, before being utilised by the caption generation module, which can also provide feedback information useful for the filtering process.

Given a video input $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, where each frame $\mathbf{x}_t$ ($t = 1 \ldots T$) is denoted by a column vector, the visual interpretation stage uses a set of feature extraction methods to encode each frame into a visual feature vector. In general, the visual feature vector corresponding to the t^{th} frame has the structure $\mathbf{f}_t = \left[\phi^{(\text{f})}(\mathbf{x}_t), \phi^{(\text{m})}(\mathbf{x}_t), \phi^{(\text{r})}(\mathbf{x}_t)\right]^\top$ where $\phi^{(\text{f})}(\cdot)$, $\phi^{(\text{m})}(\cdot)$ and $\phi^{(\text{r})}(\cdot)$ are vector-valued functions responsible for extracting **frame**, **motion** and **region** features. Conceptually, the region feature vector can be partitioned into features corresponding to M different regions so that $\phi^{(\text{r})}(\mathbf{x}_t) = [\mathbf{r}_1, \ldots, \mathbf{r}_M]^\top$.

The collection of visual features are then filtered and encoded by the **interpretation filtering process** which we represent by a function $\varphi(\cdot)$. The exact details of this function are algorithm dependent—most recent intellectual effort has been devoted to architecting $\varphi(\cdot)$ so that the resulting filtered features yield more pertinent information for caption generation. The **caption generation module** ψ uses the filtered features to generate a **sequence of words**. In practice, the caption generation module produces a sequence of probability distributions $(\mathbf{y}_1, \ldots, \mathbf{y}_N)$ of up to length N, where each element $\mathbf{y}_n$ of the sequence is linked with a word in a dictionary $\mathcal{D}$ via a decoding function. The dictionary is a set of words, and the size of the dictionary (the number of elements) is indicative of the vocabulary (the range of concepts the caption generation module can express). At each word generation step, a basic caption generation module relies on the **filtered visual features** $\varphi([\mathbf{f}_1, ..., \mathbf{f}_T])$ and the **past word** $\mathbf{y}_{n-1}$.

An elementary video captioning process can thus be summarised by the relation

$$\mathbf{y}_n = \psi(\varphi([\mathbf{f}_1, ..., \mathbf{f}_T]), \mathbf{y}_{n-1}). \tag{1.1}$$

As the following subsections will demonstrate, this basic video captioning model has been extended in various ways. For example, many methods incorporate information about previously generated caption words into the interpretation filtering process. Similarly, the caption generation process sometimes takes additional parameters which aim to give the caption generation process more contextual information than just its previous word.

1.1.1 Visual Interpretation

Visual interpretation is the process of transforming the raw video into a more compact representation without the loss of important information. We term this compact representation a **feature**, taking the form of a vector in continuous vector space. Features are extracted using one, or multiple Convolutional Neural Network (CNN) based model architectures designed for other tasks such as image classification, video classification and object detection. The specific models for these tasks don't output features, but rather classification labels or detection location coordinates. To extract features, we utilise the output of an intermediate layer of the model, which is contextually rich with information but also compact. Depending on the particular model architecture used, different types of features can be extracted. Such features are **frame features**, **motion features** or **region features**. We discuss each of these feature types in the following subsections. Note that the various feature models are all pre-trained on different datasets depending on their problem domain — we mention the datasets when they are utilised, however don't discuss them in detail as they are out the of scope of this chapter. The datasets which are of interest ie. in the captioning problem domain, are described in Section 1.1.4. Information about all of the datasets can be found in Appendix D.

Frame Features

The frame features extraction model generally takes the form of a 2D image classification CNN, with features extracted from some intermediate layer output near the end of the model prior to it's classification layers. These features contain richly encoded information about the contents of each frame (Figure 1.3).

Table 1.1 summarises the particular frame feature models utilised for each of the captioning works. Over time, video captioning works look to utilise the latest, most accurate, classification models as they generally lead to improved captioning performance via richer features. All of the image classification models are pre-trained on the extensive **ImageNet** classification dataset [Russakovsky et al., 2015], with some being even further fine-tuned on the image based **Visual Genome** dataset [Krishna et al., 2017b], or the video based **ActivityNet** dataset [Caba Heilbron et al., 2015].

Model	Trained On	Fine-tuned On	Captioning Works
AlexNet [Krizhevsky et al., 2017]	**ImageNet**		[Venugopalan et al., 2015b] [Pan et al., 2016b]
VGG-16 [Simonyan and Zisserman, 2014]	**ImageNet**		[Venugopalan et al., 2015a]
VGG-19 [Simonyan and Zisserman, 2014]	**ImageNet**		[Pan et al., 2016b] [Pan et al., 2017] [Wang et al., 2018b]
GoogLeNet [Szegedy et al., 2015]	**ImageNet**		[Yao et al., 2015] [Pan et al., 2017] [Wang et al., 2018b]
Inception-V3 [Szegedy et al., 2016]	**ImageNet**		[Wang et al., 2018b]
Inception-V4 [Szegedy et al., 2017]	**ImageNet**		[Wang et al., 2018a]
ResNet-50 [He et al., 2016]	**ImageNet**		[Baraldi et al., 2017] [Wang et al., 2018b]
ResNet-101 [He et al., 2016]	**ImageNet**		[Pei et al., 2019] [Pan et al., 2020]
ResNet-152 [He et al., 2016]	**ImageNet**		[Gan et al., 2017] [Chen et al., 2018b]
ResNet-101 [He et al., 2016]	**ImageNet**	**ActivityNet**	[Zhou et al., 2019]
ResNet-200 [He et al., 2016]	**ImageNet**	**ActivityNet**	[Zhou et al., 2018] [Zhang and Peng, 2019]
ResNeXt-101 [Xie et al., 2017]	**ImageNet**	**Visual Genome**	[Zhou et al., 2019]
InceptionResNet-V2 [Szegedy et al., 2017]	**ImageNet**		[Aafaq et al., 2019a] [Wang et al., 2019] [Zheng et al., 2020]

Table 1.1: Summary of usages of frame feature model architectures. All of the models are 2D CNN based image classifiers, which are fundamental for methodologies designed for more complex visual interpretation problem domains, including captioning.

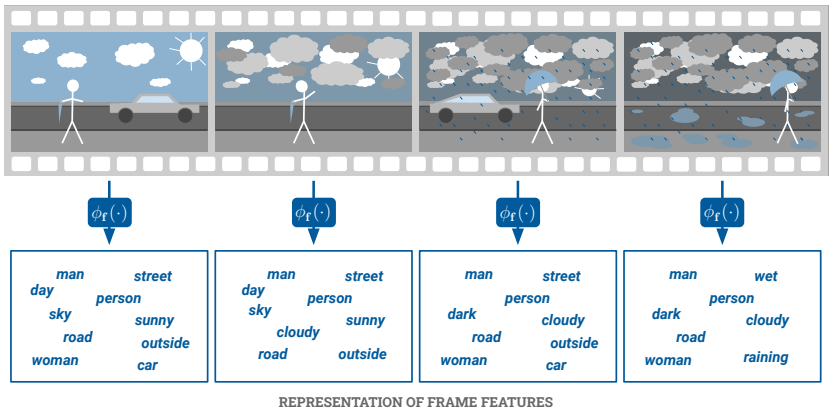


Figure 1.3: The extraction of frame features and their visual representation. Each frame is passed through the feature extraction model $\phi^{(\mathrm{f})}(\cdot)$ to generate a rich feature containing encoded information about the visual contents of that frame (shown in the blue outlined boxes). Note that the feature representations are in a continuous vector space and are not a set of explicit words — the words listed are just for explanatory purposes.

Motion Features

While image classification models can provide insight about the overall scene and how it might change over time, finer motion such as object motion or individual actions aren't captured by such models. With motion and action being particularly important for the video domain, video captioning works have also utilised motion based models, originally designed for video classification and action recognition problems. Compared to the frame feature models which run on each RGB frame individually, motion feature models extract information from multiple input frames to generate their features (Figure 1.4).

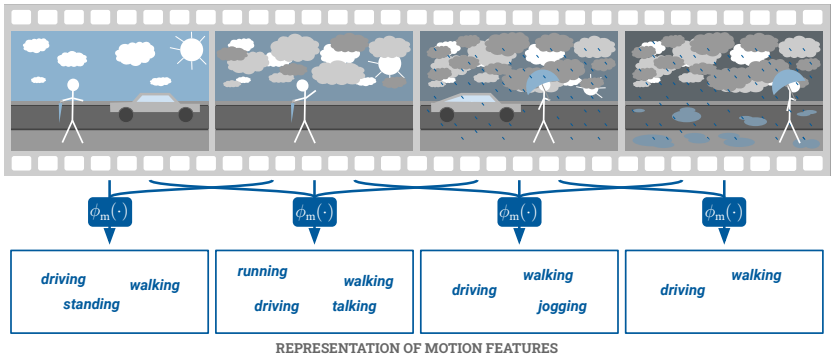


Figure 1.4: The extraction of motion features and their visual representation. Each frame is passed through the feature extraction model $\phi^{(\mathrm{m})}(\cdot)$ to generate a rich feature containing encoded information about the visual contents of the subset of temporally adjacent frames (shown in the blue outlined boxes). Unlike the frame features which contain information about static visual concepts, motion features hold information about the actions and movements across time.

Table 1.2 provides a summary of the different motion feature models utilised by various captioning works. A couple of works consider a **two-stream** approach where they utilise an optical flow (OF) model alongside a standard RGB frame model, combining features extracted from each stream. The OF models are trained on video classification datasets **UCF101** [Soomro et al., 2012] or **ActivityNet** [Caba Heilbron et al., 2015]. Another work uses hand-crafted video features

— **Histograms of Oriented Gradients (HoG)**, **Histograms of Oriented Flow (HoF)**, and **Motion Boundary Histograms (MbH)** [Dalal et al., 2006, Wang et al., 2009], encoding them further with a custom 3D CNN which is trained from scratch with the captioning module. Other works apply different 3D CNN architectures on multiple RGB frames at once. Such 3D CNN models are pre-trained on video classification datasets such as **Sports-1M** [Karpathy et al., 2014] and **Kinetics** [Kay et al., 2017].

Model	Inputs	Trained On	Captioning Works
motion-CNN [Gkioxari and Malik, 2015]	optical flow frames	**UCF101**	[Venugopalan et al., 2015a]
BN-Inception [Ioffe and Szegedy, 2015]	optical flow frames	**ActivityNet**	[Zhou et al., 2018]
3D CNN [Yao et al., 2015]	HoG HoF MbH	*jointly with captioning module*	[Yao et al., 2015]
C3D [Tran et al., 2015]	RGB frames	**Sports-1M**	[Pan et al., 2016b] [Pan et al., 2017] [Gan et al., 2017] [Baraldi et al., 2017] [Wang et al., 2018b] [Aafaq et al., 2019a] [Wang et al., 2019] [Zheng et al., 2020]
I3D [Carreira and Zisserman, 2017]	RGB frames	**Kinetics**	[Wang et al., 2019] [Pan et al., 2020]
3D ResNeXt-101 [Hara et al., 2018]	RGB frames	**Kinetics**	[Pei et al., 2019]

Table 1.2: Summary of usages of motion feature model architectures. Older architectures capture low-level motion information by processing optical flow input frames with a 2D CNN model stream running alongside a RGB stream. More recent approaches consider multiple RGB frames directly, utilising a type of 3D CNN model to capture the temporal changes of adjacent frames.

Region Features

Extracting frame features and motion features from models which consider the entire frame provide information about the global scene dynamics, however captions generally require information about individual objects and their relationships. While features related to particular scene elements can be captured in the frame features, they are not explicit and likely more noisy or conditioned on various scene elements. Therefore recent image and video captioning works also utilise spatial region features from **Regions of Interest (ROIs)** obtained from an object detection models. These models are specifically trained to localise objects, allowing for the extraction of features from explicitly determined salient spatial regions (Figure 1.5).

Table 1.3 provides a summary of the various region feature models utilised by each of the cap-

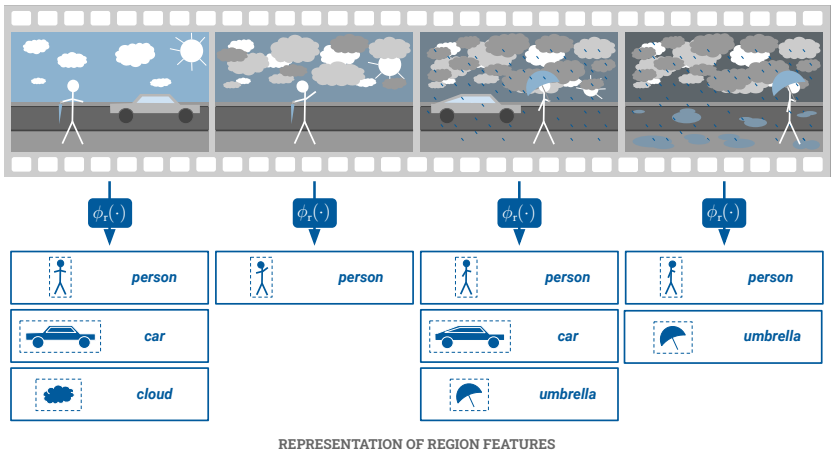


Figure 1.5: The extraction of region features and their visual representation. Each frame is passed through the feature extraction model $\phi^{(\mathrm{r})}(\cdot)$ to generate a set of rich features containing encoded information about specific salient spatial ares of the frame (shown with the sets of the blue outlined boxes). Unlike the frame and motion features which contain information about the global scene, region features contain only information extracted from the salient spatial sub-regions of the entire frame.

tioning works. These ROI generation models are trained using image based object detection datasets such as **MS-COCO** [Lin et al., 2014] and **Visual Genome** [Krishna et al., 2017b], which contain spatial coordinate and class label annotations for a range of different objects. Object detection also rely on image classification models like those seen for generating frame features, using them as backbones for informing their region detection model. The backbone models are all pre-trained on the **ImageNet** classification dataset [Russakovsky et al., 2015]. A couple of the region feature models also utilise a secondary, more powerful **feature model**, to further enrich their extracted ROI image regions. These feature models are also trained on the **ImageNet** dataset [Russakovsky et al., 2015], however the models themselves are larger and provide richer features than the backbone models. Generally, the ROIs with the highest confidence scores are utilised as region features for the captioning model.

1.1.2 Interpretation Filtering

Instead of using the raw visual features, it has been found beneficial to perform further filtering and encoding prior to caption generation. Most of the works in video and image captioning are focused on this task, with the *visual interpretation* and *caption generation* models being adopted from other problem domains such as *classification* and *machine translation* respectively.

In this section we will refer back to the example video depicted in Figure 1.1 and relate what each of the interpretation filtering techniques do in relation to their effects on the visual features and resultant caption generation in relation to the desired caption.

Model & Backbone	Trained On	Feature Model	Captioning Works
Deformable R-FCN [Dai et al., 2016] with a **ResNet-101** backbone [He et al., 2016]	**MS-COCO**	**ResNeXt-101** [Xie et al., 2017]	[Ma et al., 2018]
Faster R-CNN [Ren et al., 2015] with a **ResNeXt-101** backbone [Xie et al., 2017]	**Visual Genome**		[Zhou et al., 2019] [Pan et al., 2020] [Zheng et al., 2020]
Mask R-CNN [He et al., 2017] with a **ResNet-101** backbone [He et al., 2016]	**MS-COCO**	**ResNet-200** [He et al., 2016]	[Zhang and Peng, 2019]
YOLO [Redmon and Farhadi, 2017] with a **DarkNet-19** backbone [Redmon and Farhadi, 2017]	**MS-COCO**		[Aafaq et al., 2019a]

Table 1.3: Summary of usages of region feature model architectures. Region detection models are designed for the problems of object detection or segmentation and are designed to identify and spatially localise a set of salient objects. These models are reliant on the same 2D CNN models used to generate the frame features, using them as backbone networks for generating a rich visual features.

Mean Pooling

The most simplified filtering technique, utilised by [Venugopalan et al., 2015b] on frame features is **mean pooling** over time:

$$\tilde{\mathbf{f}} = \frac{1}{T}\sum_{t=1}^{T}\phi^{(\mathrm{f})}(\mathbf{x}_t) \tag{1.2}$$

generating a single feature $\tilde{\mathbf{f}} \in \mathbb{R}^{d^{(\tilde{\mathrm{f}})}}$ where $d^{(\tilde{\mathrm{f}})}$ is the dimension of $\tilde{\mathbf{f}}$ which is utilised by the caption generation module.

Let us consider what this means for our example video introduced at the start of the chapter. Figure 1.6 visually presents the effect of mean pooling — although features are encoded in continuous vector space, we represent the information they are likely to contain using words below each frame. The effect of mean pooling to the features across time is that the feature vector $\tilde{\mathbf{f}}$ contains a combination of all individual temporal features. Furthermore, features that are expressed in more frames will have a stronger representation in the pooled feature, as can be seen with the concept of `man` being stronger than `raining`. This results in the captioning module focusing on the most common features concepts, generating a caption which includes '`a cloudy day`'. Furthermore despite enabling the model to capture ideas such `raining`, which only appears in the final frame's feature, mean pooling also captures concepts such as `sunny` which, in this case, are likely to be counter-productive towards generating the desired caption.

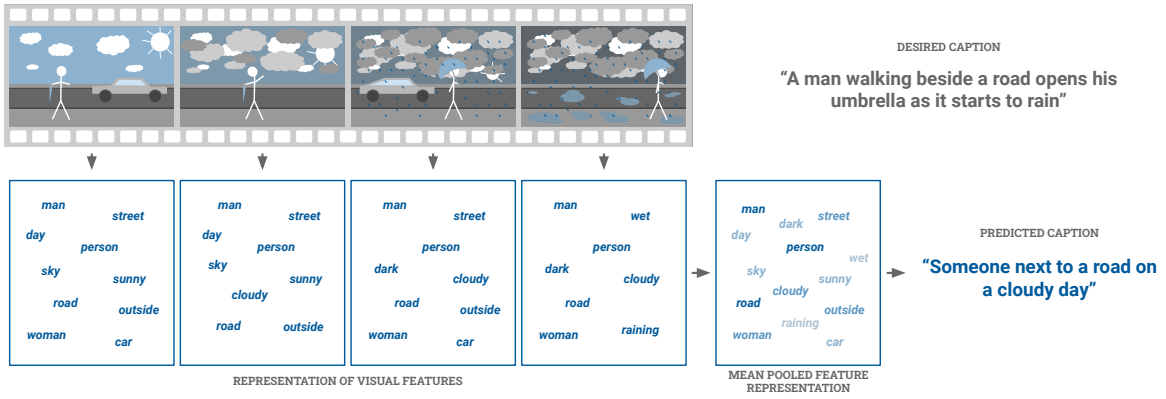


Figure 1.6: Visualisation of mean pooling effects on the visual features and caption. For each frame, there is a corresponding visual feature that contains contextual information about the visual scene (blue boxes). As aforementioned, these features don't explicitly contain the terms `man` or `street`, but rather these are likely what the continuous space vectors $\phi^{(\mathrm{f})}(\cdot)$ likely represent. The effect of temporal mean pooling on the combined feature is that although all concepts are likely represented to some extent, the most frequent concepts are represented the strongest (shown by the shading of the concepts). Therefore, the resulting caption generation is likely to be focused on the most frequent concepts.

Temporal Attention

Instead of trying to capture all of the information across the entire video, as is done in mean pooling, [Yao et al., 2015] introduce a **temporal attention mechanism** which focuses on individual frames when generating each word of the caption. Specifically, they concatenate frame and motion features for each time-step, $\mathbf{f}_t \triangleq \left[\phi^{(\mathrm{f})}(\mathbf{x}_t), \phi^{(\mathrm{m})}(\mathbf{x}_t)\right]^\top$, and calculate a weighted sum of all of these visual features based on the hidden state $\mathbf{h}_{n-1}$ of the caption generation module:

$$\tilde{\mathbf{f}}_n = \sum_{t=1}^{T} \alpha_{t,n} \mathbf{f}_t, \tag{1.3}$$

where the weights $\alpha_{t,n}$ are calculated using a normalising $\mathrm{softmax}$ function

$$\alpha_{t,n} = \mathrm{softmax}(\mathbf{c}_{t,n}^{(\alpha)}) = \frac{\exp(\mathbf{c}_{t,n}^{(\alpha)})}{\sum_{t=1}^{T} \exp(\mathbf{c}_{t,n}^{(\alpha)})} \tag{1.4}$$

and each $\mathbf{c}^{(\alpha)}$ is a **context** vector generated using a small Multi-Layer Perceptron (MLP)[1]:

$$\mathbf{c}_{t,n}^{(\alpha)} = \mathbf{W}^{(\alpha)\top} \tanh\left(\mathbf{W}^{(\mathrm{h})} \mathbf{h}_{n-1} + \mathbf{W}^{(\mathrm{f})} \mathbf{f}_t + \mathbf{b}^{(\alpha)}\right) \tag{1.5}$$

where $\mathbf{W}^{(\alpha)} \in \mathbb{R}^{d^{(\alpha)}}$, $\mathbf{W}^{(\mathrm{h})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\mathrm{h})}}$, $\mathbf{W}^{(\mathrm{f})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\mathrm{f})}}$, $\mathbf{b}^{(\alpha)} \in \mathbb{R}^{d^{(\alpha)}}$ and $d^{(\alpha)}$ is the *attention size* (usually $d^{(\alpha)} = d^{(\mathrm{h})}$).

Once again, considering our example video shown in Figure 1.7, we can visualise the effect that using temporal attention has on our visual feature utilisation. The temporal attention enables the model to focus on the first frame when generating the word `man` (where he may be more clearly visible), while focusing on the final frame when generating the word `rain` (once the puddles have formed). Furthermore, as the inclusion of the motion features allows representation for `walking` in the final three time-steps, these time-steps are likely to be most strongly weighted for the generation of the caption word `walking`.

[1] Read more about Multi-Layer Perceptrons in Appendix B.2

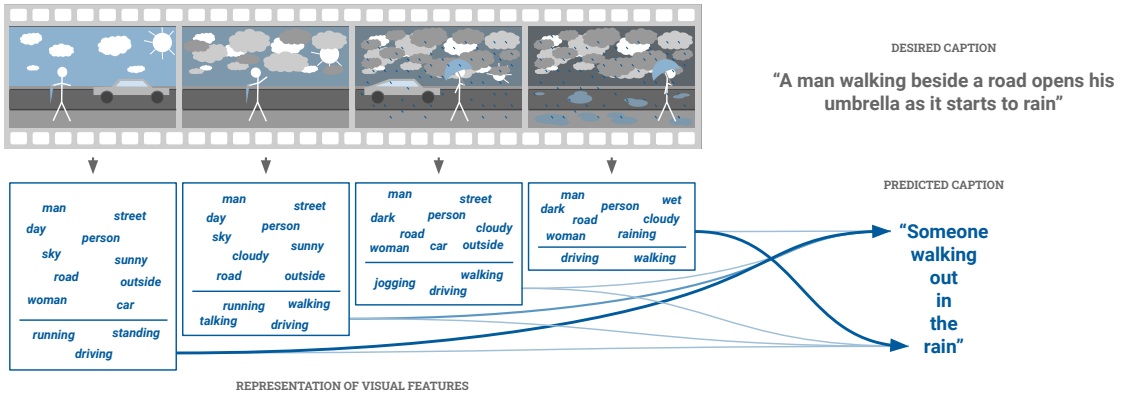


Figure 1.7: Visualisation of temporal attention effects on the visual features and caption. The visual features now contain both information obtained from the frame feature encoding as well as the motion feature encoding. We distinguish these two kinds of features by a horizontal line in each blue box. The temporal attention, conditioned on the state of the caption generator allows it to weight features from different time-steps differently for each word (shown by the different shading and weights of the arrows linking the visual features to the words). Not that we only show the weights for two interesting words, however in practice they are calculated for all words in the caption.

Recurrent Neural Network Encoder

While temporal attention allows the caption generation module to focus on particular temporal positions, it is unable to capture the temporal ordering of the visual features. [Venugopalan et al., 2015a] introduce a Recurrent Neural Network (RNN) model as a **temporal feature encoder**. RNNs are designed to handle sequences of data, encoding and utilising past time-steps when considering the current time-step. This is what makes them the model of choice for the caption generation module—as will be seen in Section 1.1.3. Here the authors use an RNN with Long Short Term Memory (LSTM) units, as they increase the RNN's recollection abilities over longer sequences[2]. Their method works by first passing the sequence of visual features through a LSTM RNN:

$$\mathbf{h'}_t = \text{LSTM}(\mathbf{f}_t, \mathbf{h'}_{t-1}) \tag{1.6}$$

where $\mathbf{f}_t = \phi^{(\text{f})}(\mathbf{x}_t)$ (or $\mathbf{f}_t = \phi^{(\text{m})}(\mathbf{x}_t)$), $\mathbf{h'}_t \in \mathbb{R}^{d^{(\text{h'})}}$. After processing all time-steps, the final hidden state of the encoder $\mathbf{h'}_T$ is used as the initial hidden state $\mathbf{h}_0$ for the caption generation module:

$$\mathbf{h}_0 = \tilde{\mathbf{f}} = \mathbf{h'}_T \tag{1.7}$$

Even with the use of LSTM units, RNNs can still struggle to recover information from significantly earlier iterations. For video it has been found that less than 80 frames is the most optimal for retaining information [Venugopalan et al., 2015a, Yue-Hei Ng et al., 2015], meaning that the final encoding $\tilde{\mathbf{f}}$ may not include earlier important information in longer videos. To somewhat alleviate this issue [Pan et al., 2016a] use a **temporally hierarchical LSTM RNN** to capture information over short and long temporal periods. They stack two RNNs, so the hidden state $\mathbf{h'}_t$ from layer-1 is passed as input into layer-2 after a certain temporal stride s. This significantly reduces the number of LSTM operations between the final hidden state of layer-2 $\mathbf{h'}_T$ and the input features

[2] More information about RNNs, LSTMs and their recollection abilities are discussed in Section 1.2.2

$\mathbf{f}_t$ particularly for small t values, strengthening the recollection of earlier time-steps. [Baraldi et al., 2017] extend this hierarchical encoder to split on **learnt boundary detections** rather than equally spaced strides. Also utilising an RNN encoder is [Ballas et al., 2016] who incorporate convolutional operations into a Gated Recurrent Unit (GRU)[3] RNN unit, producing a stacked **Recurrent Convolutional Network (RCN)**.

Considering the RNN encoder methodologies above with our example video, shown in Figure 1.8, the ability of the RNN model to consider temporal ordering allows the captions to include information about ordered processes seen in the video. Most obviously for our case, unlike the mean pooling or temporal attention, an RNN encoder can determine that the scene went from *sunny* to *rainy*, and not the other way around.

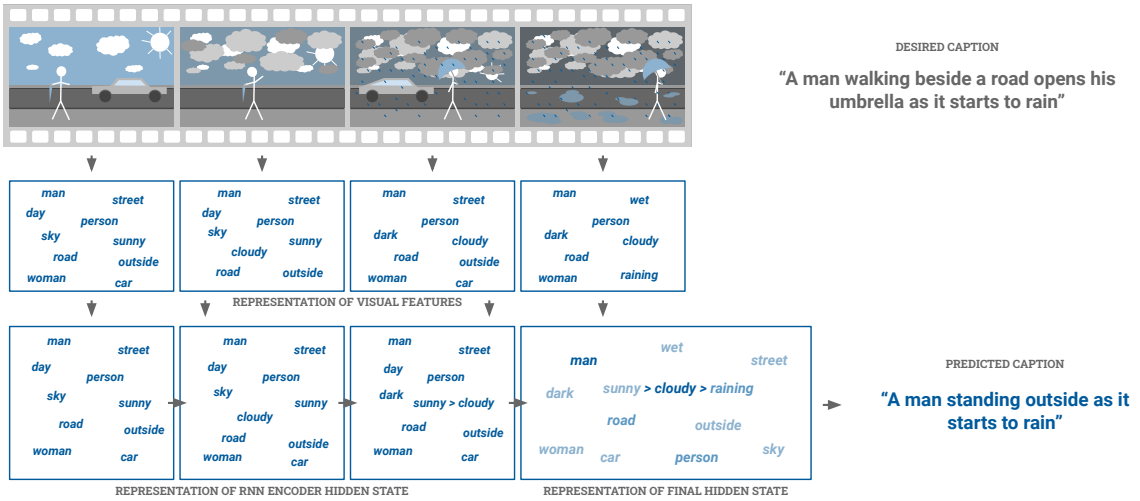


Figure 1.8: Visualisation of the RNN encoder effects on the visual features and caption. Given the visual features for each time-step the RNN encoder encodes each with information from past time-steps. This means that more frequent concepts are likely to have a more significant impact on the caption generator, however now temporal ordering can provide important insights into how the scene changes over time, in this case from `sunny` to `cloudy` to `raining`.

Transformer Network Encoder

Similar to the RNN encoder just described, [Zhou et al., 2018] utilise a Transformer Network (TN) [Vaswani et al., 2017] encoder for encoding visual features across time (Figure 1.9). Unlike the RNN encoder which iteratively evolves its hidden state over time to encode the temporal changes, the TN instead performs several **self-attention** and encoding iterations with global consideration of all of time-steps at once:

$$\tilde{\mathbf{f}}_t = \text{TransformerEncoder}(\phi^{(\text{f})}(\mathbf{x}_t), \left[\phi^{(\text{f})}(\mathbf{x}_1), \ldots, \phi^{(\text{f})}(\mathbf{x}_T)\right]) \tag{1.8}$$

Compared to the RNN encoder, the TN can provide richer features, as they are able to consider all features from past time-steps equally, while still allowing for focus on particular time-steps. Furthermore, the diverse information across time isn't continuously re-encoded into a limited sized hidden state, resulting in greater ability to focus on concepts in earlier time-steps.

[3] Gated Recurrent Units are just another RNN unit type, like the LSTM (more in Section 1.2.2)

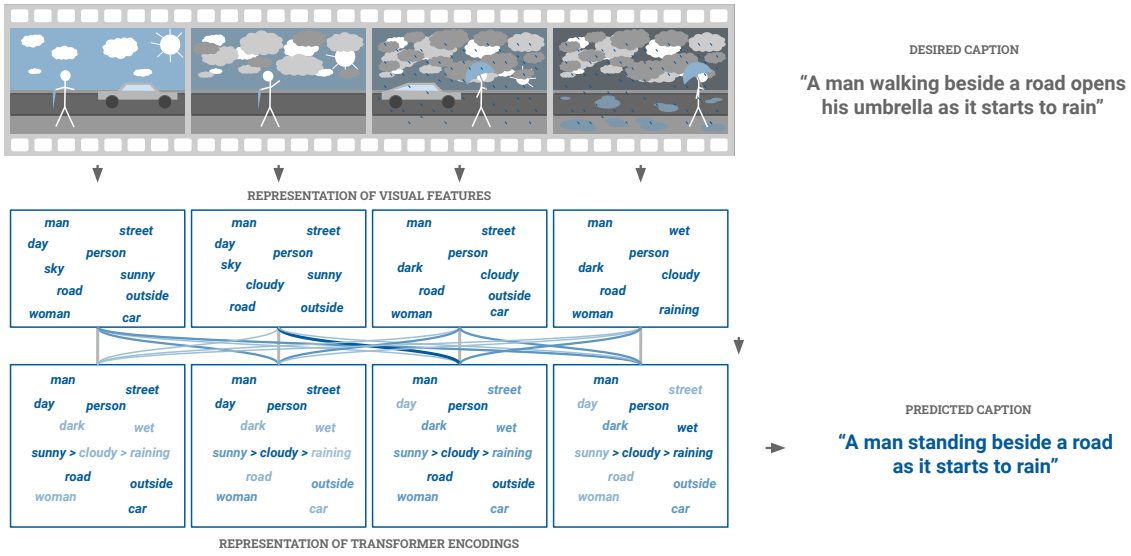


Figure 1.9: Visualisation of the TN decoder and its effects on the caption. Given the visual features for each time-step the TN decoder enriches the visual features by applying self-attention and further encoding. Compared to an RNN which iteratively encodes the time-steps in its hidden state, the TN can take a more global interpretation and combination, considering all time-steps. This allows for the TN to be able to focus on concepts from earlier time-steps while also capturing temporal ordering of events. For example, the concepts `car` and `road` which are strongest in the first feature, are now more strongly linked to the caption generator helping promote the generation of the word `road` in the caption. Furthermore, it allows for the encoding of later concepts in earlier features, permitting for the existence of all concepts available across all time-steps. For example, the transition `sunny>cloudy>raining` now appears across all time-steps, with stronger focus on the concepts which appear at those specific time-steps (shown with shading).

POS Tagging & Detection

To provide more textual context support for each word generation [Wang et al., 2019] introduce a **Parts-of-Speech (POS) tag generator** to guide the caption generation module (Figure 1.10). Their POS tags are processed by the *Stanford Log-linear Part-Of-Speech Tagger* [Toutanova et al., 2003] into 14 categories — verb(VERB), noun(NOUN), adjective(ADJ), adverb(ADV), conjunction(CONJ), pronoun(PRON), preposition(PREP), article(ART), auxiliary(AUX), participle(PRT), number qualifier(NUM), symbol(SYM), unknown(UNK) and the end-of-sentence (EOS) token. They utilise temporal attention and an LSTM RNN to generate the POS tags prior to each word generation.

Attributes Detection

Following on from work addressing image based captioning [You et al., 2016], [Gan et al., 2017, Pan et al., 2017, Zheng et al., 2020] look to jointly **learn attribute detectors** with their captioning modules. Attributes themselves are just a set of the most common words in a captioning dataset. The attribute detectors are implemented as small multi-label classification models which produce a set of confidence scores $\rho^{(\mathrm{a})}$ for the existence of each of the attributes being present in the video:

$$\rho^{(\mathrm{a})} = \sigma(f(\mathbf{f}_t)) \tag{1.9}$$

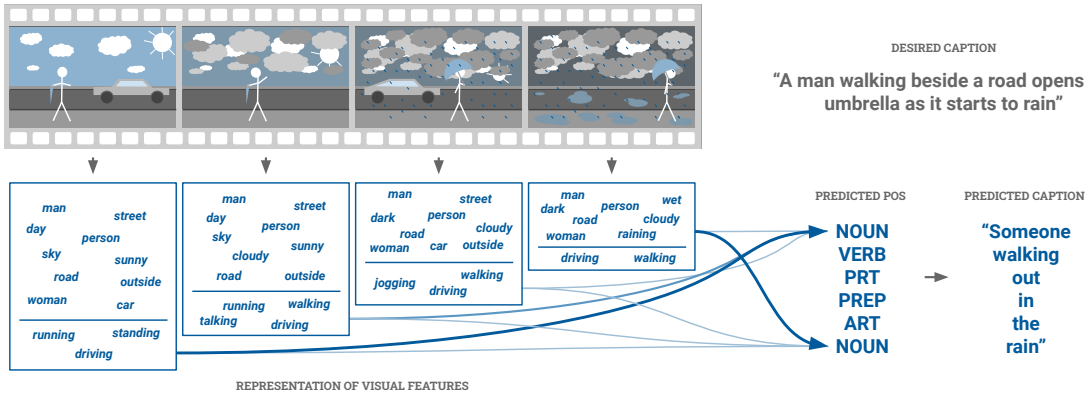


Figure 1.10: Visualisation of the POS detection and its effects on the caption. Extending the idea of temporal attention for not just word generation, here the same temporal attention process is utilised to generate POS tags. The POS are used to guide the caption generation module towards generating a particular type of word at a particular generation step. The caption generator also uses temporal attention directly on the features, but that is omitted for clarity.

where $\sigma(\cdot)$ is the logistic sigmoid function and $f(\cdot)$ is a MLP. Each of the works incorporate the scores into their captioning modules in slightly different ways. Broadly speaking though, they use them to weight and attend to the corresponding *word embeddings*, forming a new feature which persuades the captioning model to generate captions featuring the highest scoring attributes. **Word embeddings** are specific features for particular words, learnt from textual data, and can be jointly fine-tuned with the captioning model.

Figure 1.11 presents a visual representation of this filtering approaches effect on the caption generation. Classifying attributes allows for stronger linking between the visual features and caption word generations due to the explicit nature of the attribute category scoring. However, this can also hinder, more than help, if the attribute detections are noisy and erroneous. Furthermore, with a broad range of attributes, it may force the caption generation model to use more diverse language, which is a challenge for data-driven captioning models.

Region Feature Attention

Similarly to how attention can be applied to visual features across time it can also be used to attend over the region features $\phi^{(\mathrm{r})}(\mathbf{x}_t)$. [Ma et al., 2018, Zhou et al., 2019] both utilise two attention schemes to perform self-attention and caption conditioned attention over region features (Figure 1.12). For the self-attention mechanism they employ **scaled dot-product attention** ($\mathrm{SelfAtt}(\cdot)$), while for captioned conditioned attention they use **alpha-attention** — the same attention used for the temporal attention filtering technique. The self-attention allows the regions to be weighted based on the other regions for each frame, strengthening the relationship representations between regions, allowing regions to support other regions. The caption conditioned attention strengthens the link between visually salient image regions and generated caption words, promoting better visual support for particular words.

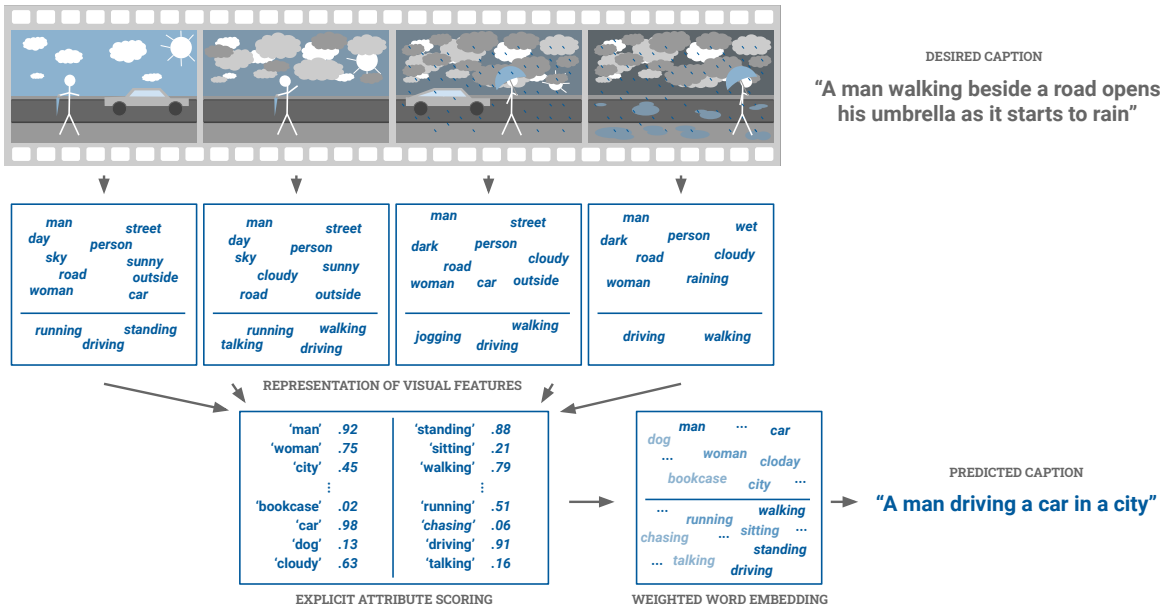


Figure 1.11: Visualisation of attribute detection and its effects on the caption. Given the visual features for each time-step a classifier is learnt to determine the probabilities in which a particular attribute exists in the video. In the case that frame and motion features are used, attribute classifiers are learnt for each type of feature. Here the attributes are explicitly categorised so are presented as *quoted non-italicised* words. The attribute scores are used to weight word embeddings to encourage the caption generator to produce the most common attributes in the caption. In this case the combination of high scores for `car` and `driving` being scored slightly higher than `walking` results in the caption suggesting the `man` is `driving`.

$$\tilde{\mathbf{f}}_{t,n} = \sum_{m=1}^{M} \alpha_{t,m,n} \text{SelfAtt}(\mathbf{r}_m, \phi^{(\text{r})}(\mathbf{x}_t), \phi^{(\text{r})}(\mathbf{x}_t)) \tag{1.10}$$

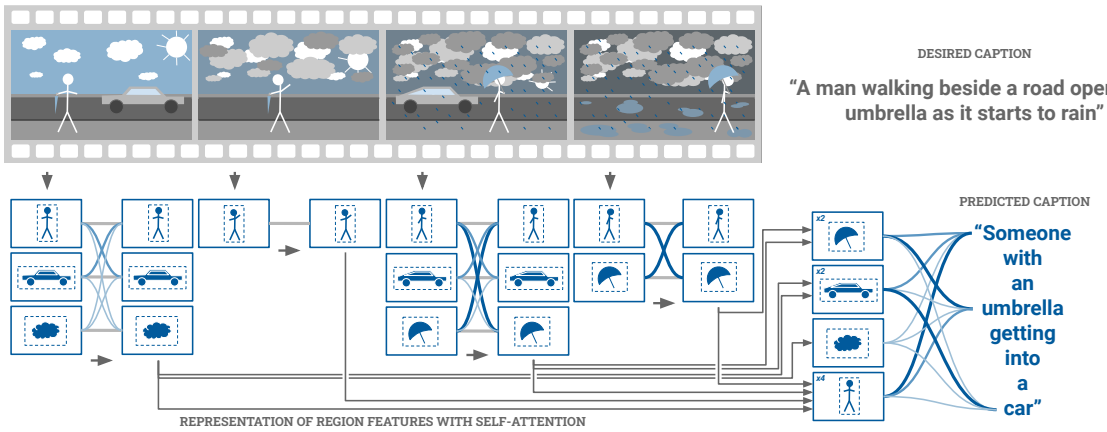


Figure 1.12: Visualisation of region attention and its effects on the caption. Given the region features for each time-step, a dot-product self-attention is applied to weight each of the regions in relation to themselves for the particular frame. The culmination of these attended regions are then attended to again conditioned on the current caption generation state. The self-attention strengthens the relationships between specific regions, with regions supporting one another (there are stronger weights between `man` and `umbrella`). The caption conditioned attention strengthens the support for particular caption words with specific salient image regions (there are stronger weights between words and the regions which contain what the word describes).

Semantic Embedding Alignment

Somewhat disjointed from the aforementioned filtering approaches, [Pan et al., 2016b] use a **visual-semantic embedding** to further encode the visual features into a vector space where they are more closely aligned with an embedding of the desired caption (Figure 1.13). Specifically, they generate a visual embedding $\tilde{\mathbf{f}}$ and a sentence embedding $\tilde{\mathbf{s}}$ with transformation matrices $\mathbf{T}^{(\mathrm{f})}$ and $\mathbf{T}^{(\mathrm{s})}$ that are jointly learnt with the caption generation module:

$$\tilde{\mathbf{f}} = \mathbf{T}^{(\mathrm{f})} f^{(\mathrm{f})}(\left[\phi^{(\mathrm{f})}(\mathbf{x}_1), \ldots, \phi^{(\mathrm{f})}(\mathbf{x}_T)\right]) \text{ and } \tilde{\mathbf{s}} = \mathbf{T}^{(\mathrm{s})} f^{(\mathrm{s})}([\mathbf{y}_1, \ldots, \mathbf{y}_N]) \tag{1.11}$$

where $f^{(\mathrm{f})}(\cdot)$ is some function to compact the temporal features to a vector eg. mean pooling, and $f^{(\mathrm{s})}(\cdot)$ is some function which encodes the ground truth caption $[\mathbf{y}_1, \ldots, \mathbf{y}_N]$ into a continuous vector. During training of the embedding matrices they look to minimise the squared Euclidean distance $\left\|\tilde{\mathbf{f}} - \tilde{\mathbf{s}}\right\|_2^2$ between the visual and sentence embeddings.

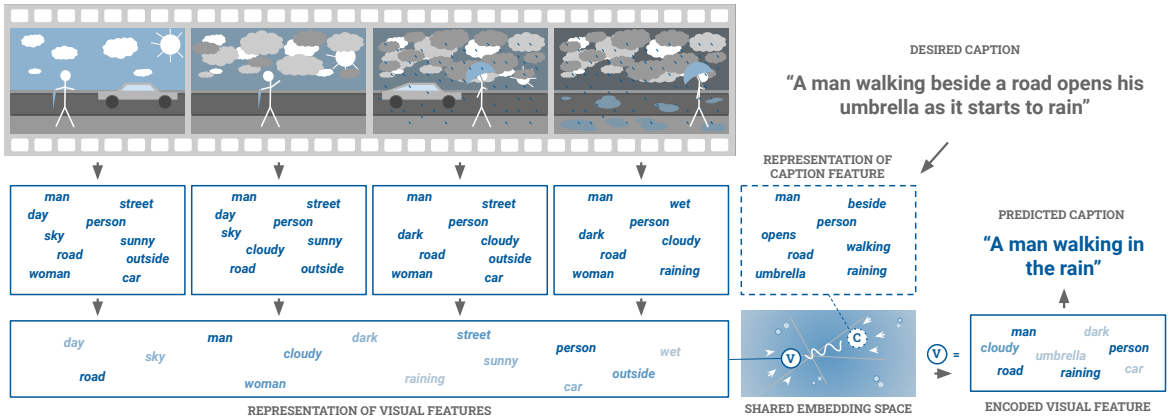


Figure 1.13: Visualisation of region attention and its effects on the caption. During training of this captioning model the visual features for the entire video are embedding into a shared space with an embedding of the desired caption. The model then learns how to embed the visual features such that they more closely aligned with the caption features in the shared space *ie. are closer together in feature space*. When generating new captions the new visual feature encoding process is more able to distinguish between visual features allowing for more precise caption generation. In the case here, `raining` in the original features wasn't very dominant, however after being encoded it is much stronger and is hence utilised in the caption.

Memory Modules

Utilising region attention, semantic attributes or semantic embeddings do provide improvements towards the semantic connection between the visual and textual modalities, however due to the diversity of language and visual concepts, captioning methods still struggle to identify, represent and generate many various concepts. To further improve the recollection of the captioning generator some works have utilised memory models [Weston et al., 2014, Sukhbaatar et al., 2015] which are able to store more explicit information relating to a broader range of concepts [Wang et al., 2018b, Pei et al., 2019] (Figure 1.14). Using attention mechanisms and feature transformations the memory can be read from, and written to, by with visual feature information and caption information. Furthermore with the explicit storing of concepts across different

video samples, it is able to encode richer features and promote the generation of rarer concepts which would get ignored in other feature encoding representations, such as the hidden state of an RNN encoder.

[Wang et al., 2018b] utilise a bi-modal (shared visual and textual) memory which gets written to, and read from, by both a temporal attention mechanism over the visual features, and the caption generation module. The order of processes are first the caption generation writes to the memory, then the temporal attention reads from the memory, then the temporal attention writes to the memory, before lastly the caption generation module reads from the memory. [Pei et al., 2019] also utilise a memory module for better retention of rarer words across training data, strengthening the caption models ability to generate more diverse captions. The memory structure in this case is more structured, with a memory entry for each word in the vocabulary with a corresponding descriptive feature composed of visual context information, word embeddings and optional auxiliary features.

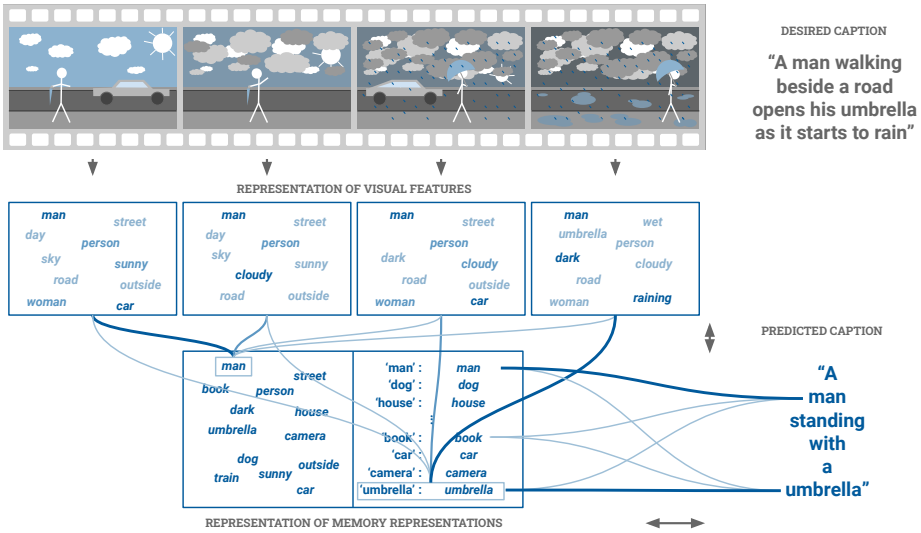


Figure 1.14: Visualisation of memory modules and their effects on the caption. The individual memory modules are represented side by side, with that of [Wang et al., 2018b] on the left, and that of [Pei et al., 2019] on the right. The memory module allows for improved direct access to features related to explicit concepts. The concepts represented with the visual features are now shaded to represent their presence and clarity within the features ie. the concept `man` is likely the most impactful part of the visual features. In this case, `umbrella` would be a relative rare concept within the dataset, resulting in it being more difficult to identify and represent in the visual features — it's shaded lightly in the final feature suggesting it's there but its presence is weak. With the explicit encoding of `umbrella` in the memory however, the concept is able to be better determined in the visual feature inputs, and hence generated in the caption.

1.1.3 Caption Generation

Based on the filtered features, the caption generation stage utilises another neural network to generate captions one word at a time. Just as videos are sequences of frames, captions are similarly sequences of words. For this sequence task two architectures are adopted by captioning works — **Recurrent Neural Networks (RNNs)** and **Transformer Networks (TNs)**. We briefly describe how they each work, how they incorporate the filtered features, and how they can pro-

vide feedback to the feature filtering process.

The collection of textual words (made up of letters) in the dictionary $\mathcal{D}$ need to be transformed into a continuous vector space. This is achieved in two steps: (1) *one-hot encoding* and (2) *word embedding*. **One-hot encoding** involves turning each word in the vocabulary into a binary vector via a mapping $\omega\colon \mathcal{D} \to \{0,1\}^{|\mathcal{D}|}$, which yields a length-$|\mathcal{D}|$ vector (one dimension for each word in the dictionary) consisting of zeros in all dimensions with the exception of a single 1 at the k^{th} dimension, corresponding to the k^{th} word. To lower the dimensionality of the one-hot vector, and to map it from a discrete to continuous representation, one transforms the vector into a **word embedding** space:

$$\mathbf{e}_k = \mathbf{E}\omega(y_k) \tag{1.12}$$

where $\mathbf{e}_k$ is the embedding vector for the k^{th} vocabulary word, $\mathbf{E} \in \mathbb{R}^{d^{(\text{e})} \times |\mathcal{D}|}$ is an embedding matrix, $d^{(\text{e})}$ is the dimension of the embedding space and $y_k \in \mathcal{D}$ is the k^{th} dictionary word. Word embeddings are generally trained using textual datasets that place contextually similar words near to one another in the embedding space. Two popular pre-trained word embeddings are **GloVe** [Pennington et al., 2014] and **Word2Vec** [Mikolov et al., 2013], but they can be learnt or fine-tuned with the captioning model training process.

Included in the vocabulary are three special tokens useful for the generation task. The *beginning of sentence* token `<BOS>` is used for the input for the first word generation ie. for $\mathbf{e}_0$. Similarly, the *end of sentence* token `<EOS>` is used to identify where the caption ends and where the captioning model should halt generation. Finally, the *unknown* token `<UNK>` is used in ground truth captions when the particular word or symbol is not found to be present in the vocabulary, which may be limited to words that are present in some threshold number of captions.

Furthermore as all captions are not the same length, the maximum sequence length N is specified for each model. Captions which are longer than this N will get their ends cut off, while any that are shorter will be **padded** up until N, where padding is using all zeros for the embedding vector.

Recurrent Neural Network Decoder

We start with the more common of the two, the Recurrent Neural Network (RNN) architecture (Figure 1.15). As their name suggests, RNNs are **recurrent** by nature, as they iteratively generate words $\mathbf{y}_n$ ($n = 1, \ldots, N$). They use the embedding for the previously generated word $\mathbf{e}_{n-1}$, their own hidden state $\mathbf{h}_{n-1}$ and the filtered visual features $\tilde{\mathbf{f}}_n$ via the relation

$$\begin{bmatrix} \rho_n \\ \mathbf{h}_n \end{bmatrix} = \text{RNN}(\mathbf{h}_{n-1}, \mathbf{e}_{n-1}, \tilde{\mathbf{f}}_n), \tag{1.13}$$

where ρ_n is a probability distribution over the vocabulary, from which the most probable word is chosen as $\mathbf{y}_n$.

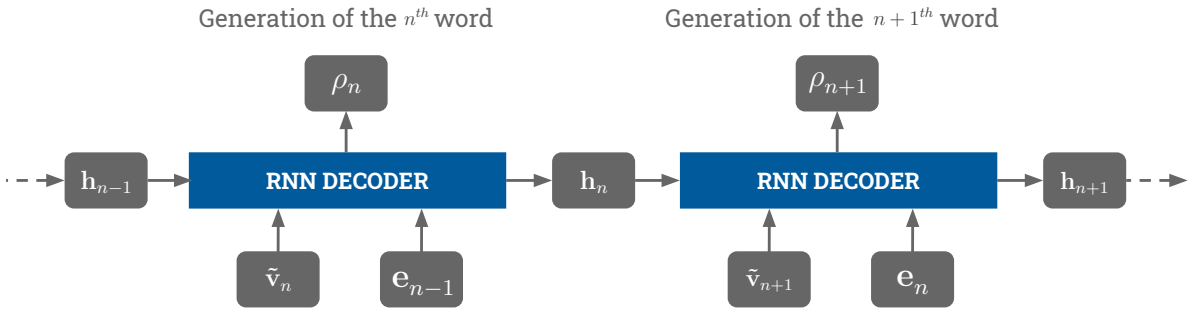


Figure 1.15: Simplified visualisation of a Recurrent Neural Network. For the generation of the n^{th} word the filtered visual features $\tilde{\mathbf{f}}$ are input along with the embedding of the previous word $\mathbf{e}_{n-1}$. Furthermore, the hidden state from the previous word generation $\mathbf{h}_{n-1}$ is recurrently passed through the RNN, providing the contextual history of past words.

Following trends in image captioning [Donahue et al., 2015, Vinyals et al., 2015], and originally machine translation tasks [Cho et al., 2014, Bahdanau et al., 2015, Sutskever et al., 2014], almost all video captioning methods adopt an RNN as a decoder. However the standard RNN just described has difficulty retaining temporally distant information in its hidden state over longer sequences. To alleviate this issue to some degree it's common practice to use gating-based **Long Short Term Memory (LSTM)** [Hochreiter and Schmidhuber, 1997] or **Gated Recurrent Unit (GRU)** [Cho et al., 2014] units. [Venugopalan et al., 2015b, Venugopalan et al., 2015a, Rohrbach et al., 2015a, Yao et al., 2015, Pan et al., 2016a, Zanfir et al., 2016, Pan et al., 2016b, Pan et al., 2017, Gan et al., 2017, Ma et al., 2018, Zhou et al., 2019, Wang et al., 2019, Zheng et al., 2020] all utilise LSTM units, while [Zhang and Peng, 2019, Aafaq et al., 2019a] utilise GRU units. The inner workings of the RNN captioning pipeline will be described in more detail in Section 1.2.2

Transformer Network Decoder

A more recently introduced method for dealing with sequences is the Transformer Network (TN) architecture (Figure 1.16) [Vaswani et al., 2017]. TNs have been utilised for captioning in both image [Herdade et al., 2019] and video based works [Zhou et al., 2018, Yu et al., 2019]. The TN is an attention based method utilising numerous dot-product attention mechanisms and MLP encoding operations. Unlike the RNN that only sees the previously generated word $\mathbf{y}_{n-1}$ and is dependent on its hidden state to hold past information, the TN performs attention over **all** previously generated words at once when generating new words. Furthermore, with the use of masking during training, TNs can be trained in parallel across N words, unlike RNNs that rely on each different $\mathbf{h}_{n-1}$, resulting in TNs being more efficient. Similar to the RNN models the TN outputs a probability distribution over the vocabulary:

$$\rho_n = \text{TransformerDecoder}([\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], \tilde{\mathbf{f}}_n) \tag{1.14}$$

The inner workings of the TN decoder pipeline, as well as the training and inference process, will be described thoroughly in Section 1.2.2.

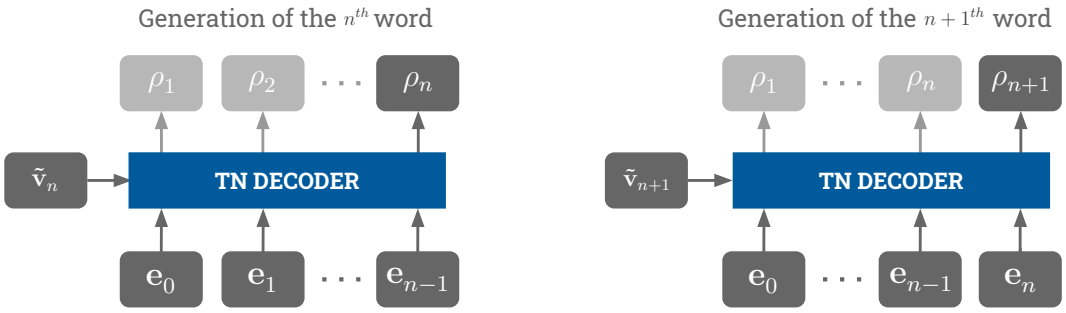


Figure 1.16: Simplified visualisation of an Transformer Network. For the generation of the n^{th} word the filtered visual features $\tilde{\mathbf{f}}$ are input along with the embeddings of all of the previous words $[\mathbf{e}_1, \ldots, \mathbf{e}_{n-1}]$. Note how $\mathbf{e}_0$ is passed in as initialisation (the same is done for the RNN), recall that this is an embedding of the `<BOS>` token. By inputting all of the previous word generations we don't reacquire holding the history in a hidden state. Also note that at every n^{th} word generation, word probabilities ρ are calculated for every step, with all but ρ_n being disregarded.

Beam Search

Beam Search is a *search* strategy for improving caption generation quality by comparing the cumulative probabilities of various sequences of word generations. At each word generation RNNs and TNs base their prediction on past words. If only the most probable word at each iteration is considered (a *greedy* approach), the captioners may push themselves into an suboptimal position due to errornous earlier word generations. Beam Search looks to alleviate this to some extent by taking the top $k^{(\text{b})}$ scoring word sets at each iteration, where $k^{(\text{b})}$ is considered the beam size. Note that this isn't taking the top $k^{(\text{b})}$ words at each iteration, but top combination of words. For example, for a beam size of $k^{(\text{b})} = 3$, say the top words at $n = 1$ are ['*I*', '*Me*', '*They*'], then at $n = 2$ the top combinations might be ['*I have*', '*I hold*', '*They have*'], so the word '*Me*' from the first iteration is discarded.

1.1.4 Datasets

All of the model architectures mentioned above require data for training and evaluation. Over the years there have been several datasets released that are designed specifically for the task of video captioning (see Table 1.4 for a summary).

MSVD [Chen and Dolan, 2011] is a generalised video captioning dataset containing YouTube clips annotated with Amazon Mechanical Turk (AMT)[4], with multiple people hired to write captions for each clip. **MSR-VTT** [Xu et al., 2016] is similar to MSVD, containing relatively short clips which are quite object centric, however it contains over eight times the video data seen in MSVD.

ActivityNet Captions is an extension of the **ActivityNet** action detection dataset [Caba Heilbron et al., 2015] and was introduced to be more action centric, with longer clips suitable for temporally dense captioning. Recently the ActivityNet and it's associated Captions extension have been extended again with **ActivityNet Entities** [Zhou et al., 2019]. Similar to **Flickr 30k Entities**

[4] https://www.mturk.com/

Dataset	Year	Context	Cap Src	# Clips	# Hours	# Caps	# Words	Vocab	# Obj. Inst.
MSVD	2011	General	AMT	1970	5.3	70028	607339	13010	0
MPII-MD	2015	Movie	DVS, Script	68337	73.6	68375	653467	24549	0
M-VAD	2015	Movie	DVS	48986	84.6	55905	519933	18269	0
MSR-VTT	2016	General	AMT	10000	41.2	200000	1856523	29316	0
ActivityNet	2017	Action	AMT	20000	849	37421	504234	11006	158000

Table 1.4: Summary of video captioning datasets. We show the year of release, the conext of the videos, the captioning source, and the numbers of clips, hours, captions, words, unique words and object instance annotations. Amazon Mechanical Turk (AMT) is an online labelling service and Descriptive Video Services (DVS) are used to describe movies to visually impaired viewers. As the years have progressed the datasets have become larger and more comprehensive. Links to each of the datasets are listed in Appendix D.

[Plummer et al., 2015], this grounds caption nouns in the video with bounding boxes and labels from 432 categories.

MPII-MD [Rohrbach et al., 2015b] and **M-VAD** [Torabi et al., 2015] use clips from Hollywood films, with captions sourced from Descriptive Video Services (DVS) and scripts. The captions in these sets are less visually descriptive, and are more story and character centric.

The focus of this work is on generalised video understanding and therefore will not consider using the film datasets, MPII-MD and MVAD. Furthermore, due to the length of the clips in ActivityNet being approximately 150 seconds long on average we leave this for future work. MSVD and MSR-VTT are the most commonly used datasets with clip lengths of around 10-15 seconds, and therefore are the two datasets we utilise for training and evaluation of our models.

Noun & Verb Analysis

To attain a deeper understanding of the makeup of these two datasets, we perform an analysis on the noun and verb occurrences according to the Python Natural Language Toolkit (PNLTK)[5]. We investigate two main properties, *firstly* **how many nouns and verbs are used per caption**, and *secondly* **the variation of nouns and verbs across captions of the same video**.

Let us consider the count statistics for the total word occurrences (Table 1.5) and the unique word occurrences (Table 1.6). On a per caption basis, it can be seen there is basically no change, with on average two or three nouns and none or one verb per caption for MSVD and MSR-VTT respectively. This suggests the captions are relatively brief and focused on only a couple of concepts. If we instead compare the total occurrences with the unique occurrences on a per video basis it can be seen that there are significant differences with approximately less than half of the words being unique. This is a result of captions which describe the same video, containing similar language. If we compare the unique per video counts with the unique per caption counts however, we can also identify that there is also variation in how, or what, is described in relation

[5] https://www.nltk.org/

to the concepts. This natural human variation in how to interpret and caption a video is one of the challenges that computers also face.

Dataset	Nouns	Verbs	Nouns/Video	Verbs/Video	Nouns/Caption	Verbs/Caption
	# (%)	# (%)	min | μ | max	min | μ | max	min | μ | max	min | μ | max
MSVD	116724 (34%)	48218 (14%)	37 | 97 | 194	3 | 40 | 105	0 | 2 | 12	0 | 0 | 6
MSR-VTT	409525 (33%)	158164 (13%)	29 | 62 | 122	3 | 24 | 65	0 | 3 | 17	0 | 1 | 9

Table 1.5: Noun and verb statistics for the MSVD and MSR-VTT datasets. We present the total occurrences for nouns and verbs, as well as the minimum, maximum and mean (μ) counts of the nouns and verbs per video and per caption. For both MSVD and MSR-VTT nouns are about twice as common as verbs, with approximately 30% and 15% of words for both sets, being classified as nouns and verbs respectively. The mean occurrences of 2-3 nouns and 0-1 verbs per caption suggest relatively brief and focused captions.

Dataset	Nouns	Verbs	Nouns/Video	Verbs/Video	Nouns/Caption	Verbs/Caption
	# (%)	# (%)	min | μ | max	min | μ | max	min | μ | max	min | μ | max
MSVD	6111 (64%)	3413 (36%)	4 | 23 | 54	2 | 13 | 37	0 | 2 | 12	0 | 0 | 6
MSR-VTT	15091 (63%)	8568 (36%)	6 | 27 | 66	0 | 13 | 36	0 | 3 | 16	0 | 1 | 9

Table 1.6: Unique word (vocabulary) statistics for the MSVD and MSR-VTT datasets. We present the occurrences for the *unique* nouns and verbs, again showing the minimum, maximum and mean numbers per video and per caption. If we compare the per video and per caption counts, where there are approximately 10$\times$ the unique concepts per video than per caption, we can determine that there is natural variation of the human annotations in the same video.

Taking a closer look at the most common nouns and verbs in each dataset (Figure 1.17) we can get a clearer idea of what kinds of videos can be found in each set, and what concepts should be best understood for good captioning outcomes.

1.1.5 Evaluation Protocol

Letting $\mathcal{D}$ be a dictionary of words, a length-N caption is a sequence of words $(\omega_n)_{n=1}^{N}$, where $\omega \in \mathcal{D}$. For each video there can be several corresponding ground truth (GT) captions, where the number of captions per video may vary. Hence, for the i^{th} video we associate a set of **reference** (GT) captions $\mathcal{C}_i = \left\{ \{(\omega_n)_{n=1}^{N_j}\}_{j=1}^{J_i} \right\}$. Similarly for each video we have the predicted **candidate** caption $\hat{c}_i = \left\{ (\hat{\omega}_n)_{n=1}^{\hat{N}_i} \right\}$.

BLEU

BLEU [Papineni et al., 2002] utilises a modified form of precision for n-gram matches between reference and candidate captions. An **n-gram** is a continuous sequence of N words. For example 'a dog walks' has three 1-grams ('a', 'dog', 'walks'), two 2-grams ('a dog', 'dog walks'), and one 3-gram ('a dog walks'). A BLEU score is calculated per n-gram length, with lengths normally ranging from 1 to 4.

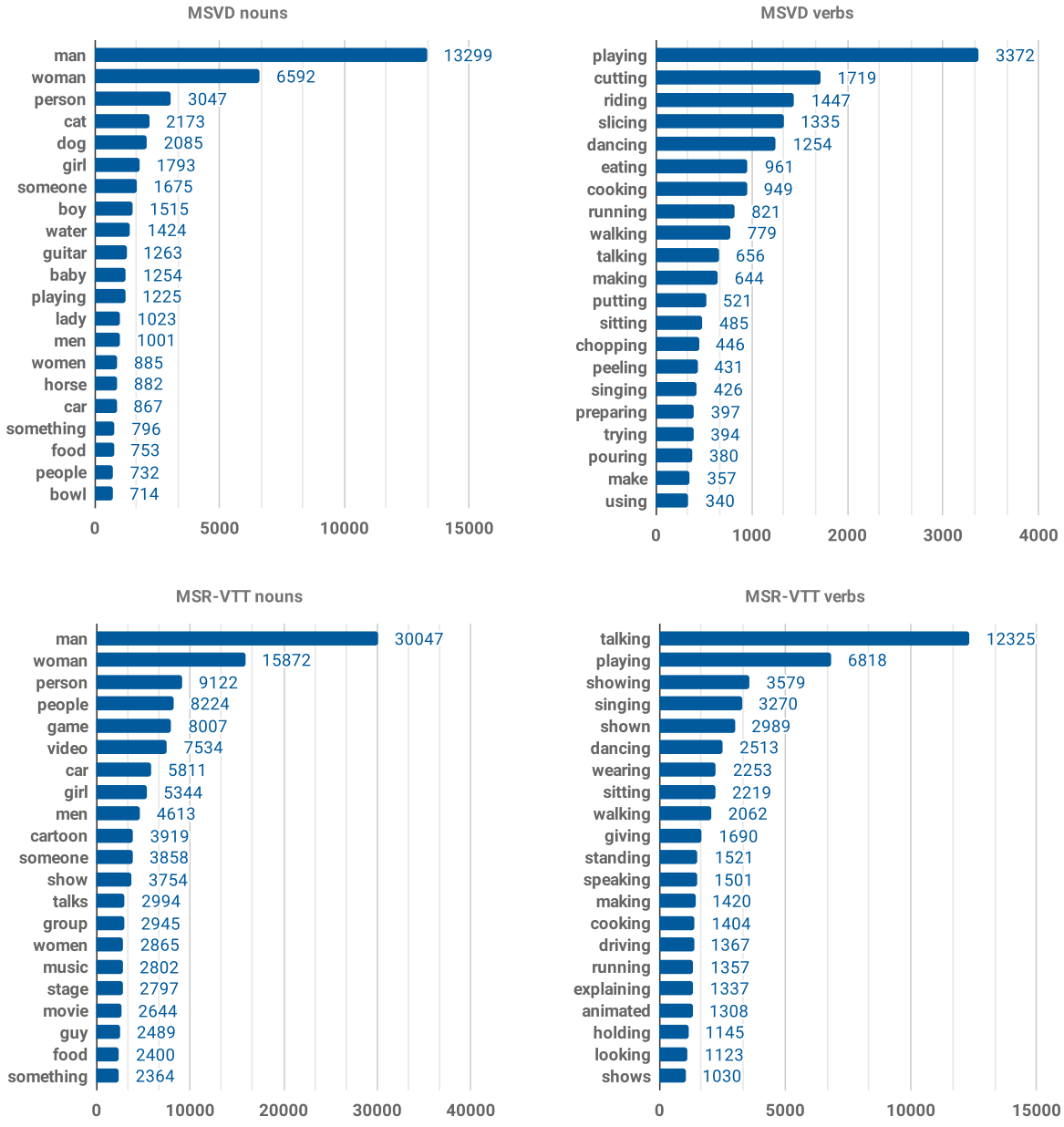


Figure 1.17: The 20 most common nouns and verbs for the MSVD and MSR-VTT datasets. Top left is the MSVD nouns, top right the MSVD verbs, bottom left the MSR-VTT nouns and bottom right the MSR-VTT verbs. For both sets it can be seen that people related nouns such as `man`, `woman`, `person`, `girl`, `boy`, `someone` are the most common, with most videos consisting of a person interacting with one or two other objects. Furthermore, you can see how most words are infrequent, resulting in a *long-tail* distribution. For MSVD cooking related verbs `cutting`, `slicing`, `eating`, `cooking`, `chopping`, `peeling`, `pouring` occur relatively often as there are a significant number of cooking videos. For MSR-VTT `talking`, `showing` occur relatively frequently reflecting many tutorial style videos in the dataset, along with `playing` and the highly occurring nouns `game` and `video`, relating to the many videos of computer gameplay recordings.

We represent the set of n-grams with length N in $\hat{c}_i$ as $\widehat{\mathcal{N}}^{(\mathrm{N})}$. We also define the function $f(\cdot)$ to count the number of times each n-gram $\hat{\nu} \in \widehat{\mathcal{N}}^{(\mathrm{N})}$ exists in either a candidate caption $\hat{c}_i$ or reference caption $c_j^{(i)} \in \mathcal{C}_i$ $(j = 1 \dots J_i)$. Then for the n-gram length of N and i^{th} sample, a precision score $p_i^{(\mathrm{N})}$ is calculated:

$$p_i^{(\mathrm{N})} = \frac{\sum_{\hat{\nu} \in \widehat{\mathcal{N}}^{(\mathrm{N})}} \min\left(f(\hat{\nu}, \hat{c}_i), \max\{f(\hat{\nu}, c_1^{(i)}), \dots, f(\hat{\nu}, c_{J_i}^{(i)})\}\right)}{\sum_{\hat{\nu} \in \widehat{\mathcal{N}}^{(\mathrm{N})}} f(\hat{\nu}, \hat{c}_i)} \tag{1.15}$$

In other words, precision is the sum of the counts for all candidate n-grams clipped with their maximum counts in the reference captions, divided by the sum of their counts in the candidate caption. For example, the precision for the candidate caption '*through a park a dog walks in a park*' and reference caption '*a dog wanders into a park*' for 2-grams is 0.286 (see Figure 1.18).

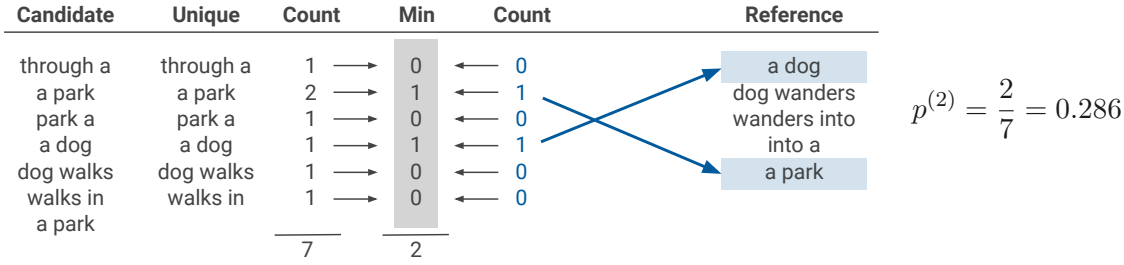


Figure 1.18: An example of the BLEU precision procedure. Assuming we are working with an n-gram length of 2, we pair up adjacent words in both the candidate and the reference captions. We then count how many times the unique pairs from the candidate appear in the candidate and clip it (take the $\min$) with the occurrence count of the same pair in the reference caption. These clipped counts are then divided by the total number of pairs to determine the precision $p^{(2)}$.

With $T^{(\text{pred})}$ denoting the total number of predicted words across all captions, and $T^{(\text{ref})}$ denoting the total number of words in all the GT captions, the BLEU score for the entire dataset for a particular n-gram length N is:

$$\text{BLEU}^{(\mathrm{N})} = \exp\left(1 - \frac{T^{(\text{ref})}}{T^{(\text{pred})}}\right) \exp\left(\sum_{i=1}^{|I|} \frac{1}{\mathrm{N}} \log(p_i^{(\mathrm{N})})\right) \tag{1.16}$$

METEOR

METEOR [Banerjee and Lavie, 2005] is based on a weighted harmonic mean of 1-gram precision and recall. A mapping (called an *alignment*) is first generated between the same words in the candidate and reference captions, such that every word in the candidate $\hat{c}_i$ maps to exactly one or none of the words in a reference caption $c_j^{(i)} \in \mathcal{C}_i$. The alignment with the most mappings, or least crossover in mapping connections, is chosen for the basis of METEOR calculation. Precision $p_j^{(i)}$ and recall $r_j^{(i)}$ are then used to calculate the harmonic mean $h_j^{(i)}$ which has recall weighted nine times more than precision:

$$p_j^{(i)} = \frac{a_j^{(i)}}{|\hat{c}_i|} \quad r_j^{(i)} = \frac{a_j^{(i)}}{|c_j^{(i)}|} \quad h_{i,j} = \frac{10 p_j^{(i)} r_j^{(i)}}{r_j^{(i)} + 9 p_j^{(i)}} \tag{1.17}$$

where $a_j^{(i)}$ is the number of alignments (ie. the number of words that have been mapped together in the alignment), $|\hat{c}_i|$ is the number of words in the candidate caption, and $|c_j^{(i)}|$ is the number of words in the jth reference caption of the ith video. In order to take into account n-grams larger than one, a penalty $\bar{p}$ is computed:

$$\bar{p}_j^{(i)} = 0.5 \left(\frac{\bar{c}_j^{(i)}}{a_j^{(i)}} \right)^3 \tag{1.18}$$

where $\bar{c}_j^{(i)}$ is the number of chunks, which are sets of sequential words that are adjacent in the candidate and reference captions. The penalty and harmonic mean are then used to calculate the METEOR score for a reference-candidate pair $(c_j^{(i)}, \hat{c}_i)$:

$$\text{METEOR}_j^{(i)}(c_j^{(i)}, \hat{c}_i) = h_j^{(i)}(1 - \bar{p}_j^{(i)}) \tag{1.19}$$

Using the previous example sentences, Figure 1.19 presents an example of alignments and calculations for the METEOR metric:

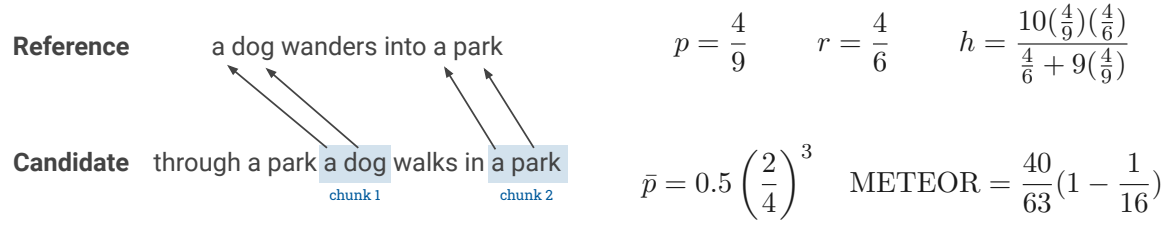


Figure 1.19: An example of the METEOR alignments and chunking. On the left we show an example of the chunking process where two chunks, consisting of two alignments each (with no alignment crossovers — the arrows don't cross one another), have been matched between the candidate and reference captions. On the right we show an example of the calculations carried out to get the METEOR score for this candidate-reference pair.

To calculate the METEOR score for a set of reference captions $\mathcal{C}_i$ the reference caption that achieves the highest score is taken:

$$\text{METEOR}^{(i)}(\mathcal{C}_i, \hat{c}_i) = \max_j(\text{METEOR}_j^{(i)}(c_j^{(i)}, \hat{c}_i)) \tag{1.20}$$

To calculate the final METEOR score for an entire dataset I, aggregate precision $p^{(I)}$, recall $r^{(I)}$ and penalty $\bar{p}^{(I)}$ are combined, using the particular reference captions as defined in the step above:

$$\text{METEOR} = \frac{10p^{(I)}r^{(I)}}{r^{(I)} + 9p^{(I)}}(1 - \bar{p}^{(I)}) \tag{1.21}$$

ROUGE

ROUGE [Lin, 2004] are a set of metrics based around n-gram co-occurences in candidate and reference captions. The set includes five metrics:

- ROUGE-N: Overlap of n-grams across the candidate and reference
- ROUGE-L: Longest common sequence between the candidate and reference

- ROUGE-W: Weighted ROUGE-L favouring consecutive sequences
- ROUGE-S: Skip-bigram[6] (2-gram) co-occurences
- ROUGE-SU: Skip-bigram and unigram (1-gram) co-occurences

In this work, when ROUGE is mentioned it refers to **ROUGE-L**, which is the F-measure of the longest common sequence (LCS) of words between a candidate $\hat{c}_i$ and reference caption $c_j^{(i)}$. Word sequences need to be sequential but not necessarily strictly successive (see example in Figure 1.20). Once the LCS is calculated (with $\mathrm{LCS}(\cdot)$) it is used to calculate the precision $p_j^{(i)}$, recall $r_j^{(i)}$ and F-measure $f_j^{(i)}$ (which is the ROUGE-L score):

$$p_j^{(i)} = \frac{\mathrm{LCS}(c_j^{(i)}, \hat{c}_i)}{|\hat{c}_i|} \quad r_j^{(i)} = \frac{\mathrm{LCS}(c_j^{(i)}, \hat{c}_i)}{|c_j^{(i)}|} \quad \mathrm{ROUGEL}_j^{(i)} = f_j^{(i)} = \frac{(1+\beta^2) p_j^{(i)} r_j^{(i)}}{r_j^{(i)} + \beta^2 p_j^{(i)}} \tag{1.22}$$

where $\beta = p_j^{(i)} / r_j^{(i)}$, and $|\hat{c}_i|$ and $|c_j^{(i)}|$ are the number of words in the candidate and reference captions respectively. Using the example captions from above the ROUGE-L score can be calculated:

Reference a dog wanders into a park
Candidate through a park a dog walks in a park

Length 2 a dog, a a, a park, dog a, dog park
Length 3 a dog a, a dog park, dog a park
Length 4 a dog a park

$$p = \frac{4}{9} \qquad r = \frac{4}{6}$$

$$\mathrm{ROUGEL} = f = \frac{(1-\frac{2}{3}^2)(\frac{4}{9})(\frac{4}{6})}{\frac{4}{6} + (\frac{2}{3})^2(\frac{4}{9})} = \frac{4}{21} = 0.19$$

Figure 1.20: An example of the LCS lengths and ROUGE-L calculation. On the left we show an example of the various LCSs for lengths 2, 3 and 4, none exist for more than 4 in this case. On the right we show an example of the calculations carried out to get the ROUGE-L score for this candidate-reference pair.

To calculate the final ROUGE-L score for an entire dataset I, the ROUGE-L (f) scores are averaged across the dataset:

$$\mathrm{ROUGEL} = \frac{1}{|I|} \sum_{i}^{|I|} \frac{1}{|\mathcal{C}_i|} \sum_{j}^{|\mathcal{C}_i|} \mathrm{ROUGEL}_j^{(i)}(c_j^{(i)}, \hat{c}_i) \tag{1.23}$$

where $|I|$ is the number of samples in the dataset, and $|\mathcal{C}_i|$ are the number of reference captions for sample i.

CIDEr

CIDEr [Vedantam et al., 2015] uses Term Frequency-Inverse Document Frequency (TF-IDF) for weighting n-grams. Given a j^{th} reference caption $c_j^{(i)}$ for the i^{th} sample, we represent the set of n-grams with length N in $c_j^{(i)}$ as $\mathcal{N}^{(\mathrm{N})}$. Similarly as for BLEU, we also define the function $f(\cdot)$ to count the number of times each n-gram $\nu \in \mathcal{N}^{(\mathrm{N})}$ exists in the reference caption $c_j^{(i)} \in \mathcal{C}_i$. The

[6] Skip-bigrams are any pair of words in their caption ordering (even if they aren't strictly successive)

TF-IDF weighting $w_{i,j}^{(\nu)}$ for each n-gram ν in a reference caption $c_j^{(i)}$ is:

$$w_j^{(i,\nu)} = \frac{f(\nu, c_j^{(i)})}{\sum_{\nu' \in V^{(\mathrm{N})}} f(\nu', c_j^{(i)})} \log\left(\frac{|I|}{\sum_{p=1}^{|I|} \min(1, \sum_q^{|\mathcal{C}_i|} f(\nu, c_q^{(p)}))}\right) \tag{1.24}$$

where $\nu' \in V^{(\mathrm{N})}$ is the vocabulary of all n-grams of length N for all reference captions in I. The first term of the above measures the term frequency (TF) of each n-gram ν, while the second term measures the rarity of ν in the dataset using its inverse dataset frequency (IDF).

The CIDEr score for the n-grams of length N is computed with the average cosine similarity between the candidate caption $\hat{c}_i$ and the reference captions $c_j^{(i)} \in \mathcal{C}_i$:

$$\mathrm{CIDEr}^{(i,\mathrm{N})}(\mathcal{C}_i, \hat{c}_i) = \frac{1}{|\mathcal{C}_i|} \sum_j \frac{\mathbf{g}^{(\mathrm{N})}(\hat{c}_i) \cdot \mathbf{g}^{(\mathrm{N})}(c_j^{(i)})}{\left\|\mathbf{g}^{(\mathrm{N})}(\hat{c}_i)\right\| \left\|\mathbf{g}^{(\mathrm{N})}(c_j^{(i)})\right\|} \tag{1.25}$$

where $\mathbf{g}^{(\mathrm{N})}(\hat{c}_i)$ is a vector formed by concatenating the weights $w_j^{(i,\nu)}$ corresponding to all n-grams of length N within the candidate caption $\hat{c}_i$. $\mathbf{g}^{(\mathrm{N})}(c_j^{(i)})$ is the same but for n-grams of length N with the j^{th} reference caption in $\mathcal{C}_i$. This score is calculated across n-gram lengths ($\mathrm{N} = 1, 2, 3, 4$) to determine score for a sample $i \in I$:

$$\mathrm{CIDEr}^{(i)}(\mathcal{C}_i, \hat{c}_i) = \sum_{n=1}^{\mathrm{N}} \frac{1}{\mathrm{N}} \mathrm{CIDEr}_i^{(i,n)}(\hat{c}_i, \mathcal{C}_i) \tag{1.26}$$

To calculate the final CIDEr score for an entire dataset I, the scores are averaged across the dataset:

$$\mathrm{CIDEr} = \frac{1}{|I|} \sum_i^{|I|} \mathrm{CIDEr}^{(i)}(\mathcal{C}_i, \hat{c}_i) \tag{1.27}$$

SPICE

SPICE [Anderson et al., 2016] (visualised in Figure 1.21) performs comparisons using based semantic propositional content by breaking captions up into scene graphs using a parser. The scene graph for a candidate caption $\hat{c}_i$ is denoted as $G(\hat{c}_i)$ while the scene graph for the reference captions $\mathcal{C}_i$, denoted $G(\mathcal{C}_i)$ is formed by the union of scene graphs $G(c_j^{(i)})$ for each $c_j^{(i)} \in \mathcal{C}_i$. The graphs are built as tuples:

$$G(\hat{c}_i) = \langle O(\hat{c}_i), E(\hat{c}_i), K(\hat{c}_i) \rangle \tag{1.28}$$

where $O(\hat{c}_i)$ is a set of object mentions in $\hat{c}_i$, $E(\hat{c}_i)$ is a set of hyper-edges representing relationships between objects in $\hat{c}_i$, and $K(\hat{c}_i)$ is a set of attributes associated with objects in $\hat{c}_i$. Based on the graph tuple representation, a logical tuple representation is formed:

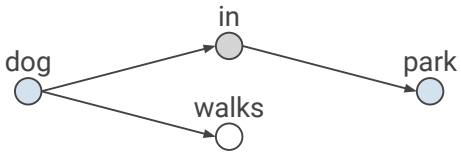
$$T(G(\hat{c}_i)) = O(\hat{c}_i) \cup E(\hat{c}_i) \cup K(\hat{c}_i) \tag{1.29}$$

This logical tuple is also generated for the reference caption before both being used to calculate the precision p_i, recall r_i and F-measure f_i (which is the SPICE score):

$$p_i = \frac{|T(\hat{c}_i) \otimes T(\mathcal{C}_i)|}{|T(\hat{c}_i)|} \quad r_i = \frac{|T(\hat{c}_i) \otimes T(\mathcal{C}_i)|}{|T(\mathcal{C}_i)|} \quad \mathrm{SPICE}^{(i)} = f_i = \frac{2 p_i r_i}{p_i + r_i} \tag{1.30}$$

where $\otimes$ is a binary matching operator. Using the example captions from above the SPICE score can be calculated:

Candidate through a park a dog walks in a park



$$T(G(\hat{c})) = \{(\text{park}), (\text{dog}), (\text{dog}, \text{walks}), (\text{dog}, \text{in}, \text{park})\} \qquad \text{SPICE} = f = \frac{2\frac{2}{4}\frac{2}{3}}{\frac{2}{4}+\frac{2}{3}} = \frac{9}{7}$$

Figure 1.21: An example of the scene graph, logical tuple and SPICE calculation. We first show what a scene graph might look like for our candidate caption, with relations `dog` $\rightarrow$ `in` $\rightarrow$ `park` and `dog` $\rightarrow$ `walks`. Below this on the left we show what such a graph representation looks like in logical tuple form. While bottom right shows an example of the SPICE calculation.

To calculate the final SPICE score for an entire dataset I, the scores are averaged across the dataset:

$$\text{SPICE} = \frac{1}{|I|}\sum_{i}^{|I|}\text{SPICE}_i(\mathcal{C}_i, \hat{c}_i) \tag{1.31}$$

Precision & Recall

The precision and recall calculations described thusfar have been unique to the individual captioning metrics, whereas here we present the standard forms utilised in classification problems. Despite not normally being utilised for captioning evaluations, we will utilise them later in the chapter to determine the accuracy for specific word generations.

The standard precision p and recall r values are calculated as:

$$p = \frac{\text{TP}}{\text{TP}+\text{FP}} \quad r = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{1.32}$$

where TP, FP, and FN are true positives, false positives and false negatives respectively:

- **True Positives (TP)** are when a model correctly predicts a word that exists in a ground truth caption;
- **False Positives (FP)** are when a model incorrectly predicts a word, that doesn't actually exist in the ground truth caption;
- **False Negatives (FN)** are when a model doesn't predict a word that exists in the ground truth caption.

Precision and recall values listed in this chapter are obtained on a per word/class basis — averaged across all of the samples. The final results are then obtained from averaging over all of the words that appear in the union of the predicted and ground truth captions.

1.2 Transformers for Video Captioning

After consideration of the various components of the three-stage video captioning frameworks discussed in Section 1.1, we develop our own framework, and investigate various configurations and additions. Keeping in line with the three-stage modulisation of past video captioning works ours also follows this structure. Presented in Figure 1.22, our initial model consists of two or three parts — the visual feature extraction and encoder pipeline, an optional secondary feature encoder, and a captioning network in the form of an RNN or TN. Details and experimental analysis of each of these parts can be found in Section 1.2.1, Section 1.2.3, and Section 1.2.2 respectively. In Section 1.2.4 we also investigate the effects of reducing the model size for the Transformer Network based model. Finally, in Section 1.2.5, we perform a thorough analysis of both model and human captioning performance, including an analysis of particular visual concepts. In Section 1.3 we will expand on this model further to promote grounding of important visual and textual concepts.

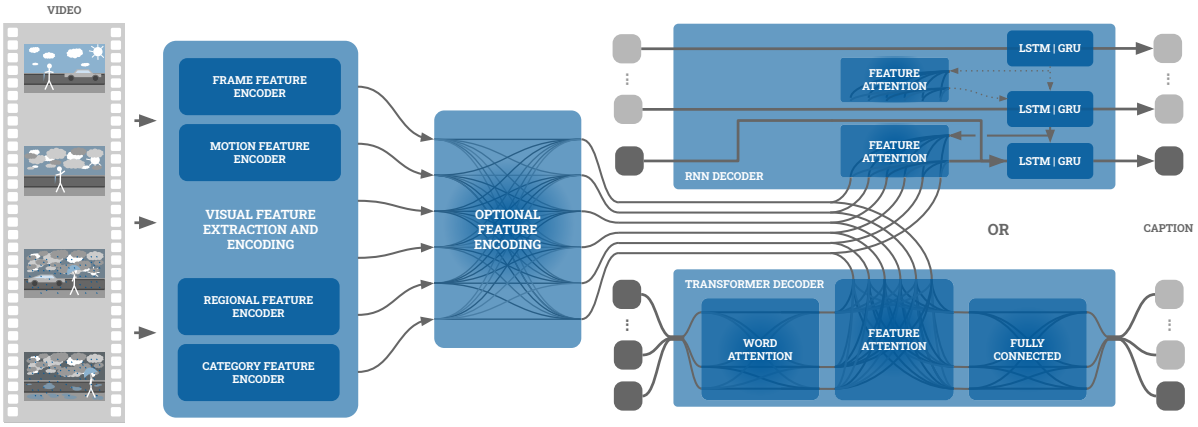


Figure 1.22: A visual representation our captioning pipeline. Our pipeline is structured into three main parts — the visual feature extraction and encoder pipeline, an optional secondary feature encoder, and a captioning module in the form of an RNN or TN. We perform attention over the visual feature encodings within each captioning module.

1.2.1 Input Features

Similar to past works we just take pre-defined models to extract our feature sets. Following on from two of the most recent works [Pan et al., 2020, Zheng et al., 2020], we look to utilise four types of input feature inputs:

1. **Frame Feature**: For each clip we constructed a set $\mathcal{X}$ of 28 uniformly spaced frames and passed them through a frame-based feature extractor $\phi^{(\mathrm{f})}\colon \mathcal{X} \to \mathbb{R}^{d^{(\mathrm{f})}}$ ($d^{(\mathrm{f})} = 1536$), such as a CNN model pre-trained on the ImageNet [Russakovsky et al., 2015] dataset. For MSVD we use **ResNet-101** [He et al., 2016], while for MSR-VTT we use **InceptionResnetV2** [Szegedy et al., 2017], taking the outputs from the last `avg-pooling` layer for both models[7]. The 28 features are then mean pooled as to form a single frame feature representing

[7] The use of different CNN models for different datasets is consistent with previous works.

the entire video:

$$\mathbf{f}^{(\mathrm{f})} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}_t \in \mathcal{X}} \phi^{(\mathrm{f})}(\mathbf{x}_t). \tag{1.33}$$

2. **Motion Feature**: To gather the necessary data for motion feature extraction, we follow the procedure of [Aafaq et al., 2019a] and form the set $\mathcal{X}' = \{[\mathbf{x}_{t-8}, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_{t+7}] \mid \mathbf{x}_t \in \mathcal{X}\}$. That is, we construct groups of 16 frames each centred on the 28 frames in $\mathcal{X}$. The sequence of frames in $\mathcal{X}'$ are passed through a motion-based feature extractor $\phi^{(\mathrm{m})} \colon \mathcal{X}' \to \mathbb{R}^{d^{(\mathrm{m})}}$ ($d^{(\mathrm{m})} = 4096$). In particular, we use the **C3D** model [Tran et al., 2015] pre-trained on Sports-1M [Karpathy et al., 2014], with the output of the `fc6` layer taken as the feature. The 28 features are mean pooled to form a single motion feature representing the entire video:

$$\mathbf{f}^{(\mathrm{m})} = \frac{1}{|\mathcal{X}'|} \sum_{\mathbf{x}_t \in \mathcal{X}'} \phi^{(\mathrm{m})}(\mathbf{x}_t). \tag{1.34}$$

3. **Region Features**: We construct a set $\mathcal{X}''$ by sampling the 15$^{\text{th}}$ frame of each video and passing it through region-based feature extractors $\phi^{(\mathrm{b})} \colon \mathcal{X}'' \to \mathbb{R}^{d^{(\mathrm{b})} \times R}$ ($d^{(\mathrm{b})} = 4$) and $\phi^{(\mathrm{r})} \colon \mathcal{X}'' \to \mathbb{R}^{d^{(\mathrm{r})} \times R}$ ($d^{(\mathrm{f})} = 1024$), where R denotes the total number of regions. Specifically, we use **Faster R-CNN** [Ren et al., 2015] pre-trained on MS-COCO [Lin et al., 2014] with a ResNet-101 [He et al., 2016] backbone. We extract box coordinates[8] ($\phi^{(\mathrm{b})}$) and the standard ROI-pooled feature outputs ($\phi^{(\mathrm{r})}$) from the Faster R-CNN for the top $R = 10$ most confident object detections resulting in our region features $\mathbf{F}^{(\mathrm{r})}$:

$$\mathbf{F}^{(\mathrm{r})} \triangleq \begin{bmatrix} \phi^{(\mathrm{b})}(\mathbf{x}_t) \\ \phi^{(\mathrm{r})}(\mathbf{x}_t) \end{bmatrix}. \tag{1.35}$$

4. **Category Feature**: For experiments with the MSR-VTT dataset we also use video category labels, which categorise videos into one of 20 categories. The set of categories $\mathcal{C}$ consists of $\{$*music*, *people*, *gaming*, *sports*, *news*, *education*, *tv shows*, *movie*, *animation*, *vehicles*, *how-to*, *travel*, *science*, *animals*, *kids*, *doc*, *food*, *cooking*, *beauty*, *ads*$\}$. We represent the category that a video belongs to by a one-hot vector $\mathbf{f}^{(\mathrm{c})} \in \{0, 1\}^{d^{(\mathrm{c})}}$ ($d^{(\mathrm{c})} = 20$) consisting of zeros in all dimensions with the exception of a single 1 at the k^{th} dimension, corresponding to the k^{th} category.

Despite the fact that we introduced mean pooling as the most basic temporal encoding scheme in Section 1.1.2, it has recently been adopted by many top-performing approaches as the encoding of choice. This choice is for two main reasons:

1. Compacting the temporal aspect, by any means, is useful for simplifying the implementation and results in significant processing efficiency benefits, as models do not scale well with many time-steps of high dimensional feature operations.

2. Although capturing temporal information is important, the videos in the MSVD and MSR-VTT datasets are relatively short and relatively static in terms of the global scene across

[8] Box coordinates are of form $[x, y, w, h]$ as ratios of the image size ie. $0 < x, y, w, h < 1$

the video. This means that not a significant amount of information is actually lost if we simply mean pool over time. What's important to identify in these videos, is the low-level motions and movements to help determine actions and verbs, which is the purpose of the inclusion of the motion feature.

To summarise, while performing a more complex temporal encoding process will likely see some minor performance gains, the negative effects on simplicity and efficiency are too significant, with focus and computational resources being devoted to other parts of the captioning framework. Furthermore, due to resource limitations, and as is common practice, we precompute all of the above features. This results in the inability to fine-tune the visual networks for the captioning problem. However, it allows for much more efficient training of much smaller non-visual models.

As an initial experiment, we look to ascertain the impact of each type of feature on the captioning performance for each of the datasets MSVD and MSR-VTT. We train and evaluate our model using different combinations of features, with results shown in Table 1.7 utilising the caption metrics discussed in Section 1.1.5. We find that region features are of most significance, highlighting the necessity to focus on spatial regions for effective caption generation. This makes sense as captions tend to refer to specific spatial regions, so more focused features are more informative than the global and mean pooled singular frame and motion features. Interestingly, while motion features perform better on their own compared to frame features for MSR-VTT, the opposite is true for MSVD. This suggests a core difference in the types of videos in the datasets, with MSR-VTT having more captions reliant on understanding the motions within the videos compared to MSVD. Finally, and somewhat unsurprisingly, we show that utilising all features attains the best performance.

From this point onward we will utilise the full set of visual features for all experiments. As the features are of variable dimensions we embed them to have the same dimension so they can be stacked into a feature matrix $\mathbf{F} \in \mathbb{R}^{d^{(\mathrm{F})} \times F}$,

$$\mathbf{F} = \left[\mathbf{W}^{(\mathrm{f})}\mathbf{f}^{(\mathrm{f})}, \mathbf{W}^{(\mathrm{m})}\mathbf{f}^{(\mathrm{m})}, \mathbf{W}^{(\mathrm{r})}\mathbf{F}^{(\mathrm{r})}, \mathbf{W}^{(\mathrm{c})}\mathbf{f}^{(\mathrm{c})}\right] \tag{1.36}$$

where $\mathbf{W}^{(\mathrm{f})} \in \mathbb{R}^{d^{(\mathrm{F})} \times d^{(\mathrm{f})}}$, $\mathbf{W}^{(\mathrm{m})} \in \mathbb{R}^{d^{(\mathrm{F})} \times d^{(\mathrm{m})}}$, $\mathbf{W}^{(\mathrm{r})} \in \mathbb{R}^{d^{(\mathrm{F})} \times (d^{(\mathrm{r})} + d^{(\mathrm{b})})}$ and $\mathbf{W}^{(\mathrm{c})} \in \mathbb{R}^{d^{(\mathrm{F})} \times d^{(\mathrm{c})}}$ are embedding matrices that will be learned during the optimisation process. This process of feature extraction and embedding is shown in Figure 1.23.

1.2.2 Captioning Module

As discussed in Section 1.1.3, the majority of video captioning works have utilised a Recurrent Neural Network as the captioning module, however there has been some promise in utilising Transformer Networks, particularly in regards to training efficiency. We implement both strategies looking to investigate the performance and efficiency differences between them. Before we discuss our findings we provide a more thorough technical description of both processes, as well as highlighting the training strategy used for training our models.

$\mathbf{f}^{(\mathrm{f})}$	$\mathbf{f}^{(\mathrm{m})}$	$\mathbf{F}^{(\mathrm{r})}$	$\mathbf{f}^{(\mathrm{c})}$	B1	B2	B3	B4	MT	RG	Cr	SP
MSVD											
✓				.698	.536	.415	.287	.252	.609	**.238**	.029
	✓			.664	.466	.345	.215	.233	.597	**.161**	.026
✓	✓			.696	.535	.410	.288	.252	.607	.285	.031
		✓		.726	.574	.470	.364	.273	.632	**.471**	.034
✓		✓		.759	.625	.526	.424	.309	.667	.640	.043
	✓	✓		.738	.587	.479	.371	.283	.644	.509	.038
✓	✓	✓		.768	.642	.542	.436	.313	.674	.665	.045
MSR-VTT											
✓				.632	.431	.299	.189	.194	.493	**.141**	.034
	✓			.717	.541	.395	.285	.233	.530	**.224**	.045
✓	✓			.666	.516	.402	.305	.222	.529	.221	.043
		✓		.700	.515	.378	.276	.227	.523	**.262**	.045
✓		✓		.750	.592	.458	.348	.261	.570	.381	.058
	✓	✓		.770	.620	.487	.376	.274	.585	.438	.063
✓	✓	✓		.784	.643	.513	.398	.280	.600	.468	.065
✓	✓	✓	✓	.787	.643	.511	.396	.280	.600	.474	.066

Table 1.7: The impact of the different input features on captioning performance. Here we show results for captioning models trained with different subsets of the input features $\mathbf{f}^{(\mathrm{f})}$, $\mathbf{f}^{(\mathrm{m})}$, $\mathbf{F}^{(\mathrm{r})}$ and $\mathbf{f}^{(\mathrm{c})}$. Metric shorthands are **B1-4: BLEU1-4, MT: METEOR, RG: ROUGE-L, Cr: CIDEr, SP: SPICE**, with best performing scores per dataset underlined. **For simplicity, let's focus our attention to the CIDEr scores from now on, as it tends to be the most insightful metric currently available.** Comparing the models which only use one feature (**bolded** results), we can see that region features are the most insightful with scores of .471 versus .238 and .161 for MSVD, and .262 versus .141 and .224 for MSR-VTT. In a similar vain, it looks as if motion features are more important than frame features for MSR-VTT (.224 vs .141) but not for MSVD (.161 vs .238), suggesting that the MSVD involves less discriminatory motion. Lastly, it is clear that including all features achieves the best performance, achieving the highest scores across almost all metrics.

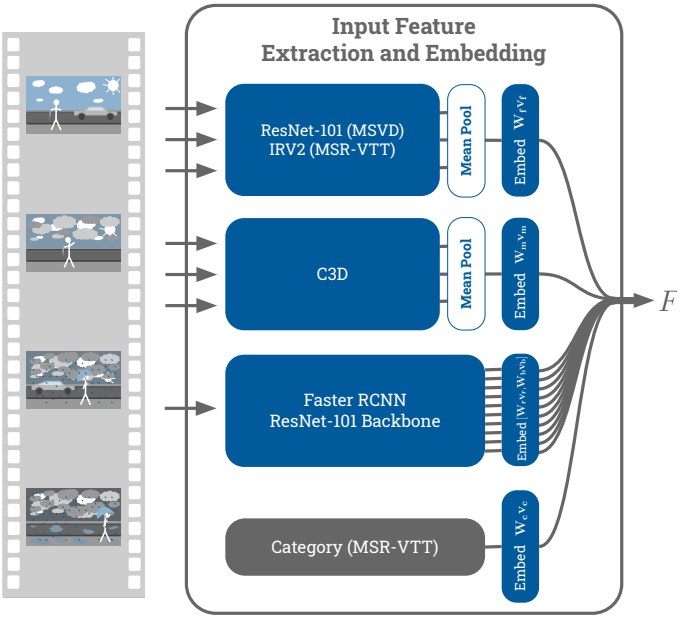


Figure 1.23: A visual representation of the visual feature extraction and embedding process. Our set of visual features $\mathbf{F}$ include frame features extracted from a ResNet-101 or IRV2 CNN, motion features extracted from a C3D CNN, region features extracted from a Faster R-CNN, and for MSR-VTT, category features related to the video.

Recurrent Neural Network Decoder

As outlined in Section 1.1.3, an RNN iteratively generates words using the embedding for the previously generated word $\mathbf{e}_{n-1}$, their own hidden state $\mathbf{h}_{n-1}$ and the filtered visual features $\tilde{\mathbf{f}}_n$:

$$\begin{bmatrix} \rho_n \\ \mathbf{h}_n \end{bmatrix} = \mathrm{RNN}(\mathbf{h}_{n-1}, \mathbf{e}_{n-1}, \tilde{\mathbf{f}}_n) \tag{1.37}$$

where ρ_n is a probability distribution over the vocabulary, from which the most probable word is chosen as $\mathbf{y}_n$. This $\mathrm{RNN}(\cdot)$ function is actually performing embeddings and additions over the inputs and is equivalent to:

$$\mathbf{h}_n = \tanh(\mathbf{U}^{(\mathrm{h})}\mathbf{h}_{n-1} + \mathbf{W}^{(\mathrm{x})}\check{\mathbf{x}}_n + \mathbf{b}^{(\mathrm{h})}) \tag{1.38}$$

where the input $\check{\mathbf{x}}$:

$$\check{\mathbf{x}}_n = \begin{bmatrix} \mathbf{e}_{n-1} \\ \tilde{\mathbf{f}}_n \end{bmatrix} \tag{1.39}$$

and $\mathbf{U}^{(\mathrm{h})} \in \mathbb{R}^{d^{(\mathrm{h})} \times d^{(\mathrm{h})}}$, $\mathbf{W}^{(\mathrm{x})} \in \mathbb{R}^{d^{(\mathrm{h})} \times d^{(\mathrm{x})}}$, $\mathbf{b}_h \in \mathbb{R}^{d^{(\mathrm{h})}}$, $\check{\mathbf{x}}_n \in \mathbb{R}^{d^{(\mathrm{x})}}$ where $d^{(\mathrm{x})} = d^{(\mathrm{e})} + d^{(\mathrm{F})}$. Using the hidden state $\mathbf{h}_n$, the probability distribution across the vocabulary of the n word generation can be determined with:

$$\rho_n = \mathrm{softmax}(\mathbf{W}^{(\mathrm{k})}\mathbf{h}_n + \mathbf{b}^{(\mathrm{k})}) \tag{1.40}$$

where $\mathbf{W}^{(\mathrm{k})} \in \mathbb{R}^{|\mathcal{D}| \times d^{(\mathrm{h})}}$, $\mathbf{b}^{(\mathrm{k})} \in \mathbb{R}^{|\mathcal{D}|}$, with $|\mathcal{D}|$ being the vocabulary size. Figure 1.24 presents a visualisation of the RNN decoding process as described above.

The above formulation is the default RNN, however in practice one normally uses either Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) units. This is because after a few

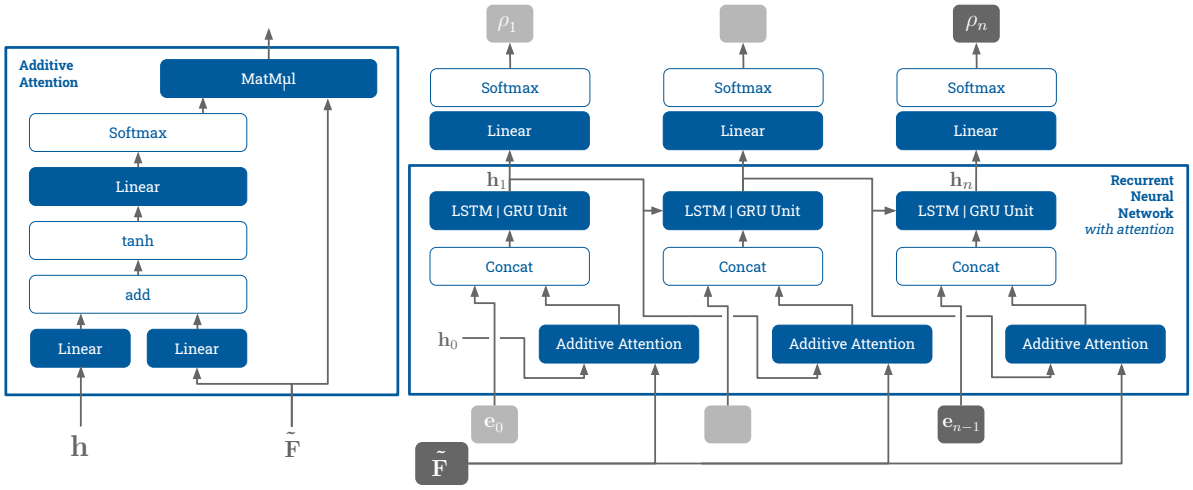


Figure 1.24: A visual representation of the Recurrent Neural Network decoder. This figure presents the additive attention module on the left and an RNN unrolled $n = 3$ steps. For each step the previous hidden state $\mathbf{h}_{n-1}$ is used to attend over the filtered visual features $\tilde{\mathbf{F}}$ with the output being concatenated with the embedding for the previously generated word $\mathbf{e}_{n-1}$ and used as input $\check{\mathbf{x}}_n$ to the LSTM or GRU unit. These units output a new hidden state vector $\mathbf{h}_n$ which is passed for context to next stage as well as being used to generate the word probability distribution ρ_n. For $n = 1$, the embedding vector $\mathbf{e}_0$ representing the `<BOS>` is passed in. The RNN will continue to generate word probabilities until it either reaches N iterations (the maximum number permissible), or if the probability distributions suggests the `<EOS>` token.

iterations data within the hidden state gets overwritten and forgotten in favour of more recent information. When training these recurrent models, gradients that are passed to earlier time-steps get smaller and smaller leading to what is called the **vanishing gradient problem**, which prevents the networks from learning longer sequences. LSTM and GRU units address these issues by adding gates to determine when to remember and when to forget information.

Despite whether the standard, LSTM or GRU unit type is used, the hidden states $\mathbf{h}$ still hold the record of the previously generated words through the generation process. Furthermore, no matter the unit type, only vector inputs are permitted for $\check{\mathbf{x}}_n$, meaning the visual features $\tilde{\mathbf{f}}_n$ also need to be a vector. We however, have a matrix of visual features $\tilde{\mathbf{F}} = \varphi(\mathbf{F})$, recalling that $\mathbf{F} = \left[\mathbf{W}^{(\mathrm{f})}\mathbf{f}^{(\mathrm{f})}, \mathbf{W}^{(\mathrm{m})}\mathbf{f}^{(\mathrm{m})}, \mathbf{W}^{(\mathrm{r})}\mathbf{F}^{(\mathrm{r})}, \mathbf{W}^{(\mathrm{c})}\mathbf{f}^{(\mathrm{c})}\right]$. To compress this set into a single vector $\tilde{\mathbf{f}}_n$ per word generation step we utilise **additive attention** [Bahdanau et al., 2015]:

$$\tilde{\mathbf{f}}_n = \sum_{f=1}^{F} \alpha_f \tilde{\mathbf{F}}[:, f] \tag{1.41}$$

where

$$\alpha_f = \mathrm{softmax}(\mathbf{W}^{(\alpha)\top} \tanh(\mathbf{W}^{(\mathrm{h})}\mathbf{h}_{n-1} + \mathbf{W}^{(\mathrm{f})}\tilde{\mathbf{F}}[:, f])) \tag{1.42}$$

and $\mathbf{W}^{(\alpha)} \in \mathbb{R}^{d^{(\alpha)}}$, $\mathbf{W}^{(\mathrm{h})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\mathrm{h})}}$ and $\mathbf{W}^{(\mathrm{f})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\mathrm{F})}}$.

Transformer Network Decoder

As outlined in Section 1.1.3, a TN performs attention over all previously generated words at once ($[\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$) when generating each n^{th} word probability distribution ρ_n:

$$\rho_n = \text{TransformerDecoder}([\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], \tilde{\mathbf{F}}_n) \tag{1.43}$$

while also considering the set of filtered visual features $\tilde{\mathbf{F}}_n$. This $\text{TransformerDecoder}(\cdot)$ function is actually quite complex and generally consists of multiple repetitions of a $\text{DecoderBlock}_l(\cdot)$ for a certain number of layers L:

$$\begin{aligned}
\mathbf{z}_{n,1} &= \text{DecoderBlock}_1(\mathbf{e}_n, [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], \tilde{\mathbf{F}}) \\
\mathbf{z}_{n,2} &= \text{DecoderBlock}_2(\mathbf{z}_{n,1}, [\mathbf{z}_{1,1}, \ldots, \mathbf{z}_{n,1}], [\mathbf{z}_{1,1}, \ldots, \mathbf{z}_{n,1}], \tilde{\mathbf{F}}) \\
&\vdots \\
\mathbf{z}_{n,L} &= \text{DecoderBlock}_L(\mathbf{z}_{n,L}, [\mathbf{z}_{1,L-1}, \ldots, \mathbf{z}_{n,L-1}], [\mathbf{z}_{1,L-1}, \ldots, \mathbf{z}_{n,L-1}], \tilde{\mathbf{F}})
\end{aligned} \tag{1.44}$$

where $\mathbf{z}_{n,l} \in \mathbb{R}^{d^{(\text{h})}}$. The final $\mathbf{z}_{n,L}$ is used to calculate the word probability distribution for the n^{th} word generation:

$$\rho_n = \text{softmax}(\mathbf{W}^{(\text{k})}\mathbf{z}_{n,L} + \mathbf{b}^{(\text{k})}) \tag{1.45}$$

where $\mathbf{W}^{(\text{k})} \in \mathbb{R}^{d^{(\text{h})} \times |\mathcal{D}|}$, $\mathbf{b}^{(\text{k})} \in \mathbb{R}^{|\mathcal{D}|}$, with $|\mathcal{D}|$ being the vocabulary size.

This $\text{DecoderBlock}_l(\mathbf{q}_n, \mathbf{K}, \mathbf{V}, \tilde{\mathbf{F}})$ is made up of a number of sequential operations:

$$\begin{aligned}
\mathbf{u}_n''' &= \text{MultiHeadAttention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) \\
\mathbf{u}_n'' &= \text{LayerNorm}(\mathbf{q}_n + \mathbf{u}_n''') \\
\mathbf{u}_n' &= \text{MultiHeadAttention}(\mathbf{u}_n'', \tilde{\mathbf{F}}, \tilde{\mathbf{F}}) \\
\mathbf{u}_n &= \text{LayerNorm}(\mathbf{u}_n'' + \mathbf{u}_n') \\
\mathbf{z}_n' &= \text{FullyConnected}(\mathbf{u}_n) \\
\mathbf{z}_n &= \text{LayerNorm}(\mathbf{u}_n + \mathbf{z}_n')
\end{aligned} \tag{1.46}$$

where $\mathbf{q}_n$ is the **query** for word generation step n, $\mathbf{K}$ is a set of **keys**, and $\mathbf{V}$ is a set of **values**, which are dependent on the layer l. For the first layer $l = 1$ we use the word embeddings for the queries, keys, and values ($\mathbf{q}_n = \mathbf{e}_n$, $\mathbf{K} = [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$ and $\mathbf{V} = [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$), while for later layers $l > 1$ they take the output from the previous layer ($\mathbf{q}_n = \mathbf{z}_{n,l-1}$, $\mathbf{K} = [\mathbf{z}_{1,l-1}, \ldots, \mathbf{z}_{n,l-1}]$ and $\mathbf{V} = [\mathbf{z}_{1,l-1}, \ldots, \mathbf{z}_{n,l-1}]$).

The $\text{FullyConnected}(\cdot)$ is defined as a small **fully connected** neural network:

$$\mathbf{z}_n' = \text{FC}(\mathbf{u}_n) = \mathbf{W}^{(2)}\text{ReLU}(\mathbf{W}^{(1)}\mathbf{u}_n) \tag{1.47}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{d^{(\text{h})} \times d^{(\text{i})}}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{d^{(\text{i})} \times d^{(\text{h})}}$.

The $\text{MultiHeadAttention}(\cdot)$ is the most important part of the TN and is based on repeating **in-**

dependent scaled dot-product attentions ($\mathrm{DotProdAttention}(\cdot)$) carried out H times in parallel:

$$\mathrm{MultiHeadAttention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) = \mathbf{W}^{(\mathrm{O})} \begin{bmatrix} \mathrm{DotProdAttention}(\mathbf{W}_1^{(\mathrm{q})}\mathbf{q}_n, \mathbf{W}_1^{(\mathrm{K})}\mathbf{K}, \mathbf{W}_1^{(\mathrm{V})}\mathbf{V}) \\ \vdots \\ \mathrm{DotProdAttention}(\mathbf{W}_H^{(\mathrm{q})}\mathbf{q}_n, \mathbf{W}_H^{(\mathrm{K})}\mathbf{K}, \mathbf{W}_H^{(\mathrm{V})}\mathbf{V}) \end{bmatrix} \tag{1.48}$$

where $\mathbf{W}^{(\mathrm{q})}, \mathbf{W}^{(\mathrm{K})}, \mathbf{W}^{(\mathrm{V})} \in \mathbb{R}^{\frac{d^{(\mathrm{h})}}{H} \times d^{(\mathrm{h})}}$ are independent projection matrices for each head $1, ..., H$, while $\mathbf{W}^{(\mathrm{O})} \in \mathbb{R}^{d^{(\mathrm{h})} \times d^{(\mathrm{h})}}$.

Each $\mathrm{DotProdAttention}(\cdot)$ attention performs a dot-product attention on the values via the dot-product of the query with the keys:

$$\mathrm{DotProdAttention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) = \mathbf{V}\mathrm{softmax}\left(\frac{\mathbf{K}\mathbf{q}_n}{\sqrt{d^{(\mathrm{h})}}}\right) = \mathbf{V}\frac{\exp\left(\frac{\mathbf{K}\mathbf{q}_n}{\sqrt{d^{(\mathrm{h})}}}\right)}{\sum_{p=1}^{T}\exp\left(\frac{\mathbf{k}_p\mathbf{q}_n}{\sqrt{d^{(\mathrm{h})}}}\right)} \tag{1.49}$$

The intuition behind this *query*, *key*, and *value* approach is that the $\mathrm{DotProdAttention}(\cdot)$ is looking to apply attention to the set of values $\mathbf{V}$, which are represented by a set key signatures $\mathbf{K}$ for each value, based on how well a particular query $\mathbf{q}_n$ aligns with the representative key signatures. So in the $\mathrm{DecoderBlock}_1(\cdot)$ for $L = 1$, the first $\mathrm{MultiHeadAttention}(\mathbf{e}_n, [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}])$ is performing a self-attention where each query word is compared to all of the other words (and itself) in the sequence $[\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$, while the second $\mathrm{MultiHeadAttention}(\mathbf{u}''_n, \tilde{\mathbf{F}}, \tilde{\mathbf{F}})$ poses how well does each of the attended words $\mathbf{u}''_n$ relate to each visual features in $\tilde{\mathbf{F}}$, then weighting the visual features appropriately. In other words, the first attention is generating an encoded sequence of all of the past and current words where some are more prominent than others, while the second uses this to determine what features are the most important to these highly weighted words.

Above we have described the process for the TN at inference time, where all of the word embeddings for the *past* generated words ($[\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$) are passed in as input. This results in an output sequence of probability distributions $(\rho_1, \ldots, \rho_n)$, however at each n^{th} iteration we only consider the latest probability distribution ρ_n, ignoring the others. This is different to an RNN which only considers input of the $\mathbf{e}_{n-1}$ embedding and generates output of the ρ_n probability distribution. Similar to an RNN however, is that we iteratively loop through the n steps until we either reach the maximum permissible length N or generate the `<EOS>` token.

The training scheme of the TN is what sets it apart from the RNN, allowing parallel processing across the N steps without the need for the iteration as is needed in RNNs. Recall that during training, for both RNNs and TNs that GT captions are either clipped or padded to ensure they are always of length N. For training TNs we simply input the entire sequence and do one pass (rather than N passes like for the RNN). We can do this because we are generating the full sequence of probabilities at once, and are not dependent on the calculation of a hidden state which is calculated iteratively. For the TN, to prevent the model considering and attending to future words, eg. looking at word embedding $\mathbf{e}_{n+4}$ from the n^{th} position, a binary masking matrix

$\mathbf{M}$ is applied within the $\mathrm{DotProdAttention}(\cdot)$:

$$\mathrm{DotProdAttention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) = \mathbf{V}\mathrm{softmax}\left(\frac{\mathbf{K}\mathbf{q}_n}{\sqrt{d^{(\mathrm{h})}}}\mathbf{M}\right) = \mathbf{V}\frac{\exp\left(\frac{\mathbf{K}\mathbf{q}_n}{\sqrt{d^{(\mathrm{h})}}}\right)}{\sum_{p=1}^{T}\exp\left(\frac{\mathbf{k}_p\mathbf{q}_n}{\sqrt{d^{(\mathrm{h})}}}\mathbf{M}\right)} \tag{1.50}$$

where $\mathbf{M} \in \mathbb{R}^{N\times N}$ takes the form:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.51}$$

This sets the attention weights of future values (in our case word embeddings) to 0.

The Transformer Network decoder architecture as described above, for inference at step n, is visually presented in Figure 1.25, showing the $\mathrm{DotProdAttention}(\cdot)$, $\mathrm{MultiHeadAttention}(\cdot)$ functions as well as the full stack of them together.

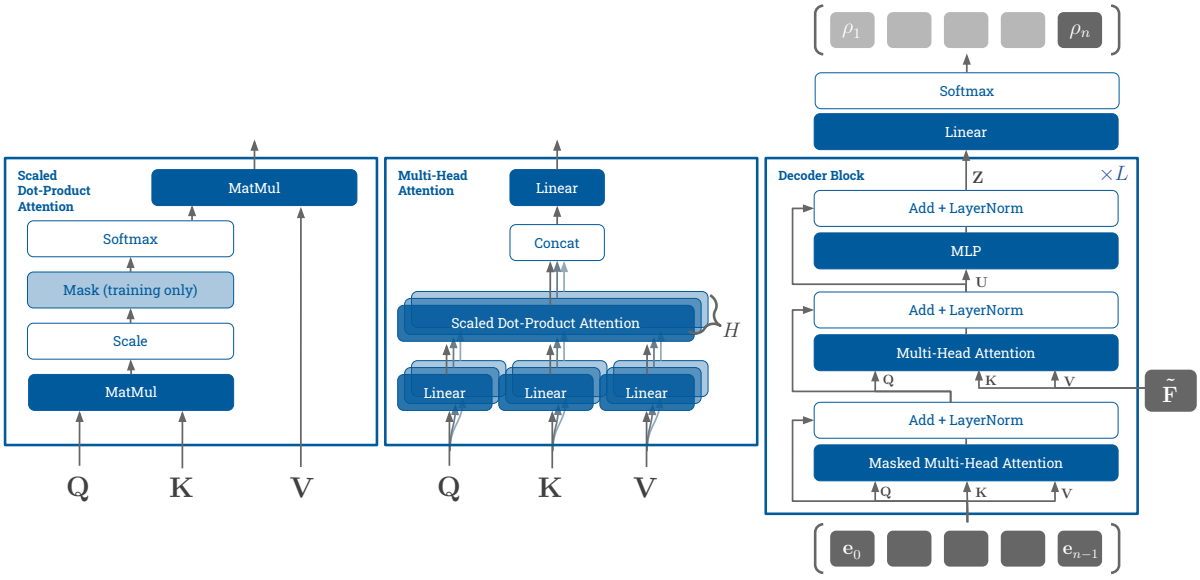


Figure 1.25: A visual representation of the Transformer Network decoder during inference. From left to right we have the $\mathrm{DotProdAttention}(\cdot)$, the $\mathrm{MultiHeadAttention}(\cdot)$, and the full $\mathrm{DecoderBlock}(\cdot)$ processes. The full sequence of past words $[\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$ is processed on the n^{th} word generation, producing an output of n word probability distributions $(\rho_0, \rho_1, \ldots, \rho_n)$ over the vocabulary, where all but the n^{th} distribution is discarded. The initial input embedding $\mathbf{e}_0$ is the embedding for the `<BOS>` token. The model will continue generating word probability distributions until it reaches the limit N or generates the `<EOS>` token. During training we pass in the full ground truth caption embedding sequence, cut or padded to length N ($[\mathbf{e}_0, \ldots, \mathbf{e}_N]$). We then do a single pass through the network rather than the iterative process as done in inference and is done for RNN training. Masking is used to prevent the attention mechanisms from attending to future word representations ($[\mathbf{e}_n, \ldots, \mathbf{e}_N]$).

Training Setup

We train and evaluate all of our models in similar conditions, keeping most hyper-parameters unchanged between experiments. Firstly, the loss function employed is the standard cross-

entropy loss used in all other captioning works, and is defined per data sample as:

$$L^{(\text{caption})} = -\sum_{n=1}^{N} \log(\Pr(\hat{y}_n = y_n)) = -\sum_{n=1}^{N} \log(\rho_n^\top \omega(y_n)) \tag{1.52}$$

where $\omega(y_n)$ is the one-hot vector for our desired ground truth word y_n.

In terms of parameters, the RNNs are trained with hidden size $d^{(\text{h})} = 1024$, while the TNs use a hidden size of $d^{(\text{h})} = 512$ (and $d^{(\text{i})} = 2048$), aligning with the visual input features encoding size of $d^{(\text{F})} = 512$. For the TN models we experiment with varying numbers of heads $H = 1|4|8$ and layers $L = 1|2$. Furthermore, its important to point out that in the following *initial captioner comparison* experiments we are yet to incorporate a filtering function $\varphi(\cdot)$, therefore we are utilising the unfiltered features $\mathbf{F}$ in-place of their filtered counterparts $\tilde{\mathbf{F}}$.

Due to hardware limitations we use a relatively small batch size of 8 for our experiments. For the MSVD experiments we train for 100 epochs, while for the MSR-VTT experiments we train for 200 epochs, both include early stopping with a patience of 50 epochs. We use the *Adam* optimiser with an initial learning rate of 10^{-4}. The maximum sequence length for MSVD experiments is $N = 20$, and $N = 17$ for MSR-VTT — we cut ground truth captions to N, and only allow the caption generators to process N iterations. Any ground truth captions shorter than N get padded with 0s after their `<EOS>` token. For more details on the training parameters used for each experiment hereon, see Appendix C.

Captioner Comparison

We perform an evaluation of the RNN and TN captioning strategies in relation to performance and efficiency, with results shown in Table 1.8. We show that LSTM units are more effective than GRU units for our RNN model, and that both outperform the TNs. Adding more heads and layers to the TN models improve their scores, however their training and inference times also increase similar or worse than the RNN models. It's unclear why there is a decrease in performance for the TNs relative to the RNNs. Potentially it's due to the MSVD and MSR-VTT datasets being too small, as TNs are known for requiring more data than other architectures, including RNNs [Liu et al., 2019, Raffel et al., 2020]. If we compare the relative drops in performance in the CIDEr scores between the best TN model (2,8) and the LSTM model across the two datasets, we find that there is a 10% drop for MSVD while only a 7% drop for MSR-VTT, indication the TN performs relatively better on the larger dataset.

Looking beyond the testing metric evaluations, we also consider the training and validation metrics, in the form of CIDEr scores, across the training periods. Figure 1.26 and Figure 1.27 provide the metric scores for the LSTM, GRU, and 1- and 2-layer 8-head TNs, for MSVD and MSR-VTT respectively. By considering these graphs we can get a sense of how the models learn in comparison to one another, and also their behaviour towards over-fitting[9].

[9] **Over-fitting** refers to when a model over-fits to a training set, indicated by continued increase in training performance, but not in validation performance

Model	Layers	Heads	B1	B2	B3	B4	MT	RG	Cr	SP	TSpS	ISpS
						MSVD						
LSTM	**1**	**-**	.768	.642	.542	.436	.313	.674	**.665**	.045	92	10.8
GRU	**1**	**-**	.773	.647	.547	.439	.309	.668	.631	.043	92	9.2
TN	**1**	**1**	.741	.613	.512	.404	.294	.652	.557	.040	110	5
		8	.751	.629	.531	.424	.300	.656	.591	.043	120	9.4
	2	**1**	.741	.613	.515	.409	.296	.654	.546	.039	80	6.7
		8	.749	.626	.527	.419	.299	.654	**.602**	.042	80	10
						MSR-VTT						
LSTM	**1**	**-**	.787	.643	.511	.396	.280	.600	**.474**	.066	54	10.4
GRU	**1**	**-**	.776	.637	.502	.384	.266	.595	.435	.058	57	11.1
TN	**1**	**1**	.761	.615	.482	.365	.261	.581	.390	.057	77	8.3
		8	.771	.635	.506	.390	.270	.591	.439	.060	79	8.6
	2	**1**	.766	.626	.496	.383	.269	.588	.432	.060	60	6.7
		8	.776	.630	.492	.374	.271	.585	**.439**	.062	59	4.3

Table 1.8: Caption decoder strategy comparisons for performance and efficiency. Here we compare the various caption generation models in relation to their performance (with the metric scores) and efficiency (with **training samples per second** and **inference samples per second**). It can be seen that the RNN models outperform the TN models across all metrics, with LSTM units being most effective. Such a performance drop may be a result of a lack of data as suggested by the proportion decrease of the CIDEr scores shown in **bold** between the datasets, with less of a drop for the MSR-VTT models. Efficiency, at least on our hardware, is similar no matter the type of model, with TNs only being more effective when they contain less parameters.

Firstly, considering the MSVD training results in Figure 1.26, it is clear that over-fitting starts occurring after about 20 epochs, with minor and rare validation improvements after this stage. Furthermore the validation results reflect the testing evaluations with the LSTM RNN outperforming the other models to a similar degree.

Considering the MSR-VTT training statistics in Figure 1.27 we can see similar over-fitting as was observed in the MSVD training. However, interestingly after reaching peak validation performance the RNNs start to become more inaccurate as they over-fit.

The over-fitting for both datasets is quite significant as both graphs taper off, with large performance gaps between the training versus validation data evaluations. The significant size of this gap is likely due to poor generalisation of specific concepts. As most concepts are rare (as shown in Section 1.1.4), the over-fitting models have learnt how to determine the specific visual features that relate to specific rare words, however such features are not good general representations for that word. This could potentially be remedied by more data, particularly of the rarest concepts, however it is a significant challenge for the captioning problem as the terms used for expressing the same thing can be vastly different.

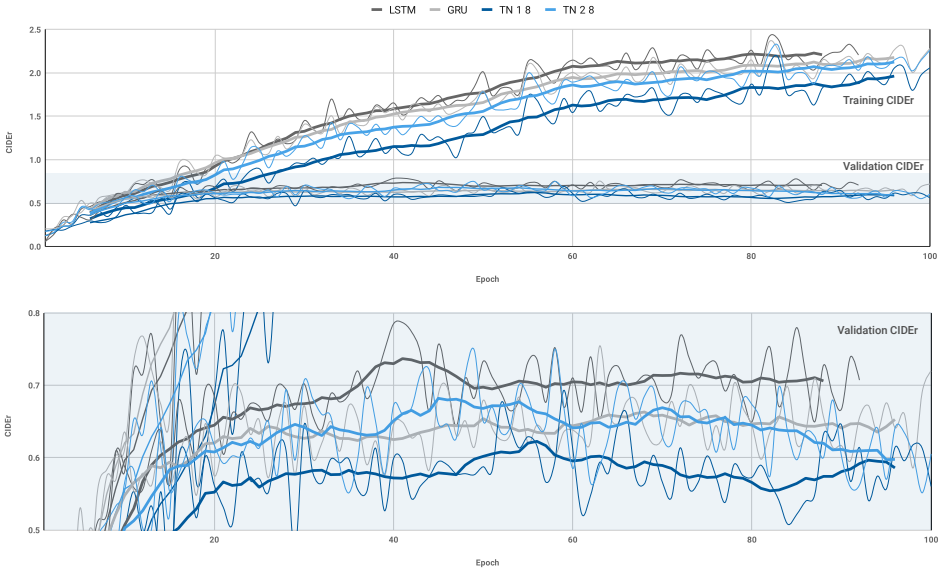


Figure 1.26: The training statistics for RNN vs TN on MSVD. Here we show the CIDEr scores of our models as evaluated with the training and validation data splits of MSVD throughout the full training procedure. The bottom graph is an inset of the top graph, zoomed in on the validation curves, as indicated by the blue shading. Also, the thick lines are the running average considering the 10 surrounding scores, as show by the smoothed thin line. We can see fairly early on in the training (within 20 epochs), that the training metrics start to pull away from the validation metrics showing over-fitting to the training data. Looking at the zoomed in bottom section of the graph it can be seen there is some slight improvements in validation performance after epoch 20 however it is minor in comparison to the training accuracy. Furthermore, the differences in model performance seen in the testing scores is reflected in these graphs, with the LSTM RNN consistently performing better than the other models, while for validation performance the GRU and 2-layer TN model's scores are similar.

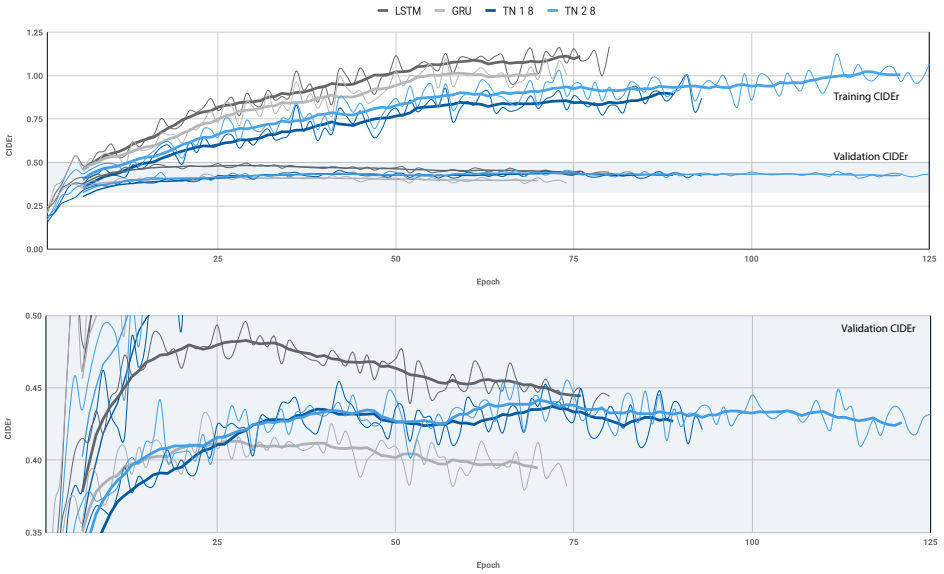


Figure 1.27: The training statistics for RNN vs TN on MSR-VTT. Here we show the CIDEr scores of our models as evaluated with the training and validation data splits of MSR-VTT throughout the full training procedure. The bottom graph is again an inset of the top graph, zoomed in on the validation curves, as indicated by the blue shading. Also, the thick lines are the running average considering the 10 surrounding scores, as show by the smoothed thin line. In this case we see that the RNN models quickly reach their peak validation performance at around epoch 25, before decreasing as the model over-fits. The TNs are slower to learn, with peak validation performance around epoch 70. All models are stopped by the early stopping process, hence why they don't reach the right side of the graph or 200 epochs.

1.2.3 Input Feature Encoder

Thus far, unfiltered features $\mathbf{F}$ have been utilised as input to our two captioning schemes — the RNN and TN. While these features are somewhat encoded by the RNN's inner encoding with $\mathbf{W}_x$, or the TN's MLP, both of these encodings occur after each captioners corresponding attention mechanism, resulting in the encoding of a single attended vector. Indeed each of the individual features are encoded when forming $\mathbf{F}$ (recall $\mathbf{F} = \left[\mathbf{W}^{(\mathrm{f})}\mathbf{f}^{(\mathrm{f})}, \mathbf{W}^{(\mathrm{m})}\mathbf{f}^{(\mathrm{m})}, \mathbf{W}^{(\mathrm{r})}\mathbf{F}^{(\mathrm{r})}, \mathbf{W}^{(\mathrm{c})}\mathbf{f}^{(\mathrm{c})}\right]$), however this is an encoding for ensuring all features are of the same dimensionality, with the encodings being independent of one another.

Following practices seen in machine translation tasks, where the TN architecture originated, we look to utilise a TN encoder to act as an initial filtering process of our set of features — $\tilde{\mathbf{F}} = \varphi(\mathbf{F})$. This filtering process can be utilised as a pre-cursor to the concept based filtering processes to be described in Section 1.3. We define the $\mathrm{EncoderBlock}_l(\cdot)$ which performs self-attention and encoding across the features, resulting in a set of filtered visual features that are encoded contextually conditioned on each other. The $\mathrm{EncoderBlock}_l(\cdot)$ is similar to the $\mathrm{DecoderBlock}_l(\cdot)$ as described in Section 1.2.2, but only performs a single multi-head attention:

$$\begin{aligned}
\mathbf{u}''_n &= \mathrm{MultiHeadAttention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) \\
\mathbf{u}'_n &= \mathrm{LayerNorm}(\mathbf{q}_n + \mathbf{u}'_n) \\
\mathbf{z}'_n &= \mathrm{FullyConnected}(\mathbf{u}_n) \\
\mathbf{z}_n &= \mathrm{LayerNorm}(\mathbf{u}_n + \mathbf{z}'_n)
\end{aligned} \tag{1.53}$$

We utilise it to encode each of our visual features $\mathbf{F} \mapsto \tilde{\mathbf{F}}$:

$$\begin{aligned}
\mathbf{z}_{f,1} &= \mathrm{EncoderBlock}_1(\mathbf{f}_f, \mathbf{F}, \mathbf{F}) \\
\mathbf{z}_{f,2} &= \mathrm{EncoderBlock}_2(\mathbf{z}_{f,1}, [\mathbf{z}_{1,1}, \ldots, \mathbf{z}_{F,1}], [\mathbf{z}_{1,1}, \ldots, \mathbf{z}_{F,1}]) \\
&\vdots \\
\mathbf{z}_{f,L} &= \mathrm{EncoderBlock}_L(\mathbf{z}_{f,L}, [\mathbf{z}_{1,L-1}, \ldots, \mathbf{z}_{F,L-1}], [\mathbf{z}_{1,L-1}, \ldots, \mathbf{z}_{F,L-1}])
\end{aligned} \tag{1.54}$$

$$\tilde{\mathbf{F}} = [\mathbf{z}_{1,L}, \ldots, \mathbf{z}_{F,L}] \tag{1.55}$$

Table 1.9 presents a performance comparison between the models with and without the TN feature encoder. Results are mixed, with the encoder leading to a reduction in performance of the LSTM RNN for both MSVD and MSR-VTT. In the case of the TN decoders, including a TN encoder does improve results in terms of some metrics, will better or equal results for CIDEr and SPICE, with most improvement coming from the larger MSR-VTT dataset. This once again might be an indication that the transformer requires more data to generalise in comparison to the RNN, with the encoder degrading the features for the RNN case.

1.2.4 Network Reduction

As seen in Section 1.2.2, over-fitting and generalisation for these captioning models is a significant challenge. One of the procedures for addressing over-fitting is reducing model size, so as

Decoder	Layers	Heads	Encoder	B1	B2	B3	B4	MT	RG	Cr	SP
MSVD											
LSTM	**1**	**-**	✗	.768	.642	.542	.436	.313	.674	**.665**	.045
			✓	.760	.625	.519	.411	.305	.664	.637	.043
TN	**1**	**8**	✗	.751	.629	.531	.424	.300	.656	.591	.043
			✓	.742	.608	.509	.407	.296	.646	**.599**	.043
MSR-VTT											
LSTM	**1**	**-**	✗	.787	.643	.511	.396	.280	.600	**.474**	.066
			✓	.783	.640	.499	.374	.273	.591	.470	.064
TN	**2**	**8**	✗	.776	.630	.492	.374	.271	.585	.439	.062
			✓	.778	.635	.494	.374	.271	.589	**.458**	.062

Table 1.9: Input feature encoder performance comparisons. Here we present the performance for both LSTM RNN and TN captioning models with and without the addition of a TN based visual feature encoder. With the inclusion of the encoder, performance for the RNN model on both MSVD and MSR-VTT is decreased across all metrics. Conversely, for the TN model, the encoder does see improvements in the CIDEr metric for both datasets.

to force the model to try and generalise more. Furthermore, TNs can have varying numbers of layers L and heads H, where different combinations may perform better than others. We therefore investigate the use of more layers and heads but smaller dimensionality, in our TN based encoder-decoder using several different combinations. Table 1.10 presents the results showing that while scores for the MSVD dataset are mixed, our 512-dimensional 2-layer 8-head model is clearly the better choice for MSR-VTT.

Dim	Layers	Heads	B1	B2	B3	B4	MT	RG	Cr	SP
MSVD										
512	**1**	**8**	.742	.608	.509	.407	.296	.646	.599	.043
	2		.750	.623	.522	.415	.303	.653	**.617**	.042
256	**4**		.758	.625	.519	.409	.299	.660	.598	.041
	8		.764	.638	.542	.435	.293	.657	.561	.038
	4	**16**	.761	.628	.527	.421	.301	.665	.593	.040
	8		.753	.624	.519	.410	.297	.648	.574	.041
MSR-VTT										
512	**1**	**8**	.749	.605	.473	.358	.261	.580	.423	.060
	2		.778	.635	.494	.374	.271	.589	**.458**	.062
256	**4**		.761	.618	.484	.368	.265	.586	.432	.057
	8		.774	.621	.480	.360	.265	.575	.406	.061
	4	**16**	.771	.628	.490	.370	.267	.586	.427	.060
	8		.757	.614	.477	.359	.261	.571	.415	.061

Table 1.10: Transformer Network parameter variation performance comparisons. This table shows the metric evaluations of our model using the TN caption generator with TN feature encoder, using either $d^{(\mathrm{F})} = 256|512$ dimensional features, $L = 1|2|4|8$ layers, and $H = 8|16$. The MSR-VTT results show that the $512, 2, 8$ combination, which contains the most parameters is the most suitable. Results on the MSVD dataset are more mixed, however the more reliable CIDEr metric score is again the highest for the $512, 2, 8$ combination model.

1.2.5 Caption Generation Analysis

Although the metrics give some indication of captioning performance, they don't tell the whole story. We therefore take a closer look at the predicted versus the ground truth (GT) captions to uncover more information how the models are actually performing.

Human Performance Analysis

When describing things, such as captioning videos, humans naturally vary in how they describe the same occurrence. This was shown to some extent with the general versus unique noun and verb occurrences for the datasets in Section 1.1.4 and is a difficult factor to evaluate. We therefore utilise the multiple GT captions for each of the videos, and compare them against one another to get a measure of human performance. Specifically, we randomly select one caption per video to be the prediction and evaluate it against the rest which are kept as GT. This is repeated 100 and 20 times for the MSVD and MSR-VTT datasets test splits respectively, with the results averaged to get the scores presented in Table 1.11.

BLEU 1	BLEU 2	BLEU 3	BLEU 4	METEOR	ROUGE-L	CIDEr	SPICE
MSVD							
.874 ±.008	.765 ±.010	.660 ±.012	.562 ±.013	.423 ±.005	.758 ±.007	1.194 ±.030	.097 ±.002
MSR-VTT							
.761 ±.003	.578 ±.005	.441 ±.005	.345 ±.006	.296 ±.003	.565 ±.004	.500 ±.005	.092 ±.001

Table 1.11: Human performance anaylsis for video captioning. This table shows the metric evaluations of the GT captions for the MSVD and MSR-VTT datasets. These results are for the test splits, however as to be expected we find equivalent results for the training and validation sets. Listed are the mean and standard deviations across 100 and 20 runs of randomly sampling a caption from the set of GT captions from each sample in MSVD and MSR-VTT respectively.

Intuition would suggest that these human evaluations would act as a soft upper-bound on our model performances, considering we are comparing a human caption to a set of other human captions, however this isn't the case. If we directly compare these human scores with scores from our testing, validation, and training procedures we find that the models do better, particularly on the training data.

Let's consider the LSTM RNN captioning model which doesn't include a feature encoder. Table 1.12 presents the testing and validation results from the best validation epoch, as well as the training results from the final epoch which is heavily over-fitted. It can be seen that for MSVD the human evaluations are better for the validation and test splits, however for the relatively well-fitted training data, the model outperforms the human annotations. This result is found for MSR-VTT also, with the model even performing better than humans on the validation and test data with the BLEU and ROUGE-L metrics. These results may seem counter-intuitive at first, but if we take a closer look at the predicted versus the ground truth captions it becomes clearer why this might be the case.

Split	Type	B1	B2	B3	B4	MT	RG	Cr	SP
					MSVD				
test	**Human**	.874	.765	.660	.562	.423	.758	1.194	.097
	LSTM	.768	.642	.542	.436	.313	.674	.665	.045
val	**Human**	.883	.783	.682	.588	.435	.777	1.344	.098
	LSTM	.770	.650	.552	.458	.314	.689	.787	.042
train	**Human**	.874	.767	.663	.566	.422	.759	**1.190**	.099
	LSTM	.950	.919	.889	.851	.554	.904	**2.207**	.104
					MSR-VTT				
test	**Human**	.761	.578	.441	.345	.296	.565	.500	.092
	LSTM	.787	.643	.511	.396	.280	.600	.474	.066
val	**Human**	.762	.582	.446	.351	.299	.571	.511	.093
	LSTM	.800	.659	.523	**.405**	.289	**.609**	.496	.068
train	**Human**	.762	.582	.446	.353	.302	.576	**.514**	.094
	LSTM	.939	.868	.780	.695	.425	.800	**1.168**	.107

Table 1.12: Human versus LSTM RNN performance for video captioning. This table compares the human annotation evaluations and the LSTM RNN model generation evaluations on the test, validation and training splits of MSVD and MSR-VTT. Humans outperform the model for most metrics when considering the testing or validation data*, however on the training data the highly over-fitted model performs significantly better. This result is consistent across both datasets with CIDEr increases from 1.190 to 2.207 and .514 to 1.168 for MSVD and MSR-VTT respectively. *With the exception of the BLEU and ROUGE-L metrics, which score higher than human evaluations for the MSR-VTT validation split, which the model isn't over-fitting to.

Let us consider the word, noun and verb count percentages for each of the datasets, as shown with histograms in Figure 1.28. By considering these graphs we can see that the word, noun and verb distributions in the GT captions are higher and more long-tailed relative to the predicted captions. This highlights that the predicted captions are shorter and more concise than the human annotated captions. One might initially assume that this reduction in predicted caption length, as well as in noun and verb usage, would be the cause for poorer metric accuracy on the testing data — since the shorter captions could miss important details found in the GT captions. Considering the training split distributions however, which are similar to the testing split distributions, and considering that the over-fitting model significantly outperforms the human captions, we can confirm that the more concise captions don't have significant negative effects on metric scoring. These results suggest that the model has a tendency to prefer generating shorter more concise captions, which have no adverse affect on the metric scoring. In fact, if we assume that the humans know what they are talking about, and use relatively correct language to refer to concepts, and that the model, even when over-fitted is likely to be more incorrect overall, then it suggests the metrics are suited to the brevity of the model generated captions.

Considering this idea further, we believe that potentially the random sampling of human captions may unfairly prevent them from achieving the kind of performance as seen with the model results on the over-fitted data. If we once again consider the variation of language in human annotations (as shown back in Section 1.1.4), and that we randomly sample singular captions and compare to all others per video, then there is a relatively high chance (0 - 50%) that we sample a caption that is relatively poor compared to the majority of the other GT captions for that video. For example, say that for each video, 75% of the GT captions say the same thing, while the other 25% say something else, then with our random sampling we'd expect to get captions that disagree with the majority 25% of the time. When we sample and evaluate one of these minority captions the resulting metric scores are relatively low, compared to an over-fitting model. As our model learns it would be trying to best fit all GT captions per video, so it tends towards generating an '*average*' caption that tries to match the majority of the GT captions. In terms of our example, this results in our model generating captions that are more like the 75% majority, resulting in *potentially* higher likelihood of good metric performance in comparison to random sampling where 25% of the time we get a poorer score. To what extent this actually plays a role is hard to determine, after all we are averaging over the dataset, so there should be more majority based scores rather than minority. Also, considering the MSR-VTT validation results in Table 1.12, this suggests that for some metrics (BLEU, ROUGE-L), the shorter '*average-like*' captions score higher regardless of over-fitting. Other sampling methods could potentially be utilised, such as taking the best human scoring caption per sample, however 'cherry-picking' like this may give too high of a score to the human captions, not appropriately reflecting *average* human performance. We conclude that it's non-trivial and ambiguous to determine what approach should be used for evaluating the human performance. Further thought and study is needed on this analysis.

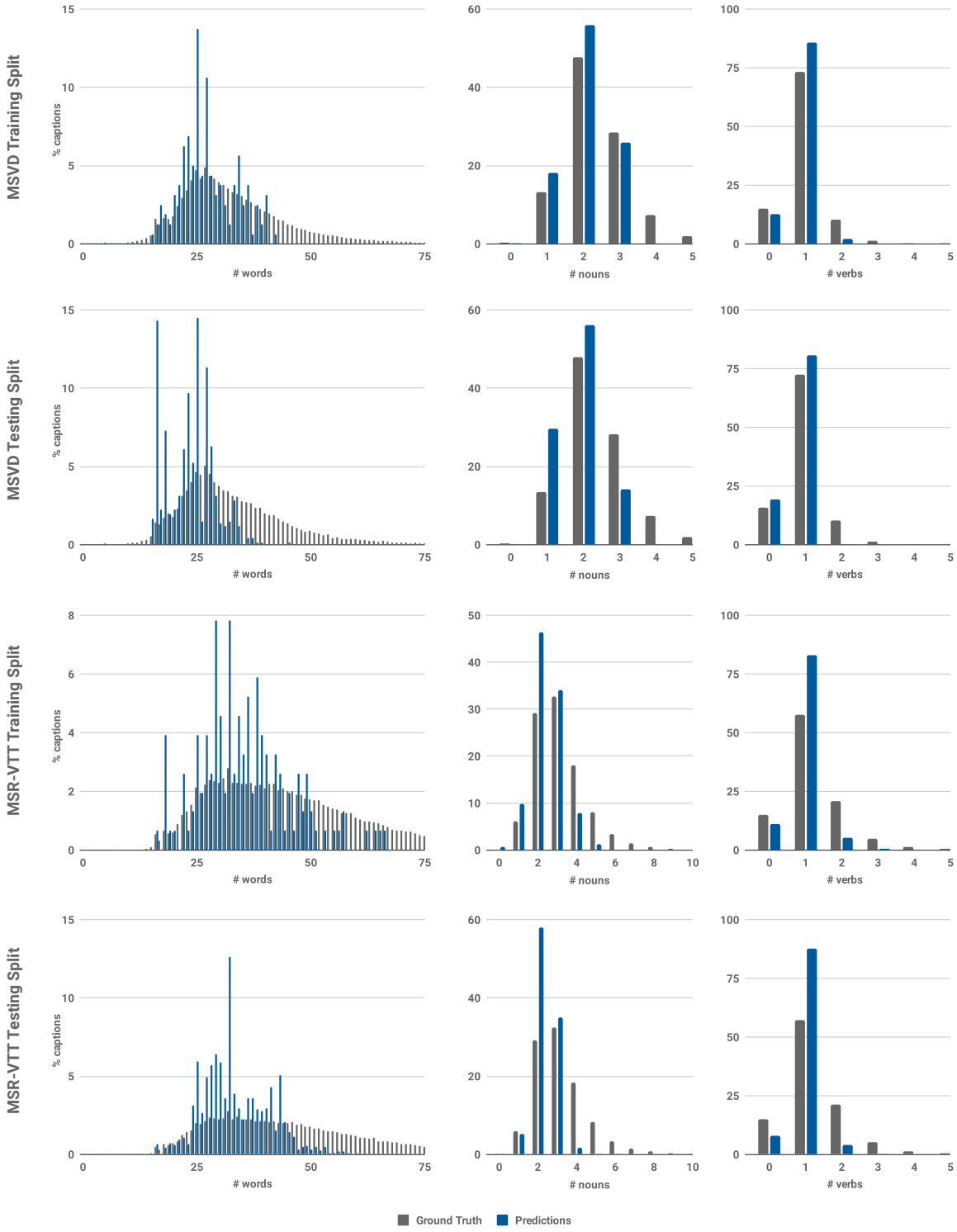


Figure 1.28: Caption percentage histograms for human and model generated captions in regards to word, noun and verb counts. From left to right this figure shows the % of captions that contain a certain number of word, noun, and verbs for the GT (grey) and RNN model generations (blue). From top to bottom these counts are on the MSVD training, MSVD testing, MSR-VTT training and MSR-VTT data splits. It shows that, no matter the dataset or split, the model tends to generate shorter more concise captions.

Concept Analysis

We have just seen how the metrics prefer more succinct, concise captions, however the performance disparity between the training split generations and the testing split generations despite similar word, noun and verb distributions suggest that the specific words, nouns and verbs play a significant role in model performance. We therefore take a closer look at the comparisons between the generated captions and the ground truth captions for both the training and testing splits, looking to determine how wrong individual nouns or verbs are.

For this comparison, for each video we extract the nouns and verbs from the generated captions and compare them to the top three nouns and top verb from the collection of GT captions. We calculate the precision, recall and F1-score for exact matches, as well as the euclidean Glove[10] embedding distance. Similarly to the human evaluations above, we include *pseudo*-human results by using the same random sample hold-one-out method.

Table 1.13 presents the results averaged across the datasets, showing the comparison between the testing and training splits and the human and model performance. We find that for the MSVD training, and MSR-VTT training and testing splits, that the LSTM RNN model performs significantly better at determining concepts. The MSR-VTT test split result is particularly interesting as it suggests that despite the model captions being shorter they still cover the important concepts. In fact, the conciseness of the model captions may be beneficial in this evaluation as the longer GT captions might introduce more than three nouns and one verb, resulting in a drop in precision. This wouldn't effect the recall however, and indeed the noun recall is higher for the human captions for MSR-VTT.

Lastly, to get an indication of what concepts are and aren't being learnt, we take a look at the error rates of the nouns and verbs that are predicted on the test splits of each dataset. As the model is bound by what concepts appear in the training sets, we first determine individual noun and verb vocabularies for both the MSVD and MSR-VTT datasets based on the GT captions. We parse the training sets extracting the unique nouns and verbs that occur in the GT captions, resulting in 6111 nouns and 3413 verbs for MSVD, and 15091 nouns and 8568 verbs MSR-VTT[11]. From these we take only those that occur in at least 20 GT captions, resulting in 697 nouns and 324 verbs for MSVD, and 1921 nouns and 797 verbs for MSR-VTT. As the NLTK tokenizer is imperfect, 78 and 300 words appear as both nouns and verbs for MSVD and MSR-VTT respectively. We manually examine these intersecting words and assign them to either the noun or verb vocabularies. Our final sets of unique nouns and verbs consist of 647 nouns and 296 verbs for MSVD, and 1750 nouns and 668 verbs for MSR-VTT.

Similarly for this comparison we are comparing generated concepts to the top three nouns and top verb from the collection of GT captions for each video. The noun and verb true positive (TP), false positive (FP) and false negative (FN) counts for our LSTM RNN model on MSVD and MSR-VTT are shown in Figure 1.29 and Figure 1.30 respectively.

[10] We use the 300-dimensional Glove-6B

[11] these statistics match up with those in Table 1.6 in Section 1.1.4

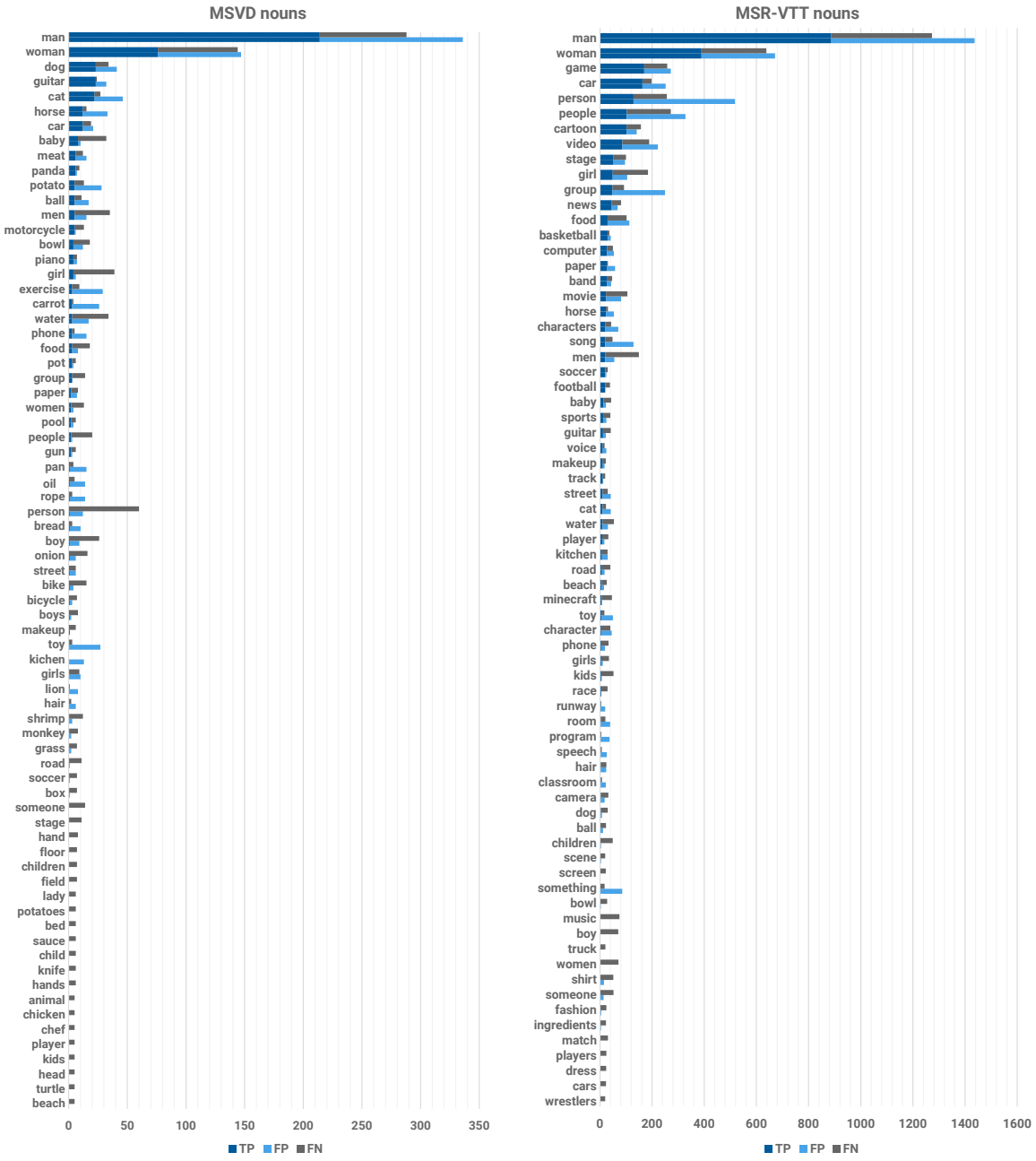


Figure 1.29: Noun TP, FP and FN counts on generated captions for the test splits of MSVD and MSR-VTT. These bar charts show the generated caption nouns true positive (TP), false positive (FP) and false negative (FN) counts in order of highest F1-score for MSVD (left) and MSR-VTT (right). For brevity, we only list predicted nouns that appear in the training vocabularies for their corresponding dataset, and that either have k predictions or GT video occurrences, where $k = 5$ for MSVD and $k = 20$ for MSR-VTT. **Note that the bars are stacked**, so by considering the dark blue and the dark grey bars together we can see the *proportion of positive GT samples* that are actually detected compared to those that are missed ie. for the noun *man* in MSVD, about 70% of the *man* occurrences are predicted correctly. Similarly, by considering the dark blue and the light blue bars together we can see the *proportion of predictions* that are actually attributed correctly to a GT ie. again for *man* in MSVD, about 60% of the predicted "*man*" nouns are correct.

Split	Type	Precision		Recall		F1 Score		Distance	
		Nouns	Verbs	Nouns	Verbs	Nouns	Verbs	Nouns	Verbs
					MSVD				
test	Human	.581	.349	.433	.375	.480	.358	2.903	4.114
	LSTM	.416	.248	.249	.248	.303	.248	4.008	5.088
train	Human	.582	.336	.439	.362	.485	.344	2.896	4.260
	LSTM	.818	.613	.559	.619	.647	.615	1.228	2.220
					MSR-VTT				
test	Human	.377	.186	**.355**	.234	.351	.201	4.488	5.422
	LSTM	.431	.330	**.315**	.336	.357	.332	3.901	4.323
train	Human	.373	.190	.352	.237	.347	.204	4.522	5.375
	LSTM	.648	.463	.488	.484	.541	.469	2.430	3.253

Table 1.13: Comparison of human and RNN performance for caption concept generation. In this table we present the concept generation performance of human captioners and our LSTM RNN. Performance is measured for nouns and verbs individually using classification metrics precision, recall, F1 Score and Euclidean distance in GloVe embedding space. It can be seen that other than the MSVD test set, the RNN model achieves better performance, with the exception of the nouns recall (**bolded**). This is likely because our choice of three nouns suit the short model generated captions, causing drops in precision but not recall when the longer GT captions include four or more nouns.

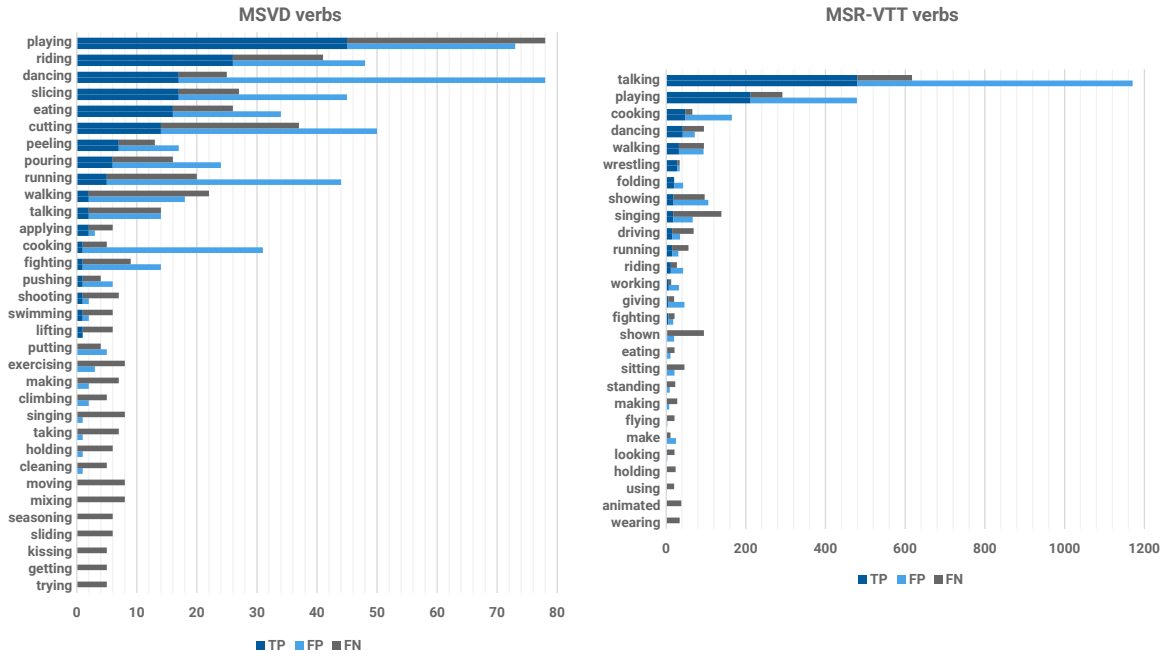


Figure 1.30: Verb TP, FP and FN counts on generated captions for the test splits of MSVD and MSR-VTT. These bar charts show the generated caption verbs true positive (TP), false positive (FP) and false negative (FN) counts in order of highest F1-score for MSVD (left) and MSR-VTT (right). For brevity, we only list predicted verbs that appear in the training vocabularies for their corresponding dataset, and that either have k predictions or GT video occurrences, where $k = 5$ for MSVD and $k = 20$ for MSR-VTT. Once again, note that the bars are stacked, ie. the number of FN for the verb "*playing*" in MSVD is $78 - 45 = 33$.

From these graphs we can note a number of interesting things. Firstly, it is clear that the majority of nouns and verbs are rare, which is to be expected with the diversity of the language and the content of the two datasets. Secondly, the more common nouns and verbs look to be predicted the most accurately, further suggesting that the diversity of the datasets is an issue for model generalisation and performance. For the nouns *baby*, *men*, *girl* and *person* are often missed when generating a caption suggesting cues for these different kinds of humans are lacking. Furthermore, for nouns *man*, *woman*, *person* (in MSR-VTT), and *people*, there are many false positives suggesting a tendency to just use these relatively common words when unsure, likely when specifically unsure of the kind of human. In comparison to the nouns, the verbs particularly for MSVD, are more evenly spread, however are more difficult to determine with greater false positive and false negative proportions. Visually, verbs are naturally more difficult to identify in comparison to nouns due to them often relating on contextual and motion cues, rather than just visual appearance cues. This remains a challenge for the visual feature extraction process to provide insightful features that can more easily allow for verb and action determination.

1.3 Video Captioning with Concepts

As seen in the previous section, correctly identifying individual nouns and verbs when generating a caption is critically important. We have also seen how all models are prone to over-fitting, with the testing and validation scores quickly hitting their peak during the training while the training scores continue to improve significantly thereafter. A potential means of addressing both of these issues is with the addition of individual noun and verb groundings prior to caption generation. Adding a classification loss on individual nouns and verbs acts as regularisation on the model, giving it another objective to solve. This idea is similar to what past works have done with attributes [Gan et al., 2017, Lu et al., 2018, Anderson et al., 2018, Gan et al., 2017] and Parts-of-Speech (POS) [Wang et al., 2019, Zheng et al., 2020], however we introduce a multi-stage and a Transformer Network (TN) to generate concepts conditioned on past concepts. Furthermore, we investigate two different strategies for incorporating our concept predictions with the captioning modules — as a concatenation with the input features (as described in Section 1.3.1) and a decoupled combination (as described in Section 1.3.2). For each of these approaches we compare three different module architectures for concept prediction, a single-stage attribute classifier (like that seen in [Gan et al., 2017, Pan et al., 2017]), a multi-stage subject-verb-object (SVO) classifier (which is a more general version of [Zheng et al., 2020]) and an iterative SVO classifier which utilises a TN and is akin to a mini-caption generator. The number of concepts we produce and utilise varies depending on the approach.

We utilise the noun and verb vocabularies introduced in Section 1.2.5, which consists of 647 nouns and 296 verbs for MSVD, and 1750 nouns and 668 verbs for MSR-VTT. In practice we combine the noun and verb vocabularies together for each dataset, allowing generation of both nouns or verbs at the same time. This results in vocabulary sizes of $|\mathring{\mathcal{D}}| = 943$ for MSVD and $|\mathring{\mathcal{D}}| = 2418$ for MSR-VTT.

1.3.1 Input Feature Stacking

Incorporating the concepts into the caption generation modules as to best exploit the concept information is an important problem to consider. We embed our concepts with the same embedding as our caption words, resulting in our set concept embeddings $[\mathring{\mathbf{e}}_1, \dots, \mathring{\mathbf{e}}_C]$ being $\in \mathbb{R}^{d^{(\mathring{\mathrm{e}})}}$. As our visual features are of the same dimension where $d^{(\mathring{\mathrm{e}})} = d^{(\mathrm{F})} = 512$, we first investigate stacking the concept word embeddings as extra input features:

$$\tilde{\mathbf{F}} = [\mathbf{F}, \mathring{\mathbf{e}}_1, \dots, \mathring{\mathbf{e}}_C] \tag{1.56}$$

We investigate three concept generation strategies which are described in the following subsections, before comparing their performance in the final subsection.

Single-Stage Attributes Classification

We begin with performing attributes classification, where an attribute is simply a noun or a verb. We consider the attribute classification task as a multi-label classification problem, where we look to classify a set of attributes for each video in a single classification stage. For each video we take the top five ($C = 5$) occurring words from the GT captions which also exist in our vocabularies as GT concepts. We choose five as we believe this should adequately cover the three nouns and two verbs which is the maximum the model tends to generate as was shown in Figure 1.28 in Section 1.2.5. For each video with predicted concept probabilities $\mathring{\rho}_{\mathring{k}}$ and GT labels $\mathring{y}_{\mathring{k}} \in \{0, 1\}$ for each of the $\mathring{k}$ concepts in $\mathring{\mathcal{D}}$, equalling 0 when the concept isn't present and 1 when it is. The concept loss is a binary cross-entropy loss:

$$L^{(\text{concept})} = -\frac{1}{|\mathring{\mathcal{D}}|} \sum_{\mathring{k}}^{|\mathring{\mathcal{D}}|} w_{\mathring{k}} \cdot \mathring{y}_{\mathring{k}} \cdot \log(\mathring{\rho}_{\mathring{k}}) + (1 - \mathring{y}_{\mathring{k}}) \cdot \log(1 - \mathring{\rho}_{\mathring{k}}) \tag{1.57}$$

where $w_{\mathring{k}}$ is a weighting for each word in the concept vocabulary and is calculated as each words inverse occurrence proportion:

$$w_{\mathring{k}} = \frac{|I^{(\text{s})}| - o_{\mathring{k}}}{o_{\mathring{k}}} \tag{1.58}$$

where $|I^{(\text{s})}|$ is the total number of captions in the dataset and $o_{\mathring{k}}$ is the total number of word occurrences for word $\mathring{k}$ across the dataset.

The final loss is calculated by adding the concept loss with the caption loss which was introduced in Equation (1.52) in Section 1.2.2:

$$L = L^{(\text{caption})} + \alpha L^{(\text{concept})} \tag{1.59}$$

where α is a balancing term, which is set to $\alpha = 2$ for our experiments.

The framework is shown in Figure 1.31 where at first a dot-product attention over all of the input features is applied, with the image feature used as the query to the attention:

$$\mathbf{u} = \text{DotProdAttention}(\mathbf{W}^{(\text{f})}\mathbf{f}^{(\text{f})}, \mathbf{F}, \mathbf{F}) \tag{1.60}$$

The attended input feature is then encoded with a small MLP which can have one or more layers of $\text{MLP}(\cdot) = \text{Dropout}(\text{ReLu}(\text{Linear}(\cdot)))$, we however just use one layer:

$$\begin{aligned} \mathbf{u}' &= \text{MLP}(\mathbf{u}) \\ &= \text{Dropout}(\text{ReLu}(\text{Linear}(\mathbf{u}))) \\ &= \text{Dropout}(\text{ReLu}(\mathbf{W}^{(\text{u})}\mathbf{u} + \mathbf{b}^{(\text{u})})) \end{aligned} \tag{1.61}$$

where the linear weight $\mathbf{W}^{(\text{u})} \in \mathbb{R}^{d^{(\text{F})} \times d^{(\text{F})}}$ and bias $\mathbf{b}^{(\text{u})} \in \mathbb{R}^{d^{(\text{F})}}$.

We then transform the $\mathbf{u}'$ into the concept vocabulary space, before applying a $\text{sigmoid}(\cdot)$ to determine the concept probabilities across $\mathring{\mathcal{D}}$:

$$\mathring{\rho} = \text{sigmoid}(\mathbf{W}^{(\mathring{\rho})}\mathbf{u}' + \mathbf{b}^{(\mathring{\rho})}) \tag{1.62}$$

where $\mathbf{W}^{(\mathring{\rho})} \in \mathbb{R}^{d^{(\mathrm{F})} \times |\mathring{\mathcal{D}}|}$ and $\mathbf{b}^{(\mathring{\rho})} \in \mathbb{R}^{|\mathring{\mathcal{D}}|}$.

The top 5 concepts, based on their confidence scores, are embedded and stacked onto the end of the visual input features, as shown further above. The $\tilde{\mathbf{F}}$ is then passed into to captioning modules as before.

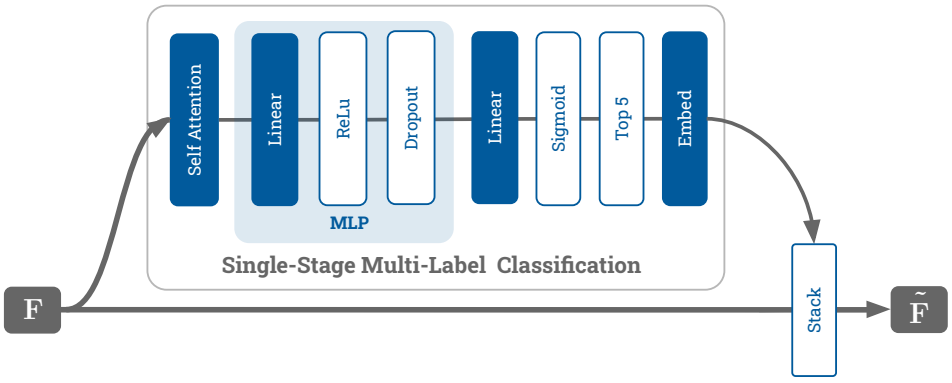


Figure 1.31: Visual representation of the single-state multi-label attributes classifier module. Given the visual features $\mathbf{F}$ we generate a set of concepts which are stacked with the initial features to form $\tilde{\mathbf{F}}$. The visual features are first passed through a self-attention mechanism before being passed through a small MLP, before being transformed and classified with a $\mathrm{sigmoid}(\cdot)$ to generate concept probabilities $\mathring{\rho}$. The top $C = 5$ concepts are then taken and embedded before being stacked with the initial features.

Multi-Stage Ordered POS Classification

While unordered attributes classification is useful, we also investigate whether asserting order for the attribute classification improves captioning results. When working with ordered concepts, we condition each generated concept based on the previously generated concepts. As the ordering of our ground truth concepts are simply in order of popularity they don't contain any contextual structure to them. To improve the contextual structure, instead of using the top five concepts, for these experiments we use contextually ordered Parts-of-Speech (POS) tags, namely subject-verb-object (SVO) triplets, where the $C = 3$ concepts used are ordered in a meaningful and constructive way. Other works [Wang et al., 2019, Zheng et al., 2020] have found improvements with utilisation of SVO triplets. Furthermore considering the findings from Section 1.2.5 that show the captioning models look to generally predict two nouns and one verb, despite human captioners using more descriptive language, suggests that SVO triplets are sufficient for these two datasets.

To generate the SVO triplets we again utilise the Python Natural Language Toolkit, which uses a trained[12] tri-gram subject tagger for generating the subjects. Then for each subject the corresponding caption is checked for a verb and noun to complete the SVO triplets. This results in the potential to have multiple SVO triplets per caption, and per clip. For our experiments we utilise the same SVO data as [Zheng et al., 2020].

This ordered classification of concepts, namely SVOs, can be considered like a *mini-captioning*

[12] Trained on Brown, conll2000, and TreeBank corpuses — see https://www.nltk.org/book/ch02.html

problem. With concept words being generated one at a time conditioned on the past words. For these ordered experiments we use the same cross-entropy loss as used for the captioning:

$$\begin{aligned} L^{(\text{concept})} &= -\sum_{c=1}^{C} \log(\Pr(\hat{\mathring{y}}_c = \mathring{y}_c)) \\ &= -\sum_{c=1}^{C} \log(\mathring{\rho}_c^\top \mathring{\omega}(\mathring{y}_c)) \end{aligned} \tag{1.63}$$

where $\mathring{\omega}(\mathring{o}_c)$ is the one-hot vector for our desired ground truth concept $\mathring{y}_c$. This is combined with the caption loss in the same way as previously discussed.

A simplified way of doing this is to use individual attention and MLP networks for each of the C concepts based on the visual features $\mathbf{F}$. So for each concept c we begin by again utilising a dot-product attention mechanism which uses either the previous word embedding $\mathring{\mathbf{e}}_{c-1}$ for the previously generated concept $\mathring{y}_{c-1}$, or the image feature $\mathbf{W}^{(\text{f})}\mathbf{f}^{(\text{f})}$ as the query and the visual features $\mathbf{F}$ as the keys and values:

$$\mathbf{u}_c = \begin{cases} \text{DotProdAttention}(\mathbf{W}^{(\text{f})}\mathbf{f}^{(\text{f})}, \mathbf{F}, \mathbf{F}) & \text{if } c = 1 \\ \text{DotProdAttention}(\mathring{\mathbf{e}}_{c-1}, \mathbf{F}, \mathbf{F}) & \text{if } c > 1 \end{cases}. \tag{1.64}$$

The attended input feature is then encoded with a small MLP which can have one or more layers of $\text{MLP}(\cdot) = \text{Dropout}(\text{ReLu}(\text{Linear}(\cdot)))$, we however just use one layer:

$$\begin{aligned} \mathbf{u}'_c &= \text{MLP}(\mathbf{u}_c) \\ &= \text{Dropout}(\text{ReLu}(\text{Linear}_c(\mathbf{u_c}))) \\ &= \text{Dropout}(\text{ReLu}(\mathbf{W}_c^{(\text{u})}\mathbf{u} + \mathbf{b}_c^{(\text{u})}))), \end{aligned} \tag{1.65}$$

where the linear weights $\mathbf{W}_c^{(\text{u})} \in \mathbb{R}^{d^{(\text{F})} \times d^{(\text{F})}}$ and biases $\mathbf{b}_c^{(\text{u})} \in \mathbb{R}^{d^{(\text{F})}}$.

We then transform the $\mathbf{u}'_c$ into the concept vocabulary space, before applying a $\text{softmax}(\cdot)$ to determine the concept probabilities across $\mathring{\mathcal{D}}$:

$$\mathring{\rho}_c = \text{softmax}(\mathbf{W}_c^{(\mathring{\text{k}})}\mathbf{u}'_c + \mathbf{b}_c^{(\mathring{\text{k}})}) \tag{1.66}$$

where $\mathbf{W}_c^{(\mathring{\text{k}})} \in \mathbb{R}^{d^{(\text{F})} \times |\mathring{\mathcal{D}}|}$ and $\mathbf{b}_c^{(\mathring{\text{k}})} \in \mathbb{R}^{|\mathring{\mathcal{D}}|}$.

Each concept as determined by the highest probability, is then embedded and individually stacked onto the set of features as done for the previous method. Figure 1.32 presents the approach with our SVO triplets, so there are three attention mechanisms followed by three MLP networks.

Ordered POS Classification with Transformers

The multi-stage POS classification module discussed in the previous subsection is able to condition concept probability predictions based on both the visual features and the previously generated concept, however it doesn't consider all previously generated concepts, *eg. the object doesn't know about the subject*. Furthermore, increasing the number of concepts in such a

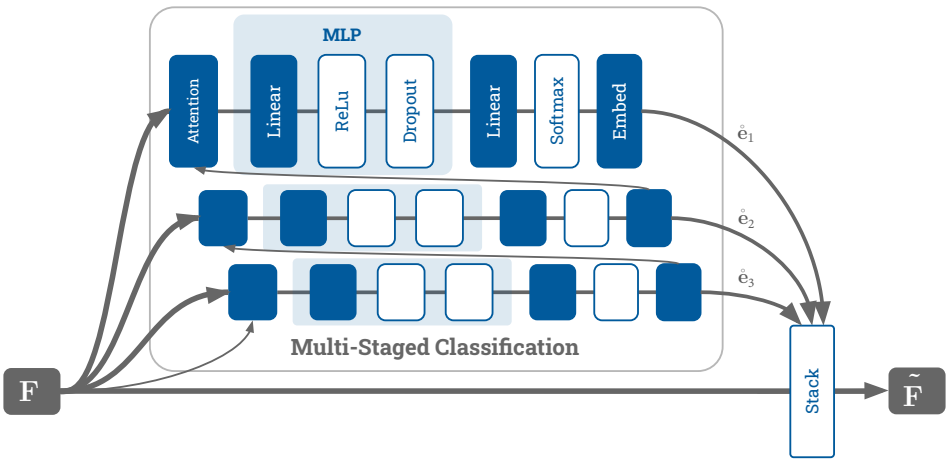


Figure 1.32: Visual representation of the multi-staged POS classifier module. Given the visual features $\mathbf{F}$ we generate a set of concepts (subject-verb-object triplets) which are stacked with the initial features to form $\tilde{\mathbf{F}}$. Here we generate concept probabilities and embeddings one at a time in subject-verb-object order, conditioning each generation on the previous generation ie. the object is conditioned on the verb and the verb is conditioned on the subject. Such conditioning is achieved via using word embeddings from the previous concept as dot-product attention queries over the set of features $\mathbf{F}$. The attended features are then passed through a single layer MLP with each being trained to encode either a subject, verb, or object.

model is difficult with new pathways needing to be implemented and learnt. To alleviate these problems, and considering concept and SVO prediction is like mini-caption generation, we also experiment with utilising a TN decoder module for generating SVO triplets. The concept probabilities per concept generation step is given by:

$$\mathring{\rho} = \text{TransformerDecoder}([\mathring{\mathbf{e}}_0, \ldots, \mathring{\mathbf{e}}_{c-1}], \tilde{\mathbf{F}}) \tag{1.67}$$

where $\mathring{\mathbf{e}}_c$ is the word embedding for the c^{th} concept. We utilise the same loss that is used for the multi-stage classification model discussed previously.

Figure 1.33 provides an overview of the framework with the TN decoder module, which works in the same way as the TN captioning modules, but for SVO triplets rather than length-N captions.

Model Comparisons

Table 1.14 and Table 1.15 present the results for the three model architectures described above, with attributes, SVOs and GT concepts, for MSVD and MSR-VTT respectively. The standard caption metrics are presented as well as the average Precisions (p) and Recalls (r) of concepts averaged across the vocabulary. The trends in results are similar for both MSVD and MSR-VTT.

Firstly, all three of the concept detection methods tested have difficulty learning correct concepts, with almost all precision and recall values being below 5%. Such inaccurate concept detection translates into the captioning performance, with all models performing worse except for SVO TN on the MSVD dataset. In the case of our single stage approach, the effects were detrimental, despite achieving the highest concept precision and recalls for the MSR-VTT dataset with the TN captioner. These reductions in performance suggest that having inaccurate concept features distracts the caption modules in negative ways.

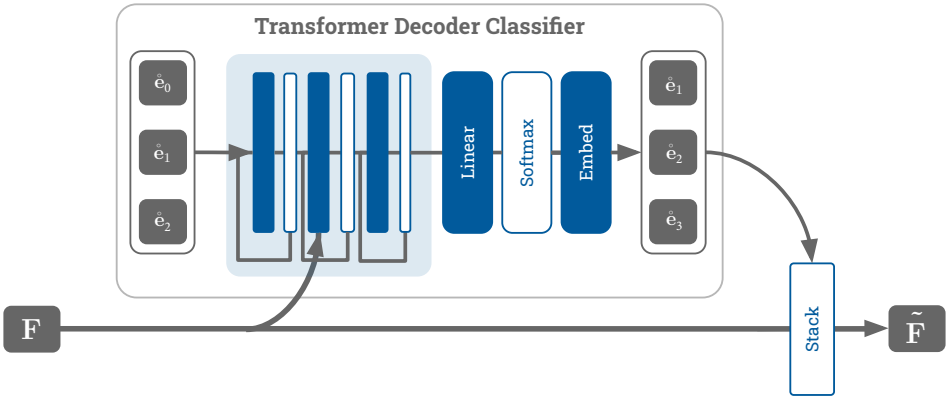


Figure 1.33: Visual representation of the TN decoder POS classifier module. Given the visual features $\mathbf{F}$ we generate a set of concepts (subject-verb-object triplets) which are stacked with the initial features to form $\tilde{\mathbf{F}}$. Here we generate concept probabilities and embeddings one at a time in subject-verb-object order, conditioning each generation on all previous generations. Such conditioning is achieved via using word embeddings from the previous concept as dot-product attention queries over all previous concept embeddings. These attended concept embeddings are then used as queries to attend to the set of features $\mathbf{F}$. The attended features are then passed through an MLP (which shares parameters across all steps, unlike the multi-stage approach).

Secondly, if we consider the performances when we pass our captioners the GT concepts, rather than inaccurate predictions from our detection models, we find significant increases in captioning performance — particularly on MSVD when using the five attributes. This suggests that despite the structure of the SVO triplets, having more concepts is the more beneficial option. As the majority of captioning metrics are based off of word matching, and the majority of words are rare and hence difficult to identify and generate, providing five concepts likely enables the generation of more of the rarer concepts in the caption leading to the significant captioning performance increases. The consistent improvements with both types of concepts highlights how they can be of great use to generating more accurate and particularly more diverse captions.

Lastly, and most notably, we note that while the LSTM RNN captioning model consistently outperforms the TN captioning model for the inaccurate concept detection modules, the opposite is true when using GT concepts. This suggests that the two models have different reliance on past word embeddings and the visual input features. The RNN captioner is likely more impacted by word history compared to visual features as the previous word embedding $\mathbf{e}_{n-1}$ makes up half of the input feature vector $\check{\mathbf{x}}$. While the other half does contain the attended encoding of all visual features $\tilde{\mathbf{f}}$, so much more information has to be squeezed into the same vector size for the visual feature case. Compare this with TNs where the visual features are more evenly used, with the attended word features $\mathbf{u}''$ being added to the attended visual features $\mathbf{u}'$. Together this explains why when concepts (and in general any features in $\tilde{\mathbf{F}}$) are poorer, the RNN can recover, whereas the transformer is more susceptible. This is however a benefit for TN moving forward, for as input features get more reliable and representative, the TN can make better use of the richer information compared to an RNN which may get stuck generating poorly grounded captions.

Captioner	Concepts	Method	B1	B2	B3	B4	MT	RG	Cr	SP	Pr	Re
LSTM	-	-	.722	.576	.473	.380	.300	.646	**.604**	.045		
	Att	Single	.677	.494	.375	.281	.246	.589	.290	.030	.016	.031
	Att GT		.911	.845	.782	.715	.454	.818	**1.395**	.077	1.0	1.0
	SVO	Multi	.733	.594	.487	.373	.287	.647	.524	.039	.017	.028
		TN	.747	.609	.508	.411	.304	.658	**.641**	.042	.034	.043
	SVO GT		.748	.613	.513	.415	.307	.668	.660	.043	1.0	1.0
TN	-	-	.742	.618	.521	.415	.302	.654	.573	.042		
	Att	Single	.640	.471	.358	.248	.234	.574	.236	.028	.021	.034
	Att GT		.930	.878	.815	.743	.481	.859	**1.582**	.085	1.0	1.0
	SVO	Multi	.718	.583	.475	.367	.287	.630	.514	.039	.033	.040
		TN	.746	.618	.516	.411	.299	.645	.566	.042	.034	.046
	SVO GT		.768	.646	.543	.432	.319	.673	.684	.045	1.0	1.0

Table 1.14: Grounded captioning and concept detection results on MSVD. This table presents the captioning performances (with the usual metrics) and the concept detection performances (using precision and recall) for the various concept detection methods (single, multi, and TN). We show captioning performance for both the LSTM RNN and the TN captioners. We also show captioning performance when the captioners are given the GT attributes or concepts. Regarding the LSTM captioner results it can be seen that we see a performance gain across all but the SPICE metric for the TN SVO generation model. For all other concept prediction models we see decreases in performance. Interestingly despite the detrimental captioning performance using the single concept prediction method, using the five attributes performs significantly better than using the SVO triplets when providing either of the captioners with the GT. Considering the precision and recall values we find all are very low (less than 5%), likely due to the large vocabulary and caption language variation.

Captioner	Concepts	Method	B1	B2	B3	B4	MT	RG	Cr	SP	Pr	Re
LSTM	-	-	.761	.628	.507	.397	.273	.597	.465	.061		
	Att	Single	.709	.535	.394	.279	.226	.521	.253	.049	.027	.033
	Att GT		.875	.775	.652	.532	.330	.681	**.710**	.082	1.0	1.0
	SVO	Multi	.717	.588	.470	.367	.256	.577	.410	.054	.018	.016
		TN	.763	.618	.487	.378	.269	.588	.447	.061	.018	.022
	SVO GT		.792	.655	.523	.410	.285	.609	**.497**	.066	1.0	1.0
TN	-	-	.772	.631	.497	.379	.268	.589	.441	.059		
	Att	Single	.673	.506	.373	.265	.215	.507	.264	.046	.041	.061
	Att GT		.886	.790	.667	.542	.336	.683	**.781**	.086	1.0	1.0
	SVO	Multi	.730	.593	.470	.361	.256	.570	.398	.054	.019	.018
		TN	.737	.600	.477	.368	.259	.577	.416	.055	.017	.020
	SVO GT		.794	.676	.553	.439	.287	.619	**.512**	.063	1.0	1.0

Table 1.15: Grounded captioning and concept detection results on MSR-VTT. This table is the same as the proceeding table but the results are for the MSR-VTT dataset. Unlike the MSVD results no caption prediction models aid in the captioning performance, suggesting that when they are inaccurate they distract the captioning models with misleading and incorrect information. Considering the captioning performances when GT concepts are utilised two things are noticeable. First, as was the case for the MSVD results also, having more attributes compared to the structured SVO triplets is more beneficial. Second, while the LSTM RNN consistently outperforms the TN captioner when concepts are inaccurate, the opposite is true when the GT concepts are utilised (this was also true for the MSVD results).

1.3.2 Decoupling the Concepts from the Input Features

Thus far we have incorporated concepts by stacking them to the input feature encoding, allowing the caption generation models to attend across both the concept and visual features at the same time. This combination may result in a counter-productive competition between visual and concept features in the caption generation module. This is because, despite the visual features and concept features being of the same dimension, they are contextually very different and likely disjointed in their shared encoding space. Therefore we also propose decoupling the visual and concept features, combining and attending between the concept features and the previous word embeddings, while keeping the visual features separate from the language features. As the incorporation of the concept features is now quite specific to each of the RNN and TN captioning modules we describe their integration over two subsections, and then discuss the results in a third.

Recurrent Neural Network Captioner

Our original Recurrent Neural Network captioning model utilised additive attention per word generation step to attend to the visual features $\tilde{\mathbf{F}}$, generating a compressed attended visual feature:

$$\tilde{\mathbf{f}}_n = \sum_{f=1}^{F} \alpha_f \tilde{\mathbf{F}}[:, f] \tag{1.68}$$

where

$$\alpha_f = \text{softmax}(\mathbf{W}^{(\alpha)\top} \tanh(\mathbf{W}^{(\text{h})} \mathbf{h}_{n-1} + \mathbf{W}^{(\text{F})} \tilde{\mathbf{F}}[:, f])) \tag{1.69}$$

and $\mathbf{W}^{(\alpha)} \in \mathbb{R}^{d^{(\alpha)}}$, $\mathbf{W}^{(\text{h})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\text{h})}}$ and $\mathbf{W}^{(\text{F})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\text{F})}}$.

As presented in Figure 1.34, with the addition of concepts, we also utilise a second additive attention mechanism to attend across the set of our concepts stacked with the previous word:

$$\tilde{\mathbf{e}}_n = \sum_{c=1}^{C+1} \alpha_e [\mathbf{e}_{n-1}, \mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C] \tag{1.70}$$

where

$$\alpha_e = \text{softmax}(\mathbf{W}^{(\alpha)\top} \tanh(\mathbf{W}^{(\text{h})} \mathbf{h}_{n-1} + \mathbf{W}^{(\text{e})} [\mathbf{e}_{n-1}, \mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C])) \tag{1.71}$$

and $\mathbf{W}^{(\alpha)} \in \mathbb{R}^{d^{(\alpha)}}$, $\mathbf{W}^{(\text{h})} \in \mathbb{R}^{d^{(\alpha)} \times d^{(\text{h})}}$ and $\mathbf{W}^{d^{(\text{e})}} \in \mathbb{R}^{d^{(\alpha)} \times (C+1)}$.

The hidden state for the n^{th} iteration is then determined as:

$$\mathbf{h}_n = \tanh(\mathbf{U}^{(\text{h})} \mathbf{h}_{n-1} + \mathbf{W}^{(\text{x})} \check{\mathbf{x}}_n + \mathbf{b}^{(\text{h})}) \tag{1.72}$$

where the input $\check{\mathbf{x}}$:

$$\check{\mathbf{x}}_n = \begin{bmatrix} \tilde{\mathbf{e}}_n \\ \tilde{\mathbf{f}}_n \end{bmatrix} \tag{1.73}$$

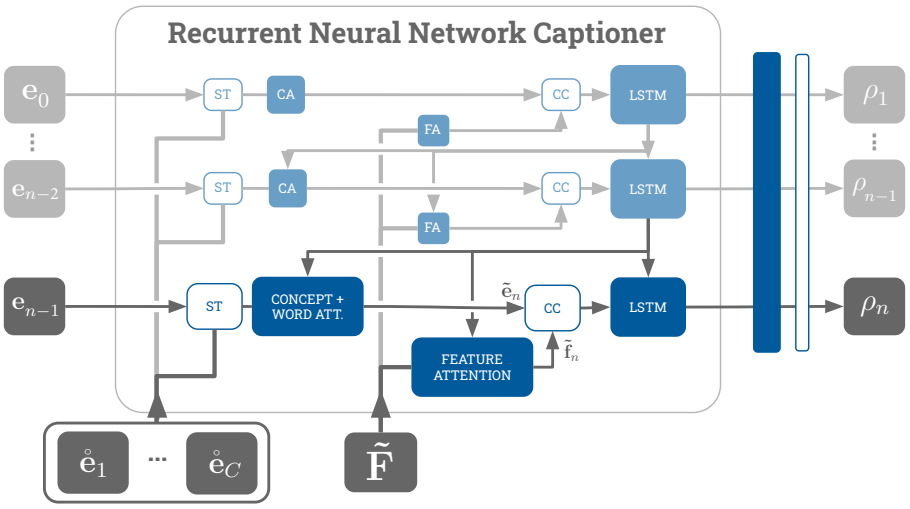


Figure 1.34: Visual representation of our decoupled RNN captioner. We now utilise two additive attention mechanisms, one for the set of visual features $\tilde{\mathbf{F}}$ as was done before, and now a second for the previous word stacked (ST) with the set of concepts $[\mathbf{e}_{n-1}, \mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C]$. These generate attended features $\tilde{\mathbf{f}}_n$ and $\tilde{\mathbf{e}}_n$ respectively, which are each concatenated (CC) and passed into an LSTM unit as our new $\check{\mathbf{x}}_n$.

Transformer Network Captioner

For the decoupling of the features for the Transformer Network captioner is relatively straightfoward, we simply add an extra decoder module ($\mathrm{DecoderBlock}(\cdot)$) prior to the visual feature decoder module, which attends across the set of concepts $[\mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C]$ in relation to all of the previously generated words $[\mathbf{e}_1, \ldots, \mathbf{e}_{n-1}]$ (see Figure 1.35):

$$
\begin{aligned}
\mathring{\mathbf{z}}_{n,1} &= \mathring{\mathrm{DecoderBlock}}_1(\mathbf{e}_n, [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], [\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}], [\mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C]) \\
\mathbf{z}_{n,1} &= \mathrm{DecoderBlock}_1(\mathring{\mathbf{z}}_{n,1}, [\mathring{\mathbf{z}}_{1,1}, \ldots, \mathring{\mathbf{z}}_{n,1}], [\mathring{\mathbf{z}}_{1,1}, \ldots, \mathring{\mathbf{z}}_{n,1}], \tilde{\mathbf{F}}) \\
&\vdots \\
\mathring{\mathbf{z}}_{n,L} &= \mathring{\mathrm{DecoderBlock}}_L(\mathbf{z}_{n,L}, [\mathbf{z}_{1,L-1}, \ldots, \mathbf{z}_{n,L-1}], [\mathbf{z}_{1,L-1}, \ldots, \mathbf{z}_{n,L-1}], [\mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C]) \\
\mathbf{z}_{n,L} &= \mathrm{DecoderBlock}_L(\mathring{\mathbf{z}}_{n,L}, [\mathring{\mathbf{z}}_{1,L-1}, \ldots, \mathring{\mathbf{z}}_{n,L-1}], [\mathring{\mathbf{z}}_{1,L-1}, \ldots, \mathring{\mathbf{z}}_{n,L-1}], \tilde{\mathbf{F}})
\end{aligned}
\tag{1.74}
$$

for a certain number of layers L (we use $L = 1$), where $\mathbf{z}_{n,L} \in \mathbb{R}^{d^{(\mathrm{h})}}$ and $\mathring{\mathbf{z}}_{n,L} \in \mathbb{R}^{d^{(\mathrm{h})}}$. As was done with the standard TN captioner, the final $\mathbf{z}_{n,L}$ is used to calculate the word probability distribution for the n^{th} word generation:

$$
\rho_n = \mathrm{softmax}(\mathbf{W}^{(\mathrm{k})} \mathbf{z}_{n,L} + \mathbf{b}^{(\mathrm{k})}) \tag{1.75}
$$

where $\mathbf{W}^{(\mathrm{k})} \in \mathbb{R}^{d^{(\mathrm{h})} \times |\mathcal{D}|}$, $\mathbf{b}^{(\mathrm{k})} \in \mathbb{R}^{|\mathcal{D}|}$, with $|\mathcal{D}|$ being the vocabulary size.

Model Comparisons

We perform experiments for the same three concept grounding architectures as introduced in Section 1.3.1, with results shown in Table 1.16 and Table 1.17 for MSVD and MSR-VTT respec-

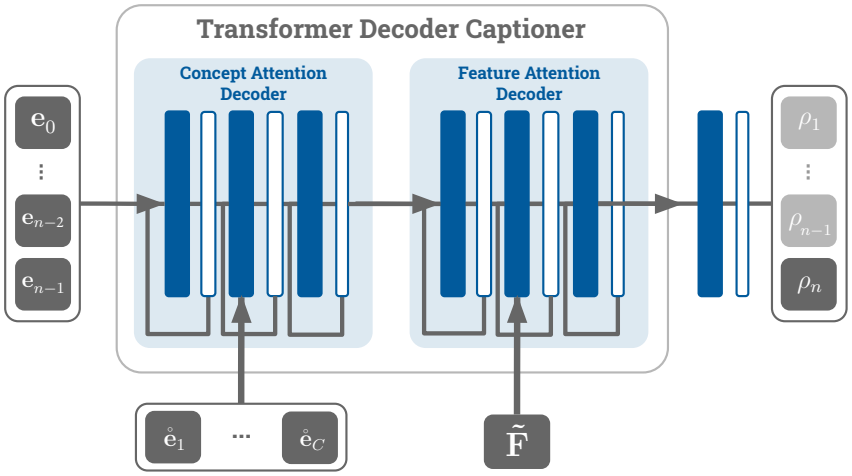


Figure 1.35: Visual representation of our decoupled TN captioner. We now add a secondary $\mathrm{Decod}\mathring{\mathrm{e}}\mathrm{rBlock}(\cdot)$ which performs attention between the set of caption word embeddings $[\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}]$ and the set of concept word embeddings $[\mathring{\mathbf{e}}_1, \ldots, \mathring{\mathbf{e}}_C]$ prior to the visual feature attention $\mathrm{DecoderBlock}(\cdot)$.

tively. Comparing the decoupled cases to the coupled cases from Section 1.3.1 we find a mix of performance decreases and increases.

Firstly, considering the attributes detection model, the detection model fails to learn concepts, resulting in detrimental effects to the caption generator, where its once reliable previous word embedding $\mathbf{e}_{n-1}$ is contaminated with five errornous and disruptive concept vectors. When testing this model with the GT concept embeddings we can see significant drops in performance compared to the stacked feature models due to the captioning model struggling to learn from the problematic attribute detector. These results occur for both MSVD and MSR-VTT evaluations, however the performance drops are somewhat lessened for MSR-VTT.

Secondly, when considering the SVO models, decoupling seems more favourable, particularly for the MSVD dataset, and particularly when the GT concept embeddings are used. It isn't completely clear why the decoupling aids the SVO models on MSVD but not MSR-VTT. Nonetheless, to summarise, these results suggest that both techniques of considering concepts are useful, particularly when the concepts are accurately detected.

Capt.	Conc.	Method	B1	B2	B3	B4	MT	RG	Cr	CrDf	SP	Pr	Re
LSTM	-	-	.722	.576	.473	.380	.300	.646	.604		.045		
	Att	Single	.471	.183	.096	.048	.135	.448	.075	*-.215*	.007	0.0	0.0
	Att GT		.754	.622	.523	.411	.293	.665	.430	***-.965***	.033	1.0	1.0
	SVO	Multi	.738	.602	.493	.382	.291	.643	.536	*+.012*	.039	.030	.033
		TN	.751	.621	.522	.421	.306	.662	.661	*+.020*	.042	.033	.044
	SVO GT		.810	.708	.616	.515	.363	.726	.925	***+.265***	.055	1.0	1.0
TN	-	-	.742	.618	.521	.415	.302	.654	.573		.042		
	Att	Single	.648	.426	.313	.218	.202	.564	.240	*-.333*	.024	0.0	0.0
	Att GT		.652	.548	.436	.294	.295	.640	.535	***-1.047***	.041	1.0	1.0
	SVO	Multi	.742	.611	.510	.404	.292	.651	.544	*+.030*	.040	.031	.035
		TN	.748	.615	.517	.417	.302	.649	**.598**	*+.032*	.041	.032	.041
	SVO GT		.822	.730	.642	.541	.362	.736	.877	***+.193***	.053	1.0	1.0

Table 1.16: Grounded captioning and concept detection with decoupling results on MSVD. This table is the same as the proceeding tables but the results are for the decoupling feature injection method for the MSVD dataset. We add an additional column (**CrDf**) for the **CIDEr score differences** between the scores here and the corresponding scores for the feature stacking models from Table 1.14 on page 89. If we first consider these scores we can see that the decoupled injection has positive effects on the SVO models but significantly negative effects on the attributes models. Furthermore, for the staked feature approach, the coupling makes the TN captioner with TN concept detector competitive with the model that doesn't consider concepts.

Capt.	Conc.	Method	B1	B2	B3	B4	MT	RG	Cr	CrDf	SP	Pr	Re
LSTM	-	-	.761	.628	.507	.397	.273	.597	.465		.061		
	Att	Single	.693	.529	.389	.271	.228	.530	.286	*+.033*	.051	.034	.040
	Att GT		.858	.743	.614	.490	.317	.658	.628	***-.082***	.076	1.0	1.0
	SVO	Multi	.746	.614	.491	.379	.260	.586	.422	*+.012*	.056	.020	.019
		TN	.759	.618	.491	.380	.260	.587	.431	*-.016*	.058	.019	.022
	SVO GT		.804	.669	.534	.416	.285	.615	.495	***-.002***	.065	1.0	1.0
TN	-	-	.772	.631	.497	.379	.268	.589	.441		.059		
	Att	Single	.685	.506	.365	.256	.215	.505	.272	*+.013*	.048	.046	.069
	Att GT		.820	.687	.542	.411	.294	.619	.593	***-.188***	.072	1.0	1.0
	SVO	Multi	.741	.598	.473	.362	.257	.573	.405	*+.007*	.054	.019	.017
		TN	.757	.617	.490	.380	.262	.585	.433	*+.017*	.056	.016	.020
	SVO GT		.785	.659	.534	.421	.283	.613	.486	***-.026***	.061	1.0	1.0

Table 1.17: Grounded captioning and concept detection with decoupling results on MSR-VTT. This table is the same as the proceeding table but the results are for the decoupling feature injection method for the MSR-VTT dataset. The **CIDEr score differences** (**CrDf**) are between the scores here and the corresponding scores for the feature stacking models from Table 1.15 on page 89. If we first consider these scores we can see that the decoupled injection has less of an effect here than it does on the MSVD results, with mixed minor increases and decreases. The effects on the GT attributes models are still relatively negative though, as was the case for the MSVD results. Furthermore, compared to the MSVD results which see significant performance variations for the use of GT concepts between the stacking and decoupled methods (*-1.0*s and *+.2*s), here the performance differences are much smaller (*-.15*s and *-.01*s).

1.4 Summary

In this chapter we have investigated the problem of video captioning as a form of generalised video understanding. We provided a comprehensive overview of the three stage video captioning process — visual interpretation (Section 1.1.1), interpretation filtering (Section 1.1.2), and caption generation (Section 1.1.3). We discussed all of the major works related to these three stages, with an emphasis on the *interpretation filtering* stage as the heart of the captioning process, where most works focus their attention (*literally*). We also gave an overview of the datasets (Section 1.1.4) that are crucial to the learning of these video captioning frameworks, as well as the standard evaluation protocols (Section 1.1.5) used to determine the accuracy of generate captions.

We performed a number of experiments investigating the effects of different model input features Section 1.2.1, model architectures (Section 1.2.2 and Section 1.2.3), and model sizes (Section 1.2.4). We found that spatial region features are crucial for accurate captioning (about twice as good as standalone image features as shown in Table 1.7), highlighting the importance of having specific visual representations for objects which relate to words in the captions. We also found that an LSTM RNN with additive attention over the input features is more effective at caption generation than a multi-layered TN, with relatively similar training and inference speeds (CIDEr scores of .665 and .602 respectively shown in Table 1.8 in Section 1.2.2). Furthermore we discovered that both models quickly over-fit to the training data (Figure 1.26 and Figure 1.27), with the over-fitted training models performing significantly better on the over-fitted training data in comparison to human annotators (approximately two times better shown in Table 1.12). A more in-depth analysis of the computer generated versus the human captions revealed that the model captions are much shorter, more succinct, and less descriptive. Furthermore, they tended to form as if they were an *average* of the majority of ground truth captions, leading them to attain relatively high scores with the current metrics in comparison to randomly sampling human captions. Following on from these insights, we discussed how it's non-trivial and ambiguous to determine human performance for the captioning problem due to the interprebility of the captioning process (see Section 1.2.5 for the discussion). Lastly, we took a more thorough look at the specific noun and verb accuracies for our captioning model, considering that most concepts appear rarely in the dataset (Section 1.1.4 and Figure 1.17), we found the identification and usage of the rarer concepts is more inaccurate (Figure 1.29 and Figure 1.30).

With our findings of the importance of image region features for object identification, the metric benefits of shorter more concise captions, and the challenges of noun and verb diversity, it's suggestive that the key to effective captioning is correct identification and usage of a broad range of visual diverse nouns and verbs. With this in mind we investigated several grounding processes to strengthen the connection between the diverse nouns and verbs with the visual input features. We extracted the top concepts and structured SVO concept triplets, using them to teach three different filtering processes to determine the key concepts in a video (Section 1.3.1). Captioning using the GT concepts showed that using the top five concepts is more beneficial

than using SVO triplets (1.395 and .660 respectively as shown in Table 1.14), as well as that there is always improvement in captioning performance when concepts are correctly identified (see GT results in Table 1.14 and Table 1.15). We discussed the abilities of the RNN captioner to better recover from poor concept features, but for the TN to be more flexible and accurate when the input concepts and visual features are less noisy and richer (Section 1.3.1). Lastly, we investigated two ways of injecting the concept features with the captioning module, finding mixed performance (Table 1.16 and Table 1.17 in Section 1.3.2).

We believe that improvements for video captioning models should come from focusing on reliably generating the diverse noun and verb sets found in the datasets. Recalling rare concepts at caption generation time is also at the crux of the challenge, suggesting more work related to models which can utilise a feature library to extract discriminative information related to rare concepts is likely a promising research direction. Furthermore, improving the input features could also lead to performance gains, particularly for the Transformer Network models which are more reliant on them, and reliant on larger volumes of data in general. Such improvements could come from more information with region features, including classification labels, or using features from spatio-temporal action detection models. Lastly, although longer-term temporal information doesn't seem to be overly influential for the short clips in the MSVD and MSR-VTT datasets (relative to spatial region and short-term low-level motion features), for future problems with longer videos more focus would need to be put on the temporal sampling of visual features with longer-term temporal modeling.

2 Concept Detection & Localisation

This chapter explores approaches for performing concept detection and spatio-temporal localisation in videos. Specifically, when provided with a video clip the goal is to **detect and identify a set of concepts**, localising them with a bounding box and classifying them with a label (Figure 2.1).

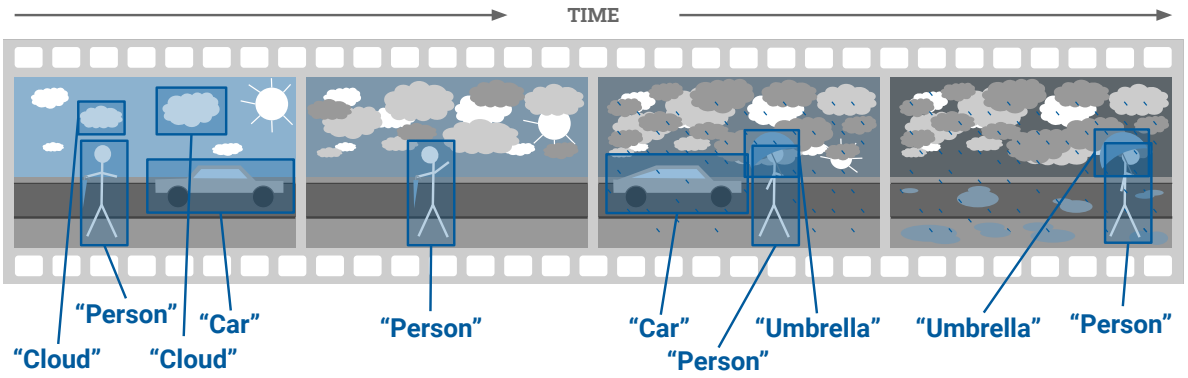


Figure 2.1: An example of the video object detection problem. Provided a video portraying a scene, the goal is to generate a set of bounding box coordinates and a classification label for each visible object.

Currently, the research community handles the detection and classification of objects and actions separately, each being considered in two disjoint problem domains. **Object detection** [Liu et al., 2020] is a problem that has been well studied in the still image domain, and has recently been extended into the video domain. In images, detection involved finding objects spatially, however in videos objects also need to be found temporally. In **video object detection** works [Jiao et al., 2021], such temporal localisation happens automatically as every frame is processed, however temporal association of detections isn't addressed. Temporal association, is the process of associating detections across time for individual object instances. Association is considered in another disjoint problem domain called **tracking** [Marvasti-Zadeh et al., 2021], which has been one of the fundamental research areas related to videos for many decades. **Action recognition** [Pareek and Thakkar, 2021] has also been one of the fundamental video domain problems for a long time, and has for the most part been considered a classification problem. In comparison to detection, classification involves labelling a single clip with no spatial or temporal localisation. More recent studies have evolved to focus on **temporal action localisation and event detection**, which localise actions in the temporal domain, however still not in the spatial domain.

2.1 Framewise Object Detection

Videos are naturally just a set of sequential still images or '*frames*', meaning that still image object detection models can be applied to videos on a per-frame basis. We begin by investigating the main model architectures that are utilised for image-based object detection, as well as providing an overview of image-based object datasets and explanations of the evaluation protocols used for benchmarking detection models. We then perform a number of initial experiments with image-based object detectors on both the image and video datasets, investigating their performance in terms of both speed and accuracy.

2.1.1 Object Detection in Images

There are two categories of modern neural network based object detection architectures:

- **Two-stage detectors**, which firstly propose image regions that are likely to contain objects based on exhaustively positioned anchor boxes, before classifying and refining the locations of these region proposals;
- **One-stage detectors**, which skip the proposal step, performing classification and box refinement directly on the anchor boxes.

Anchor boxes are multi-scale and multi-ratio boxes which are exhaustively positioned across the input image, such that for each pixel (i, j) there are A anchors. The particular anchor scales and ratios are determined by measuring the sizes and ratios of the ground truth objects in the training split of the dataset, so as to best accommodate the majority of objects. Anchors are used as an initial step to place boxes across an input image, and are refined by the detectors in different ways.

Two-Stage Detectors

Region Convolutional Neural Networks & Region Proposal Networks

The first modern object detection model was a two-stage architecture called **Region Convolutional Neural Network (R-CNN)** [Girshick et al., 2014]. In the R-CNN model, selective search [Uijlings et al., 2013] is used to generate 2000 region of interest (ROI) proposals that are considered likely to contain objects. These region proposals are warped into 227×227 image regions that are passed into a pre-trained classification Convolutional Neural Network (CNN). This CNN is used as a feature extractor with the last *dense* layer output of dimension 4096 used for classification by individual Support Vector Machine (SVM) binary classification models. Furthermore, a small regression network is trained to learn four offset values for box adjustments. There are two key problems with the R-CNN model — firstly, selective search is a fixed algorithm that doesn't permit any learning, and secondly, training and inference is relatively slow since each of the 2000 proposals need to be individually passed through the CNN.

The latter aforementioned issue was resolved in **Fast R-CNN** [Girshick, 2015] which instead passes the entire image through the feature extraction CNN. The proposal boxes, as generated by the selective search method, are then projected onto the feature output of the CNN, being cropped and warped by a ROI pooling layer. Each ROI feature is then classified and regressed with a small neural network, doing away with the individual class SVMs. These modifications not only improved accuracy but also resulted in approximately a $10\times$ speed-up.

Speeding up the process even a further $10\times$, while also improving accuracy was **Faster R-CNN** [Ren et al., 2015]. The selective search method is replaced with a Region Proposal Network (RPN) which takes the spatial feature map from the CNN to regress adjustments and provide an objectness score to a set of *multi-scale anchor boxes*. These anchor boxes are exhaustively placed across the image, with multiple scales and aspect ratios used in an attempt to handle variously sized and shaped objects. The ROIs generated by the RPN are then used to extract regions from the CNNs spatial feature to be warped, classified and adjusted in the same way as Fast R-CNN. Figure 2.2 presents an overview of the Faster R-CNN architecture.

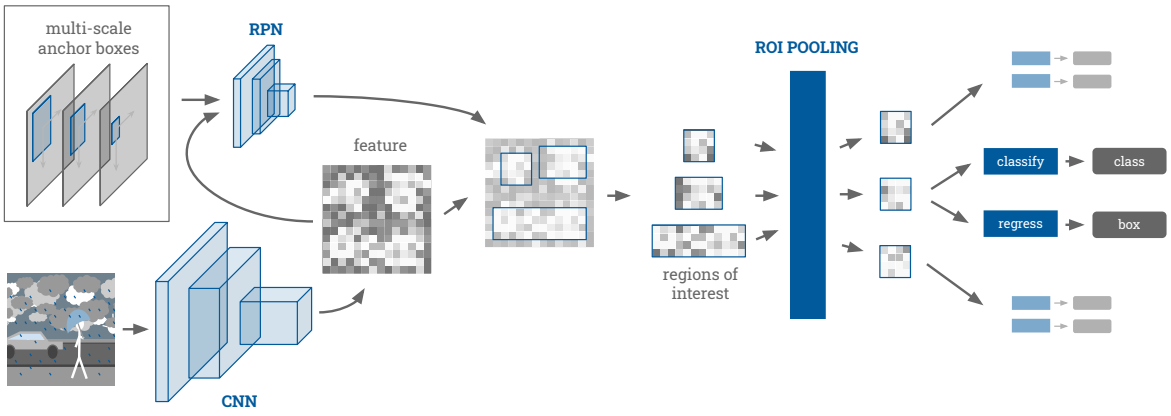


Figure 2.2: An overview of the Faster R-CNN object detection pipeline. Initially an image is passed through a Convolutional Neural Network (CNN) to generate a spatially reduced feature encoding. Using this feature encoding as input, as well as the coordinates for exhaustively placed anchor boxes, a Region Proposal Network (RPN) is used to adjust the anchor boxes and apply an objectness score to each box. The highest scoring refined boxes are then projected onto the feature encoding, with the various sub-regions relating to the boxes being extracted from the encoding, resulting in a set of region of interest (ROI) boxes. As these ROIs can be of various sizes and shapes they are passed through an ROI pooling layer which squashes them into the same spatial dimension. They are then flattened into vectors and passed through individual classification and box regression networks to get final class labels and box coordinates.

Region Fully Convolutional Neural Networks

While Fast R-CNN removed the per region CNN feature extraction of R-CNN, Faster R-CNN still relied on per ROI classification and regression with a small densely connected network after the ROI pooling layer, which was slow to train and predict with. Addressing this problem, [Dai et al., 2016] introduced the **Region Fully Convolutional Network (R-FCN)** [Dai et al., 2016] which utilises a fully convolutional architecture, sharing computation across all ROIs. This is achieved by performing a convolutional operation to produce position-sensitive score maps of $A^2(C+1)$ channels where C is the number of classes and $A = 3$ representing different spatial region anchors. A vote is taken across the A^2 regions to determine the class for an ROI.

[Singh et al., 2018] extend the R-FCN architecture by separating the detection and classification pathways, learning shared filters for agnostic class localisation, and then classifiers for 3000 object classes. The position-sensitive filters are re-purposed to produce *super-class* scores — a super-class being visually similar classes.

Feature Pyramid Networks

Despite utilising multiple scaled anchor boxes, the aforementioned detectors still struggle to handle objects of varying sizes, especially smaller objects. This issue is a result of only taking, and ROI pooling over, a single sized feature from late in the CNN. As data moves further through CNN networks the features get spatially smaller, for example the final feature of Faster R-CNN is only 7×7 pixels spatially. Considering this, the feature that is extracted and used is of such low resolution relative to the input image that is has lost much of the finer detailed information necessary to represent spatially small image regions. To help alleviate this issue, and inspired by pyramids of multi-scaled input images, [Lin et al., 2017] introduced the **Feature Pyramid Network (FPN)**. FPNs generate features of varying scales by combining upscaled smaller, more processed, feature maps with larger, more spatially detailed, feature maps. [Zeiler and Fergus, 2014] showed that as you go deeper into a CNN model, the layers capture not only a greater spatial area of the input image, but also more complex patterns and concepts. Considering this, having both the forward then backward generation of feature maps allows the more detailed larger maps to gain more complex contextual insights from the more processed feature maps. The FPN has no effect on the Faster R-CNN framework design, it just replaces the feature extraction CNN, with ROIs extracted from their corresponding sized feature.

Looking to improve both the capability and efficiency of the FPN architecture, [Ghiasi et al., 2019] use Neural Architecture Search (NAS) to find a new FPN (**NAS-FPN**) architecture structure which includes both top-down and bottom-up connections. Extending NAS-FPN, with a focus on efficiency, [Tan et al., 2020] introduce **EfficientDet** which uses a Bi-directional FPN (BiFPN) for more efficient cross-scale feature fusion. They also scale the resolutions, depths and widths for the backbone, feature network and prediction heads jointly using a compound coefficient.

One-Stage Detectors

SSD

With a focus on speed and efficiency [Liu et al., 2016] introduced the **Single-Shot Multi-box Detector (SSD)**, which is $3\times$ faster than Faster R-CNN. The efficiency benefits are achieved by removing the RPN and performing direct classification and regression of anchor boxes. Given a feature output of spatial size $S \times S$, there are $A = 4$ anchors for each pixel in the feature. Each anchor at each pixel gets predicted with four box coordinate adjustments and C class probabilities, resulting in $S^2A(4 + C)$ outputs per feature map. Similar to how FPNs are used in the two stage models, for SSD outputs are calculated at different scaled feature maps from different depths of the feature extraction CNN.

YOLO, YOLO9k & YOLOv3

In a very similar vein to SSD, **You Only Look Once (YOLO)**, introduced by [Redmon et al., 2016], also directly regresses and classifies anchor boxes, removing the need for the RPN. Differently to SSD, YOLO only takes a feature map from a single scale, and only calculates class probabilities per feature pixel rather than per anchor box $A = 2$, resulting in $S^2(4A + C)$ outputs. Furthermore, YOLO uses a dense layer for this calculation rather than fully-convolutional layers, and also a specific fast feature extraction CNN called **DarkNet**, while SSD uses a standard CNN architecture.

Extending upon YOLO, **YOLO9k** [Redmon and Farhadi, 2017] aligns closer with SSD, calculating output boxes based per anchor rather than feature pixel, as well as calculating an objectness score, resulting in $S^2A(5 + C)$ outputs. Furthermore, positional offsets and size ratios are calculated rather than direct box positions and sizes as learning is found to be more stable. To better handle multi-sized objects YOLO9k uses a FPN to add a larger feature map, while also performing multi-resolution training — changing the input image size every ten batches. For efficiency improvements, the DarkNet is reduced from 26 layers to 19, maintaining a similar structure. The most important difference with YOLO9k however is its ability (or *attempt*) to classify 9000 classes, adopting a specific training strategy to jointly learn from both detection and classification data. When training, if a detection sample is seen, backpropogation is normal, whereas if a classification sample is seen, loss is only backpropogated through the classification parts of the network. This training scheme allows their model to learn from the **ImageNet** image classification dataset [Russakovsky et al., 2015]. To handle the cross-over between the very fine-grained ImageNet dataset and the more generalised detection datasets, a hierarchical classification tree is formulated based on the **WordNet** database. When calculating the probability of a class all of the probabilities from that class to the root of the classification tree are multiplied together.

The most current[1] version of YOLO, **YOLOv3** [Redmon and Farhadi, 2018] (Figure 2.3) makes further improvements to YOLO and YOLO9k, however focuses on only object detection data. The two key improvements come from the use of a new 53 layer DarkNet feature CNN, which has a more modern design of repeating *residual* blocks of 1×1 and 3×3 convolutions, as well as the use of features from three scales, rather than one or two.

Alternatives to Using Anchors

One of the biggest downsides of modern detectors, both one- and two-stage alike, is their reliance on **anchors**. Anchors require pre-computation on the training dataset to determine what scales and aspect ratios should be used to best suit the particular types of objects in the dataset. Furthermore, as our classifications and box refinements are all in respect to the anchors, it's necessary to perform these operations A times, A being the number of anchor boxes.

[1] was current at time of writing. Post-review the latest version is YOLOv5 – github.com/ultralytics/yolov5

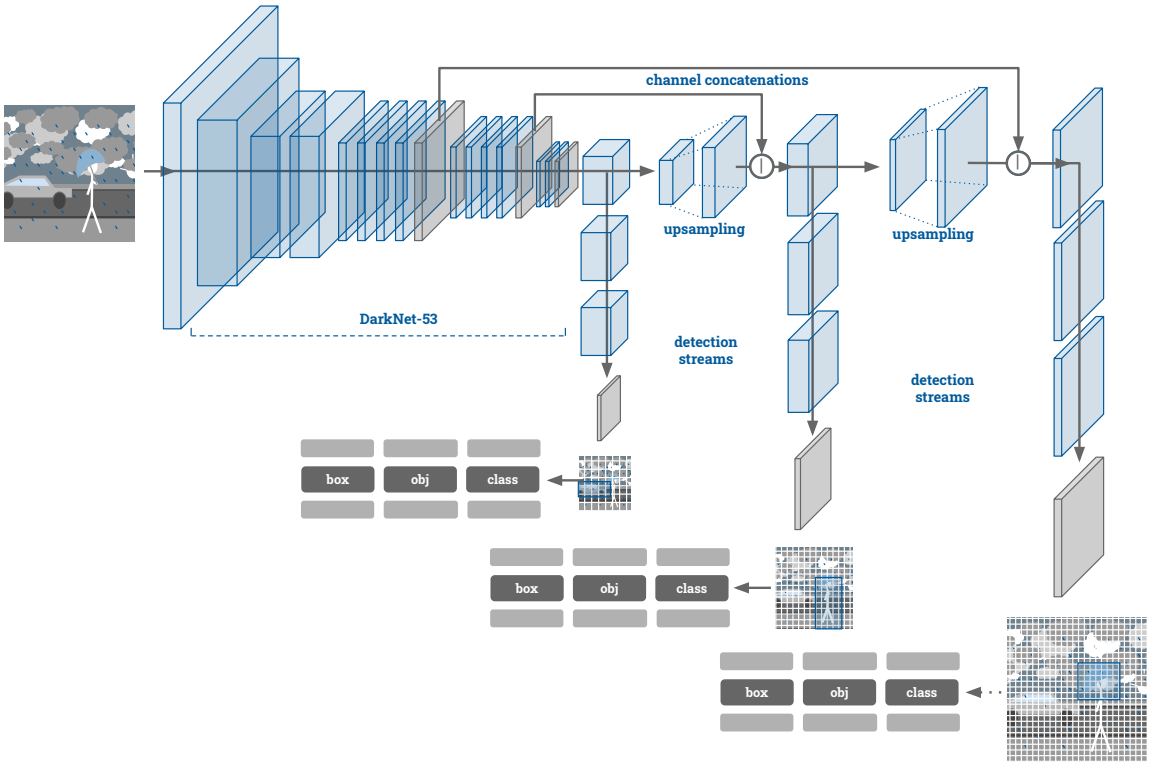


Figure 2.3: An overview of the YOLOv3 object detection pipeline. Initially an image is passed through the DarkNet-53 Convolutional Neural Network (CNN) to generate three spatially reduced feature maps of varying scales. Starting from the smallest scale feature map, the network splits on two paths. Following on horizontally is the upsampling and concatenation operations, which work like the FPN, passing through the contextually rich upsampled features and concatenating them with the spatially detailed features. Following on vertically are the three detection streams which perform the box refinement, object scoring, and class category classification for each of the three scales. For clarification on the final features, they don't actually look like the initial image but they do retain spatial information representing the particular objects in the original image. Furthermore the pixeling shows how they are of different scales, with larger objects such as the `car` being detected in the small feature map, and smaller objects like the `umbrella` in the large feature map.

Looking to not be reliant on anchor boxes, [Law and Deng, 2018] pose the object detection problem as one of determining two keypoints, the top-left and bottom-right of a box surrounding an object. Their **CornerNet** model produces a heatmap for each corner for each class, and an embedding layer to associate corner pairs. CornerNet is extended into **CenterNet** [Duan et al., 2019] which addresses CornerNet's inability to consider global information when generating keypoint predictions. They add another central keypoint which reinforces the corner points. Another alternative to pre-computing anchor boxes is to regress them directly, which is done by [Tian et al., 2019] in their **fully convolutional one-stage detector (FCOS)**. To recover ambiguities related to pixels being part of two boxes, as well as issues related to large strides, they use a deeper FPN with shared classification and regression heads across the different scales. Lastly they include a centerness head to prevent low quality boxes being predicted far from the centre of an object.

Non-Maximal Suppression

Due to the high number and density of anchor boxes, objects are generally covered by many anchors, resulting in the object detection model producing many overlapping boxes for a single object instance. To handle the repetition of boxes for a single object instance, a method called **Non-Maximal Suppression (NMS)** is used. NMS works by taking the highest confidence box for a class and removing any boxes of the same class that overlap with a certain area threshold, normally by at least 50%. Similar to how selective search was an explicit algorithm and untrainable, the NMS process is the same. Addressing this issue is [Hu et al., 2018] who introduce **Relation Networks**, which utilise multi-head attention with appearance and geometry features of proposals to perform instance recognition and duplicate removal. This resulted in Relation Networks being the first object detection framework to be fully end-to-end trainable.

2.1.2 Datasets

All of the model architectures mentioned above require data for training and evaluation. Over the years there have been a number of datasets released that are designed specifically for the object detection problem. Table 2.1 presents a summary of the **image and video object detection datasets**.

One of the first large-scale object detection datasets to arise was the **Pascal VOC** [Everingham et al., 2010] dataset. Pascal VOC consists of over 20k images with over 50k labelled object instances covering 20 object categories.

Currently, the most widely used object detection dataset is **MS-COCO** [Lin et al., 2014] which extended the number of categories to 80, ranging from animals to vehicles to household items. The set also consists of over six times the number of images (120k), and over sixteen times the number labelled object instances (800k) in comparison to Pascal VOC.

Part of the ImageNet group of datasets, **ImageNet-DET** [Russakovsky et al., 2015] consists of greater class diversity, with 200 object categories. ImageNet-DET also consists of close to

Dataset	# Classes	# Images (Clips)	# Boxes (Object Instances)	B per I (I per C)
Image Datasets				
Pascal VOC	20	21503	62199	2.89
MS-COCO	80	122266	886729	7.25
ImageNet-DET	200	476688	534308	1.12
Flickr 30k Entities	-	31783	193020	6.07
Open Images	600	1910098	15851536	8.30
Visual Genome	-	108077	5403850	50
Video Datasets				
ImageNet-VID	30	1693473 (5314)	2655058 (11575)	1.57 (2.18)

Table 2.1: A summary of image and video object detection datasets. We present the number of classes, images, and boxes for each dataset. We also present ratio of boxes per image (B per I), which along with the number of classes, gives an indication of each datasets diversity and complexity. For the video datasets we also show the number of clips, the number of individual object instances, and the instances per clip (I per C). Links to each of the datasets are listed in Appendix D.

four times the number of images of that in MS-COCO (470k), however it only has approximately two-thirds of the number of object instances (530k). This means there are many less objects in a single image, with most samples in ImageNet-DET consisting of one object that is large, centralised and unoccluded in the image. This results in ImageNet-DET being useful for learning greater class diversity, but less useful for learning spatial localisations, especially in crowded scenes. This has led to it not being utilised as much as the aforementioned sets for object detection benchmarks.

Flickr 30k was introduced as an image captioning dataset, however since its introduction and widespread use for captioning benchmarks, interest arose to extend it for other problem domains. **Flickr 30k Entities** [Plummer et al., 2015] extends Flickr 30k by introducing 190k annotated bounding boxes for the 30k images in Flickr 30k. Unlike the aforementioned object detection sets where specific objects are localised, in Flickr 30k Entities boxes are labelled with snippets of the captions from the original Flickr 30k set. Hence, such a dataset lends itself more towards the problem of visually grounding captions, however nouns could be extracted from these textual snippets to create object-related box annotations.

Currently available in its sixth version, the **Open Images** [Kuznetsova et al., 2020] dataset is currently the largest image based dataset with object localisations. It consists of close to 2 million images with over 15 million box annotations for 600 categories. Similarly to MS-COCO, the images are very diverse, and regularly contain complex scenes with multiple objects. The dataset also has 2.8 million instance segmentations for 350 classes, 59.9 million image-level labels for 20k classes and 3.3 million relationship annotation triplets covering relations (*woman playing guitar*), object properties (*table is wooden*) and human actions (*woman is jumping*). Relationship annotations are becoming more useful for modern machine learning problems, where

contextual and inter-object interactions can give insights towards more accurate predictions.

Centered around the idea of object relationships, **Visual Genome** [Krishna et al., 2017b] is designed as a structured graph-style knowledge base. It consists of over 100k images with 3.8 million object instances, 2.8 million object properties or attributes, and 2.3 million relationships. All of the object instance and attribute labels are mapped to their WordNet [Miller, 1995] synset. WordNet is a lexical database of semantic relations between words, with a synset being a group of synonyms that have the same meaning. Furthermore, Visual Genome consists of 5.4 million region descriptions and 1.7 million visually based question answers, making it useful in a number of vision and language problem domains.

For video, the **ImageNet-VID** dataset [Russakovsky et al., 2015] is currently the only large-scale generalised video object detection dataset. The dataset consists of over 5k clips made up of over 1.5 million frames, with more than 2.5 million boxes for over 10k individual object instances across 30 categories. On average only one or two objects appear in each frame at a time, and 2-3 instances across an entire clip.

2.1.3 Evaluation Protocol

For object detection problems the metric used for measuring a model's accuracy is **mean Average Precision (mAP)**. The mAP is the mean of the individual class average precisions:

$$\text{mAP} = \frac{1}{C}\sum_{c} \text{AP}_c \tag{2.1}$$

where **average precision (AP)** for each class is calculated by taking the average of the **interpolated precision (IP)** values across a range of equidistant recall levels $\bar{r} \in \{0, 0.1, ..., 1\}$:

$$\text{AP}_c = \frac{1}{11}\sum_{\bar{r}\in\{0,0.1,...,1\}} \text{IP}_c(\bar{r}) \tag{2.2}$$

To calculate the IPs for each $\bar{r}$, boxes are ordered based on their confidence level and then **precision** and **recall** values are calculated per box (p_c, r_c). The precision and recall values are based on the boxes with higher confidences — for example, the third most confident box calculates its precision and recall values based on the top three most confident boxes. The interpolated precision at the recall value $\bar{r}$ is the maximum precision p_c for all of the precisions associated with all of the boxes with recalls $r_c \geq \bar{r}$:

$$\text{IP}_c(\bar{r}) = \max_{r_c \geq \bar{r}} \; p_c^{(r_c)} \tag{2.3}$$

$$p_c = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.4}$$

$$r_c = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.5}$$

where TP, FP, and FN are true positives, false positives and false negatives respectively. To calculate these values for detection it's necessary to consider the localisations of boxes. For

this purpose an **intersection over union (IoU)** score is calculated between each pair of predicted and ground truth (GT) boxes:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{2.6}$$

The TP, FP and FN values are then based on a IoU threshold (usually set to 0.5) resulting in:

- **True Positives (TP)** are when a model correctly predicts a box with an IoU greater than the threshold, with the correct class label;
- **False Positives (FP)** are when a model predicts a box with the correct label, but its IoU is less than the threshold, or it is a duplicate prediction where another predicted box has a higher IoU with the underlying GT box;
- **True Negatives (TN)** are not considered as we expect every image being tested to contain an instance of the class;
- **False Negatives (FN)** are when a model correctly predicts a box with an IoU greater than the threshold, but with the incorrect class label. These are considered FP for the incorrectly labelled prediction class.

Along with the MS-COCO dataset, a set of six new mAP metrics were introduced. Furthermore, instead of measuring over 11 recall levels they instead use a much finer 101 ranges:

- **AP**[2]/**AP$_{50\text{-}95}$** takes the average APs over a range of IoUs from 0.5 to 0.95 with 0.05 increments;
- **AP$_{50}$** is the mAP with IoU of 0.5;
- **AP$_{75}$** is the mAP with IoU of 0.75;
- **AP$_{S}$** is the mAP for small objects (area $< 32^2$ pixels);
- **AP$_{M}$** is the mAP for medium objects ($32^2 \leq$ area $\leq 96^2$ pixels);
- **AP$_{L}$** is the mAP for large objects (96^2 pixels $<$ area).

Considering a key challenge in video based object detection is handling motion blur and other artifacts associated with fast moving objects, [Zhu et al., 2017b] introduced a video based mAP metric which aggregates results based on an object's motion. They also use different scales for the small, medium and large evaluations to suit the larger size of the objects found in the ImageNet-VID video object detection dataset:

- **AP$_{SL}$** is the mAP for slow objects (0.9 $<$ motion IoU);
- **AP$_{MO}$** is the mAP for moderate objects (0.7 $\leq$ motion IoU $\leq$ 0.9);
- **AP$_{FA}$** is the mAP for fast objects (motion IoU $<$ 7.0).
- **AP$_{S}$** is the mAP for small objects (area $< 50^2$ pixels);
- **AP$_{M}$** is the mAP for medium objects ($50^2 \leq$ area $\leq 150^2$ pixels);

[2] following standard practice, AP is used as shorthand for mAP in table headers

- **AP_L** is the mAP for large objects (150^2 pixels $<$ area).

The **motion IoU** value is related to each GT instance, and is calculated as the average IoU of an instance with itself across ± 10 frames (excluding the current frame where IoU would be 1).

2.1.4 Performance Considerations

In comparison to image data, one of the key challenges with video data is its sheer size. Generally videos consist of between 24 and 30 frames per second, leading to challenges with its transmission, storage and processing. In fact, processing an entire full length video with even the latest modern neural network models is still significantly restricted by hardware limitations. As a result, frameworks either slice videos into shorter clips, processing each individually, or take only a small number of frames spaced evenly apart across the entire video. Furthermore, different models require different hardware resources, so it is particularly important when working with video data to consider a models processing speed and memory requirements. We therefore carry out a performance and efficiency evaluation with three of the key object detection pipelines — **Faster R-CNN**, **SSD** and **YOLO**. There are numerous different implementations of these pipelines using various deep learning libraries, however we utilise the **MXNet** and **GluonCV** implementations [Zhang et al., 2019b] for all three models as they promoted the most consistency between the implementations.

Table 2.2 presents details about the three models and their individual performance and efficiencies. We run evaluations using batch sizes of 1 and 4 as we are limited by memory resources. We note that such batch sizes are relatively small, potentially having negative impacts on performance (normally bigger is better), however as best we can we keep them similar across all experiments. We find that while the SSD network utilises the least memory with its smaller model size, it is not nearly as accurate as Faster R-CNN or YOLOv3. The YOLOv3 network, being single shot, is much quicker than the Faster R-CNN network while only having a minor loss in accuracy. **For these reasons we utilise the YOLOv3 model as our basis for video object detection, referring to it simply as YOLO from hereon.**

2.1.5 A Closer Look at YOLO

Structure

The **YOLO** model, specifically version 3, as described in [Redmon and Farhadi, 2018] and shown in detail in Figure 2.4, consists of two CNNs. The first is a feature extraction network called **Darknet-53**, which is an improved extension of the Darknet-19 network used in YOLO9k [Redmon and Farhadi, 2017]. The second is a **detection network** which takes feature maps of three different scales from the DarkNet-53 and processes them through individual branches to generate mutli-scaled box predictions.

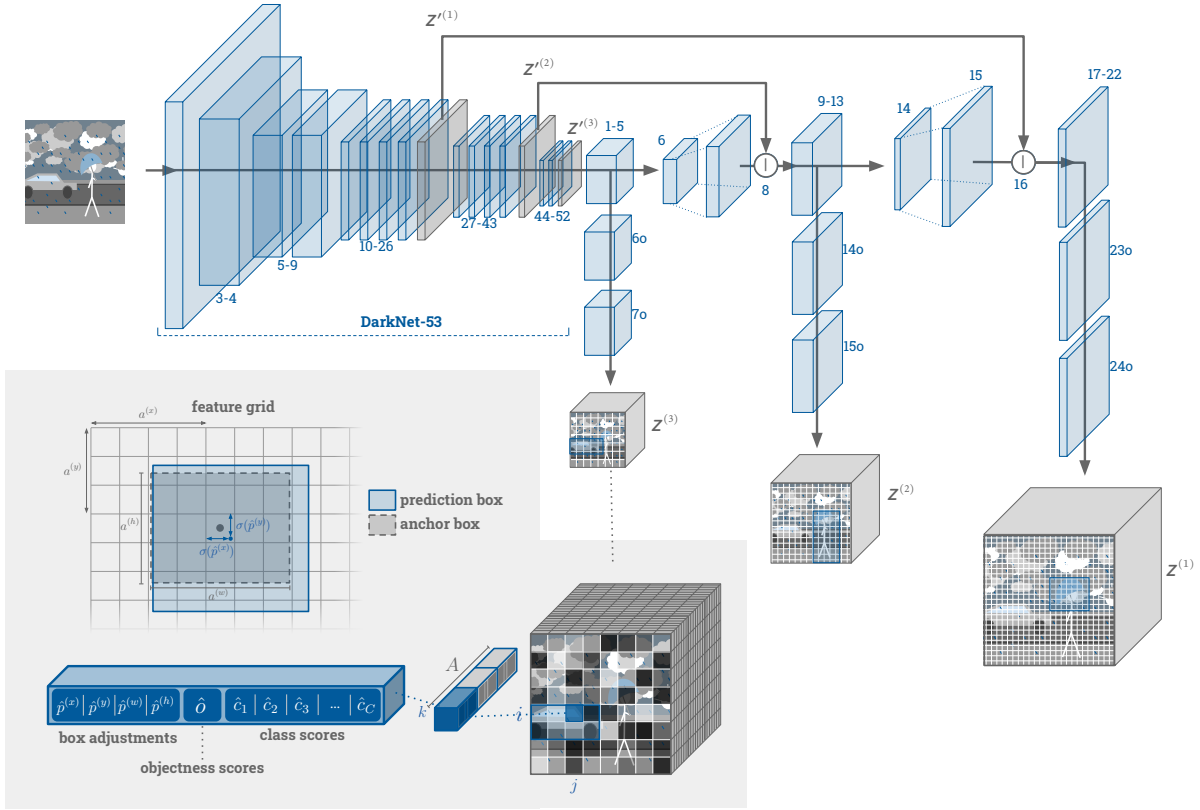


Figure 2.4: A more detailed look at the YOLO architecture. Starting from the top-left, an image is passed through the DarkNet-53 CNN generating three varying scaled features ($\boldsymbol{Z'}^{(1)}$: 52 × 52, $\boldsymbol{Z'}^{(2)}$: 26 × 26, and $\boldsymbol{Z'}^{(3)}$: 13 × 13). The features are then used in a secondary detection CNN. Continuing horizontally, the detection CNN consists of more convolutions followed by upsampling and concatenation operations, where $\boldsymbol{Z'}^{(3)}$ gets upsampled and concatenated channel-wise with $\boldsymbol{Z'}^{(2)}$, and $\boldsymbol{Z'}^{(2)}$ with $\boldsymbol{Z'}^{(1)}$. Performing these operations allows the spatially larger feature maps to gain rich contextual information from the smaller and deeper spatial features. Each of the concatenated features then pass through their respective detection streams (the vertical branches) which consist of more convolutional operations to generate three output feature volumes $\boldsymbol{Z}^{(1)}$, $\boldsymbol{Z}^{(2)}$, and $\boldsymbol{Z}^{(3)}$. Each of the output volumes has the same widths and heights of $\boldsymbol{Z'}^{(1)}$, $\boldsymbol{Z'}^{(2)}$, and $\boldsymbol{Z'}^{(3)}$ but have $A \times (4 + 1 + C)$ channels, where A is the number of anchors and C is the number of classes. Considering the grey inset in the bottom left we will describe the components of these feature volumes. For each spatial pixel (i, j), there are A anchors (here $A = 3$), each consisting of four anchor refinements $(\hat{p}^{(x)}_{i,j,k}, \hat{p}^{(y)}_{i,j,k}, \hat{p}^{(w)}_{i,j,k}, \hat{p}^{(h)}_{i,j,k})$, an objectness score $\hat{o}$ and a score for each category class $(\hat{c}_1, \ldots, \hat{c}_C)$. The box coordinates $(\hat{b}^{(x)}_{i,j,k}, \hat{b}^{(y)}_{i,j,k}, \hat{b}^{(w)}_{i,j,k}, \hat{b}^{(h)}_{i,j,k})$ are calculated (described in text) using the k_{th} anchors centroid feature pixel grid position $(a^{(\text{x})}_k, a^{(\text{y})}_k)$, the anchors width and height $(a^{(\text{w})}_k, a^{(\text{h})}_k)$, and the predicted refinements $(\hat{p}^{(x)}_{i,j,k}, \hat{p}^{(y)}_{i,j,k}, \hat{p}^{(w)}_{i,j,k}, \hat{p}^{(h)}_{i,j,k})$. These refinements values are also used for loss calculation (described in text) rather than the box coordinates. Layer numbers provided correspond to the layer numbers in Table 2.3 and Table 2.4 which outline the structures of the DarkNet and detection CNNs in more detail respectively.

Model	Input Size	Parameters		Batch Size	Memory	Samples / Second		mAP
		Learnable	Static			Train	Inference	
Faster R-CNN	600×1000	32954544	1311680	1	9GB	5.31	9.10	37.0
				4	10GB	4.13	9.07	
YOLOv3	416×416	61626049	150930	1	7GB	13.65	52.80	36.0
				4	10GB	21.15	92.38	
SSD	512×512	27594183	405507	1	2GB	15.81	40.42	29.0
				4	4GB	18.79	48.80	

Table 2.2: Performance and efficiency baselines for image based object detection model implementations. For each model we show the input image size, the number of learnable and static parameters, the mAP score (on MS-COCO). We test using batch sizes of 1 and 4, reporting the maximum GPU memory consumption and processing speeds (in samples per second) for training and inference. These evaluations were carried out on a single desktop machine — **CPU:** i7-4790 @ 3.60GHz × 8; **RAM:** 15.6GB; **GPU:** GTX 1080 Ti 11.7GB.

The **DarkNet-53 CNN**, shown in detail in Table 2.3, follows a fairly common modern classification CNN design. It is designed for fast classification and is pre-trained on **ImageNet** [Russakovsky et al., 2015] before being utilised as a feature extractor for YOLO. Experiments in the original paper [Redmon and Farhadi, 2018] showed that DarkNet-53 achieves similar accuracy as *ResNet-152* for image classification despite having over 35% less operations and being two times faster. The feature outputs used by the YOLO detector network are extracted on the last repetitions of the last three block steps. The DarkNet-53 CNN can be conceptualised as a set of three closely-related mappings, $\mathbf{x}_t \mapsto \boldsymbol{Z'}^{(g)}$ $(g = 1 \dots 3)$, each of which take a video frame and produce a multi-dimensional array. Specifically,

$$\boldsymbol{Z'}^{(g)} \in \mathbb{R}^{S^{(g)} \times S^{(g)} \times D^{(g)}}, \tag{2.7}$$

where $S^{(g)}$ is the spatial dimension, and $D^{(g)}$ is the channels depth. By default, using the standard 416 $\times$ 416 input image size ($\mathbf{x}^t \in \mathbb{R}^{416 \times 416 \times 3}$) we get,

$$\begin{aligned} \boldsymbol{Z'}^{(1)} &\in \mathbb{R}^{52 \times 52 \times 256}, \\ \boldsymbol{Z'}^{(2)} &\in \mathbb{R}^{26 \times 26 \times 512}, \\ \boldsymbol{Z'}^{(3)} &\in \mathbb{R}^{13 \times 13 \times 1024}, \end{aligned} \tag{2.8}$$

equating to spatial downsampling ratios of 8 $\times$ 8, 16 $\times$ 16, and 32 $\times$ 32 respectively.

The **detection CNN** part of the YOLO architecture, shown in detail in Table 2.4, utilises principles from FPNs to enrich spatially large images. It utilises two pathways, one that enriches spatially larger features with the rich contextual information from spatially smaller features, and one to generate output tensors for each spatial feature scale. We can consider it as a mapping of our three tensors $\boldsymbol{Z'}^{(g)} \mapsto \boldsymbol{Z}^{(g)}$ $(g = 1 \dots 3)$, where

$$\mathbb{R}^{S^{(g)} \times S^{(g)} \times A \times (5+C)} \ni \boldsymbol{Z}^{(g)} = [z_{i,j,k,l}], \tag{2.9}$$

Repeats	Layer Type	# Filters	Size / Stride	Output Size (w,h)	Layer #	Detector
	Convolutional	32	3×3 / 1	1	1	
	Convolutional	64	3×3 / 2	1 / 2	2	
	Convolutional	32	1×1 / 1		3	
$1\times$	Convolutional	64	3×3 / 1		4	
	Residual			1 / 2		
	Convolutional	128	3×3 / 2	1 / 4	5	
	Convolutional	64	1×1 / 1		6, 8	
$2\times$	Convolutional	128	3×3 / 1		7, 9	
	Residual			1 / 4		
	Convolutional	256	3×3 / 2	1 / 8	10	
	Convolutional	128	1×1 / 1		11, ..., 25	
$8\times$	Convolutional	256	3×3 / 1		12, ..., 26	
	Residual			1 / 8		$\rightarrow \boldsymbol{Z'}^{(1)}$
	Convolutional	512	3×3 / 2	1 / 16	27	
	Convolutional	256	1×1 / 1		28, ..., 42	
$8\times$	Convolutional	512	3×3 / 1		29, ..., 43	
	Residual			1 / 16		$\rightarrow \boldsymbol{Z'}^{(2)}$
	Convolutional	1024	3×3 / 2	1 / 32	44	
	Convolutional	512	1×1 / 1		45, ..., 51	
$4\times$	Convolutional	1024	3×3 / 1		46, ..., 52	
	Residual			1 / 32		$\rightarrow \boldsymbol{Z'}^{(3)}$

Table 2.3: The structure of the Darknet-53 network architecture. Each row represents a layer, with an input flowing from top to bottom. The columns show the number of **repeats** for a particular group of layers, the **layer type**, the **number of filters** in the layer, the spatial **size and stride** of the filter kernels, the **spatial output size** relative to the input image, the **layer number ID**, and the **outputs to the detector**. DarkNet-53 is mostly made up of blocks (shown in grey), which are repeated various times throughout the network. Each block consists of a *dense* (1×1) convolution, a 3×3 convolution, and a residual connection with the blocks input. Prior to each set of block repetitions there are 3×3 convolutions with strides of 2, which each halve the spatial resolution of the feature maps. As the network gets deeper, and the spatial resolution decreases, the number of filters is increased. Every convolutional layer is made up of the convolution operation followed by a BN and a Leaky ReLU (L-ReLU) activation triplet. BN and L-ReLU are omitted from the table for brevity. Also omitted from the table are an average pooling layer, dense convolution layer (layer 53) and Softmax activation which are used for classification but not for detection. The outputs of the DarkNet-53, and the inputs to the detection CNN are shown in the last column as $\boldsymbol{Z'}^{(1)}$, $\boldsymbol{Z'}^{(2)}$ and $\boldsymbol{Z'}^{(3)}$.

where $S^{(g)}$ is the spatial dimension and remains the same as for each $\boldsymbol{Z'}^{(g)}$, A is the number of anchors, and C is the number of classes. The 5 is attributed to four anchor box refinements

$$\hat{p}^{(\mathrm{x})}_{i,j,k} \triangleq z_{i,j,k,1}, \quad \hat{p}^{(\mathrm{y})}_{i,j,k} \triangleq z_{i,j,k,2}, \quad \hat{p}^{(\mathrm{w})}_{i,j,k} \triangleq z_{i,j,k,3} \quad \hat{p}^{(\mathrm{h})}_{i,j,k} \triangleq z_{i,j,k,4} \tag{2.10}$$

and one 'objectness' score $\hat{o}_{i,j,k} \triangleq z_{i,j,k,5}$. The remaining dimensions are associated with class predictions $\widehat{\boldsymbol{C}} = [\hat{c}_{i,j,k,u}] \triangleq [z_{i,j,k,u+5}]$ (for $u = 1 \dots C$), which follow a one-hot class representation convention.

The anchor box adjustments are used with the anchor's pre-determined width and height ($a^{(\mathrm{w})}_k$, $a^{(\mathrm{h})}_k$) and the feature region offset (anchor's centre coordinates) ($a^{(\mathrm{x})}_k, a^{(\mathrm{y})}_k$) to generate box predictions that lie in the unit interval

$$\begin{aligned}
\hat{b}^{(\mathrm{x})}_{i,j,k} &= \tau_I\left(\hat{p}^{(\mathrm{x})}_{i,j,k}\right) = \frac{\sigma\left(\hat{p}^{(\mathrm{x})}_{i,j,k}\right) + a^{(\mathrm{x})}_k}{S^{(g)}} \quad & \hat{b}^{(\mathrm{w})}_{i,j,k} &= \tau_{III}\left(\hat{p}^{(\mathrm{w})}_{i,j,k}\right) = \frac{a^{(\mathrm{w})}_k e^{\hat{p}^{(\mathrm{w})}_{i,j,k}}}{S^{(g)}} \\
\hat{b}^{(\mathrm{y})}_{i,j,k} &= \tau_{II}\left(\hat{p}^{(\mathrm{y})}_{i,j,k}\right) = \frac{\sigma\left(\hat{p}^{(\mathrm{y})}_{i,j,k}\right) + a^{(\mathrm{y})}_k}{S^{(g)}} \quad & \hat{b}^{(\mathrm{h})}_{i,j,k} &= \tau_{IV}\left(\hat{p}^{(\mathrm{h})}_{i,j,k}\right) = \frac{a^{(\mathrm{h})}_k e^{\hat{p}^{(\mathrm{h})}_{i,j,k}}}{S^{(g)}}.
\end{aligned} \tag{2.11}$$

where σ is the sigmoid function.

Training

To train the YOLO CNN mappings on ground-truth bounding-box annotations one needs to convert the ground truth bounding box coordinates, to ground truth anchor refinements via the relations

$$\begin{aligned}
p^{(\mathrm{x})}_{i,j,k} &= \tau_I^{-1}\left(b^{(\mathrm{x})}_{i,j,k}\right) = \log\left(\frac{b^{(\mathrm{x})}_{i,j,k} S^{(g)} - a^{(\mathrm{x})}_k}{1 - \left(b^{(\mathrm{x})}_{i,j,k} S^{(g)} - a^{(\mathrm{x})}_k\right)}\right) \quad & p^{(\mathrm{w})}_{i,j,k} &= \tau_{III}^{-1}\left(b^{(\mathrm{w})}_{i,j,k}\right) = \log\left(\frac{b^{(\mathrm{w})}_{i,j,k} S^{(g)}}{a^{(\mathrm{w})}_k}\right) \\
p^{(\mathrm{y})}_{i,j,k} &= \tau_{II}^{-1}\left(b^{(\mathrm{y})}_{i,j,k}\right) = \log\left(\frac{b^{(\mathrm{y})}_{i,j,k} S^{(g)} - a^{(\mathrm{y})}_k}{1 - \left(b^{(\mathrm{y})}_{i,j,k} S^{(g)} - a^{(\mathrm{y})}_k\right)}\right) \quad & p^{(\mathrm{h})}_{i,j,k} &= \tau_{IV}^{-1}\left(b^{(\mathrm{h})}_{i,j,k}\right) = \log\left(\frac{b^{(\mathrm{h})}_{i,j,k} S^{(g)}}{a^{(\mathrm{h})}_k}\right)
\end{aligned} \tag{2.12}$$

When training the YOLO object detection architecture the loss L_{yolo} used is a weighted sum of five terms

$$L^{(\mathrm{yolo})} = \underbrace{\lambda_1 L_1}_{\text{centroid loss}} + \underbrace{\lambda_2 L_2}_{\text{dimension loss}} + \underbrace{\lambda_3 L_3}_{\text{objectness loss}} + \underbrace{\lambda_4 L_4}_{\text{objectness penalty}} + \underbrace{\lambda_5 L_5}_{\text{class loss}}. \tag{2.13}$$

To break down these individual losses one by one we will make use of two kinds of indicator functions

$$\chi^{(I)}_{i,j,k}(\cdot) = \begin{cases} 1, & \text{if the } k\text{th anchor at position } (i,j) \text{ has an associated ground truth box} \\ 0, & \text{otherwise} \end{cases}$$

$$\chi^{(II)}_{i,j,k}(\cdot) = \begin{cases} 0, & \text{if the } k\text{th anchor at position } (i,j) \text{ overlaps with a ground truth box by } >70\% \\ 1, & \text{otherwise.} \end{cases}.$$

In	Repeats	Layer Type	# Filters	Size / Stride	Output Size (w,h)	Layer #	Out
$\boldsymbol{Z'}^{(3)} \rightarrow$	$\times 3$	Convolutional	512	1×1 / 1		1, 3, 5	
		Convolutional	1024	3×3 / 1	1 / 32	2, 4, 6o	
		Convolutional*	$A \times (5 + C)$	1×1 / 1		7o	$\rightarrow$
$5 \rightarrow$		Convolutional	256	1×1 / 1	1 / 32	6	
		Upsample		$\times 2$	1 / 16	7	
$\boldsymbol{Z'}^{(2)} \rightarrow$		Ch. Concatenation		(7 | $\boldsymbol{Z'}^{(2)}$)	1 / 16	8	
	$\times 3$	Convolutional	256	1×1 / 1		9, 11, 13	
		Convolutional	512	3×3 / 1	1 / 16	10, 12, 14o	
		Convolutional*	$A \times (5 + C)$	1×1 / 1		15o	$\rightarrow$
$13 \rightarrow$		Convolutional	128	1×1 / 1	1 / 16	14	
		Upsample		$\times 2$	1 / 8	15	
$\boldsymbol{Z'}^{(1)} \rightarrow$		Ch. Concatenation		(15 | $\boldsymbol{Z'}^{(1)}$)	1 / 8	16	
	$\times 3$	Convolutional	128	1×1 / 1		17, 19, 21	
		Convolutional	256	3×3 / 1	1 / 8	18, 20, 22o	
		Convolutional*	$A \times (5 + C)$	1×1 / 1		23o	$\rightarrow$

Table 2.4: The structure of the YOLO detection network architecture. Once again, each row represents a layer, with an input flowing from top to bottom (however now there are some skips as the model branches). The columns show the **inputs to each layer** — if blank it means inputs are from the row above, the number of **repeats** for a particular group of layers, the **layer type**, the **number of filters** in the layer, the spatial **size and stride** of the filter kernels, the **spatial output size** relative to the input image, the **layer number ID**, and the **outputs**. The detection CNN consists of three branches that generate output volumes at the three different input feature scales. Starting from the $\boldsymbol{Z'}^{(3)}$ feature input, the detection network performs three sets of 1×1 then 3×3 convolutions, splitting into two pathways before the final 3×3 convolution. One pathway performs another 3×3 convolution, followed by a dense 1×1 output convolution, while the other performs upsampling and concatenation with the $\boldsymbol{Z'}^{(2)}$ feature input. This process is repeated again for $\boldsymbol{Z'}^{(1)}$ to generate three output volumes, one from each branch, with $A \times (5 + C)$ channels per pixel. Similarly to DarkNet-53, all convolutional layers have BN and L-ReLU activations, with the exception of the output convolutional layers (*) which don't utilise BN.

The **centroid loss** is the loss calculated in relation to the differences between the ground truth box offsets ($p^{(\mathrm{x})}$, $p^{(\mathrm{y})}$) with the anchor centroid ($a^{(x)}$, $a^{(y)}$) and the predicted offsets ($\hat{p}^{(\mathrm{x})}$, $\hat{p}^{(\mathrm{y})}$) with the anchor centroid. For this the mean squared error (MSE) is calculated:

$$L_1 = \sum_i \sum_j \sum_k \chi^I_{i,j,k} \left(\frac{\left(\hat{p}^{(\mathrm{x})}_{i,j,k} - p^{(\mathrm{x})}_{i,j,k}\right)^2}{2} + \frac{\left(\hat{p}^{(\mathrm{y})}_{i,j,k} - p^{(\mathrm{y})}_{i,j,k}\right)^2}{2} \right). \tag{2.14}$$

The **dimensions loss** is very similar to the centroid loss however it is calculated in relation to the differences between the GT dimensions ($p^{(\mathrm{v})}$, $p^{(\mathrm{h})}$) in relation to the anchor dimensions ($a^{(w)}$, $a^{(h)}$) and the predicted offsets ($\hat{p}^{(\mathrm{w})}$, $\hat{p}^{(\mathrm{h})}$) in relation to the anchor. For this the MSE is again calculated:

$$L_2 = \sum_i \sum_j \sum_k \chi^I_{i,j,k} \left(\frac{\left(\hat{p}^{(\mathrm{w})}_{i,j,k} - p^{(\mathrm{w})}_{i,j,k}\right)^2}{2} + \frac{\left(\hat{p}^{(\mathrm{h})}_{i,j,k} - p^{(\mathrm{h})}_{i,j,k}\right)^2}{2} \right). \tag{2.15}$$

The **objectness loss** is calculated using the predicted objective scores $\hat{o}_{i,j,k}$ and is compared to $o_{i,j,k}$ which is a binary label $(0, 1)$ of whether this particular anchor relates to a GT image. For this the binary cross-entropy (BCE) is calculated:

$$\begin{aligned} L_3 &= \sum_i \sum_j \sum_k \chi^I_{i,j,k} \left(o_{i,j,k} \log(\hat{o}_{i,j,k}) + (1 - o_{i,j,k}) \log(1 - \hat{o}_{i,j,k}) \right) \\ &= \sum_i \sum_j \sum_k \chi^I_{i,j,k} \log(\hat{o}_{i,j,k}). \end{aligned} \tag{2.16}$$

The **no-objectness loss** is used to prevent the model from just proposing objects everywhere. It is calculated similarly to the objectness loss however we use the secondary indicator function $\chi^{(II)}_{i,j,k}(\cdot)$ which prevents this loss from penalising the model for being close but not exact with its predictions. Once again the BCE is calculated:

$$L_4 = \sum_i \sum_j \sum_k \chi^{II}_{i,j,k} \left(o_{i,j,k} \log(\hat{o}_{i,j,k}) + (1 - o_{i,j,k}) \log(1 - \hat{o}_{i,j,k}) \right). \tag{2.17}$$

The **class loss** is calculated across the set of individual class prediction scores $\widehat{\boldsymbol{C}}$ and the individual binary class labels $\boldsymbol{C}$. Again the BCE loss is used:

$$L_5 = \sum_i \sum_j \sum_k \sum_u \chi^I_{i,j,k} \left(c_{i,j,k,u} \log(\hat{c}_{i,j,k,u}) + (1 - c_{i,j,k,u}) \log(1 - \hat{c}_{i,j,k,u}) \right). \tag{2.18}$$

The **loss weightings** for the centroid and dimension losses are $\lambda_1 = \lambda_2 = 5$ to give more weighting to the box localisations over the classifications. For the non-objectness loss, $\lambda_4 = 0.5$ to prevent the network from focusing too much on feature pixels that don't have objects. While the other two losses are not weighted ($\lambda_3 = 1$ and $\lambda_5 = 1$).

2.1.6 Framewise Baseline Establishment

To get an idea of the initial baseline accuracy of the YOLO model, we train and evaluate with the three main image object detection datasets — Pascal VOC, MS-COCO, ImageNet-DET and the

main video object detection dataset — ImageNet-VID. For the training and testing of ImageNet-VID all frames are extracted and then used as individual image samples, mimicking the structure of the image datasets. Table 2.5 presents the results with the image based mAP evaluation metrics as well as the video based mAP metric for the ImageNet-VID dataset evaluation.

Our **training setup** for the results in Table 2.5 are as follows. Firstly, for all training and evaluations we sample the ImageNet-VID dataset taking only every 25^{th} frame, making the dataset more manageable to work with. We train using standard stochastic gradient descent (SGD) with a learning rate of 0.001, momentum of 0.9 and weight decay of 0.0005. We utilise four GPUs, enabling the use of a higher batch size of 64, training all models from scratch except for the ImageNet pre-training in the DarkNet-53 backbone which gets fine-tuned during training. For the Pascal VOC model we train for 200 epochs, with a warm-up of 4 epochs, and a learning decay of 0.1 on epochs 160 and 180. For the MS-COCO model we train for 280 epochs, warming-up with 2 epochs and decaying at epochs 220 and 250. For the ImageNet-DET model we train for 140 epochs with a warm-up of 3 epochs and decaying at epochs 100 and 120. While finally, for the ImageNet-VID model we train for 80 epochs, with 2 warm-up epochs and learning rate decay occurring at epochs 50 and 70. More details about our training setup for this experiment and all others are available in Appendix C.

Dataset	Image Based Metrics						
	VOC_{12}	$AP_{50\text{-}95}$	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Pascal VOC	83.6	46.2	73.5	50.0	11.3	30.4	56.4
MS-COCO	57.9	36.0	57.1	38.7	17.3	38.7	52.2
ImageNet-DET	54.3	34.1	50.4	37.6	4.8	18.6	45.2
ImageNet-VID	50.4	29.9	47.9	33.3	3.7	14.5	35.5
Dataset	**Video Based Metrics**						
	AP	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L
ImageNet-VID	46.1	52.9	47.7	29.5	14.8	36.2	57.7

Table 2.5: Framewise YOLO mAP baselines on the Pascal VOC, MS-COCO, ImageNet-DET and ImageNet-VID datasets. Concerning the image based metrics, the ImageNet-VID scores are the lowest, suggesting it is a more difficult dataset than the still image datasets.

Table 2.5 is able to provide some insight into what datasets might be harder to learn from and or predict on than others, with ImageNet-VID having the lowest mAP scores suggesting it is more difficult than the image datasets. To help gauge whether it is more difficult to *learn from*, or more difficult to *predict on*, *or both*, we experiment with using the models trained on the image datasets to predict on the ImageNet-VID dataset. Table 2.6 presents the per-class video AP for each class in the ImageNet-VID dataset. It's important to note that many of the classes in ImageNet-VID don't exist in Pascal VOC or MS-COCO, so for these classes an AP score is unobtainable and is ignored in the calculation of the mAPs.

Class		Training Set			
Wordnet Synset ID	ImageNet-VID Label	Pascal VOC	MS-COCO	ImageNet-DET	ImageNet-VID
n02691156	airplane	68.1	73.0	40.9	**79.7**
n02419796	antelope	-	-	78.0	58.5
n02131653	bear	-	45.8	63.5	46.1
n02834778	bicycle	56.4	61.3	28.9	58.9
n01503061	bird	44.3	47.2	53.5	47.9
n02924116	bus	68.7	74.6	41.3	66.7
n02958343	car	54.5	57.6	25.2	49.5
n02402425	cattle	-	-	49.9	45.2
n02084071	dog	31.7	37.9	52.3	41.5
n02121808	domestic cat	38.7	40.6	43.9	31.1
n02503517	elephant	-	58.1	61.7	51.6
n02118333	fox	-	-	52.8	48.3
n02510455	giant panda	-	-	50.7	**61.5**
n02342885	hamster	-	-	58.9	57.4
n02374451	horse	49.5	63.1	55.1	59.0
n02129165	lion	-	-	21.0	**22.5**
n01674464	lizard	-	-	67.8	39.9
n02484322	monkey	-	-	43.9	28.2
n03790512	motorcycle	41.2	47.3	22.2	43.9
n02324045	rabbit	-	-	59.3	45.1
n02509815	red panda	-	-	15.5	**16.3**
n02411705	sheep	22.0	27.3	14.8	**30.4**
n01726692	snake	-	-	52.9	18.8
n02355227	squirrel	-	-	29.0	**34.8**
n02129604	tiger	-	-	59.5	**60.2**
n04468005	train	64.4	69.9	56.1	**74.9**
n01662784	turtle	-	-	43.4	**48.9**
n04530566	watercraft	-	-	39.6	**43.5**
n02062744	whale	-	-	41.7	**44.9**
n02391049	zebra	-	25.8	27.1	**28.4**
	Mean (Best)	**49.0 (0/11)**	**52.2 (5/14)**	**45.2 (13/30)**	**46.1 (12/30)**
	Pascal VOC Subset	**49.0 (0/11)**	**54.5 (5/11)**	**39.5 (3/11)**	**53.0 (3/11)**
	MS-COCO Subset		**52.2 (5/14)**	**41.9 (5/14)**	**50.7 (4/14)**

Table 2.6: Image vs video trained models on ImageNet-VID with per class AP. This table presents the mAP evaluation results on the ImageNet-VID dataset using models trained on the image datases Pascal VOC, MS-COCO and ImageNet-DET, versus the model trained on the ImageNet-VID dataset. Best scores per class are underlined, with the means and best counts per dataset listed at the bottom. Also at the bottom are average scores over the Pascal VOC and MS-COCO class subsets. Considering the averages, even though not trained on the ImageNet-VID data, the models trained on the image datasets Pascal VOC and MS-COCO outperform the specifically trained ImageNet-VID models, with the ImageNet-DET model also competitive. Performance on the ImageNet-DET and ImageNet-VID datasets which both contain all 30 classes are relatively equivalent.

Table 2.6 shows that, with higher or competitive mAPs, the still image trained models are highly effective on the ImageNet-VID data despite not being trained on it. This can likely be attributed to two factors:

1. **The missing classes are more difficult**
   The mAPs for the Pascal VOC and MS-COCO trained models are relatively much higher than those of ImageNet-DET and ImageNet-VID, suggesting that the classes which are unavailable may be more difficult to detect. If we only consider the mAP scores from the ImageNet-VID model on the Pascal VOC and MS-COCO class subsets, we find the ImageNet-VID model scores in **53.0** (versus the Pascal VOC model's 49.0) and **50.7** (versus the MS-COCO model's 52.2) respectively. We observe that, with equivalent classes, the ImageNet-VID model is more accurate than the Pascal VOC model, and only slightly less accurate than the MS-COCO model. The remaining minor accuracy differences between the MS-COCO and ImageNet-DET datasets, as well as the increased performance on Pascal VOC, can likely be attributed to the second factor;

2. **There is better model generalisation from greater training diversity**
   When broken down into individual frames, ImageNet-VID contains over 2.6 million boxes for 30 classes (approximately 85000 per class), however since it is video, there is significant repetition across the boxes. A better indication of diversity for ImageNet-VID is the number of individual object instances, which is only 11.5k for 30 classes (approximately **380 per class**). When this is compared to the image sets, with Pascal VOC having 62k boxes for 20 classes (approximately **3100 per class**), MS-COCO having 900k boxes for 80 classes (approximately **11250 per class**), and ImageNet-DET having 530k boxes for 200 classes (approximately **2650 per class**), ImageNet-VID has relatively poor intra-class diversity. *Intra-class diversity is the diversity of samples for the same class*. Furthermore with MS-COCO and ImageNet-DET having 80 and 200 classes each, compared to the 30 of ImageNet-VID, the MS-COCO and ImageNet-DET models are likely to also have better inter-class diversity. *Inter-class diversity is the diversity of samples across varying classes*.

We look to verify the second factor as discussed above by looking to see if there is any correlation between the *inter-* and *intra*-class diversity and the class APs from all four of the datasets. Classes that are not in all four datasets are omitted from this experiment. By considering the number of boxes per class per dataset we can get a good idea of the extent of intra-class diversity, however this gives no indication of inter-class diversity. For inter-class diversity we need to factor in the number of classes in each dataset. We therefore measure intra- and inter-class diversity as:

$$\frac{\text{the number of boxes for a class in a dataset}}{\text{the number of classes for that dataset}} \tag{2.19}$$

We present results in Figure 2.5 where we compare the diversity scores for the classes across the four datasets and compare it to their AP scores. We find that despite some scaling of the measure, and some noisy per class results, on average the diversity scores are proportionate to the AP scores. This result supports the argument that greater *intra-* and *inter*-class diversity

leads to better generalisation and AP performance at test time.

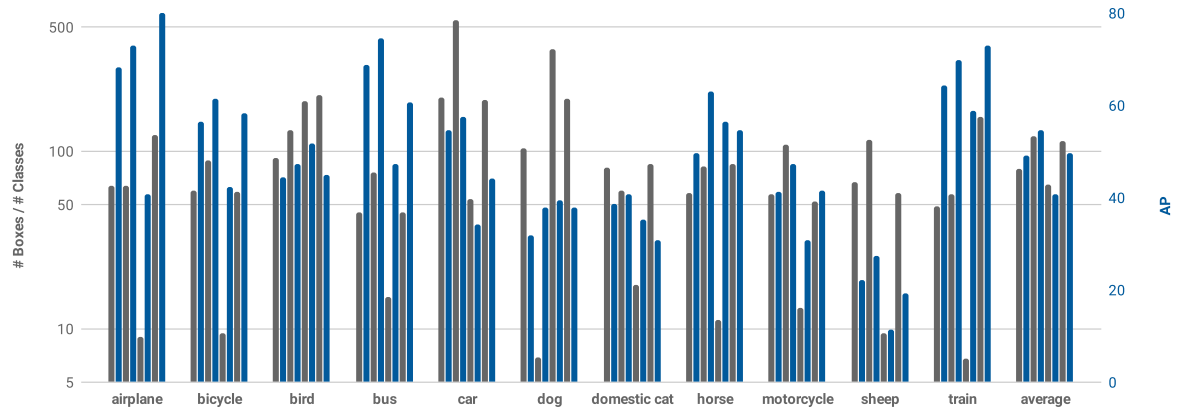


Figure 2.5: Class diversity compared with individual class APs for object localisation datasets on ImageNet-VID. Here we look to compare individual class diversities for the object datasets and its the relation of diversity to AP. Each class is listed along the bottom as well as an average of the 11 classes on the far right. The grey bars correspond to the class diversity, the higher the bar the more diverse, as determined by our heuristic. The blue bars present the AP scores for each of the classes. The bars are meant to be considered in pairs, from left to right for each class we have the diversity and AP for Pascal VOC, then MS-COCO, ImageNet-DET, and ImageNet-VID. Results should be considered on a class-by-class basis with the comparisons coming between each dataset or bar pairs. Despite some variation for the individual classes, if we consider the average we observe that the APs are proportionate to the diversity scores.

2.1.7 Transfer Learning from Image Data

It has been found that training a model or part of a model on a separate, generally more diverse, dataset before training on the problem dataset can lead to improved accuracy. This idea is called **transfer learning** and is utilised in the standard YOLO architecture, with the DarkNet-53 backbone CNN being pre-trained on the ImageNet classification dataset before being fine-tuned in the detection model on the detection datasets. Since the DarkNet-53 is used for feature extraction, exploiting the highly diverse and expansive ImageNet dataset enables the DarkNet-53 to learn very generalised image filters that are also highly useful for object detection. This idea of transfer learning can be taken a step further with pre-training on one or multiple of the image datasets, before training again (fine-tuning) on the ImageNet-VID dataset.

We perform fine-tuning experiments for each of the image datasets, pre-training with an image set and then fine-tuning with the ImageNet-VID dataset. For these experiments we fine-tune for another 40 epochs with 2 warm-up epochs, a learning rate of 0.001 decaying by a factor of 0.1 at epochs 20 and 30. Table 2.7 shows that pre-training on any of the image datasets prior to training on ImageNet-VID is beneficial to model accuracy, with higher mAPs in all cases (48.2, 49.9, and 51.4 compared to 45.0 on average).

Pre-training with the ImageNet-DET dataset is especially successful (with mAP 51.4), likely due to it including all of the classes in ImageNet-VID. If we consider per class APs as are shown in Table 2.8 we can see that for the Pascal VOC and MS-COCO classes, if we simply consider the

Trained On	mAP						
	AP	AP_{SL}	AP_{MO}	AP_{FA}	AP_{S}	AP_{M}	AP_{L}
Pascal VOC	48.2	55.3	50.3	30.4	16.0	38.2	59.7
MS-COCO	49.9	58.5	50.8	31.6	18.5	39.4	61.1
ImageNet-DET	51.4	60.8	52.4	32.0	16.3	39.2	64.1
ImageNet-VID	46.1	52.9	47.7	29.5	14.8	36.2	57.7

Table 2.7: Image dataset fine-tuning of YOLO results on ImageNet-VID. This table presents the results of fine-tuning each of the individual image dataset models on the ImageNet-VID data. All fine-tuned models outperform the ImageNet-VID only trained model, with the ImageNet-DET model performing particularly well.

averages we see that fine-tuning has decreased performance from 49.0 to 48.2 and from 52.2 to 49.9 for Pascal VOC and MS-COCO respectively. However, as these means consider the classes that exist *only* in the ImageNet-VID dataset, this fine-tuning comparison isn't particular fair. So instead we consider the class subsets separately and find the ImageNet-VID only classes attain significantly lower mAPs (43.9 vs 48.2 and 45.0 vs 49.9) than the actual fine-tuned classes, which see performance boosts (48.2 to 55.6 and 49.9 to 55.6) for Pascal VOC and MS-COCO respectively.

Class		Training Set						
Wordnet Syn.	ImageNet-VID Lab.	Pascal VOC		MS-COCO		ImageNet-DET		ImageNet-VID
n02691156	airplane	68.1	83.4	73.0	84.8	40.9	83.8	79.7
n02419796	antelope	-	61.8	-	67.4	78.0	63.2	58.5
n02131653	bear	-	52.0	45.8	51.4	63.5	50.4	46.1
n02834778	bicycle	56.4	66.1	61.3	67.4	28.9	66.0	58.9
n01503061	bird	44.3	51.4	47.2	55.6	53.5	54.0	47.9
n02924116	bus	68.7	68.2	74.6	74.3	41.3	72.4	66.7
n02958343	car	54.5	50.5	57.6	52.5	25.2	51.1	49.5
n02402425	cattle	-	42.3	-	54.4	49.9	50.4	45.2
n02084071	dog	31.7	45.4	37.9	45.0	52.3	52.4	41.5
n02121808	domestic cat	38.7	39.0	40.6	36.8	43.9	43.8	31.1
n02503517	elephant	-	52.0	58.1	53.3	61.7	56.2	51.6
n02118333	fox	-	46.2	-	55.0	52.8	54.0	48.3
n02510455	giant panda	-	65.8	-	68.4	50.7	69.2	61.5
n02342885	hamster	-	57.7	-	60.0	58.9	65.2	57.4
n02374451	horse	49.5	62.5	63.1	67.2	55.1	61.4	59.0
n02129165	lion	-	22.6	-	23.9	21.0	28.1	22.5
n01674464	lizard	-	37.8	-	36.9	67.8	48.6	39.9
n02484322	monkey	-	30.9	-	27.7	43.9	34.4	28.2
n03790512	motorcycle	41.2	44.9	47.3	46.0	22.2	48.5	43.9
n02324045	rabbit	-	48.7	-	45.5	59.3	51.5	45.1
n02509815	red panda	-	22.4	-	9.3	15.5	16.5	16.3
n02411705	sheep	22.0	25.0	27.3	38.0	14.8	30.8	30.4
n01726692	snake	-	22.8	-	22.1	52.9	27.1	18.8
n02355227	squirrel	-	38.0	-	37.4	29.0	43.7	34.8
n02129604	tiger	-	65.0	-	59.8	59.5	66.3	60.2
n04468005	train	64.4	74.9	69.9	76.4	56.1	76.1	74.9
n01662784	turtle	-	52.0	-	49.7	43.4	52.3	48.9
n04530566	watercraft	-	46.0	-	53.4	39.6	46.2	43.5
n02062744	whale	-	41.3	-	48.7	41.7	49.5	44.9
n02391049	zebra	-	28.8	25.8	29.3	27.1	29.6	28.4
Mean		**49.0**	**48.2**	**52.2**	**49.9**	**46.1**	**51.4**	**45.0**
ImageNet-VID Only Classes			**43.9**		**45.0**			
Fine-Tuned Only Classes		**49.0**	**55.6**	**52.2**	**55.6**	**46.1**	**51.4**	

Table 2.8: Image dataset fine-tuning of YOLO individual class AP results on ImageNet-VID. This table presents the results of fine-tuning each of the individual image dataset models on the ImageNet-VID data with respect to individual class APs. We also show the means overall and for the ImageNet-VID only class subset, and fine-tuned only class subset. We also show the results without fine-tuning (from Table 2.7) in light grey. We see performance improvements on average for the classes that exist in both the pre-training image dataset and the fine-tuning ImageNet-VID video dataset. This leads to pre-training on the ImageNet-DET dataset being most beneficial.

2.2 Temporal Object Detection

Thus far, despite training and testing on the video ImageNet-VID dataset, each frame is considered as a separate input sample. Although this allows for the trivial extension of image based architectures into the video domain, performance is limited due to challenges relating specifically to video. Firstly, the dynamic nature of video generally leads to deteriorated object appearance because of motion blur and video defocusing. Secondly, there tends to be more occlusions due to objects of interest often moving behind other scene elements. Lastly, dynamic objects such as people or animals can appear in rarer poses because a video captures their full motion patterns, whereas still imagery tends to caption certain poses most often *eg. sitting, standing, walking*. These differences can make it more difficult to detect some objects in certain frames, leading to many missed detections. While individual frames may be degraded and difficult to perform detection on, the benefit of video is that surrounding frames may be clearer. Therefore if frames are considered and processed together, there is more opportunity to recover detections that otherwise would have been missed in the framewise case. Furthermore, even if *clear* or *insightful* frames are selected and processed, low level motion information is still lost if frames aren't considered in dense temporal blocks. While losing such information may not be detrimental to the captioning task, we argue that it is still insightful for general video understanding, and hence worthwhile to explore.

2.2.1 Object Detection in Videos

Object detection in videos is an extension of object detection in images, where objects must be found temporally in addition to spatially. Object detection in video doesn't have the requirement of instance identification and tracking across time, which are currently seen as an extra processing technique called **object tracking**. For this reason evaluation and mAP scoring is still carried out on a per-frame basis, however now the per-frame detections will be based on information from multiple frames rather than a singular frame. The frame we are evaluating on and trying to detect on is called the **reference frame**.

When it comes to exploiting the temporal information to improve the detection performance on the reference frame researchers have focused their efforts on two complimentary approaches (Figure 2.6):

1. **Box-level refinement** — involves trying to link and align detection boxes or *tubelets* of the same object instance across time. Tubelets are boxes through time ie. imagine aligning bounding boxes next to each other through time they would generate tubes in space-time. Such methods are encroaching on solutions to the *tracking* problem and can generally be added as a post-processing technique to the second set of approaches;
2. **Feature-level refinement** — involves utilising temporally adjacent or nearby frames to augment and enrich the features of the reference frame. This enriching looks to provide in-

formation from surrounding frames which might be deteriorated in the reference frame, promoting better detection in the latter detection part of the model, which is reliant on the features per frame.

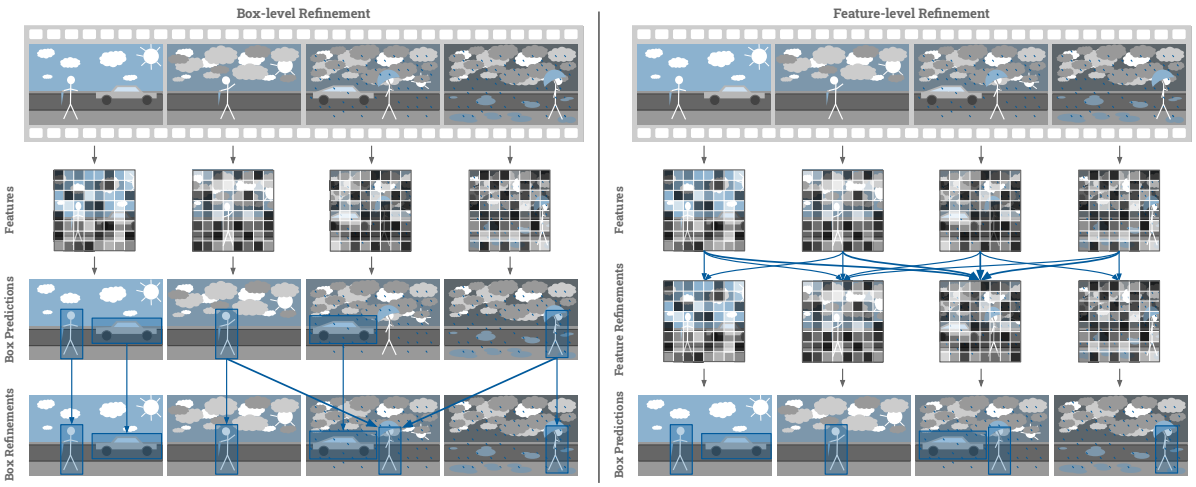


Figure 2.6: The comparison between box-refinement and feature-refinement approaches for object detection in video. Given a video, frames individually get processed by some CNN to attain features. In our example, while the man is well represented by the these features in frames 1,2, and 4, in frame 3 it's difficult to determine his presence. For the box-refinement approach, this leads to the man being missed in the box predictions. To overcome this, the box-refinement techniques consider the box predictions from the surrounding frames. In comparison, the feature-refinement methods alter the features themselves, augmenting them with features from nearby frames. This augmented feature is then sufficient for the detection of the man in the detection stage.

Box-level Refinement

Seq-NMS

Introduced by [Han et al., 2016], **Seq-NMS** is an extension of the image based NMS into the temporal dimension. It has three stages, selection of high scoring boxes, box re-scoring, and box suppression. Box re-scoring is based on linking the high scoring boxes across frames, which is achieved by linking boxes with IoU scores over 50%. The scores for the linked boxes are then averaged over time to generate tubelet scores.

TCN, T-CNN & TPN

Leaning heavily on the principles of the R-CNN network, [Kang et al., 2016] generate 2000 object proposals per frame using selective search [Uijlings et al., 2013], removing most using an AlexNet extracted from a ImageNet-DET pre-trained R-CNN. Remaining proposals are scored using a set of SVMs, like as is done for the R-CNN, using a GoogLeNet [Szegedy et al., 2013] pre-trained on ImageNet and fine-tuned on ImageNet-DET as a feature extractor. The highest confidence box proposals are then bi-directionally tracked through time with a tracking algorithm [Wang et al., 2015], creating tubelet proposals. The tubelet proposals are refined by replacing proposal boxes with original object detection boxes that have a high IoU score. Due to object detectors being sensitive to pose variation, large confidence differences are often present across

time for a tubelet, making it necesarry for them to be re-scored. Here the authors use a set of **1D temporal convolutional networks (TCNs)** for each class, each individually trained to predict the probability that a tubelet contains that class.

Extending the method above is [Kang et al., 2018], who introduce **Tubelets with CNN (T-CNN)**, adding *multi-context suppression (MCS)* and *motion-guided propagation (MGP)* stages. The purpose of MCS is to remove false positive spatial region proposals, by subtracting a set value from low confidence proposals, while keeping high confidence proposals the same. They decide the thresholds and subtraction amount by greedily searching on the validation set. Their intuition behind this is based on the fact that the ImageNet-DET dataset almost always has only 1-2 object instances. The purpose of MGP is to recover false negatives, by adding spatial proposals from adjacent frames, since often objects don't move far spatially in adjacent frames. These proposals are then tracked and further processed in the same way as [Kang et al., 2016].

[Kang et al., 2017] extend upon the previous methods by introducing a **Tubelet Proposal Network (TPN)** which replaces the computationally expensive and time consuming tracker of [Wang et al., 2015] with a neural network. Using object proposals on the initial frame as spatial anchors, features are temporally pooled across temporal windows from the static object proposals. This idea exploits the receptive field of CNNs, where deep features cover information from large spatial regions — so over small enough temporal windows the static regions at the feature level spatially cover the movements of the object. They train their TPN to regress the relative movements of the anchor boxes across time, resulting in tubelets. To generate long tubelets, the final adjusted box of the previous window is used as the static anchor for the next window, iteratively growing the tubelets through time. To classify tubelets they utilise an encoder-decoder LSTM RNN, taking the ROI features across time as the input sequence.

Feature-level Refinement

Optical Flow based Bilinear Feature Warping

[Zhu et al., 2017b] propose **flow-guided feature aggregation (FGFA)**, which utilises optical flow (OF) frames to warp and aggregate feature maps between a set of subsequent frames. Using FlowNet [Dosovitskiy et al., 2015], they generate flow fields between a reference frame and a neighbouring frame, and then use *bilinear warping* to warp the neighbouring frame to the reference frame. For each reference frame, multiple neighbouring frames are warped and then aggregated using a spatially weighted sum. The weights are estimated using the cosine similarity between the warped feature and an embedded version of the reference feature. The resulting aggregated feature is then passed into the detection network, in their case an R-FCN [Dai et al., 2016].

One of the key problems with the above FGFA [Zhu et al., 2017b] is that reference features are generated for every frame, which is computationally expensive. [Zhu et al., 2017c] improve the FGFA process with their **deep feature flow (DFF)** model, which only performs the expensive fea-

ture extraction on sparse key frames. To generate features for non-key frames, they similarly use a OF-guided bilinear warping, which is a much faster process than the CNN feature computation. To further speed-up the OF calculation process they design two smaller FlowNet models that are one-quarter and one-eighth the complexity of the standard FlowNet [Dosovitskiy et al., 2015]. Furthermore, to improve the FlowNet's ability to produce useful OF fields, they jointly learn it with the rest of their framework.

[Zhu et al., 2018] **combine the FGFA and DFF approaches**, and further look to improve the speed and accuracy trade-off with the addition of three new ideas. Firstly, they perform recursive feature warping and aggregation like that in FGFA but only on sparse key frames. Secondly, they only update particular spatial regions of non-key frame features, which are considered to be low quality, with *quality* being learnt by the network. Thirdly, they perform temporally-adaptive key frame selection based on the predicted feature quality.

[Wang et al., 2018c] also utilise OF fields to perform feautre warping, however their focus is on not only using pixel-level information, but also using instance-level information. They introduce their **motion-aware network (MANet)** to jointly consider both pixel-level and instance-level features across time.

Correlation Filter Encoding

Acknowledging the complexity of the OF warping approaches, [Feichtenhofer et al., 2017] look to simplify the video object detection process by focusing on generating short temporally localised tracklets in an end-to-end learnable framework. Rather than utilising flow to warp and aggregate features across time, they instead use **correlation features**, which are calculated between pairs of adjacent frames. These correlation features are particularly good at representing object co-occurences through time, and are utilised in a novel ROI-tracking layer which regresses box transformations between frames. This results in tracklets between correlated frames, which are joined into tubelets via a linking process like that from [Gkioxari and Malik, 2015].

Recurrent Neural Network Encoding

With a focus on fast inference speeds [Liu and Zhu, 2018] perform feature refinement with a **Convolutional-LSTM RNN** mixed in with the latter layers of a SSD object detection model [Liu et al., 2016]. Similarly, [Ning et al., 2017] directly use a **LSTM RNN** on the output features of a YOLO detection network [Redmon and Farhadi, 2017].

Spatio-Temporal Hierarchical Refinement

Using a Faster R-CNN as a base detector, [Chen et al., 2018a] introduce a scale-time lattice which introduces **Propagation and Refinement Units (PRUs)** that combine and refine two features into a new feature. The PRUs are used in a hierarchical binary-tree structure, going from lower detailed at small scales and higher detailed and higher scales, allowing for the expensive base detector to only be run on sparse frames.

De-convolutional Operations Refinement

[Bertasius et al., 2018] introduce **spatio-temporal sampling networks (STSN)** which works by concatenating the features of two frames, obtained from the backbone of a Deformable CNN [Dai et al., 2017] detection network. A set of deconvolutional layers are then used to determine offsets which are used to combine the features.

Relation Networks Refinement

Extending upon Relation Networks [Hu et al., 2018], which contextually consider all ROIs in a still image, [Deng et al., 2019b] introduce the **Relation Distillation Network (RDN)** which contextually considers ROIs in individual frames and across multiple frames. This is done in two stages, a basic stage which considers all ROIs from the reference frame to surrounding frames, and then an advanced stage which considers the top ROIs from all frames with those in the reference frame.

Memory Networks Refinement

[Deng et al., 2019a] utilise an external memory module to guide the feature aggregations based on features and ROI features from past frames. A pixel memory is read to augment the frame features prior to a RPN, and then an object instance memory is read to augment the ROI features, before standard regression, classification and NMS. The memory is then updated with the new object detection and global frame features.

[Chen et al., 2020] introduce the **memory enhanced global-local aggregation (MEGA)** network which combines Relation Network modules with a long-range memory module. While other approaches only take either a global or local approach for comparing and aggregating features across time, here both are jointly carried out. Furthermore, thanks to the simplified memory module, features can be aggregated across the entire video clip, rather than just the 20-30 frames as was done in past approaches.

2.2.2 Turning YOLO Temporal

The detection works discussed in Section 2.2 take two complimentary approaches to object detection in video. With the utilisation of some of the still image object detection pipelines presented in Section 2.1, they look to either refine the box outputs from these detectors over time, or look to refine the feature outputs before continuing on with the detection process. In this thesis we consider the second of these approaches as we believe errors should be resolved as early as possible in the detection process, with making the job easier for down-stream processes.

Compared to the previously discussed approaches we take a step back and look to determine the effects of various standardised temporal information encoding methodologies from other video problem domains including **video classification** and **event detection**. Furthermore, unlike the past works which utilise the image detection models practically as-is, only extracting features for refinement, we look to modify the architecture of YOLO itself in an attempt to make it more suitable for videos. The down-side to this approach is that we break it's ability to be fine-tuned from the image object detection datasets. There may be solutions to this such as

specific network design, weight initialisations and training techniques, however we leave these studies for future work.

In this section we only modify the detection CNN section of the YOLO architecture, allowing us to still attain the benefits of the DarkNet-53 pre-training on the ImageNet image classification dataset. Specifically, we investigate four standardised procedures for combining the DarkNet features across time — **pooling**, **channel concatenation**, **feature correlations**, and **3D convolutions**.

For the temporal case it is still necessary to generate detections for every frame, however unlike as was done in Section 2.1, where only the one frame was utilised, now a window of W frames is used. This window is centered around the reference frame at time-step t, with frames included either side with some stride s. We remind the reader that we sample the ImageNet-VID dataset at every 25$^{\text{th}}$ frame, but **only for the reference frames**. So in this case, for example, a stride of $s = 1$ is the directly adjacent frame, not the next 25$^{\text{th}}$ frame.

Freezing DarkNet

Passing a window of $W > 1$ frames into the DarkNet and YOLO detection model results in more computational overhead. Therefore a common tactic to improve model efficiency is to **freeze** parts of the model architecture. Freezing means that the weights are unchanged during training and gradients don't need to be stored and backpropogated for the frozen model parts. While this improves model efficiency there is a trade-off in that the frozen layers are unable to learn and fine-tuned themselves, generally resulting in a decrease in accuracy.

We explore the effect of freezing the DarkNet feature network on model efficiency and accuracy. Table 2.9 shows by freezing the DarkNet we can almost double our processing speed with about three-quarters of the RAM usage. Freezing doesn't effect inference speeds, as the freezing has no affect on the data flow during the inference process.

Frozen DarkNet	Parameters		Batch Size	Memory (GB)		Samples / Second	
	Trainable	Non-trainable		RAM	GPU	Train	Inference
✘	61626049	150930	1	7	7	14	144
			4	8	10	23	250
✓	21041121	40735858	1	6	5	25	144
			4	6	10	41	250

Table 2.9: YOLO model efficiency with and without freezing the DarkNet. Here we show the effects on the YOLO network and its training time and resources usage. By freezing the DarkNet we reduce the number of parameters by two-thirds resulting in about three-quarters of the RAM usage, and an almost two times speed-up in training.

Table 2.10 shows the decrease in model accuracy when freezing the DarkNet, with single frame mAP dropping from 46.1 to 42.0. As we are looking to perform many experiments with limited resources, we freeze the DarkNet for all of the following experiments. We expect all results

would see further performance increases with the fine-tuning of the DarkNet with the YOLO training.

Frozen DarkNet	mAP						
	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP
✘	52.9	47.7	29.5	14.8	36.2	57.7	**46.1**
✓	51.5	42.0	23.3	12.0	31.4	53.8	**42.0**

Table 2.10: YOLO model accuracy with and without freezing the DarkNet. Here we show the effects of freezing the DarkNet on the YOLO network's mAP scores on the ImageNet-VID dataset. We see a decrease across all of the scores, and a 4.1 (10%) decrease in the main mAP when the DarkNet is frozen.

A second option to increase model efficiency is by pre-computing the DarkNet output features ($\boldsymbol{Z'}^{(1)}$, $\boldsymbol{Z'}^{(2)}$, $\boldsymbol{Z'}^{(3)}$) for every frame $\mathbf{x}_t$ and just passing them in to the YOLO detection layers, completely removing the DarkNet data flow from the training process. While this would speed up processing significantly, it requires more storage as both the features and frames need to be stored. Furthermore, it practically prohibits the ability to perform data-augmentation on the input frames. Due to limited storage and the desire for data-augmentation we don't explore this pathway, however still consider it a viable method for other works.

Pooling

The simplest way to merge data across time is to perform **temporal pooling**. Temporal pooling has been used in works for video classification [Yue-Hei Ng et al., 2015, Donahue et al., 2015] and video captioning [Venugopalan et al., 2015b, Zanfir et al., 2016, Zhou et al., 2019, Zheng et al., 2020]. We experiment with both **mean** and **max** pooling, at both an early and a late pooling stage. We start by outlining the steps associated with **early pooling**.

Let $\left(\boldsymbol{Z'}_t^{(g)}\right)_{t=1}^{T}$ be a sequence of feature tensors (one for each video frame) produced by the DarkNet feature outputs. At each time-step t we sample a collection of frames from the time interval $t \pm s\lfloor \frac{W}{2} \rfloor$ and form a sequence of tensors $\left(\boldsymbol{W}_t^{(g)}\right)_{t=1}^{T}$, where for $t' = 1 \dots W$,

$$\begin{aligned} \mathbb{R}^{W \times S^{(g)} \times S^{(g)} \times D^{(g)}} \ni \boldsymbol{W}_t^{(g)} &= \left[w_{t',i,j,d}^{(g)} \right] \\ &= \left[z'_{t-s\lfloor \frac{W}{2} \rfloor + st',i,j,d} \right], \end{aligned} \tag{2.20}$$

and s is the stride or gap between consecutive frames. This sequence is, in turn, transformed by pooling operations into another sequence $\left(\boldsymbol{U}_t^{(g)}\right)_{t=1}^{T}$. In particular, $\boldsymbol{U}_t^{(g)} \in \mathbb{R}^{S^{(g)} \times S^{(g)} \times D^{(g)}}$ where each element is:

$$[u_{i,j,d}^{(g)}] = \begin{cases} \frac{1}{W} \sum_{t'} [w_{t',i,j,d}^{(g)}] & \text{if we apply mean pooling to } \boldsymbol{W}_t \\ [w_{t^*,i,j,d}^{(g)}] & \text{if we apply max pooling to } \boldsymbol{W}_t, \end{cases} \tag{2.21}$$

where $t^* = \underset{t'}{\operatorname{argmax}} [w_{t',i,j,d}^{(g)}]$.

We dub the aforementioned steps as early pooling because we operate on the DarkNet features before they are passed into the YOLO detection layers. The steps for **late pooling** are analogous, the only difference being that instead of using DarkNet features, we use the YOLO features directly prior to the final output convolutional layers of the YOLO network. Figure 2.7 presents the two pooling positions with blue circles, which will be the same positions for concatenation and correlation experiments in the next subsections.

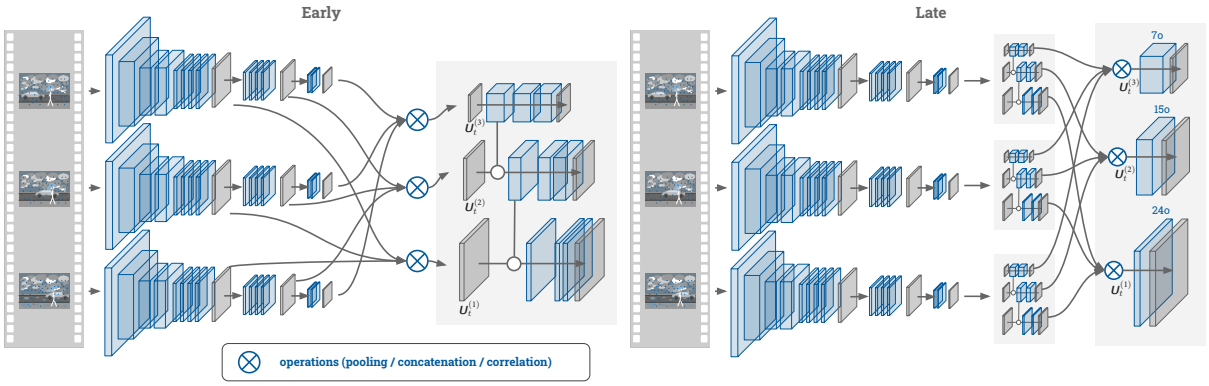


Figure 2.7: Early vs late positioning for temporal merging operations in YOLO. Here we present the position differences between the early and late pooling, concatenation and correlation operations as indicated by the blue outlined circles. Early pooling merges the DarkNet feature outputs ($\boldsymbol{Z'}^{(1)}$, $\boldsymbol{Z'}^{(2)}$, $\boldsymbol{Z'}^{(3)}$), while late pooling merges features in the detection CNN right before the last convolutional operation. The operations generate features $\boldsymbol{U}^{(1)}$, $\boldsymbol{U}^{(2)}$, $\boldsymbol{U}^{(3)}$ where the W dimension is condensed to 1.

Table 2.11 presents the pooling results using a window size of $W = 3$ and strides of $s = 1$ and $s = 25$. Comparing the results between the two stride sizes it can be seen that performing pooling on a small stride of $s = 1$ improves results over the single frame baseline, while the larger stride of $s = 25$ worsens them. This highlights that **simple pooling can be effective for only very short localised temporal regions**. The relative variation of temporally sparse frames is likely too high, resulting in blurred or noisy features upon pooling, decreasing the mAP. Considering the individual AP_{SL}, AP_{MO} and AP_{FA} scores, it's clear that the faster an object moves the more effect that the larger window stride deteriorates the feature. Otherwise, variation in performance between early and late, or mean and max is relatively minor, with slightly better performance for max over mean, and late over early.

Channel Concatenation

Temporal **channel concatenation** was utilised for FlowNet-S [Dosovitskiy et al., 2015] to capture temporal information between two subsequent frames for optical flow generation. Despite its simplicity, its effectiveness for generating relatively accurate optical flow was quite impressive. We experiment with performing concatenation along the filters ($D^{(g)}$) channel, generating a new

Pooling		Window		mAP							# Paramaters	
Pos.	**Type**	**Size**	**Str.**	**AP_{SL}**	**AP_{MO}**	**AP_{FA}**	**AP_{S}**	**AP_{M}**	**AP_{L}**	**AP**	**Learnable**	**Static**
early	**mean**	**3**	**1**	50.0	43.9	26.7	13.0	31.9	54.7	42.8	21094971	40735858
		3	**25**	53.2	35.7	10.0	6.1	22.7	51.2	36.9		
	max	**3**	**1**	52.4	45.0	**27.6**	13.1	33.6	56.3	44.1		
		3	**25**	51.3	34.1	**8.7**	6.2	21.9	48.5	35.6		
late	**mean**	**3**	**1**	53.1	44.6	26.8	13.1	33.7	56.5	44.2		
		3	**25**	51.8	35.0	8.4	7.6	23.5	49.4	36.0		
	max	**3**	**1**	53.5	45.2	25.9	13.2	33.7	56.6	**44.7**		
		3	**25**	52.3	35.0	9.9	7.3	23.7	48.8	**36.3**		
baseline		**1**	**1**	51.5	42.0	23.3	12.0	31.4	53.8	**42.0**	21094971	40735858

Table 2.11: YOLO mAP evaluations using different pooling strategies. On the left we have the **position** and **type** of the pooling, and the window **size** and **stride**. On the right we show the number of **learnable** and **static** parameters. Results are fairly similar between the various types of pooling, the main factor playing a role here is the window stride. Using the higher stride of 25 is about 6 mAP points lower than the baseline, while using a stride of 1 gives up to a 3 point improvement. This effect is exacerbated for the fast moving objects (**AP_{FA}**) which on average see differences between the two stride values of 17 mAP points, compared to 1-2 points for the slow moving objects (**AP_{SL}**).

feature $\boldsymbol{U}_t^{(g)} \in \mathbb{R}^{S^{(g)} \times S^{(g)} \times W D^{(g)}}$ per scale g, where each element is:

$$[u_{i,j,d}^{(g)}] = \begin{cases} [w_{1,i,j,d}^{(g)}] & \text{if } d \in [1, 2, \ldots, D^{(g)}] \\ [w_{2,i,j,d}^{(g)}] & \text{if } d \in [D^{(g)} + 1, D^{(g)} + 2, \ldots, 2D^{(g)}] \\ \vdots & \\ [w_{W,i,j,d}^{(g)}] & \text{if } d \in [(W-1)D^{(g)} + 1, (W-1)D^{(g)} + 2, \ldots, WD^{(g)}] \end{cases} \tag{2.22}$$

The concatenation along the filters channel keeps the spatial structure intact, while allowing specific weighting towards the different timesteps. Without modification of the YOLO convolutional layers, and their respective number of channels, most of the processing is required from the first convolutional layers after temporal channel concatenation. We perform concatenation in the same positions as pooling, either on the DarkNet output features (early) or before the final output convolutional layers (late).

Table 2.12 presents the results for the early and late concatenation for different window strides $s \in \{1, 5, 10, 25\}$. Results are similar for all experiments, with an improvement of about 2-3 mAP points over the baseline. The late pooling position seems slightly favourable compared to the early pooling, as well as a larger window stride. This shows that larger strides can be beneficial provided the encoding is dynamically suited for larger strides, as is the case here with the convolutional operations directly proceeding the concatenations.

Concatenation	Window		mAP							# Parameters	
Position	Size	Str.	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	Learnable	Static
early	3	1	52.0	44.6	25.7	12.5	33.3	55.4	43.6	22471227	40735858
	3	5	53.4	44.7	24.9	12.5	33.3	56.6	44.2		
	3	10	51.1	44.7	25.0	12.2	32.8	55.3	43.2		
	3	25	52.2	45.7	27.2	14.0	33.9	55.9	44.8		
late	3	1	52.8	44.9	26.6	12.2	33.3	56.6	44.2	21471291	
	3	5	54.9	45.9	25.0	14.2	34.6	56.9	45.3		
	3	10	54.0	45.9	24.1	13.4	34.1	56.5	44.7		
	3	25	55.5	45.9	25.9	13.1	34.1	58.0	45.4		
baseline	1	1	51.5	42.0	23.3	12.0	31.4	53.8	42.0	21094971	40735858

Table 2.12: YOLO mAP evaluations using different concatenation strategies. On the left we have the **position** of the concatenation and the window **size** and **stride**. On the right we show the number of **learnable** and **static** parameters. Results are all relatively similar with slight (1-4 points) mAP improvements.

Feature Correlation

Utilised in FlowNet-C for optical flow generation [Dosovitskiy et al., 2015] and for video object detection [Feichtenhofer et al., 2017] with good effect, **feature correlation** allows for the modeling of small spatial translations in feature activations over time, by comparing local spatial neighbourhoods of two temporally spaced feature maps.

Depicted in Figure 2.8, feature correlation involves performing a *convolution* between two feature maps, rather than a feature map and a filter (as is done in standard convolutional operations). In this case, *filters* are extracted from one feature map at every spatial pixel location (i, j). Each filter is then applied to the second feature map, performing the standard 2D convolutional operation. Rather than allowing the filters to be applied to every spatial location in the second feature map, the filters are bounded to a spatial area specified by a maximum displacement limit $\bar{d}$. Specifically the (i, j)th filter from the first feature map is only applied to the 2D pixel space $(i - \bar{d}, j - \bar{d}), \ldots, (i + \bar{d}, j + \bar{d})$ in the second feature map. This results in $(2\bar{d} + 1)^2$ 2D convolutional operations, and hence $(2\bar{d} + 1)^2$ output feature channels. Since there are no learnt filters, performing correlation adds no extra parameters to the model, which is one of the benefits of its usage.

As we are working with a window size of length $W = 3$, we need to perform two feature correlations, one between the first frame feature (at time-step $t - s$) and the reference frame feature (at time-step t), and one between the last frame feature (at time-step $t + s$) and the reference frame feature. These each result in one output tensor each — denoted by $\boldsymbol{U'}^{(g)} \in$

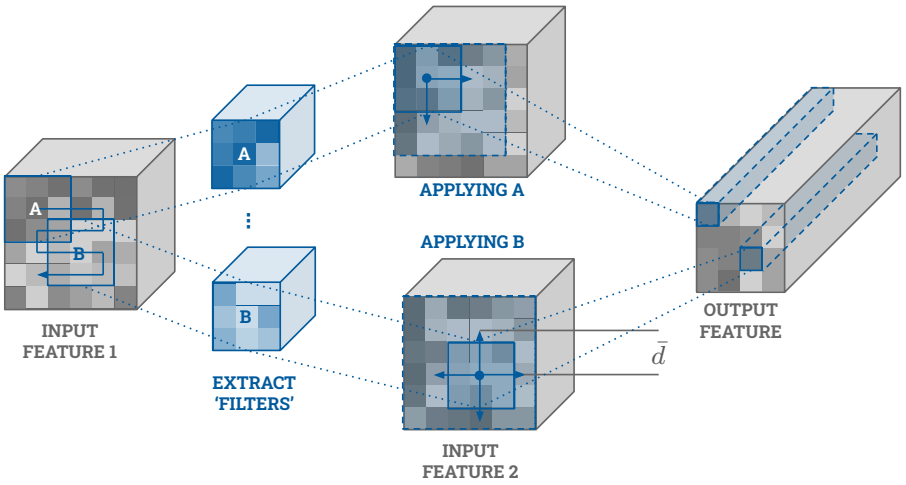


Figure 2.8: A visual representation of the feature correlation process. Feature correlation involves performing a *convolution* between two feature maps (1 and 2), rather than a feature map and a filter. In this case, *filters* are extracted from one feature map (1) at every spatial pixel location (i, j), resulting in $S^{(g)} \times S^{(g)}$ separate filters (these are the A and B, each of size 3×3). Each filter is then applied to the second feature map (2), performing the standard 2D convolutional operation. To limit the filters to only focusing within spatial neighbourhoods of the pixel locations they were extracted from (i, j), they are limited to being applied to the pixels $(i-\bar{d}, j-\bar{d}), \ldots, (i+\bar{d}, j+\bar{d})$ in feature 2 with $\bar{d}$ being a maximum displacement limit. In the figure this 2D limit is shown on feature 2 with the dashed blue box, while the filter is shown with the solid blue box. This results in $(2\bar{d}+1)^2$ convolutional operations per (i, j) and also $(2\bar{d}+1)^2$ output feature channels. For simplicity this figure omits padding additions.

$\mathbb{R}^{S^{(g)} \times S^{(g)} \times (2\bar{d}+1)(2\bar{d}+1)}$ and $\boldsymbol{U''}^{(g)} \in \mathbb{R}^{S^{(g)} \times S^{(g)} \times (2\bar{d}+1)(2\bar{d}+1)}$. Each are calculated as:

$$\boldsymbol{U'}^{(g)} = \left[u'^{(g)}_{i,j,k}\right] = \sum_{\check{i},\check{j},d} w^{(g)}_{1,i+\check{i}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, j+\check{j}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, d} w^{(g)}_{2,\pi_1(k)+\check{i}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, \pi_2(k)+\check{j}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, d} \tag{2.23}$$

and

$$\boldsymbol{U''}^{(g)} = \left[u''^{(g)}_{i,j,k}\right] = \sum_{\check{i},\check{j},d} w^{(g)}_{3,i+\check{i}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, j+\check{j}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, d} w^{(g)}_{2,\pi_1(k)+\check{i}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, \pi_2(k)+\check{j}-\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor - 1, d} \tag{2.24}$$

where

$$\begin{aligned} \pi_1(k) &= k - (i-1)(2\bar{d}+1) - (j-1)(2\bar{d}+1) + i - \bar{d} - 1 \\ \pi_2(k) &= k - (i-1)(2\bar{d}+1) - (j-1)(2\bar{d}+1) - i(2\bar{d}+1) + j - \bar{d} - 1 \end{aligned} \tag{2.25}$$

with $\check{i} = 1, ..., K^{(\mathrm{S})}$ and $\check{j} = 1, ..., K^{(\mathrm{S})}$, where $K^{(\mathrm{S})}$ is the filter kernel spatial dimensions, in our case $K^{(\mathrm{S})} = 3$. While the definitions of $\pi_1(\cdot)$ and $\pi_2(\cdot)$ may look complex, they merely represent the necessary index manipulations for determining the particular pixel position in the second feature based on the output index k. Furthermore, to permit the convolution of a $K^{(\mathrm{S})} \times K^{(\mathrm{S})}$ filter offset by $\bar{d}$ on the boundary pixel positions (ie. $(1, 1)$) the input features are padded by $\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor + \bar{d}$. For simplicity in the correlation equations above, we keep the spatial indices constant despite the padding (ie. the top-left position in the padded feature would be $i = 1 - (\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor + \bar{d})$ and $j = 1 - (\left\lfloor \frac{K^{(\mathrm{S})}}{2} \right\rfloor + \bar{d})$).

The results $\boldsymbol{U'}^{(g)}$ and $\boldsymbol{U''}^{(g)}$ are concatenated together along the filters ($D^{(g)}$) channel to form

our correlated feature $\boldsymbol{U}_t^{(g)} \in \mathbb{R}^{S^{(g)} \times S^{(g)} \times 2(2\bar{d}+1)(2\bar{d}+1)}$:

$$\boldsymbol{U}_t^{(g)} = [u_{i,j,d}] = \begin{cases} [u'_{i,j,d}] & \text{if } d \in [1, 2, \ldots, (2\bar{d}+1)^2] \\ [u''_{i,j,d}] & \text{if } d \in [(2\bar{d}+1)^2 + 1, (2\bar{d}+1)^2 + 2, \ldots, 2(2\bar{d}+1)^2] \end{cases} \tag{2.26}$$

Once again we experiment with utilising correlation in the same early and late positions as pooling and concatenation. We investigate two different displacement limits $\bar{d} \in \{4, 8\}$. and two temporal strides $s \in \{1, 25\}$. Table 2.13 presents the mAP scores for different correlation strategies. Using correlation results in a minor improvement over the baseline across all model versions. Most benefit is found from using the larger displacement $\bar{d} = 8$, and the later positioning. These suggest that determining spatial relationships over time works better for the more contextually rich features late in the detection network. Furthermore, considering that we are working with spatial feature sizes of 52×52, 26×26, and 13×13, the better performing $\bar{d} = 8$ suggests relationships over larger spatial regions are more important for this dataset, ie. spatially global motions are more insightful than spatially local ones.

Correlation		Window		mAP							# Parameters	
$\bar{d}$	Pos.	Size	Stride	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	Learnable	Static
4	early	3	1	49.8	43.4	25.6	12.1	32.4	53.8	42.4	22616379	40735858
		3	25	50.8	43.3	24.1	12.3	32.0	53.8	42.5		
	late	3	1	52.1	44.1	25.9	11.8	32.7	55.8	43.7	21522321	
		3	25	51.2	45.1	26.8	12.5	34.5	55.1	43.7		
8	early	3	1	51.2	44.4	26.6	13.1	33.1	55.6	43.6	22989115	
		3	25	53.1	46.1	25.5	12.5	33.6	56.5	44.6		
	late	3	1	52.9	45.5	26.6	13.1	34.7	56.5	44.6	21653361	
		3	25	52.4	44.8	26.9	12.3	33.7	55.5	44.1		
baseline		1	1	51.5	42.0	23.3	12.0	31.4	53.8	42.0	21094971	40735858

Table 2.13: YOLO mAP scores using different correlation strategies. On the left we have the **maximum displacement** value d and the **position** of the correlation, and the window **size** and **stride**. On the right we show the number of **learnable** and **static** parameters. Results show that the larger maximum displacement of 8 is more beneficial than 4. Furthermore performing correlation late is slightly more beneficial than early.

3D Convolutions

While two-dimensional convolutions are common for image processing, allowing for spatial modelling, they can be extended into three dimensions (3D) to allow for spatio-temporal modelling. **Three dimensional convolutions**, initially utilised in [Ji et al., 2013, Tran et al., 2015] have shown great promise in processing video representations in many applications [Ji et al., 2013, Karpathy et al., 2014, Tran et al., 2015].

One of the problems with adding a third dimension to 2D convolutional layers is that it exponentially increases the number of weights in the kernels and the number of operations. This makes 3D convolutions much more computationally expensive compared to 2D. To combat this issue, inspired by depthwise separable convolutions which are used to speed up 2D convolutions [Chollet, 2017, Howard et al., 2017], [Qiu et al., 2017] and [Tran et al., 2018] decompose the 3D convolutional operation into its spatial and temporal components, creating P3D and (2+1)D convolutional operations respectively. P3D and (2+1)D operations involve splitting the $K^{(\mathrm{T})} \times K^{(\mathrm{S})} \times K^{(\mathrm{S})}$ convolution into a spatial $1 \times K^{(\mathrm{S})} \times K^{(\mathrm{S})}$ convolution and a temporal $K^{(\mathrm{T})} \times 1 \times 1$ convolution. Separating these operations results in significantly less parameters per 3D convolution allowing for faster processing or a deeper network.

We investigate the performance of the 3D, and less computationally demanding (2+1)D convolutional operations for associating temporal information. We do this by replacing the 2D convolutions in the detection network with 3D / (2+1)D convolutions. We keep the properties of the 2D convolutions and extend them into the time dimension. Using a temporal stride and padding of 1, the temporal output size is equal to the temporal input size. This design allows for the mixing of temporal information, but still requires pooling to merge data over time. Following results from our pooling experiments, we utilise late temporal max pooling.

Table 2.14 presents the mAP results for the two convolutional strategies, with improvements over the baselines in almost all cases. Despite the significantly lower number of learnable parameters, the (2+1)D convolutions perform better than standard 3D, particularly for the larger stride $s = 25$. Interestingly, the larger stride results in inferior results compared to the smaller stride $s = 1$ for the standard 3D case, whereas the opposite result occurs for the R(2+1)D case.

Convolution	Window		mAP							# Parameters	
Type	Size	Stride	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	Learnable	Static
(2+1)D	3	1	52.8	45.6	27.9	13.3	33.9	56.7	44.8	33868347	40746610
	3	25	57.5	40.0	27.0	13.3	34.5	57.6	45.6		
3D	3	1	52.1	45.3	26.9	12.8	33.5	56.2	44.3	58630203	40735858
	3	25	51.9	44.9	24.9	11.7	33.1	56.3	43.7		
baseline	1	1	51.5	42.0	23.3	12.0	31.4	53.8	42.0	21094971	40735858

Table 2.14: YOLO mAP scores using different convolutional strategies. On the left we have the **type** of convolution, and the window **size** and **stride**. On the right we show the number of **learnable** and **static** parameters. Results show minor performance improvements with either strategy, with the (2+1)D convolutions performing slightly better than the standard 3D convolutions despite having significantly less parameters.

2.2.3 Turning DarkNet Temporal

The modifications to the detection layers of the YOLO network have improved accuracy for object detection in videos against the single frame baseline, however many activity detection and classification works have found benefits to capturing temporal information at very early stages of their networks. While earlier temporal modelling may be more beneficial for identifying actions, which are composed of much finer detailed motion, we are focused on determining any benefits for identifying objects. In this section we investigate methods of permitting earlier temporal information processing prior to the YOLO detection layers. Specifically, we investigate **motion dedicated streams**, **hierarchical 2D convolutions**, and **hierarchical 3D convolutions with residual connections**.

All image and video object detection methods rely on an ImageNet pre-trained feature network, such as the DarkNet used in this case. Using a pre-trained network has significant benefits for both faster and more accurate learning, however any blatant change in network structure could break the information flow and invalidate the ImageNet pre-training. Some works including [Mansimov et al., 2015, Carreira and Zisserman, 2017] have explored techniques of modifying pre-trained network structures without being overly destructive to their representative abilities. We look to minimise any direct modifications to the DarkNet information flow to keep its representative power gained from the pre-training intact.

Motion Dedicated Streams

One way to incorporate lower level motion information, without any modification to the DarkNet is to add a completely separate network or *stream*. We then extract the usual features from DarkNet $\boldsymbol{Z'}_t^{(g)}$ and a corresponding three features from the second stream $\boldsymbol{Z''}_t^{(g)}$, concatenating them together along their channels dimension prior to input into YOLO's detection layers:

$$\mathbb{R}^{S^{(g)}\times S^{(g)}\times (D^{(g)}+D''^{(g)})} \ni \boldsymbol{U}_t^{(g)} = [u_{i,j,d}^{(g)}] = \begin{cases} [z_{i,j,d}'^{(g)}] & \text{if } d \in [1,2,\ldots,D^{(g)}] \\ [z_{i,j,d}''^{(g)}] & \text{if } d \in [D^{(g)}+1, D^{(g)}+2, \ldots, D^{(g)}+D''^{(g)}] \end{cases} \tag{2.27}$$

We investigate using a **FlowNet network** [Dosovitskiy et al., 2015] and a **R(2+1)D network** [Tran et al., 2018].

FlowNet

The **FlowNet** model, shown in detail in Table 2.15, consists of two stages — an embedding with reduction phase and a decoding with expansion phase. The first phase is a fairly standard CNN pipeline, while the second phase uses de-convolutional layers to expand the feature maps into flow maps. **De-convolutions** or *transposed convolutions* are convolutions that increase the output size by using padding and or striding on the input.

In	Layer Type	# Filters	Size / Stride	Output Size (w,h)	Layer #	Out
			Phase 1			
	Convolutional	64	7×7 / 2	1 / 2	1	
	Convolutional	128	5×5 / 2	1 / 4	2	$\rightarrow$ 26
	Convolutional	256	5×5 / 2	1 / 8	3	
	Convolutional	256	3×3 / 1		4	$\rightarrow$ 22
	Convolutional	512	3×3 / 2	1 / 16	5	
	Convolutional	512	3×3 / 1		6	$\rightarrow$ 18
	Convolutional	512	3×3 / 2	1 / 32	7	
	Convolutional	512	3×3 / 1		8	$\rightarrow$ 14
	Convolutional	1024	3×3 / 2	1 / 64	9	
	Convolutional	1024	3×3 / 1		10	
			Phase 2			
	Convolutional	2	3×3 / 1		11	
	De-convolutional	2	4×4 / 2	1 / 32	12	$\rightarrow$ 14
10 $\rightarrow$	De-convolutional	512	4×4 / 2		13	
8, 12, 13 $\rightarrow$	Concatenation				14	$\rightarrow \boldsymbol{Z''}_t^{(3)}$
	Convolutional	2	3×3 / 1		15	
	De-convolutional	2	4×4 / 2	1 / 16	16	$\rightarrow$ 18
14 $\rightarrow$	De-convolutional	256	4×4 / 2		17	
6, 16, 17 $\rightarrow$	Concatenation				18	$\rightarrow \boldsymbol{Z''}_t^{(2)}$
	Convolutional	2	3×3 / 1		19	
	De-convolutional	2	4×4 / 2	1 / 8	20	$\rightarrow$ 22
18 $\rightarrow$	De-convolutional	128	4×4 / 2		21	
4, 20, 21 $\rightarrow$	Concatenation				22	$\rightarrow \boldsymbol{Z''}_t^{(1)}$
	Convolutional	2	3×3 / 1		23	
	De-convolutional	2	4×4 / 2	1 / 4	24	$\rightarrow$ 26
21 $\rightarrow$	De-convolutional	64	4×4 / 2		25	
2, 24, 25 $\rightarrow$	Concatenation				26	
	Convolutional	2	3×3 / 1		27	

Table 2.15: The details of the FlowNet architecture. Each row represents a layer, with an input flowing from top to bottom (however now there are some skips as the model branches). The columns show the **inputs to each layer** — if blank it means inputs are from the row above, the **layer type**, the **number of filters** in the layer, the spatial **size and stride** of the filter kernels, the **spatial output size** relative to the input image, the **layer number ID**, and the **outputs** — to either later layers or to the detection network. **FlowNet** consists of two stages — an embedding with reduction phase and a decoding with expansion phase. Phase 1, at the top, resembles a relatively standard 2D CNN pipeline. Phase 2, on the other hand, includes de-convolutional layers which spatially expand the feature size. We take outputs from the first three concatenation layers.

We extract feature tensors $\boldsymbol{Z''}_t^{(1)}$, $\boldsymbol{Z''}_t^{(2)}$ and $\boldsymbol{Z''}_t^{(3)}$ from after FlowNet's **concatenation** layers, which contain both the phase 1 and the phase 2 flow features, capturing as much information at each scale as possible. We only extract features from the first three concatenations as they match the spatial resolutions of the corresponding DarkNet features. This approach of using FlowNet's inner features is different than past two-stream approaches [Simonyan and Zisserman, 2014, Donahue et al., 2015, Yue-Hei Ng et al., 2015], which take the final FlowNet output feature and pass it into another network for classification. By foregoing the use of another flow classification network we reduce the parameters and the need to train another stream. For DarkNet with FlowNet we sample three frames passing the reference frame $\mathbf{x}_t$ through the DarkNet, and two adjacent frames on either side (again with some stride s), $\mathbf{x}_{t-s}$ and $\mathbf{x}_{t+s}$, through the FlowNet. Figure 2.9 presents a visual representation of each of the stream structures and how each of the feature tensors are extracted and concatenated together prior to the detection network.

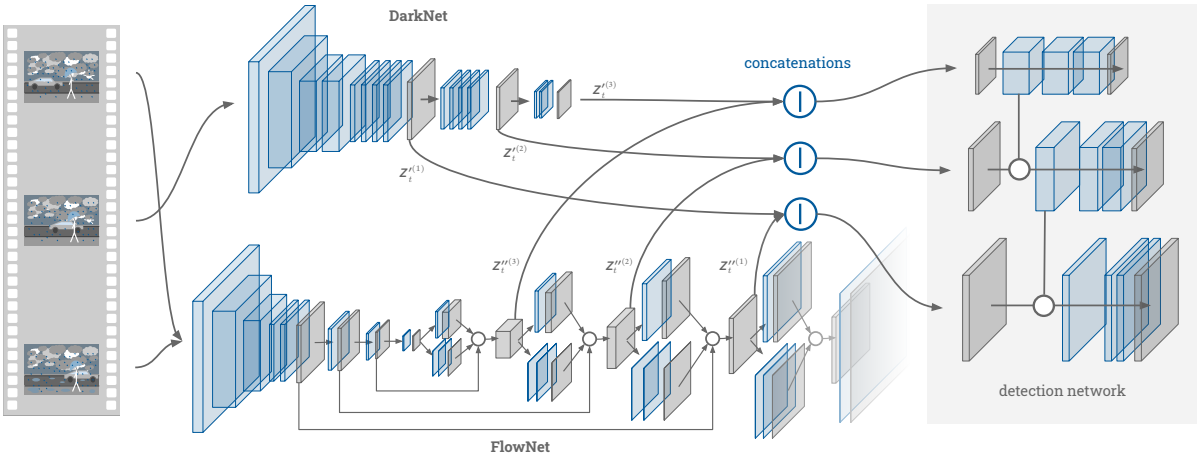


Figure 2.9: A visual representation of the DarkNet and FlowNet multi-stream approach. Given a window of frames $W = 3$, the reference frame (middle) is passed into the standard DarkNet model shown at the top, while the adjacent frames on either side are passed through the FlowNet model shown along the bottom. Spatially equivalent features are taken at three points along each of the streams and concatenated together along the channels dimension. The resultant features of the three varying scales are then passed into the standard YOLO detection network.

R(2+1)D

The **R(2+1)D** model, shown in detail in Table 2.16, modifies the ResNet network architecture [He et al., 2016], replacing the 2D convolutions with R(2+1)D convolutions. We extract feature tensors shown as $\boldsymbol{Z''}_t^{(1)}$, $\boldsymbol{Z''}_t^{(2)}$ and $\boldsymbol{Z''}_t^{(3)}$ from the **end of stages 2, 3 and 4** respectively. Each of these features are two-times too large spatially so we spatially max pool with a spatial size and stride of 2 to down-sample the feature maps to the size matching the DarkNet features. Furthermore, we use a temporal input of $W = 9$ frames, but the standard input length for R(2+1)D is 8, therefore we utilise global max pooling along the time dimension before concatenating with the DarkNet features. Specifically, features $\boldsymbol{Z''}_t^{(1)}$, $\boldsymbol{Z''}_t^{(2)}$, and $\boldsymbol{Z''}_t^{(3)}$ have temporal dimensions of 5, 3 and 1 respectively, so features $\boldsymbol{Z''}_t^{(1)}$ and $\boldsymbol{Z''}_t^{(2)}$ are max pooled across time also. The 9 sampled frames are centred on the frame we are performing detection on, passing the 5$^{\text{th}}$ frame into the DarkNet stream and all 9 into the R(2+1)D stream. Figure 2.10 presents a visual repre-

sentation of each of the stream structures and how each of the feature tensors are extracted and concatenated together prior to the detection network.

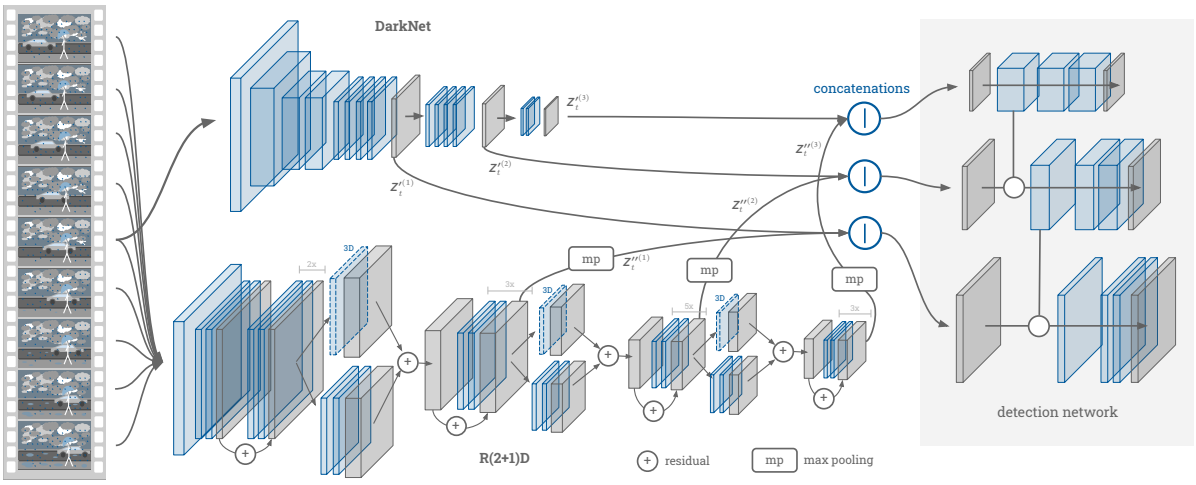


Figure 2.10: A visual representation of the DarkNet and R(2+1)D multi-stream approach. Given a window of frames $W = 9$, the reference frame (middle) is passed into the standard DarkNet model shown at the top, while all frames (including the reference) are passed through the R(2+1)D model shown along the bottom. All blue layers in the R(2+1)D network are R(2+1)D convolutional operations each consisting of a $1 \times 3 \times 3$ followed by a $3 \times 1 \times 1$ 3D convolution pair, with the exception of the three standard 3D layers with dashed outlines. As the R(2+1)D network generates features that are two-times the size of corresponding DarkNet features, we spatially max pool them to down-sample them to the correct sizes. Similarly, the earlier two features ($\boldsymbol{Z''}_t^{(1)}$ and $\boldsymbol{Z''}_t^{(2)}$) have temporal sizes of 5 and 3 respectively, so we also max pool temporally to squash them into one time-step. These max pooling operations are shown with **mp**, while the residual connections of the R(2+1)D stream are show with **(+)**. The pooled features are then concatenated with their corresponding DarkNet features (along the channels dimension), and passed into the standard YOLO detection network.

Table 2.17 presents the results for the addition of the pre-trained motion streams alongside the DarkNet stream, presenting for both frozen and unfrozen models. For the frozen models we find there is minor improvement in mAP for faster moving objects, however scores worsen for the slower moving objects. This might suggest our stride choice of $s = 1$ is too small, resulting in only tiny movements between frames within the windows. As these motion streams work best with low level motion, the slow objects were likely too slow to be detected. Interestingly, when allowing fine-tuning of the streams with the detection network we find the opposite is true, with the slower objects attaining the improvement in mAP (albeit very small). The fine-tuning of the R(2+1) model resulted in significant drops in performance, suggestive that the detection network destructively altered the pre-trained weights of the R(2+1)D model.

Hierarchical 2D

Rather than 3D convolutional layers, temporal information can be captured by performing 2D convolutions over multiple frames with feature outputs either pooled or 1D convoluted into a single 2D feature. This idea of temporal fusion has been done by [Karpathy et al., 2014] for the problem of video classification where different arrangements (early, late, slow fusion) were investigated. We introduce this idea in a hierarchical layer setup - *akin to slow fusion*, where 2D convolutions are repeated across temporal windows.

Stg.	Rep.	In	Layer Type	# Filters (Mid)	Size / Stride	Out Size	Layer #	Out
			(2+1)D Conv	64 (45)	3×7×7 / 1,2	1, 1/2	1	
1			(2+1)D Conv	64 (144)	3×3×3 / 1,1		2	
			(2+1)D Conv	64 (144)	3×3×3 / 1,1		3	
			Residual				r1	
			(2+1)D Conv	64 (144)	3×3×3 / 1,1		4, 6	
	2×		(2+1)D Conv	64 (144)	3×3×3 / 1,1		5, 7	
			Residual				r2, r3	
2			(2+1)D Conv	128 (230)	3×3×3 / 2,2	1/2, 1/4	8	
			(2+1)D Conv	128 (288)	3×3×3 / 1,1		9	→ r4
		r3 →	3D Conv	128	1×1×1 / 2,2	1/2, 1/4	d1	
		9 + d1 →	Residual				r4	
			(2+1)D Conv	128 (288)	3×3×3 / 1,1		10, 12, 14	
	3×		(2+1)D Conv	128 (288)	3×3×3 / 1,1		11, 13, 15	
			Residual				r5, r6, r7	$\rightarrow \boldsymbol{Z}''^{(1)}_t$
3			(2+1)D Conv	256 (460)	3×3×3 / 2,2	1/4, 1/8	15	
			(2+1)D Conv	256 (576)	3×3×3 / 1,1		16	→ r8
		r7 →	3D Conv	256	1×1×1 / 2,2	1/4, 1/8	d2	
		16 + d2 →	Residual				r8	
			(2+1)D Conv	256 (576)	3×3×3 / 1,1		17, ..., 25	
	5×		(2+1)D Conv	256 (576)	3×3×3 / 1,1		18, ..., 26	
			Residual				r9, ..., r13	$\rightarrow \boldsymbol{Z}''^{(2)}_t$
4			(2+1)D Conv	512 (912)	3×3×3 / 2,2	1/8, 1/16	27	
			(2+1)D Conv	512 (1152)	3×3×3 / 1,1		28	→ r14
		r13 →	3D Conv	512	1×1×1 / 2,2	1/8, 1/16	d3	
		28 + d3 →	Residual				r14	
			(2+1)D Conv	512 (1152)	3×3×3 / 1,1		29, 31	
	3×		(2+1)D Conv	512 (1152)	3×3×3 / 1,1		30, 32	
			Residual				r15, r16	$\rightarrow \boldsymbol{Z}''^{(3)}_t$
			Avg. Pool		Global			

Table 2.16: The details of the R(2+1)D 34-layer architecture. Each row represents a layer, with an input flowing from top to bottom (however now there are some skips as the model branches). The columns show the **stage number**, the number of **repeats** for a particular group of layers, **inputs to each layer** — if blank it means inputs are from the row above, the **layer type**, the **number of filters** in the layer (and the inner filter dimension for R(2+1)D), the temporal and spatial **sizes and strides** of the filter kernels, the temporal and spatial **output size** relative to the input window, the **layer number ID**, and the **outputs** — to either later layers or to the detection network. **R(2+1)D** is made up of four stages, each with similar structures — some convolutions followed by a residual connection. We take the outputs from stages 2, 3 and 4.

Frz.	Window		Motion	mAP							# Parameters	
	Size	Str.		AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	Learnable	Static
	3	1	FlowNet	50.6	42.7	**25.6**	11.7	31.7	54.2	42.0	21866811	79412364
✓	9	1	R(2+1)D	49.7	**43.6**	24.5	13.7	31.2	53.9	42.1	21439035	104278121
	1	1	baseline	**51.5**	42.0	23.3	12.0	31.4	53.8	42.0	21094971	40735858
	3	1	FlowNet	**53.2**	**49.0**	**27.9**	13.7	36.1	57.9	46.3	101128245	150930
✘	9	1	R(2+1)D	26.7	25.6	13.8	7.3	19.6	31.1	**23.1**	125515938	201218
	1	1	baseline	52.9	47.7	29.5	14.8	36.2	57.7	46.1	61679899	150930

Table 2.17: mAP evaluations of including pre-trained motion streams alongside the DarkNet stream. On the left we have the **whether the streams were frozen or not** during training, the window **size** and **stride** and **type** of motion stream. On the right we show the number of **learnable** and **static** parameters. We show that performance with the addition of the two motion streams is mixed, and not beneficial in comparison to the baseline. While there does seem some promise on the faster moving objects ($\mathbf{AP_{FA}}$) for the frozen models, the opposite is the case for the unfrozen models. Fine-tuning the R(2+1)D model with the detection network results in the breaking of the R(2+1)D pre-training leading to significantly deteriorated mAP performance.

We break the DarkNet into stages and experiment with applying max pooling or 1D convolutions with temporal width of $K^{(\mathrm{T})} = 3$ after each stage. Figure 2.11 presents the hierarchical 2D model with the hierarchical structure **2 levels deep to layer number 5**.

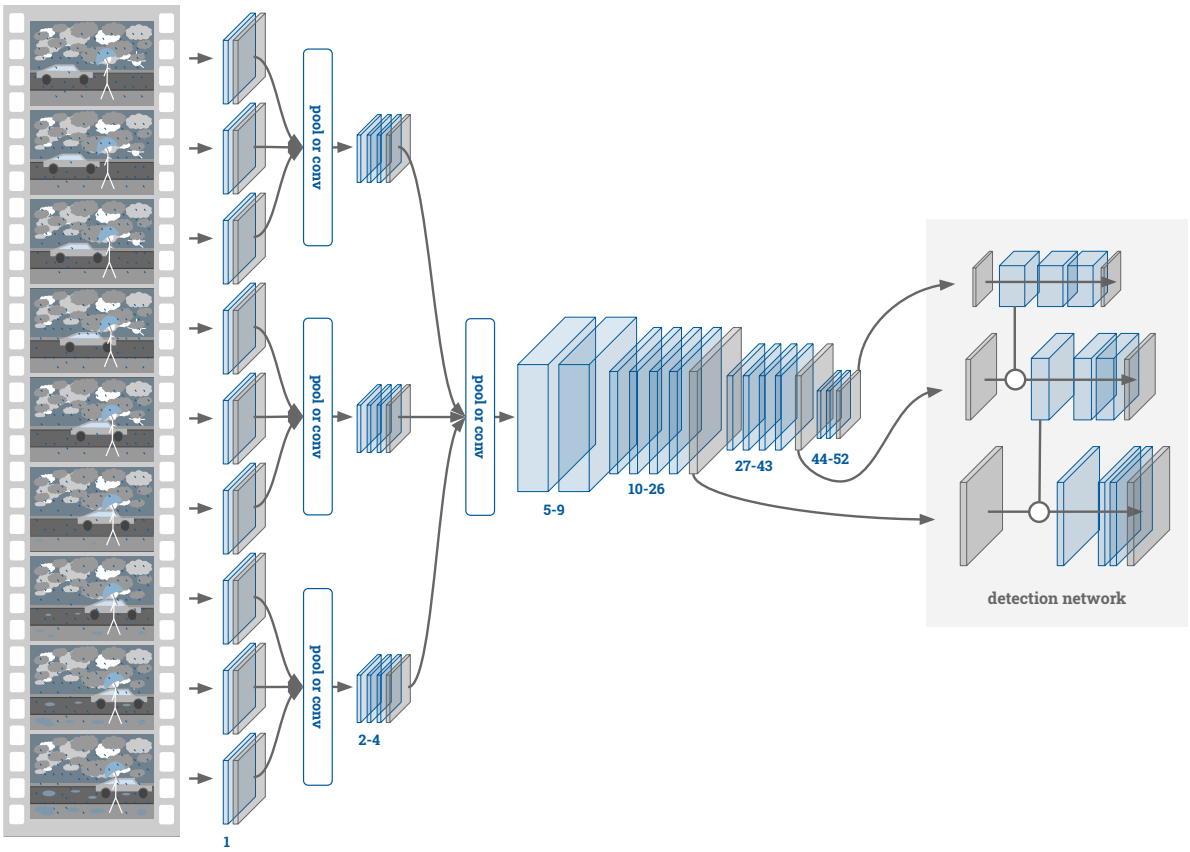


Figure 2.11: A visual representation of the proposed hierarchical DarkNet. This shows the hierarchical DarkNet **up to layer 5**, utilising a window of $W = 9$. After each stage of the DarkNet network we perform temporal max pooling or a 1D temporal convolution to reduce the temporal dimension by a factor of three.

Table 2.18 presents the results for window sizes of $W = 3$ and $W = 9$. One of the downsides of this approach is that as we deepen the hierarchical structure into the DarkNet, we require a larger initial temporal window size. Although theoretically window sizes of $W = 27$ and $W = 81$ are possible, they resulted in the models being too large to fit in memory[3] and are omitted. This highlights one of the challenges of working with video with such networks — current hardware limitations prevent many frames being able to be learnt from at once, resulting in small and or strided windows for low-level frame processing. One of the benefits of this approach however, is the small number of extra parameters that are added to the model via utilising a hierarchical structure. The results show that these modifications cause worsened mAP scores, particularly for the models with two merging operations ($W = 9$) compared to the models with one ($W = 3$). This result suggests that the pooling and convolution operations are too disruptive to the information flow in the pre-trained DarkNet model.

Type	H. upto	Window		mAP							# Parameters	
	Layer #	Size	Str.	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	Learnable	Static
max	2	3	1	53.3	47.2	26.4	14.1	34.8	56.5	**44.8**	61679899	150930
	5	9	1	48.7	41.9	19.9	12.2	31.8	50.7	**40.1**	61679899	150930
conv	2	3	1	52.8	48.2	26.0	14.2	35.5	57.3	**45.8**	61680059	150994
	5	9	1	44.5	39.8	22.4	14.0	30.1	47.1	**38.0**	61680379	151122
baseline		1	1	52.9	47.7	29.5	14.8	36.2	57.7	46.1	61679899	150930

Table 2.18: mAP evaluation for the hierarchical DarkNet architecture. On the left we show the **type of temporal merging operation**, the particular DarkNet layer that the **temporal merging follows**, as well as the window **size** and **stride**. On the right we again show the number of learnable and static parameters. As the DarkNet is unfrozen for the addition of the temporal merging operations, we utilise the unfrozen DarkNet single frame baseline. We find no benefits to this approach with the majority of mAP scores dropping in comparison to the baseline.

Hierarchical 3D with Residuals

As seen in the last subsection, altering the input features that pass through the different stages of the ImageNet pre-trained DarkNet can be too disruptive to the information flow during training and testing. To try to alleviate this, we also investigate a similar hierarchical style model, but one that uses residual connections to allow for slow and less disruptive modification to the underlying DarkNet parameters. Furthermore, we utilise 3D convolutions in place of the 2D ones, and only use them to enrich the main three feature outputs $\boldsymbol{Z}''^{(1)}$, $\boldsymbol{Z}''^{(2)}$, $\boldsymbol{Z}''^{(3)}$. More specifically, as shown in Figure 2.12, the $\boldsymbol{Z}''^{(1)}$ is taken as is, however $\boldsymbol{Z}''^{(2)}$ incorporates information from the two adjacent frames in the window, while $\boldsymbol{Z}''^{(3)}$ incorporates information from the entire window of $W = 5$ frames.

Table 2.19 presents the mAP results for varying temporal strides of input frames, for both the learnable and frozen DarkNet layers. It can be seen that there are relatively good improvements

[3] Two Nvidia K80s with combined memory of 48GB

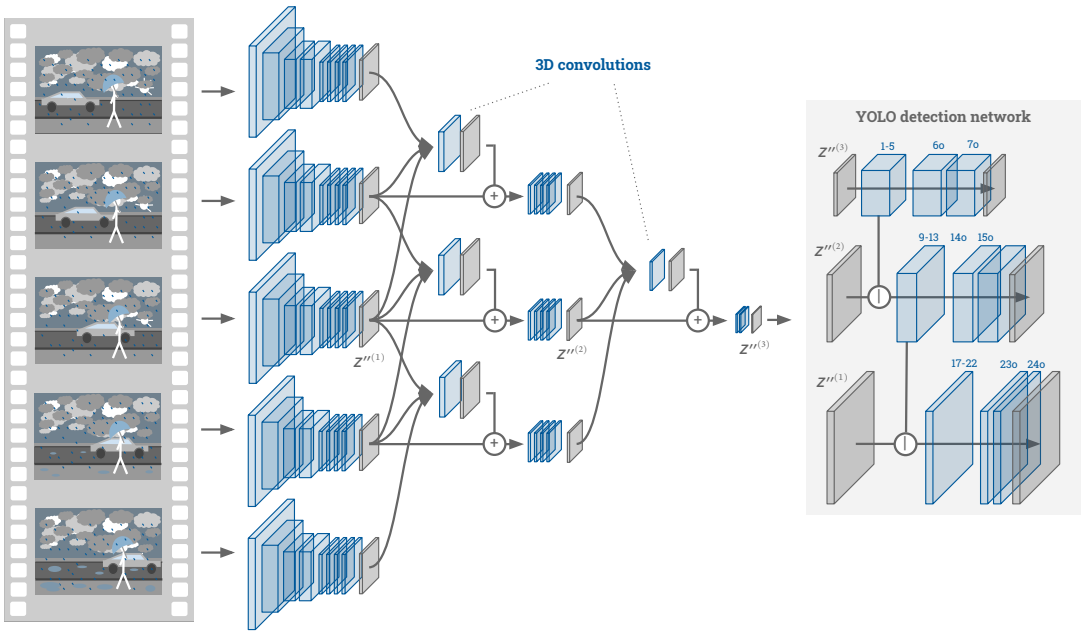


Figure 2.12: Visual representation of the 3D with residuals hierarchical DarkNet architecture. Utilising a window size of $W = 5$, all frames are passed through the first stage of the DarkNet model resulting in five features $\boldsymbol{Z''}^{(1)}_{t-2s}, \ldots, \boldsymbol{Z''}^{(1)}_{t+2s}$. The central feature (the reference frame feature) is utilised as input to the detection network for the first scale, while the others are utilised in 3D convolutions to enrich features $\boldsymbol{Z''}^{(2)}_{t-s}$, $\boldsymbol{Z''}^{(2)}_{t}$, $\boldsymbol{Z''}^{(2)}_{t+s}$ and $\boldsymbol{X}^{(3)}_{t}$. We use residual connections (+) to incorporate the temporally adjusted intermediate DarkNet features into the *untouched* DarkNet features.

for all strides when the DarkNet layers are frozen. This result is particularly interesting since the single 3D convolutional layers with additive connections are improving the input features to each stage of the frozen DarkNet. This shows that the 3D convolutions are able to learn how to develop temporally enriched features for direct use in the un-tuned DarkNet. In fact, considering the unfrozen DarkNet experiments, we find results are more variable and poor relative to the unfrozen baseline. This is suggestive that the fine-tuning that the DarkNet receives during training may be pushing them towards handling poorer-augmented features during the training process, affecting the DarkNet layers in a negative way by the end of training.

2.2.4 Class Agnostic Evaluations & Summary

While the goal of object detection is to both detect and classify an object, there may also be some benefit for some applications to just detect whether any object is present at all. Furthermore, when we consider video in comparison to images, it is often easier to determine the existence of an object, since we have multiple frames, but harder to determine class, due to blur and video artifacts. We investigate the performance of a class agnostic detector where only the objectness score $\hat{o}$ is considered, disregarding any specific class label predictions in evaluation.

Table 2.20 presents a summary of the best methods from each of the experiments in Section 2.2.2 and Section 2.2.3, along with their corresponding agnostic evaluations. Overall we see general minor improvements for the standard (non-agnostic) experiments (between 0-4

Frozen	Window		mAP							# Parameters	
	Size	Stride	AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	Learnable	Static
✓	**5**	**1**	53.7	45.8	26.8	13.0	34.5	56.9	44.9	26014779	40740466
	5	**3**	55.7	46.7	26.5	11.6	34.0	59.2	**46.0**		
	5	**10**	54.6	46.9	26.1	12.6	34.0	57.9	45.8		
	5	**25**	55.0	45.5	26.8	12.4	34.5	57.5	45.4		
baseline (✓)	**1**	**1**	51.5	42.0	23.3	12.0	31.4	53.8	42.0	21094971	40735858
✘	**5**	**1**	55.2	49.3	31.6	15.6	37.2	59.3	**47.9**	66599707	155538
	5	**3**	48.2	46.6	28.6	13.5	34.7	53.6	43.6		
	5	**10**	50.3	47.2	27.0	14.0	35.0	55.0	44.2		
	5	**25**	52.9	49.5	29.9	15.4	36.7	57.7	46.6		
baseline (✘)	**1**	**1**	52.9	47.7	29.5	14.8	36.2	57.7	46.1	61679899	150930

Table 2.19: mAP evaluation for the 3D with residuals hierarchical DarkNet architecture. On the left we present **whether the DarkNet model is frozen or not**, as well as the window **size** and **stride**. On the right we show the learnable and static parameters. Considering the frozen DarkNet results we can see improvements across practically all measures no matter the stride we used. With an average mAP improvement of 4 points compared to the baseline, in the best case of $s = 3$. In comparison, the un-frozen and fine-tuned DarkNet model results are more variable with two of the four performing better than the baseline Interestingly for strides of $s = 3$ and $s = 10$, the fine-tuning of the DarkNet results in worsened performance.

mAP point improvement), with more mixed and subdued results for the agnostic evaluations (between 0-2.3 mAP point improvement). Our hierarchical 3D model achieves the highest mAP scores in the non-agnostic case, attaining a 4 point (9.5%) and 1.8 point (3.9%) relative improvement on the framewise baseline with frozen and unfrozen DarkNet layers respectively.

Model	Table	Frz.	Agn.	mAP							Dif.
				AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	AP
Framewise	Table 2.10	✓	✘	51.5	42.0	23.3	12.0	31.4	53.8	42.0	
			✓	71.0	60.0	44.6	23.5	55.0	71.7	60.3	
Max Pooling	Table 2.11	✓	✘	53.5	45.2	25.9	13.2	33.7	56.6	44.7	*+2.7*
			✓	73.2	62.7	46.1	25.9	57.9	73.8	62.6	*+2.3*
Ch. Concat.	Table 2.12	✓	✘	55.5	45.9	25.9	13.1	34.1	58.0	45.4	*+3.4*
			✓	74.2	61.7	45.1	26.1	57.5	73.8	62.6	*+2.3*
Feat. Corr.	Table 2.13	✓	✘	52.9	45.5	26.6	13.1	34.7	56.5	44.6	*+2.6*
			✓	73.8	62.6	45.0	25.6	57.8	73.8	62.6	*+2.3*
(2+1)D Conv.	Table 2.14	✓	✘	57.5	40.0	27.0	13.3	34.5	57.6	45.6	*+3.6*
			✓	72.8	62.4	45.2	25.4	57.3	73.5	62.3	*+2.0*
Multi-Stream	Table 2.17	✓	✘	49.7	43.6	24.5	13.7	31.2	53.9	42.1	*+0.1*
			✓	72.1	61.7	45.8	26.5	56.7	72.8	61.8	*+1.5*
Hierarchical 3D	Table 2.19	✓	✘	55.7	46.7	26.5	11.6	34.0	59.2	46.0	***+4.0***
			✓	71.6	60.5	42.3	21.0	54.3	72.6	60.2	*-0.1*
				AP_{SL}	AP_{MO}	AP_{FA}	AP_S	AP_M	AP_L	AP	AP
Framewise	Table 2.10	✘	✘	52.9	47.7	29.5	14.8	36.2	57.7	46.1	
			✓	75.8	66.4	47.6	25.7	56.0	77.4	65.5	
Mulit-Stream	Table 2.17	✘	✘	53.2	49.0	27.9	13.7	36.1	57.9	46.3	*+0.2*
			✓	75.7	67.4	49.1	26.0	59.9	78.6	66.2	*+0.7*
Hierarchical 2D	Table 2.18	✘	✘	52.8	48.2	26.0	14.2	35.5	57.3	45.8	*-0.3*
			✓	75.2	65.9	47.1	25.9	58.2	77.3	64.9	*-0.6*
Hierarchical 3D	Table 2.19	✘	✘	55.2	49.3	31.6	15.6	37.2	59.3	47.9	***+1.8***
			✓	75.5	66.1	48.6	26.2	60.1	77.4	65.4	*-0.1*

Table 2.20: Experimental mAP summaries and class agnostic evaluations. Here we take the best performing model from each of our experiments, listing their mAP results for both the standard as well as additionally for the class agnostic case where only the objectness score is considered. On the left we list the **type of experimental model**, the **table with the full results**, and whether the backbone networks are **frozen** or not, and whether the evaluation is **agnostic** or not. On the far right we list the **mAP point difference** with the single frame baseline. Overall we see mostly minor improvements in average mAP with our experiments, with less improvements for the agnostic scores. Notably the proposed hierarchical 3D model attains the most significant improvements over the single frame baseline for the non-agnostic evaluations.

2.3 Exploiting Image Data for Video Detection

A significant limitation of current video detection models and datasets is their class diversity, or lack thereof. ImageNet-VID only contains 30 unique classes[4], which can be insufficient for many applications, especially captioning if we are looking to utilise detectors to identify the broad set of concepts found in captions.

In an effort towards improving model diversity for the video object detection problem, as well as to highlight an object detection models' ability to handle more class variation, we look to exploit image detection datasets and their categories. Table 2.6, back in Section 2.1.6, showed that a framewise object detection model trained on MS-COCO performs better (52.2 mAP) than one trained on ImageNet-VID (50.7 mAP) when testing on the intersecting classes of MS-COCO and ImageNet-VID. Table 2.8 then showed that the mAP is even further improved (55.6) when fine-tuning the MS-COCO trained model on ImageNet-VID, when considering the intersecting classes. These results highlight the benefits of still image datasets in the training and initialisation of video based detection models. We therefore look to combine the main three image object detection datasets Pascal VOC, MS-COCO, and ImageNet-DET with the video object detection dataset ImageNet-VID into a single dataset with a single class structure. We then train and test a framewise model on the combined dataset, comparing it to individually trained models. Lastly, we perform a class/concept coverage analysis with the captioning datasets used in Chapter 1.

2.3.1 Combining Detection Datasets

Image detection datasets Pascal VOC, MS-COCO and ImageNet-DET consist of annotations for 20, 80 and 200 categories respectively, however many of these classes are similar or equivalent. We look to merge the datasets into a single, larger dataset, therefore we need to associate categories appropriately. To combine the datasets, we manually inspect all categories along with those in ImageNet-VID, assigning each its appropriate WordNet ID. The resulting combined dataset contains **246 unique categories**, which are presented in Table 2.21 along with the datasets they appear in.

Some of the categories can be considered subcategories of a more generalised category, for example `dining-room_table` is a type of `table`. Furthermore, there are a bunch of types of objects, such as *animals*, *food*, *vehicles* and *household items*. Based on ideas presented in [Redmon and Farhadi, 2017], where a hierarchical class tree is built for 9000 object classes, we generate a smaller hierarchical class tree for the 246 classes attained from the merging of our datasets.

The hierarchical tree is generated using the 9000 class tree introduced in [Redmon and Farhadi,

[4]YouTube-BB contains 23 classes (14 of which are in ImageNet-VID), however due to its size we leave the utilisation of YouTube-BB for further work.

Class	VID	VOC	COC	DET	Class	COC	DET	Remaining classes in DET		
airplane	✓	✓	✓	✓	remote_control	✓	✓	accordion	pot	ice_lolly
bicycle	✓	✓	✓	✓	microwave	✓	✓	ant	flute	porcupine
bird	✓	✓	✓	✓	toaster	✓	✓	armadillo	french_horn	power_drill
bus	✓	✓	✓	✓	refrigerator	✓	✓	artichoke	frog	pretzel
car	✓	✓	✓	✓	hand_blower	✓	✓	ax	frying_pan	printer
domestic_cat	✓	✓	✓	✓	truck	✓		baby_bed	goldfish	puck
dog	✓	✓	✓	✓	fireplug	✓		bagel	golf_ball	punching_bag
horse	✓	✓	✓	✓	street_sign	✓		balance_beam	golfcart	purse
motorcycle	✓	✓	✓	✓	parking_meter	✓		band_aid	guacamole	racket
sheep	✓	✓	✓	✓	giraffe	✓		banjo	guitar	ray
train	✓	✓	✓	✓	platter	✓		baseball	hair_spray	rubber_eraser
elephant	✓		✓	✓	bag	✓		basketball	hamburger	rugby_ball
bear	✓		✓	✓	necktie	✓		bathing_cap	hammer	rule
zebra	✓		✓	✓	baggage	✓		beaker	harmonica	saltshaker
antelope	✓			✓	frisbee	✓		bee	harp	sax
cattle	✓			✓	snowboard	✓		bell_pepper	cowboy_hat	scorpion
fox	✓			✓	ball	✓		binder	head_cabbage	screwdriver
giant_panda	✓			✓	sport_kite	✓		bookcase	helmet	seal
hamster	✓			✓	baseball_bat	✓		bow	hippopotamus	skunk
lion	✓			✓	baseball_glove	✓		bow_tie	horizontal_bar	snail
lizard	✓			✓	skateboard	✓		bowl	ipod	snowmobile
monkey	✓			✓	aquaplane	✓		brassiere	isopod	snowplow
rabbit	✓			✓	tennis_racket	✓		burrito	jellyfish	soap_dispenser
lesser_panda	✓			✓	wineglass	✓		butterfly	koala	soccer_ball
snake	✓			✓	glass	✓		camel	ladle	spatula
squirrel	✓			✓	fork	✓		can_opener	ladybug	starfish
tiger	✓			✓	table_knife	✓		cart	lamp	stethoscope
turtle	✓			✓	spoon	✓		cello	lemon	stove
vessel	✓			✓	bowl	✓		centipede	lipstick	strainer
whale	✓			✓	sandwich	✓		chain_saw	lobster	strawberry
chair		✓	✓	✓	broccoli	✓		chime	maillot	stretcher
person		✓	✓	✓	carrot	✓		cocktail_shaker	maraca	sunglasses
display		✓	✓	✓	doughnut	✓		coffee_maker	microphone	swimming_trunks
sofa		✓	✓	✓	trifle	✓		computer_keyboard	milk_can	swine
boat		✓	✓		bed	✓		corkscrew	miniskirt	syringe
bottle		✓	✓		toilet	✓		cream	mushroom	table
cow		✓	✓		keyboard	✓		croquet_ball	nail	tape_player
dining-room_table		✓	✓		telephone	✓		crutch	neck_brace	tennis_ball
houseplant		✓	✓		oven	✓		cucumber	oboe	tick
traffic_light			✓	✓	washbasin	✓		mug	otter	windsor_tie
bench			✓	✓	book	✓		diaper	pencil_box	trombone
backpack			✓	✓	clock	✓		digital_clock	pencil_sharpener	cornet
ski			✓	✓	vase	✓		dishwasher	perfume	unicycle
banana			✓	✓	scissors	✓		dragonfly	piano	vacuum
apple			✓	✓	teddy	✓		drum	pineapple	violin
orange			✓	✓	toothbrush	✓		dumbbell	ping-pong_ball	volleyball
hotdog			✓	✓				electric_fan	pitcher	waffle_iron
pizza			✓	✓				face_powder	plastic_bag	washer
laptop			✓	✓				fig	plate_rack	water_bottle
mouse			✓	✓				file	pomegranate	wine_bottle

Table 2.21: Classes across different detection sets. This table shows a list of the 246 manually processed classes used in our combined dataset as well as the individual datasets that contain samples for them.

2017] called **WordTree**. WordTree is based on the language database **WordNet** [Miller et al., 1990] which itself is structured as a directed graph, not a tree. To build a hierarchical tree the shortest paths from the individual visual category nodes to the root `physicalobject` node are taken. To build our hierarchical class tree we start by finding the 246 categories in WordTree and extract the sub-trees from these nodes to the root. We find all but 2 of our 246 categories were in WordTree and we manually add these extra two nodes to the tree. Nodes of extracted sub-trees that are either very visually similar to their child or parent node, or are overly ambiguous and non-visual are manually modified. For example, `vegetable.n.01` → `solanaceous_vegetable.n.01` → `pepper.n.04` → `sweet_pepper.n.02` → `bell_pepper.n.02` just becomes `vegetable.n.01` → `bell_pepper.n.02`. The sub-tree is further manually modified to account for odd groupings found in the original WordTree. For example, `punching_bag.n.02` is a sub-category of *ball.n.01*, and there are two food categories `root` → `food.n.01` and `root` → `food.n.02`. After performing some manual hierarchical re-structuring, we add an extra 39 parent classes, for a total of **285 detectable object classes across 6 hierarchical levels**. Figure 2.13 (best viewed digitally) shows our hierarchical class tree, with the *root* in the centre branching out to all of the individual leaf classes.

Having this hierarchical tree allows us to correctly combine samples from Pascal VOC, MS-COCO, ImageNet-DET and ImageNet-VID to create a combined object detection dataset. Table 2.22 presents the sample counts for each of the datasets and the combined dataset.

Dataset	# Classes	# Images	# Boxes	B per I
Pascal VOC	20	21503	62199	2.89
MS-COCO	80	122266	886729	7.25
ImageNet-DET	200	476688	534308	1.12
ImageNet-VID	30	1693473	2655058	1.57
Combined	285	2313930	4138294	1.79

Table 2.22: Individual object detection dataset counts compared to the combined dataset. Due to the high number of individual frames in the ImageNet-VID dataset most of our combined dataset is made up of samples from the video set, however we increase the number of classes from 30 to 285.

2.3.2 Training & Inference on the Combined Dataset

Training an object detection network for hierarchical class labels is not as straightforward as the standard class structure as in the hierarchical case multiple class labels exist for the same object instance. To handle this, during the training process GT samples are labelled so that all nodes from the leaf to the root are labelled **1**, while all other nodes are labelled **0**. Following YOLOv3 [Redmon and Farhadi, 2017] principles, the softmax classification used in [Redmon and Farhadi, 2018] is replaced with a sigmoid across all possible classes individually, giving each their individual probability. Training is carried out as usual, with the loss calculated using

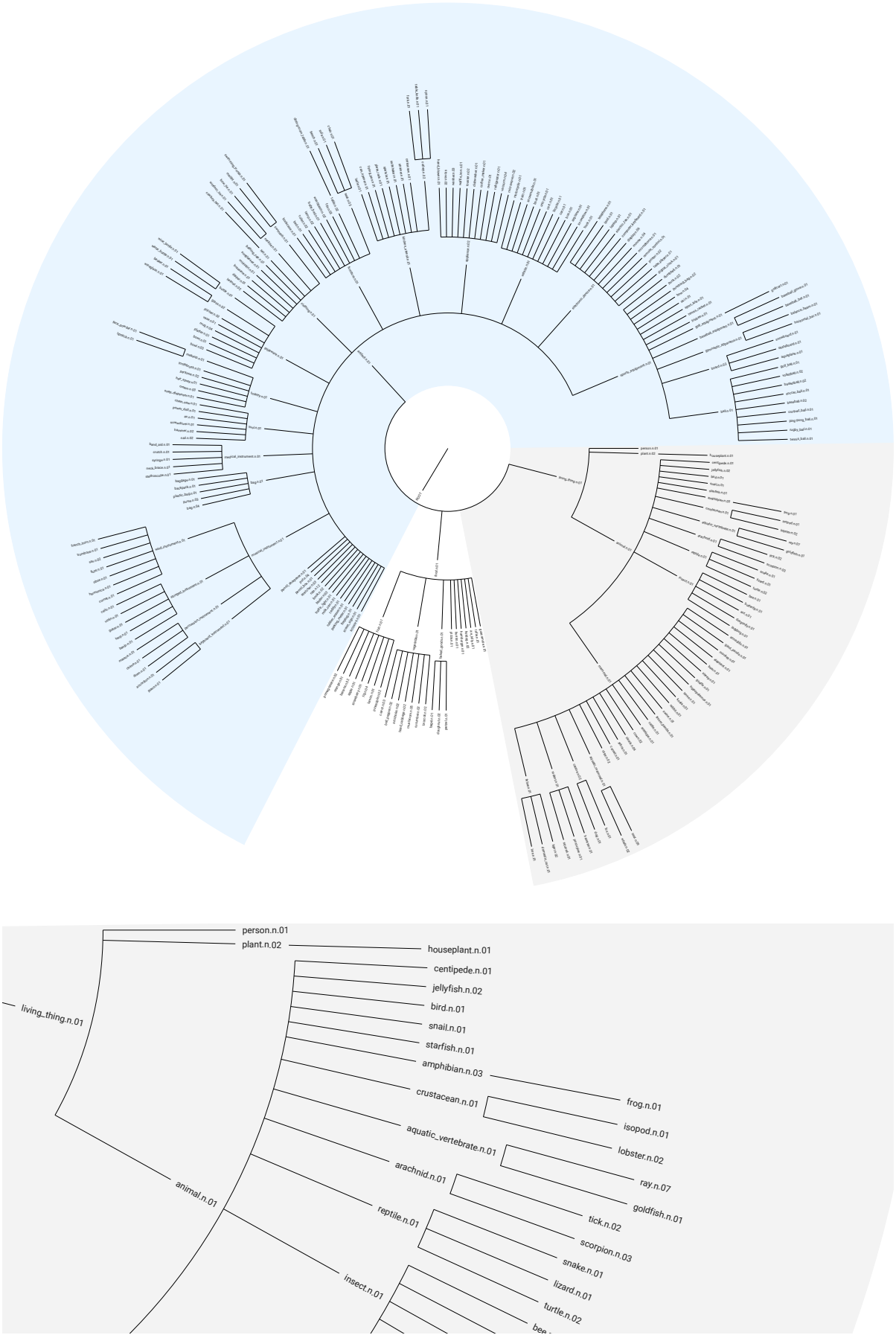


Figure 2.13: The hierarchical class tree for our combined dataset. Expanding from the central `root` class we have three child classes — `artifact.n.01` (shaded blue), `living_thing.n.01` (shaded grey), and `food.n.01` (white). Bottom is zoomed in on right side showing hierarchy of `living_thing.n.01`. ***Best viewed digitally.***

sigmoid binary cross entropy on each class and with an average taken across all classes.

Similarly, inference with hierarchical class labels is not as trivial as the single class case. For inference, the maximum score amongst all of the leaf nodes is taken, no matter their distance from the `root` node. However, if none of the leaf nodes meet the confidence threshold of 0.5, then the entire tree is pruned one level and tested again. This pruning is performed based on distance of a node from the `root`, so a leaf node that is closer to the `root` may not get pruned straight away, however if they didn't meet the threshold originally they also won't as the tree gets pruned. This *uneven* pruning is used to account for varying branch lengths, so that comparisons aren't made between overly specific and very generalised categories.

One of the benefits of a hierarchical class structure is that it allows a predictive model to be more insightful in the face of uncertainty. While no child nodes might meet the confidence threshold, their less specific parent node might. When performing evaluation on the tree, mAP is presented based on hierarchical level in order to show the impact of more lenient category labels.

Table 2.6 at the start of this chapter shows the performance of the standard framewise YOLO detector on the ImageNet-VID validation set after being trained on different datasets. Table 2.23 extends this to consider the model trained on the hierarchical combination dataset, and includes mAP results for parent categories. Notably the performance for the combined detector is relatively worse than individual dataset detectors, which could be attributed to the higher number and diversity of classes adding extra complexity that the model is unable to handle. Potentially a larger model could attain improved accuracy with the more complex data, however we leave exploration for further works.

Class	Training Set				
	Pascal VOC	MS-COCO	ImageNet-DET	ImageNet-VID	Combined
artifact					37.5
vehicle					39.0
airplane	68.1	73.0	40.9	79.7	49.9
bicycle	56.4	61.3	28.9	58.9	44.0
boat	-	-	39.6	43.5	35.8
bus	68.7	74.6	41.3	66.7	32.5
car	54.5	57.6	25.2	49.5	35.7
motorcycle	41.2	47.3	22.2	43.9	22.1
train	64.4	69.9	56.1	74.9	31.5
living_thing					51.2
animal					52.9
bird	44.3	47.2	53.5	47.9	45.8
mammal					53.1
antelope	-	-	78.0	58.5	43.4
aquatic_mammal	-	-	-	-	14.6
whale	-	-	41.7	44.9	15.0
bear	-	45.8	63.5	46.1	49.1
canine	-	-	-	-	37.8
dog	31.7	37.9	52.3	41.5	35.4
fox	-	-	52.8	48.3	25.9
cattle	-	-	49.9	45.2	30.4
elephant	-	58.1	61.7	51.6	39.5
feline	-	-	-	-	32.5
domestic_cat	38.7	40.6	43.9	31.1	42.6
lion	-	-	21.0	22.5	14.1
tiger	-	-	59.5	60.2	12.8
giant_panda	-	-	50.7	61.5	37.1
horse	49.5	63.1	55.1	59.0	52.7
lesser_panda	-	-	15.5	16.3	02.5
monkey	-	-	43.9	28.2	13.7
rabbit	-	-	59.3	45.1	34.9
rodent	-	-	-	-	12.9
hamster	-	-	58.9	57.4	09.6
squirrel	-	-	29.0	34.8	12.5
sheep	22.0	27.3	14.8	30.4	28.6
zebra	-	25.8	27.1	28.4	25.9
reptile					26.0
lizard	-	-	67.8	39.9	19.0
snake	-	-	52.9	18.8	09.5
turtle	-	-	43.4	48.9	35.1
Mean	**49.0**	**52.2**	**45.2**	**45.0**	**31.1**

Table 2.23: Varying dataset trained models evaluated on hierarchical ImageNet-VID. Shown on the left is the class hierarchical sub-tree for the 30 ImageNet-VID classes, and on the right are the individual class mAP scores for models trained on the Pascal VOC, MS-COCO, ImageNet-DET, ImageNet-VID and combined datasets respectively. We can see that the combined dataset trained model is unable to attain the scores seen in the individual models, suggesting that further investigation is needed towards how to best exploit the diverse range of data that exists in the combined dataset.

2.3.3 Detection and Captioning Concept Crossover

With the object detection pipeline being able to determine both the 30 ImageNet-VID categories and the 285 categories from the combination of the detection sets, with the idea of using detectors to improve captioning performance, it is useful to determine the coverage of these categories within the captioning sets.

ImageNet-VID Coverage

Let's firstly consider the ImageNet-VID coverage on MSVD and MSR-VTT, shown in Table 2.24. The table shows for every category the number of videos in which it appears, in terms of appearing in any of the GT captions. As direct word matching is used, some categories don't exactly appear in captions, however potentially visually similar words do, therefore for the ImageNet-VID categories some synonyms are manually added and counted. The total is made up of the union of the *exact* word matches and all of the synonym matches. The % of videos in which each category and its synonyms appear in relation to all videos in the dataset is also shown. Overall the ImageNet-VID categories are used in captions for **38.42%** of the MSVD videos and **25.38%** of the MSR-VTT videos. This results in the majority of videos in ether set not having a groundable object when a detector is trained with solely ImageNet-VID.

Combined Set Coverage

Now let's consider the coverage of the combined 285 categories introduced in Section 2.3 for the MSVD and MSR-VTT captioning sets. Table 2.25 presents a summary of the coverage statistics for the MSVD and MSR-VTT sets. The summary table contains classes that exist in at least 2% of the videos in the captioning sets, the full tables Table A.2 (MSVD) and Table A.3 (MSR-VTT) consisting of all classes can be found in Appendix E. Due to the high number of categories manual synonym specification isn't performed, however with the hierarchical nature of the 285 categories, the coverage of all children of a particular class are considered. For example, in MSVD the exact word `food` is used in 243 videos, however all of the sub-categories of the `food` class are used across 126 videos, resulting in a *food-related* class being present in 301 videos. Since only singular word matching is performed so categories that are described with two words will have no *exact* matches.

The total coverage can be seen through the `root` node, with **89.58%** and **81.16%** coverage for MSVD and MSR-VTT respectively. This coverage is significantly higher than the 30 ImageNet-VID categories of **38.42%** and **25.38%** respectively. Furthermore we find that many of the most common concepts are shared across the datasets, with the summary tables consisting of many of the same categories for both datasets. The tables also highlight the more popular categories for each captioning set, for example MSVD is very animal centric, with something animal related mentioned in 24.75% of videos, compared to just 9.77% in MSR-VTT. Lastly, it's important to note

MSVD					MSR-VTT				
Noun	Exact	Synonyms (#)	Total	% of set	Noun	Exact	Synonyms (#)	Total	% of set
dog	147	dogs (30)	151	12.58	**car**	1004	cars (375), van (58)	1063	10.63
domestic cat	0	cat (82), cats (19)	85	7.08	**dog**	259	dogs (109)	285	2.85
bird	12	chicken (42), birds (6)	55	4.58	**bird**	87	chicken (96), birds (71)	216	2.16
car	52	cars (9), van (7)	54	4.50	**domestic cat**	0	cat (192), cats (63)	205	2.05
bicycle	21	bike (38), bikes (1)	40	3.33	**bicycle**	52	bike (163), bikes (47)	180	1.80
monkey	37	monkeys (4)	37	3.08	**airplane**	123	plane (111), jet (24), planes (19)	176	1.76
horse	27	horses (8)	27	2.25	**horse**	160	horses (81)	173	1.73
airplane	13	plane (18), jet (13), planes (1)	23	1.92	**watercraft**	0	boat (134), boats (29)	140	1.40
motorcycle	19	motorcycles (2)	19	1.58	**motorcycle**	112	motorcycles (46)	124	1.24
bear	18	bears (3)	18	1.50	**bus**	78	buses (8)	83	0.83
tiger	15	tigers (1)	15	1.25	**monkey**	71	monkeys (17)	74	0.74
watercraft	0	boat (12)	12	1.00	**train**	61	trains (13)	66	0.66
lion	10	lions (2)	10	0.83	**bear**	48	bears (17)	58	0.58
rabbit	7		7	0.58	**tiger**	41	tigers (14)	43	0.43
cattle	2	cow (5), cows (3)	7	0.58	**fox**	35	foxes (2)	35	0.35
bus	7		7	0.58	**lion**	28	lions (14)	33	0.33
squirrel	5	squirrels (1)	6	0.50	**hamster**	30	hamsters (9)	32	0.32
snake	6	snakes (1)	6	0.50	**cattle**	2	cow (24), cows (9)	27	0.27
fox	6	foxes (1)	6	0.50	**snake**	24	snakes (11)	26	0.26
elephant	6	elephants (2)	6	0.50	**rabbit**	25	rabbits (4)	26	0.26
whale	5		5	0.42	**elephant**	23	elephants (7)	24	0.24
hamster	5		5	0.42	**turtle**	13	turtles (8)	15	0.15
zebra	4	zebras (3)	4	0.33	**whale**	10	whales (9)	13	0.13
train	4	trains (2)	4	0.33	**lizard**	10	lizards (1)	10	0.10
turtle	3	turtles (1)	3	0.25	**sheep**	9		9	0.09
sheep	1		1	0.08	**zebra**	6	zebras (1)	7	0.07
antelope	1		1	0.08	**squirrel**	6	squirrels (1)	6	0.06
red panda	0		0	0.00	**antelope**	6		6	0.06
lizard	0		0	0.00	**red panda**	0		0	0.00
giant panda	0		0	0.00	**giant panda**	0		0	0.00
TOTALS	**359**		**461**	**38.42**	**TOTALS**	**2054**		**2538**	**25.38**

Table 2.24: Concept overlap between ImageNet-VID and the MSVD and MSR-VTT captioning datasets. On the left we show the overlaps for the MSVD dataset, while on the right we show for the MSR-VTT dataset. A concept is counted as overlapping if it exists in any of the captions for a video, with maximally one count per video. Each table has the **class labels** on the left, with the number of **exact word matches**, then in the middle are **manually identified synonyms**, which are unioned together to form **final counts**. The final counts are then used to determine the ratios of the concepts in each of the captioning datasets. The tables are sorted by most common concepts first. Most concepts appear in very few of the samples in either MSVD or MSR-VTT, with of 38.42% and 25.38% total coverage respectively. If we were to use a detector trained on ImageNet-VID for caption groundings the majority of videos would have no object groundings.

that for both captioning sets the majority of classes appear less than 2% of the time, suggesting even with the combined dataset, some concepts couldn't be correctly grounded by an object detection model. We investigate the most common nouns in the captioning datasets which aren't associated with a combined detection set class in the following sub-section.

Missing Concepts

As seen in the previous sub-section our combined datasets classes don't completely cover all videos in the captioning datasets. Knowing the coverage of the detectable categories is only half of the picture, it's important to also determine the concepts that are utilised frequently in the captioning sets but are missing in the detectable categories. Table 2.26 presents a summary of the 100 most common concepts (nouns as determined by the NLTK) in the MSVD and MSR-VTT captioning sets, which aren't one of the detectable categories in our combined detection dataset. This table is extended to the top 204 nouns per captioning set in Table A.4 (MSVD) and Table A.5 (MSR-VTT) found in Appendix E. As has been done for the previous statistics, counting is performed per video, not per caption — if the word is in any caption for a video they are counted once for that video.

For both MSVD and MSR-VTT a significant proportion of the mission nouns are different terms for the concept of `person`, ie. *man*, *woman*, *someone*, *boy*, *girl* etc. For captioning a specific sex and age classification may result in more accurate captions, or conversely captioning with more general terminology when talking about persons. Furthermore, the words '*video*' and '*clip*' are also used in a high proportion of samples, likely due to them being used in captions to reference the video itself. Beyond these, many of the highly proportioned missing concepts aren't specific objects that can be well defined with a bounding box detector ie. *song*, *piece*, *sort*, *scene*, *front*, etc. Such concepts may be better represented by either an entire image representation or with other textual context features.

MSVD				
Class	Ext	Chd	Tot	% set
LEVEL 0				
ROOT	0	1075	1075	**89.58**
LEVEL 1				
artifact	0	669	669	55.75
food	243	126	301	25.08
living thing	0	877	877	73.08
LEVEL 2				
sports equipment	0	110	110	9.17
furniture	1	136	136	11.33
appliance	3	53	54	4.50
musical instrument	0	86	86	7.17
tableware	0	149	149	12.42
vehicle	17	118	120	10.00
kitchen utensil	0	36	36	3.00
electronic device	0	50	50	4.17
pot	69	0	69	5.75
vegetable	37	36	61	5.08
fruit	28	35	52	4.33
animal	81	282	297	24.75
person	703	0	703	58.58
LEVEL 3				
ball	51	13	53	4.42
board	46	8	49	4.08
bed	37	0	37	3.08
seat	4	50	54	4.50
table	49	0	49	4.08
stove	38	0	38	3.17
stringed instrument	0	57	57	4.75
keyboard instrument	0	27	27	2.25
glass	50	0	50	4.17
bowl	108	0	108	9.00
car	52	0	52	4.33
cutlery	0	28	28	2.33
mammal	5	255	255	21.25
LEVEL 4				
chair	26	0	26	2.17
guitar	53	0	53	4.42
piano	26	0	26	2.17
spoon	24	0	24	2.00
monkey	37	0	37	3.08
canine	1	150	150	12.50
horse	27	0	27	2.25
LEVEL 5				
dog	147	0	147	12.25

MSR-VTT				
Class	Ext	Chd	Tot	% set
LEVEL 0				
ROOT	0	8116	8116	**81.16**
LEVEL 1				
artifact	1	5522	5522	55.22
food	1102	373	1312	13.12
living thing	0	5802	5802	58.02
LEVEL 2				
clothing	202	393	569	5.69
sports equipment	0	951	951	9.51
furniture	20	1367	1375	13.75
appliance	8	242	250	2.50
musical instrument	0	358	358	3.58
toiletry	0	285	285	2.85
tableware	0	718	718	7.18
vehicle	441	1518	1553	15.53
electronic device	0	614	614	6.14
pot	230	0	230	2.30
fruit	46	235	268	2.68
animal	217	905	977	9.77
person	5272	0	5272	52.72
LEVEL 3				
hat	235	0	235	2.35
ball	553	341	674	6.74
board	224	12	231	2.31
bed	218	0	218	2.18
seat	97	385	474	4.74
table	741	0	741	7.41
stringed instrument	0	262	262	2.62
makeup	183	39	200	2.00
glass	282	6	284	2.84
bowl	392	0	392	3.92
car	1004	0	1004	10.04
microphone	265	0	265	2.65
mammal	3	716	717	7.17
LEVEL 4				
basketball	204	0	204	2.04
chair	288	0	288	2.88
guitar	250	0	250	2.50
canine	4	290	292	2.92
LEVEL 5				
dog	259	0	259	2.59

Table 2.25: Concept overlap between the combined dataset class tree with the nouns in MSVD and MSR-VTT — (summary). Here we present the most common overlaps between the classes in our combined dataset with the nouns in the MSVD (left) and MSR-VTT (right) datasets. For brevity we only show classes that exist in **at least 2%** of the captioning videos, the full tables Table A.2 and Table A.3 consisting of all classes can be found in Appendix E. We list the classes in hierarchical order starting at the `root`. For each class we show their exact word match count, the cumulative count from their children, and the union of the children and exact counts for the total and ratio %. Note this union is not the sum since a child and parent class may be in captions for the same video, which we only want to count once. Three things are important to notice in this table. Firstly, the total coverages are relatively high with 89.58 and 81.16 for the MSVD and MSR-VTT datasets respectively. Secondly, many of the most common concepts are shared across the dataset. Lastly, the majority of the classes are not listed here, meaning they are either very rare or non-existent in captioning datasets.

MSVD					
Noun	Count	% set	Noun	Count	% set
man	791	65.92	top	59	4.92
someone	500	41.67	girls	59	4.92
woman	460	38.33	movie	58	4.83
something	437	36.42	ingredients	57	4.75
lady	386	32.17	field	57	4.75
girl	266	22.17	couple	56	4.67
women	260	21.67	persons	55	4.58
men	249	20.75	instrument	54	4.50
boy	245	20.42	house	54	4.50
kichen	232	19.33	stage	53	4.42
guy	218	18.17	recipe	53	4.42
video	194	16.17	rice	50	4.17
chef	194	16.17	head	49	4.08
people	173	14.42	back	49	4.08
s	171	14.25	mixture	48	4.00
pieces	154	12.83	animals	48	4.00
playing	152	12.67	way	47	3.92
knife	133	11.08	guys	47	3.92
water	131	10.92	face	47	3.92
cook	124	10.33	egg	47	3.92
piece	121	10.08	cuts	47	3.92
dish	114	9.50	liquid	46	3.83
song	111	9.25	half	46	3.83
music	110	9.17	dance	46	3.83
pan	102	8.50	i	45	3.75
vegetables	98	8.17	grass	45	3.75
ground	97	8.08	skillet	44	3.67
kid	92	7.67	play	44	3.67
group	91	7.58	kids	44	3.67
sort	90	7.50	game	44	3.67
baby	90	7.50	camera	44	3.67
hand	89	7.42	cute	43	3.58
clip	87	7.25	area	43	3.58
road	85	7.08	frying	42	3.50
cat	82	6.83	chicken	42	3.50
slices	79	6.58	air	42	3.50
room	78	6.50	scene	41	3.42
kitchen	76	6.33	onion	41	3.42
side	74	6.17	anyone	41	3.42
front	74	6.17	film	40	3.33
kind	72	6.00	singing	39	3.25
cooking	70	5.83	onions	38	3.17
floor	69	5.75	eggs	38	3.17
meat	67	5.58	bike	38	3.17
child	67	5.58	forest	37	3.08
boys	67	5.58	mouth	36	3.00
show	65	5.42	item	36	3.00
hands	62	5.17	box	36	3.00
street	61	5.08	tricks	35	2.92
somebody	61	5.08	toy	35	2.92

MSR-VTT					
Noun	Count	% set	Noun	Count	% set
man	6707	67.07	computer	662	6.62
video	4737	47.37	road	660	6.60
people	4043	40.43	clips	654	6.54
woman	3977	39.77	water	652	6.52
someone	3035	30.35	crowd	645	6.45
show	2520	25.20	child	632	6.32
something	2487	24.87	sports	623	6.23
guy	2370	23.70	play	622	6.22
girl	2351	23.51	children	617	6.17
men	2127	21.27	kid	610	6.10
clip	2074	20.74	street	606	6.06
screen	1971	19.71	couple	593	5.93
talks	1965	19.65	players	585	5.85
group	1949	19.49	scenes	576	5.76
women	1807	18.07	dish	570	5.70
game	1758	17.58	singing	568	5.68
music	1679	16.79	animation	561	5.61
scene	1632	16.32	bunch	539	5.39
front	1561	15.61	interview	525	5.25
lady	1514	15.14	field	525	5.25
camera	1510	15.10	voice	521	5.21
s	1399	13.99	sings	513	5.13
tv	1392	13.92	pictures	509	5.09
shirt	1318	13.18	hands	500	5.00
room	1274	12.74	suit	499	4.99
song	1269	12.69	side	499	4.99
boy	1265	12.65	top	496	4.96
movie	1227	12.27	picture	488	4.88
dress	1181	11.81	persons	476	4.76
cartoon	1083	10.83	things	474	4.74
stage	1002	10.02	program	470	4.70
kids	871	8.71	place	455	4.55
characters	861	8.61	conversation	455	4.55
character	849	8.49	recipe	454	4.54
color	808	8.08	floor	450	4.50
background	808	8.08	house	445	4.45
television	789	7.89	dance	443	4.43
playing	787	7.87	match	434	4.34
footage	758	7.58	speaking	432	4.32
plays	757	7.57	home	426	4.26
audience	754	7.54	games	425	4.25
girls	749	7.49	film	421	4.21
kitchen	741	7.41	time	416	4.16
hand	724	7.24	cooking	416	4.16
hair	721	7.21	piece	413	4.13
guys	708	7.08	male	413	4.13
news	698	6.98	images	399	3.99
talk	691	6.91	gameplay	390	3.90
player	687	6.87	blue	389	3.89
ground	672	6.72	ingredients	387	3.87

Table 2.26: The 100 most common nouns from MSVD and MSR-VTT which are not part of the combined dataset class tree — (summary). Here we present the 100 most common nouns in the MSVD (left) and MSR-VTT (right) captioning datasets which are not part of our combined dataset. For brevity we only show the top 100 nouns, more comprehensive tables Table A.4 (MSVD) and Table A.5 (MSR-VTT) are found in Appendix E. We list the nouns in frequency order, presenting their count and ratio of the number of videos they appear in across the datasets. Most notably most of the missing nouns are related to different ways of saying `person`, ie. *man*, *someone*, *woman*, *people*, etc.

2.4 Summary

Following on from findings in Chapter 1 that concept detection is critically important for proficient captioning and understanding, this chapter has investigated the problem of object detection in video. As video object detection models rely on image based detection architectures we began by providing an in-depth look at the main image object detection methodologies (Section 2.1). We also discussed the datasets such models are trained on (Section 2.1.2), and the evaluation metrics used to evaluated model performance (Section 2.1.3).

We initially investigated the performance versus efficiency of three image object detection models — Faster R-CNN, SSD and YOLO (Table 2.2), choosing YOLO for our experiments due to its efficiency benefits. After taking a more comprehensive look at YOLO and its feature network — DarkNet-53 (Section 2.1.5), we carried out a framewise performance evaluation on both image detection sets as well as the video detection set — ImageNet-VID (Table 2.5). We performed a thorough analysis of the classwise performance (Table 2.6), finding that more intra-class diversity in the training data tended to result in better test performance (see Section 2.1.6). Furthermore, we found that pre-training on any of the image datasets before fine-tuning on the ImageNet-VID video dataset increases mAP scores for any and all classes (Section 2.1.7).

After establishing the framewise baselines we looked to modify the YOLO architecture to be able to consider multiple input frames when determining the detection outputs for a reference frame (Section 2.2). We began by investigating modifications to the detection part of the YOLO model, keeping the DarkNet feature network untouched (Section 2.2.2). We implemented and experimentally analysed a broad set of approaches that have seen success in combining temporal information in other video problem domains such as video classification and event detection. Such considerations included mean and max pooling (Table 2.11), channel concatenation (Table 2.12), feature correlation (Table 2.13), and 3D convolutions (Table 2.14), all of which showed performance increases of between 0-4 mAP points over the single frame baseline. These results, considered together, confirm that augmenting temporally spaced frames, in general, improves detection performance.

After experiments on the detection subnet of YOLO, we turned our attention to the DarkNet feature network (Section 2.2.3). As low-level motion features and early-network temporal modelling had been found beneficial for motion focused problems such as action recognition, we considered if performing temporal accumulation at early stages is beneficial to the video object detection problem. We experimentally investigated a number of approaches such as adding a FlowNet or R(2+1)D network as a secondary separate stream (Table 2.17), finding similar performance to the framewise baseline. We also considered hierarchical 2D convolutional (Table 2.18), and hierarchical 3D convolutional with residuals structures (Table 2.19). We found that the hierarchical 2D was too disruptive to the pre-trained DarkNet weights and information flow, leading to decreased performance. The hierarchical 3D model however attained performance improvements, likely aided by the addition of residual connections which limited the degree of change to the pre-trained model weights. Most notably, keeping the DarkNet frozen

and training convolutions to improve features for input into various stages resulted in significant improvements (10% relative improvement in Table 2.19). This finding highlights that we can learn better input features for separate stages of a frozen pre-trained network, by utilising temporally nearby features. We summarised our experiments and performed class agnostic evaluations in Table 2.20, finding that agnostic performance gains weren't as significant (0-2.5 mAP points) as the non-agnostic evaluations.

Lastly, bringing it back to the need of video object detection for the purpose of better understanding through captioning, we highlighted the lack of inter-class diversity in the video detection datasets. The video main dataset — ImageNet-VID, only contains 30 object classes which is significantly less than the 647 and 1750 nouns in our filtered MSVD and MSR-VTT vocabularies, as presented in Chapter 1. In fact, we found that the classes in ImageNet-VID are only used in captions for approximately 38% and 25% of the videos in MSVD and MSR-VTT respectively (Table 2.24). We therefore generated a hierarchical class tree and mapping for the combination of the Pascal VOC, MS-COCO, ImageNet-DET image based datasets with the ImageNet-VID dataset (Section 2.3.1). We found that coverage increases to approximately 90% and 81% for MSVD and MSR-VTT respectively (Table 2.25). We trained and evaluated a framewise YOLO network on the combined dataset finding it wasn't as effective as individually trained detectors (Table 2.23). We believe this could be due to the intra-class diversity between the individual datasets, however limited by resources we were only able to train a single model and think further optimisation of the training routine could achieve improved performance.

3 Fine-Grained Understanding

In Chapter 1 and Chapter 2 we considered video understanding in very generalised settings, where there was a wide variety of objects and actions to understand. However in the real world there is often need for models to handle much more specific circumstances (Figure 3.1).

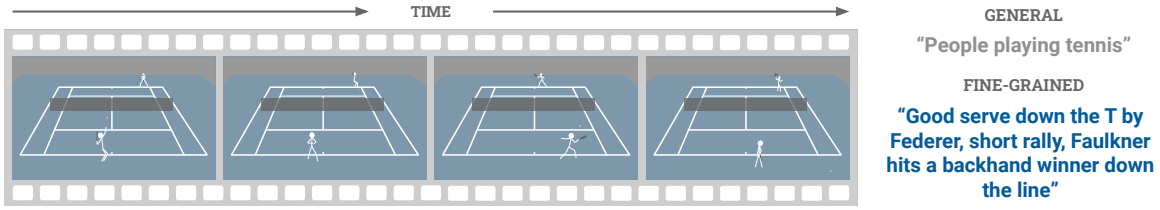


Figure 3.1: An example of generalised and fine-grained understanding. Provided a video portraying a scene, the goal is to understand it relative to the domain specific context of the video. In this example we have a video clip of a tennis play, and we want to be able to understand it in regards to the sport of tennis, so if we are to generate captions they would be more specific about the scene rather than generalistic.

One particular field of growing interest is sports analytics, where there is a desire to automate the collection of statistics using video footage. When we compare sports footage to the general video content seen in previous chapters a number of key differences are present:

- Sports footage generally takes place in a single scene, meaning we can't infer concepts from the scene representation;
- Sports footage generally contains only a few very specific objects, meaning for the most part, identifying what an object is isn't likely to aid in the understanding, rather we are much more reliant on actions and movement information;
- Sports (broadcast) footage can consist of multiple cuts with different camera angles and zooms containing a single scene or event, rather than one continuous shot for an event.

These factors mean the sports video understanding problem needs a finer tuned approach than that seen in general video understanding. Furthermore, in general approaches it's generally enough to just identify the sport, however here we want to determine the particular intricacies of the sport itself.

This chapter explores methods for performing video understanding on *fine-grained* data. We introduce a small but detailed dataset focused on the sport of tennis, which is designed for event classification, temporal detection and captioning in the form of commentary. This dataset is utilised to examine the potential of deep learning models on such a fine-grained problem.

3.1 Fine-Grained Datasets and Approaches

This section will cover some of the other fine-grained event detection and captioning datasets that exist in the literature, as well as briefly describe some of the approaches that focus on these datasets. The datasets introduced in this section are fine-grained in that they are specific to a particular domain — *eg cooking, sport, shopping*. The approaches addressing these problem domains are all different, dependent on the particular fine-grained domain, however they generally share common techniques with more general event classification, detection and captioning works as presented in Chapter 1 and Chapter 2.

3.1.1 Datasets

MPII-Cooking [Rohrbach et al., 2012a] is a dataset consisting of 44 videos of people cooking or preparing food in a kitchen. The set consists of continuous clips of a person making one of 14 dishes, ranging in length from 3 to 41 minutes. The clips are annotated with temporal events for 65 cooking activities such as *cut slices*, *pour*, or *spice*. Related to this dataset is **MPII-Composites** [Rohrbach et al., 2012b] which used Amazon Mechanical Turk (AMT) to build a text-based corpus of 2124 cooking sequences containing 12958 event descriptions. It also extends MPII-Cooking with 212 more videos for 41 composite dish creations. MPII-Cooking and MPII-Composites are combined in **MPII-Cooking 2** [Rohrbach et al., 2016], which contains a total of 273 videos of 59 different dish options. **TACoS** [Regneri et al., 2013] use AMT to generate temporally aligned captions for the MPII-Composites dataset. They only focus on 127 of the 212 videos, with 20 different descriptions each leading to 2540 captions in total. This dataset was extended to **TACoS-MultiLevel (TACoS-ML)** [Rohrbach et al., 2014] which annotates the videos with captions at three levels of granularity — a detailed description with at most 15 sentences, a short description with 3-5 sentences, and a single sentence.

The **MERL Shopping** dataset [Singh et al., 2016] consists of 96 videos each 2 minutes long, of people shopping from a set of shelves, shot from a static overhead camera. There are 5 action classes — *reach to shelf*, *retract from shelf*, *hand in shelf*, *inspect product*, and *inspect shelf*. Although the number of videos in this dataset is less than in MPII-Cooking 2, there are many action instances per video resulting in a large number frames (approximately 30k) per class.

Most similarly to our dataset, the **NCAA Basketball** [Ramanathan et al., 2016] dataset is focused on the sport of basketball and consists of 257 game videos that are each an average of 90 minutes long. The videos are annotated with temporal action localisations for 11 classes such as *3-point success*, *3-point fail*, *steal*. The set is split into 11436 training, 856 validation and 2256 test events, each of which has one of the 11 class labels. Furthermore in 850 event sub-clips of the test set their exists spatial ball localisations, as well as individual player localisations for 9000 frames in the test set.

Dataset	Domain	# Classes	# Videos	# Events	Spatial?	Temporal?	Captions?
MPII-Cooking	Cooking	65	44			✓	
MPII-Composites	Cooking	41	212			✓	
MPII-Cooking 2	Cooking	59	273			✓	
TACoS	Cooking	41	212			✓	✓
MERL Shopping	Shopping	5	96			✓	
NCAA Basketball	Basketball	11	257	14548	✓	✓	

Table 3.1: Summary of fine-grained datasets. This table summarises some statistics and properties of the fine-grained datasets. For each dataset we show its domain, the number of event classes, the number of videos, and the number of events, and whether there are spatial, temporal, or caption annotations.

3.1.2 Approaches

As aforementioned, the particular approaches used for fine-grained tasks can often be relatively tailored to their particular problem domain. In this regard we will only discuss a handfull of approaches related to some of the datasets described above in Section 3.1.1.

[Rohrbach et al., 2016] work to recognise cooking activities in MPII-Cooking 2. They explore methods for performing hand detection and pose estimation with the use of a number of hand crafted features — HoG, HoF, DT, SIFT, MbH.

[Ramanathan et al., 2016] focus on recognising basketball game events from the NCAA Basketball dataset, and the relation of particular players to the events. They utilise a CNN-based multi-box detector [Szegedy et al., 2013] to detect players, a KLT tracker [Veenman et al., 2001] to track the player detections, and a bidirectional LSTM RNN to represent track features. They then learn a temporal attention model to combine track features at each time-step before using another bidirectional RNN to perform event detection.

[Singh et al., 2016] look to also recognise cooking activities in the MPII-Cooking 2 and MERL Shopping datasets. They employ a two stream CNN approach that is applied to raw images and pixel trajectory images. They apply this CNN to the full frames, and a box cropped around the person in each clip. They then utilise a bidirectional LSTM RNN to predict action labels.

3.2 A Tennis Dataset

This section provides the details of the tennis dataset. Extending upon the reasons laid out in the chapter introduction, the particular sport of tennis was chosen for a number of reasons:

- Compared to some other sports, the actions in tennis are more discrete and easier to temporally annotate;
- Such discreteness also lends itself to caption annotation, where it is clearly beneficial to caption at the `point` level of granularity;
- Camera movement and different angles are kept to a minimum during live play, which is one less hurdle to overcome, and also distills the models focus to the intricacies that define events;
- Actions are short and dense in time, with event boundaries often occurring only a few frames apart, meaning an action detection model has a very small window of time, and hence very few frames, to make a decision;
- It is one of the bigger and more commercially viable sports in Australia, with this work having potential commercial outcomes.

If you are unfamiliar with the sport of tennis we suggest you familiarise yourself before reading further, however it is basically just two people, each hitting a ball with a racket over a net stretched across a court between them. A match is structured into components — a **match** has a number of **sets** which are themselves made up of a number of **games**, which are each made up of a number of **points**, which are what each player wins or loses. When a player wins enough points they win the **game**, when they win enough games they win the **set**, with **matches** being determined as the best of 3 sets (or best of 5 depending on tournament). Each game has a *server*, which is how a point is commenced — the server **serves** the ball into play. Following the serve, the players exchange hits, when a player hits the ball on the dominant side of their body (based on their handedness) it's called a **forehand**, and when on their less dominant side it's called a **backhand**.

Our dataset is composed of event annotations for five singles (two player) tennis matches and contains fine-grained hierarchical event annotations as well as `point` level commentary annotations. We make our dataset publicly available at hayden.faulkner.codes/tennis[1].

3.2.1 Event Annotation

Event annotations are of various granularities — from the match level all the way down to the serve and hit level. For each event type we also label the events with particular related attributes. To annotate the videos we developed a video temporal event annotation tool, which is general

[1] https://hayden.faulkner.codes/tennis

enough to be used for other event detection markups and video domains. The tool is made available for free as part of the code for this thesis — see my GitHub[2].

Table 3.2 presents count and length statistics for each event type, and shows the attributes that are annotated for each event. For the `serve` and `hit` classes we include an attribute *near* or *far* to indicate which end of the court the particular serve or hit occurred at. As the main camera remains static at one end of the court during play, the *near* and *far* annotations are in relation to the main camera position. Similarly for the `hit` class the attributes *left* and *right* are specified to indicate what side of the body a ball is hit. Once again this is relative to the main camera, allowing for the visual generalisation of forehands and backhands from players with different handedness. For example, a forehand can occur on the left-side or the right-side of a player depending on their handedness, so to a visual identification model the concept of a forehand is much more ambiguous than the visually distinct and consistent left-side or right-side.

Type	Attributes	# Events	# Frames	Avg. Frames per Event
match	winner	5	786455	157291
set	winner, score	11	765738	69613
game	winner, score, server	118	588759	4989
point	winner, score	746	159494	214
serve	near or far, in or fault or let	1017	68385	67
hit	near or far, left or right	2551	73564	29

Table 3.2: Tennis dataset annotation statistics. We annotate six event types, each having a set of particular attributes also annotated and attributed to each event instance. We show the number of event instances, the number of frames, and the average frames per instance for each event type. Keep in mind that these events are hierarchical — ie. a `match` event instance is composed of many `point` instances.

We use the attributes for the `serve` and `hit` annotations to construct a set of 10 shot classes, plus an other class (Figure 3.2). These are formed in a hierarchical manner, with the potential for individual evaluation on each level, allowing a greater understanding of a model's ability to handle different levels of specificity.

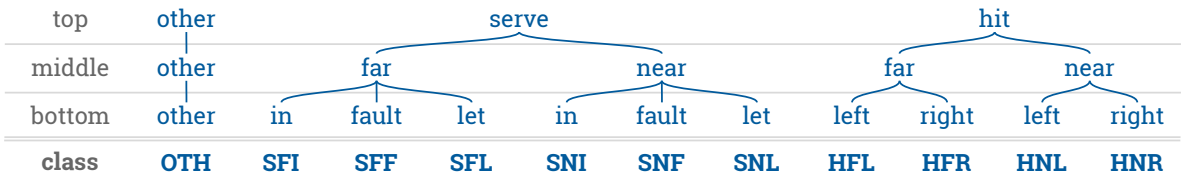


Figure 3.2: The class hierarchy for the tennis dataset for shot classification. Both the hits and serves are marked as either being carried out by the *near* or *far* player in relation to the camera. Furthermore, the hits are marked as occurring on the *left* or *right* hand side of a players body (in relation to the camera), while the serves are marked as either *in*, *let* or *fault*. With these combinations we specify 10 event classes as well as an `other` class. These 11 classes can also be considered as a three-level class hierarchically, with the top-level classes — `other`, `serve` and `hit`

[2] https://github.com/HaydenFaulkner/TemporalEventAnnotator

The dataset is split into training, validation and testing subsets — proportions are presented visually in Figure 3.3 and further detailed in Table 3.3. The dataset is partitioned in two ways — either by using four of the five videos as the training set, and split the remaining video across the validation and test set (split 01); or by splitting all five videos across all three splits (split 02). The former option is useful to determine how well a model can generalise to an unseen match, while the latter split methodology is useful for approximating the performance of a model as if it has access to enough data to generalise better. For both splits, there exists a significant class imbalance, with the *background* `other` class being 25-30 times more frequent than the next most common occurring class. Furthermore, the *let* classes (`SFL` and `SNL`) are incredibly rare, occurring approximately 1/300th as often as the `other` class. Correctly identifying the rare *let* classes correctly will likely be a highly challenging task for standard data driven methods to overcome. For the experiments carried out herein (Section 3.3), split 02 is utilised as it has *let* samples across each split, and represents the generalisation scenario as if more data was available.

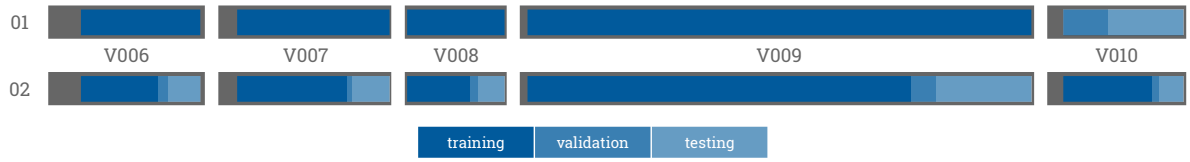


Figure 3.3: Visualisation of the tennis dataset splits. For each split (01 and 02) we show the length of each of the 5 videos with grey bars, with the individual splits shown as varying shades of blue. As the videos contain footage prior to the commencement of play, some frames at either ends of the videos aren't utilised in any split. For split 01 four videos are assigned exclusively to the training set, while keeping one solely for validation and testing. Split 2 on the other hand includes frames from all matches in all three splits.

3.2.2 Commentary Annotation

Beyond temporal event annotation, the dataset is also annotated with commentary style captions for each of the 746 `point` events. Commentary was initially scraped from the internet (tennisearth.com[3]), however after manual inspection it was found that 582 (76%) of the captions were either missing or incorrect. For example, in one case the scraped commentary read '*High kick serve, Federer returns a forehand return, short rally, Nadal cross-court backhand lands out-side the court*', however the point was actually just a double fault, so was revised to '*Double Fault*'. The structure and style of the revised language was kept as similar to the non-revised as possible.

To generalise the commentary across games and matches *player names* are replaced with `np` and `fp` for near and far player respectively. The terms *forehand* and *backhand* are also replaced with either `ls` (left shot) or `rs` (right shot) depending on a player's handedness and court position. Similarly as was done for the shot annotations, the change to `ls` and `rs` is done for visual interpretation generalisation, and ambiguity prevention with different handedness players.

[3] http://tennisearth.com/

Class	# Events - Split 01			# Frames - Split 01			# Events - Split 02			# Frames - Split 02		
	Train	Val	Test	Train	Val	Test	Train	Val	Test	Train	Val	Test
OTH	2507	133	198	573394	28538	49648	2079	160	608	470963	36932	143685
SFI	342	11	29	20114	772	1925	296	22	64	17716	1402	3693
SFF	117	2	5	7962	153	333	95	7	22	6430	577	1441
SFL	25	0	1	1596	0	72	21	1	4	1380	38	250
SNI	293	24	29	17186	1762	1994	242	18	86	14876	992	5074
SNF	111	7	10	7312	578	772	88	8	32	6020	473	2169
SNL	10	2	0	656	126	0	9	1	2	543	65	174
HFL	533	22	45	16520	648	1419	432	33	135	13530	1037	4020
HFR	576	39	41	16858	1096	1150	474	37	145	13878	1037	4189
HNL	602	29	39	16196	811	1076	514	37	119	13879	1036	3168
HNR	546	31	48	15605	882	1303	448	33	144	12686	920	4184

Table 3.3: Tennis dataset class split statistics. Here we show the event and frame counts per class for each of the two splits. Considering the most frequent class `other` and the most infrequent class (`SNL`) we can see no matter the split there are significant class imbalances in our dataset.

All captions are filtered to only contain lowercase alpha characters (a-z), spaces, mid-word hyphens and mid-word single apostrophes. Figure 3.4 presents a word frequency graph and Table 3.4 presents some diverse examples. Beyond the 746 `point` grounded captions, we also download and process another 11 thousand tennis commentary captions, which grow the vocabulary from 223 to 250. While these extra captions aren't manually verified and have no corresponding videos, they can be useful for learning textual relationships and improving captioning performance.

Point ID	Caption
P00000001	*'high kick serve fp returns a ls return short rally fp cross-court rs lands out-side the court'*
P00000012	*'quick serve is an ace'*
P00000036	*'np serves down the t fp returns a ls return brief rally np fails to keep a cross-court ls in the play'*
P00000051	*'np hits a good serve fp struggles with it returning it long'*
P00000155	*'cannon serve down the t is an ace'*
P00000172	*'sharp angled slice serve np returns a rs return fp whips a rs cross-court winner'*

Table 3.4: Tennis dataset GT caption examples. Presented are six examples of captions taken from our GT caption set. You can see their terminology is very specific to tennis, with them focusing on the order of specific events, ie. hits, serves, rallies, etc.

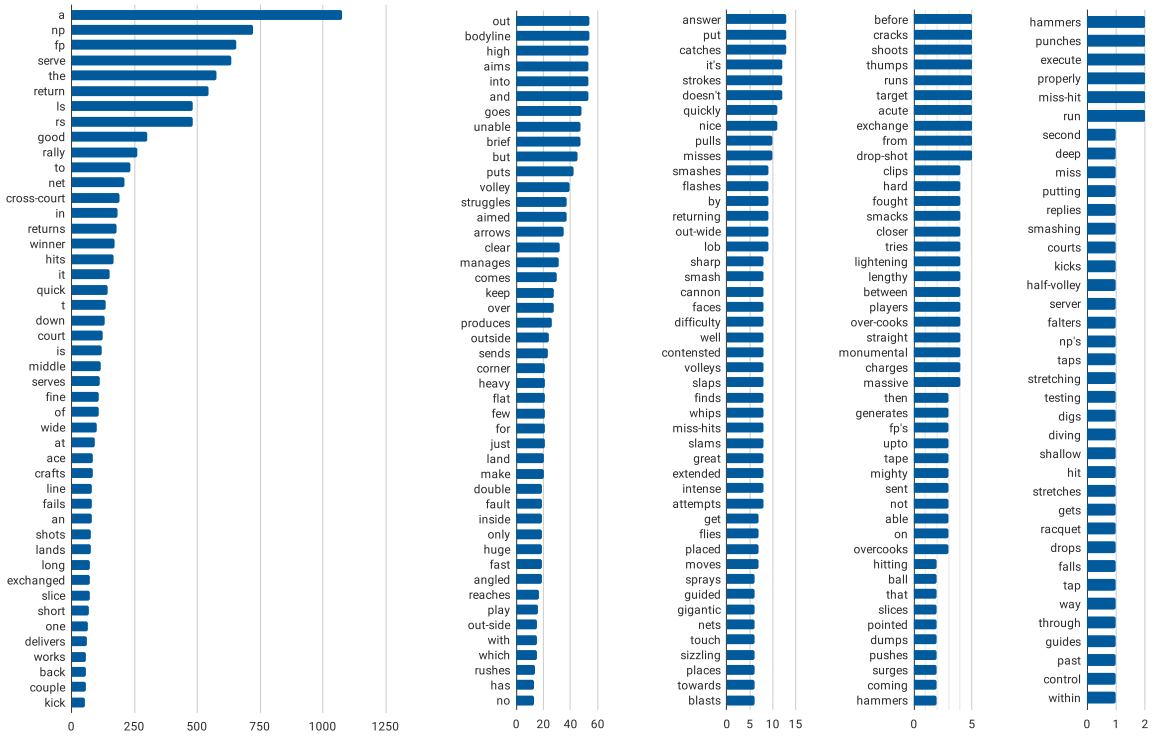


Figure 3.4: Word frequencies of the tennis datasets caption training data. As is common with vocabularies our word frequencies result in a long-tailed distribution with most words appearing infrequently. Most common to least common words are listed from top-left to bottom-right, note the decreasing scales along the individual x-axes.

3.3 A Tennis Event Detector and Commentary Model

In this section we implement and experimentally analyse numerous neural network based architectures for performing **temporal event detection** (Section 3.3.1) and **commentary generation** (Section 3.3.2) on our tennis dataset.

3.3.1 Temporal Event Detection

We investigate the event detection problem first, as this is the more fundamental lower level problem, with the caption generation model utilising a pre-trained event detection model.

Frame-wise classification with a CNN

The most straightforward approach to event detection is performing frame-wise classification, where subsequent frames of the same class can be grouped as an event. As there are a significant number of `OTH` frames in comparison to the other ten classes, we reduce the number `OTH` frames using random sampling to contain only as many as the next most prevalent class (`SFI`). During evaluation we also separate the results of the `OTH` (background) and non-`OTH` classes.

Evaluation is carried out on a per frame basis using precision, recall, F1-score and the cross-class accuracy (`OTH` GT samples ignored) for the non- `OTH` (non-background) classes, as well as the accuracy for the `OTH` class. Recalling from Section 2.1.3, the precision and recall for a class are calculated as:

$$p = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad r = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.1}$$

where TP, FP, and FN are the number of true positives, false positives, and false negatives respectively. The F1-score per class is calculated as:

$$\begin{aligned} F_1 &= 2\frac{p \cdot r}{p + r} \\ &= \frac{\text{TP}}{\text{TP} + .5(\text{FP} + \text{FN})} \end{aligned} \tag{3.2}$$

where FN is the number of false negatives. Accuracy is calculated as:

$$a = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.3}$$

In the context of frame classification, say for example if we consider the serve-far-in (`SFI`) class, a TP would be when the model correctly predicts a `SFI` frame, a TN would be when it correctly predicts the frame isn't `SFI` , a FP is when the model predicts the frame is `SFI` but it isn't, and FN is when the model predicts some other class when it should have predicted `SFI` .

We begin by training and testing a number of different Convolutional Neural Network (CNN) architectures of different sizes to determine their *out-of-the-box* effectiveness on the tennis dataset. Specifically, we experiment with various ResNet sizes [He et al., 2016], a DenseNet [Huang et al., 2017], and two MobileNet sizes [Howard et al., 2017]. Table 3.5 presents the results using the evaluation metrics aforementioned, and also includes each architectures learnable and static parameter counts. We find that the ResNet models benefit from their larger sizes in comparison to the smaller MobileNet architectures. The DenseNet, despite significantly less parameters, achieves competitive performance with the larger ResNet models. For this reason we decide to utilise the DenseNet-121 architecture as the main classification CNN for upcoming experiments.

Figure 3.5 presents a visualisation of the DenseNet-121 model which consists of four *dense blocks* and three *transition blocks*. The dense blocks each consist of multiple [*convolution, BN, ReLU*] triplets, with outputs of earlier triplets being concatenated together to form the inputs for succeeding triplets. The transition blocks each consist of a single [*BN, ReLU, convolution, average pool*] set of operations, using the average pooling to half the spatial feature dimensions. Table 3.6 presents the architecture in more detail.

Table 3.7 presents the class-wise results from the DenseNet-121 model from Table 3.5. Included is the balance of test samples per class, noting that there are very low proportions of `SNL` and `SFL` samples in the test set. It can be seen the worst performing classes are the *fault* and *let* classes (`SFF` , `SFL` , `SNF` , `SNL`), which is likely a result of the visual similarity between the

CNN	Precision	Recall	F1-Score	Accuracy		# Parameters	
				not OTH	OTH	Learnable	Static
ResNet-18	53.6	47.2	49.5	65.2	94.5	11180491	7948
ResNet-34	56.1	48.6	51.3	71.4	94.1	21288651	15372
ResNet-50	53.8	48.5	50.4	68.1	95.8	23523019	45580
ResNet-101	53.0	50.4	51.0	66.7	96.4	42515147	97804
ResNet-152	56.0	57.1	53.6	70.3	95.8	58158795	143884
DenseNet-121	54.0	53.2	**52.4**	**67.2**	96.2	**6998923**	83648
MobileNet3-Small	46.5	46.5	45.8	56.4	95.8	1678379	12112
MobileNet3-Large	50.3	45.4	47.0	59.6	94.9	4214843	24400

Table 3.5: Frame-wise evaluations comparing various CNN classification architectures. For each CNN we present the average class precision, recall, F1-scores, as well as the accuracy of the non- `OTH`, and the `OTH` classes separately. We also present the number of learnable and static parameters for each model. Results are mixed, however comparing ResNet models to the MobileNet models, we can see the ResNets benefit from their extra parameters. The DenseNet architecture achieves a relatively high F1-score (second best behind ResNet-152), despite having a significantly smaller model size.

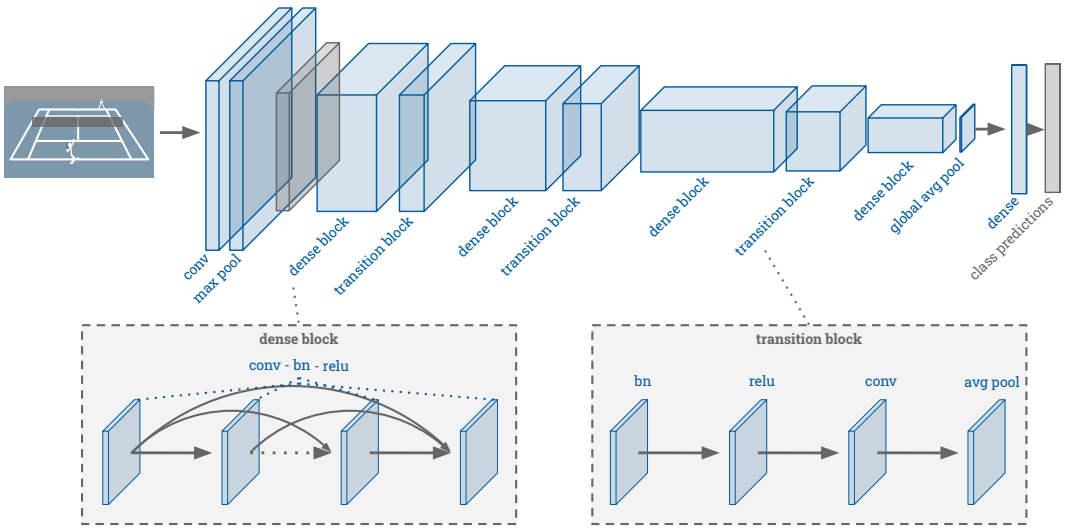


Figure 3.5: Overview of the DenseNet-121 architecture. DenseNet-121 is made up of four pairs of **dense blocks** followed by **transition blocks**. Each dense block consists of multiple sets of [*convolution, BN, ReLU*] triplets. The outputs from each of these triplets are concatenated to the inputs of the succeeding triplets within the block. Such a design allows for a more direct connection between output and input, minimising problems related to exploding or vanishing gradients for very deep networks. As to allow for concatenation within the dense blocks the spatial feature sizes remain constant within the blocks. The transition block, which is made up of the [*BN, ReLU, convolution, average pool*] set of operations, uses its average pooling layer to do a spatial 2×2 average, halving the feature map size. At the end of the network a spatially global average pooling is taken, followed by a dense layer and softmax operation to generate class probabilities.

Block	Repeats	Layer Type	# Filters	Size / Stride	Output Size (w,h)	Layer #
		Convolutional	64	7×7 / 2	1 / 2	1
		Pooling (max)		3×3 / 2	1 / 4	
Dense	6×	Convolutional	128	1×1 / 1		2, ..., 12
		Convolutional	32	3×3 / 1		3, ..., 13
Transition		Convolutional	128	1×1 / 1	1 / 4	14
		Pooling (avg)		2×2 / 2	1 / 8	
Dense	12×	Convolutional	128	1×1 / 1		15, ..., 37
		Convolutional	32	3×3 / 1		16, ..., 38
Transition		Convolutional	256	1×1 / 1	1 / 8	39
		Pooling (avg)		2×2 / 2	1 / 16	
Dense	24×	Convolutional	128	1×1 / 1		40, ..., 86
		Convolutional	32	3×3 / 1		41, ..., 87
Transition		Convolutional	512	1×1 / 1	1 / 16	88
		Pooling (avg)		2×2 / 2	1 / 32	
Dense	16×	Convolutional	128	1×1 / 1		89, ..., 119
		Convolutional	32	3×3 / 1		90, ..., 120
		Pooling (avg)		Global		
		Dense	C			121
		Softmax				

Table 3.6: The details of the DenseNet-121 architecture. Here we present a more detailed specification of the DenseNet-121 architecture. We show the block types, their associated number of repetitions, and the layers within them (excluding the ReLU and BN for brevity). For each layer we present the number of filters used, the spatial size and stide of each filter, the output size of the layer in relation to the input image size, and the layer IDs. Note the final dense layer has C filters corresponding to the number of classes we are looking to classify, in our case $C = 11$.

serve `fault`, `let` and `in` classes, as well as the proportionally small unique training examples for these classes in comparison to the `in` classes. This is the main factor contributing to the 0.0 scores of the `SNL` class in our upcoming experiments (Table 3.7, Table 3.8, Table 3.9 and Table 3.10). By considering Figure 3.6 which presents the confusion matrix showing the number of labels versus predictions per class, we can see that indeed most `serve` predictions are classified as `in`.

Class	Precision	Recall	F1-Score	Accuracy	% Test Samples
OTH	96.2	97.8	97.0	96.2	83.5
SFI	61.6	54.6	57.9		2.1
SFF	19.4	16.3	17.7		0.8
SFL	8.0	35.1	13.0		**0.1**
SNI	72.4	55.7	62.9		2.9
SNF	17.4	28.5	21.6	67.2	1.3
SNL	0.0	0.0	0.0		**0.1**
HFL	79.7	70.4	74.8		2.3
HFR	72.5	80.5	76.3		2.4
HNL	77.8	77.1	77.5		1.8
HNR	88.8	69.5	78.0		2.4

Table 3.7: Class-wise scores using the DenseNet-121 architecture. Here we show the precision, recall and F1-score per class using the DenseNet-121 architecture for frame-wise classification. We also show the cross-class accuracy for the non-`OTH` and `OTH` classes separately. Furthermore, the % of test samples that each class has in the full test set is shown to highlight that some classes (`SFL`, `SNL`) are very rare.

Optical Flow Utilisation

The use of optical flow (OF) networks and two-stream approaches have been found to be helpful in video classification and detection problems [Simonyan and Zisserman, 2014, Feichtenhofer et al., 2016, Peng and Schmid, 2016], due to OF being an informative representation of motion. We utilise FlowNet-S [Dosovitskiy et al., 2015] (described back in Section 2.2.3) to extract flow frames between every consecutive frame of our dataset. We input frames full resolution ($720 \times 1280 \times 3$) into the FlowNet-S, resulting in output flow images of $704 \times 1280 \times 3$. To align these with our RGB frames we crop the RGB inputs to $704 \times 1280 \times 3$. We experiment with three different architectures:

1. Concatenating the RGB and OF frames along their channels dimension (making a $704 \times 1280 \times 6$ input frame) before using the DenseNet-121 architecture without ImageNet pre-training for classification;
2. Only using the OF frames as input into the DenseNet-121 architecture, again without ImageNet pre-training; and
3. Inputting the RGB and OF frames into separate streams, with the RGB frames passed

		Prediction										
		O	SFI	SFF	SFL	SNI	SNF	SNL	HFL	HFR	HNL	HNR
Label	O	138259	863	424	27	1301	256	3	701	400	494	957
	SFI	433	2276	891	7	0	1	0	0	0	51	34
	SFF	313	796	279	3	0	0	0	0	0	22	28
	SFL	14	151	51	20	0	0	0	0	0	8	6
	SNI	749	1	0	0	3673	574	3	42	32	0	0
	SNF	424	0	0	0	1359	378	0	0	8	0	0
	SNL	69	0	0	0	74	28	0	3	0	0	0
	HFL	334	6	6	0	69	27	1	3203	209	44	121
	HFR	365	10	0	0	106	55	0	434	3036	25	158
	HNL	162	51	38	0	1	1	0	84	39	2466	326
	HNR	194	11	25	0	15	8	0	80	47	87	3717

Figure 3.6: Confusion matrix of class-wise labels versus predictions from the DenseNet-121 architecture. For each frame in the test set, we add it to the count in the particular cell based on what the frame's label was and what class it was predicted as. For best case we would have a diagonal matrix with the class label counts along the diagonal. Cells are shaded based on the proportion of predictions per label, with darker meaning more predictions. To interpret this matrix let's consider the `SFL` class, specifically looking at the row corresponding to the `SFL` label. We can see that `SFL` frames were mostly predicted as `SFI` frames (151) or `SFF` frames (51) compared to being correctly predicted as `SFL` (20). The effect is similar for the other `let` class and also the `fault` classes, with them most predicted as the more common `in` class. The `hit` classes on the other-hand are relatively well predicted with the majority of counts aligning between the labels and predictions.

through the DenseNet-121 architecture but pre-trained on ImageNet, while the OF frames are passed through a separate DenseNet-121 which isn't pre-trained, with the output features (after the global average pooling) from each stream being concatenated before the dense classification layer.

Table 3.8 presents the class-wise F1-scores for the three methods incorporating OF. It can be seen that utilising a two-stream approach results in significant improvements for most classes. This highlights the importance of low-level motion cues for this particular domain, where determining actions based on particular movements is important. If we consider either of the other two approaches, we see decreases in performance. For the six channel input this is likely due to the RGB part of the frame not having access to the ImageNet pre-training, whereas for the OF only model the OF on its own is likely an inferior representation compared to RGB.

Temporal Pooling

Performing frame-wise classification independently per-frame disregards any temporal information, resulting in temporally subsequent frames often being classified differently. It can be beneficial to smooth the per frame classifications to make larger event chunks. The simplest way to smooth over the temporal noise is to use pooling over a sliding window. We investigate the performance of both mean and max pooling operations for varying window sizes. We don't retrain this network but instead just temporally pool on the frame-wise CNN feature outputs

Flow Utilisation	Class-wise F1-Score											
	OTH	SFI	SFF	SFL	SNI	SNF	SNL	HFL	HFR	HNL	HNR	AVG
Six Channel Input	95.7	47.5	11.0	10.8	60.3	16.9	0.0	74.6	77.7	70.1	76.1	49.1
Only Flow	94.0	41.2	5.8	7.2	52.7	10.4	0.0	60.3	61.4	58.8	71.7	42.1
Two-Stream Nets	97.2	67.4	14.6	13.4	67.0	19.4	0.0	81.8	83.5	79.0	86.2	**55.4**
None	97.0	57.9	17.7	13.0	62.9	21.6	0.0	74.8	76.3	77.5	78.0	52.4

Table 3.8: Experimental results of utilising optical flow inputs for frame-wise classification. For each of the OF utilisation methods we present the class-wise F1-scores, as well as an average score across the classes. We find decreased performance for the two single-stream approaches, however increased performance for the two-stream approach.

(post global average pooling) inputting the temporally pooled feature into the dense classification layer.

Table 3.9 presents the results, showing some improvement for both max and mean pooling depending on the particular window size. A window of 30 frames appears to be more beneficial for the `serve` classes, while a shorter window of 9-15 frames is more suitable for `hit` classes. When we consider the average length of these events, serves being an average of 67 frames long and hits 29, it makes sense that pooling windows with similar temporal ratios with the specific events would be optimal, as any longer would consider too much past or future event information concealing the desired event. This is why a large pooling window of 120 frames has negative effects, especially with max pooling where more confident class responses can overpower the central class of interest. Mean pooling appears mildly more beneficial than max pooling, suggesting voting on what a frames class should be, based on those around it, is more effective than taking the highest confidence frame. Potential extensions to this could be using a weighted window with more weight given to the central frames, or other dynamic window selection methods, but this is left for future work.

CNN-RNN

While pooling removes much of the temporal noise, it is a static operation that is highly dependent on the window size and ratio to individual event sizes. A more dynamic approach is to utilise a Recurrent Neural Network (RNN) to consider information throughout the window dynamically. We experiment with using a bi-directional Gated Recurrent Unit (GRU) RNN to encode the CNN output features with temporally contextual information, prior to a temporal max pooling operation.

Figure 3.7 presents an overview of the CNN-RNN pipeline with a window size of three. The DenseNet features (again taken after the global average pooling layer) from each time-step are passed into the GRU RNN, which has a hidden dimension of 128. The RNN then generates feature encodings for each time-step in the window, which are conditioned on the other frame

Type	Frames	Class-wise F1-Score											
		OTH	SFI	SFF	SFL	SNI	SNF	SNL	HFL	HFR	HNL	HNR	AVG
max	3	97.2	60.4	16.9	13.7	63.1	22.0	0.0	76.5	76.4	79.3	80.0	53.2
	9	97.2	62.1	15.6	15.3	63.3	22.1	0.0	77.6	75.5	80.8	81.6	53.7
	15	97.1	62.2	14.9	13.1	63.0	23.4	0.0	76.0	69.8	79.4	81.0	52.7
	30	96.2	60.2	12.7	6.6	60.2	26.0	0.0	61.4	41.2	65.6	74.4	45.9
	120	91.6	3.1	0.0	0.0	11.6	3.2	0.0	0.7	0.0	2.0	18.7	11.9
mean	3	97.1	59.0	17.6	13.5	63.5	21.6	0.0	75.8	77.4	79.0	79.2	53.1
	9	97.3	60.9	18.9	14.6	64.7	21.3	0.0	77.0	79.0	81.0	80.4	54.1
	15	97.5	62.0	19.6	14.1	65.6	21.6	0.0	77.1	78.9	81.0	80.3	54.3
	30	97.5	63.9	20.8	13.4	65.9	23.3	0.0	74.6	73.7	77.4	77.7	53.5
	120	94.7	51.3	10.2	0.0	42.7	5.9	0.0	21.9	7.3	10.3	36.6	25.5
baseline	1	97.0	57.9	17.7	13.0	62.9	21.6	0.0	74.8	76.3	77.5	78.0	52.4

Table 3.9: Experimental results of temporal pooling for smoothing frame-wise classifications. Here we present the class-wise F1-scores, as well as an average score across the classes, for the pooling techniques across varying sliding window sizes. Results are mixed, however we find that for the `serve` classes which are temporally longer, a pooling window of 30 is most beneficial, while for the shorter `hit` classes a shorter window achieves the best results.

features in the window. The features are then temporally max pooled across the window to get the feature for a single time-step which is input into the dense classification layer.

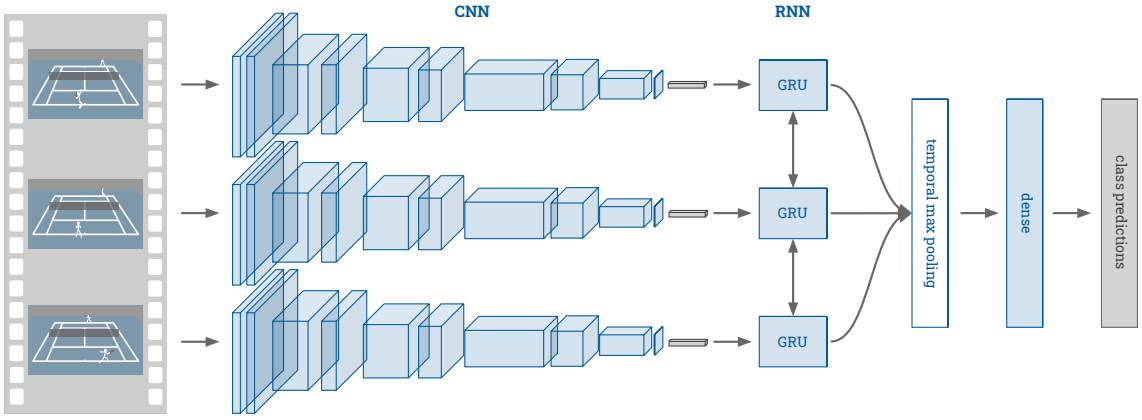


Figure 3.7: A visual representation of the CNN-RNN temporal encoding architecture. Assuming a window of size 3, each frame of the window is passed through the DenseNet-121, with the features extracted from after the global average pooling layer. These features are then passed as input into a bi-directional GRU RNN which encodes each feature with information from other time-steps within the window. The output features from the RNN at each time-step are then temporally max-pooled prior to being classified with the dense layer from the DenseNet.

Table 3.10 presents the results for varying window sizes. It can be seen that no matter the window size there is improvement using the CNN-RNN for the `OTH` class. Considering the serve and hit classes, it can be seen that there are still effects of the particular event lengths, with

window sizes of 15 and 30 being preferable for the `hit` and `serve` events respectively. Notably, the `fault` events (classes `SFF` and `SNF`) see significant F1-score increases with the high 120 window length. This is most likely a result of it utilising information directly after the serve action in the video, where if it's a fault no rally takes place and another serve is generally taken. This is an example of how in practice specific models might be most beneficial for different types of events, with consideration of the particular fine-grained problem at hand.

Frames	Class-wise F1-Score											
	OTH	SFI	SFF	SFL	SNI	SNF	SNL	HFL	HFR	HNL	HNR	AVG
3	97.1	60.5	12.6	8.9	62.0	17.3	0.0	78.8	80.1	78.2	82.8	52.6
15	97.3	62.6	12.7	12.3	64.0	21.3	0.0	81.2	82.8	81.2	86.0	54.7
30	97.6	65.0	13.4	13.5	66.2	27.9	0.0	80.6	83.0	80.3	84.8	55.7
120	97.1	63.2	**26.2**	0.0	66.0	**39.4**	0.0	60.5	61.2	63.4	64.6	49.2
baseline	97.0	57.9	17.7	13.0	62.9	21.6	0.0	74.8	76.3	77.5	78.0	52.4

Table 3.10: Experimental results of the CNN-RNN temporal encoding architecture. Once again we present the class-wise F1-scores for various window sizes. Similarly to the temporally pooling results we find the `serve` events benefit from a larger window size (30 or even 120), while the shorter `hit` events benefit from a smaller window size (15). The `fault` classes specifically benefit from a large window size of 120 frames, likely due to the fact that play stops directly after a fault, so using the frames post the serve action is providing a better cue of whether a serve is a fault or in.

3.3.2 Commentary Generation

The focus on a fine-grained problem revolves around the idea of applying these visual understanding models to real world applications. Being able to detect particular tennis events temporally is a very useful tool for statistics, however it's interesting to understand how well these detections can be used for something more sophisticated. In this sub-section we investigate the ability of basic video captioning models to generate accurate captions for tennis commentary generation.

Data Processing

There are some key differences between how we have processed input and output data for the event detection model and what is necessary for caption generation. For event detection we input either a single or a window of frames, to generate a single class label for a single frame. For captioning however, the input is a variable number of frames representing a single *point*, and the output is a single sentence with variable number of words.

As neural networks are only able to take a fixed sized input and generate a fixed sized output, for varying length sequences the use of *padding* is required when training and running the model (as also done in Section 1.1.3). As the *point* lengths are highly variable from less that 200 frames to

over 900, we also utilise the practice of *bucketing* during training. **Bucketing** involves grouping inputs of similar lengths together and passing them through the model together during training ie. in a single batch there will only ever be samples from one group or *bucket*. We partition our inputs (I) and outputs (O) into five different lengths — (I: 230, O: 10), (I: 402, O: 16), (I: 574, O: 22), (I: 746, O: 28), (I: 918, O: 34), padding intermediate samples up to these lengths.

Word Embeddings

A standard practice when using words as input is to embed them into a shared vector space, which when learnt, places similarly contextual words in similar areas of the embedding space. We embed each of the 250 words in our extended vocabulary into a 100-dimensional embedding space which is learnt prior to training the captioning model. To learn the word embeddings we utilise a Skip-gram model [Mikolov et al., 2013] which looks to capture syntactic and semantic word relationships. Figure 3.8 presents a visualisation of the learnt space, lowered from 100 to 2 dimensions using t-SNE (t-distributed stochastic neighbor embedding), showing how even in two dimensions many similar words are spaced nearby one another. For example, as `np` and `fp` are used in similar contexts they are spaced next to each other, the same can be seen for `rs` and `ls`.

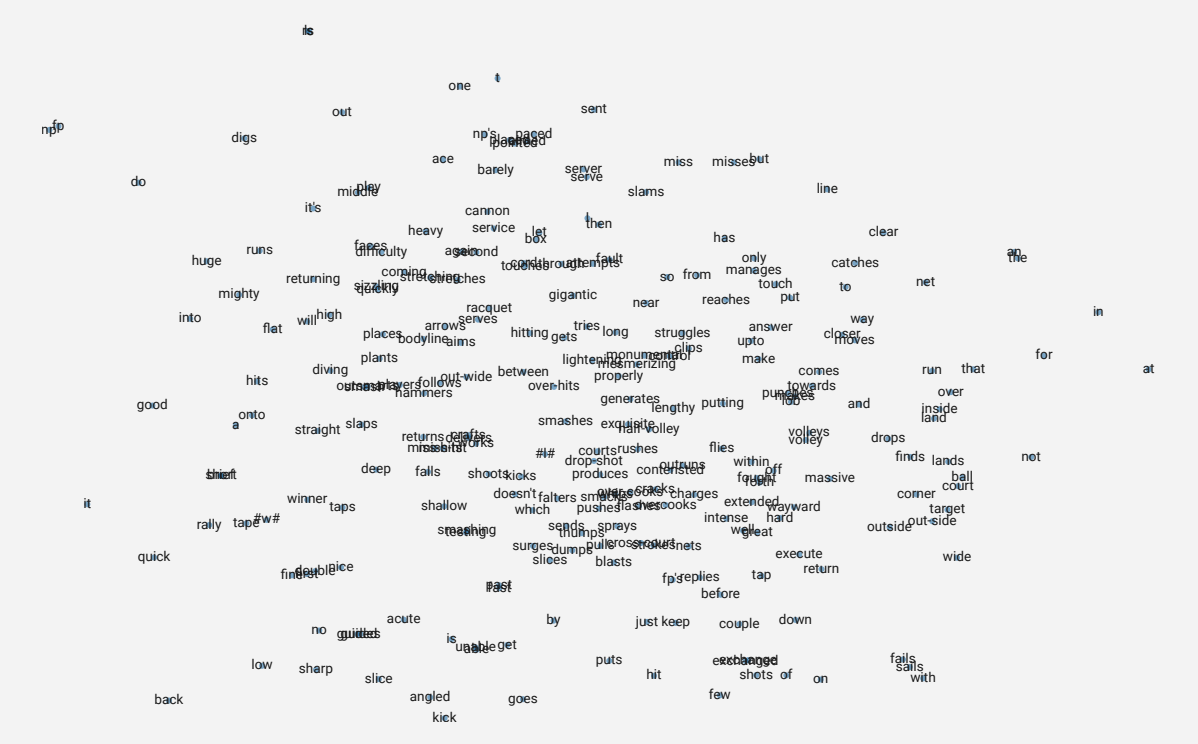


Figure 3.8: The word embeddings for the extended tennis vocabulary. This shows a visual representation of the 100-dimensional word embedding space collapsed into 2-dimensions using t-SNE. If we focus on the top left corner we can see that `np` and `fp` are grouped together as well as `rs` and `ls`. An interactive 3-dimensional t-SNE embedding result can be found on the project webpage — hayden.faulkner.codes/tennis.

Captioning Model

We utilise a sequence-to-sequence RNN [Wu et al., 2016] approach for generating commentary descriptions for each *point*. The model, visually represented in Figure 3.9, consists of an encoder bi-directional GRU RNN that takes the DenseNet CNN feature outputs, passing out an encoded feature at each time-step. Due to computational hardware restrictions the features from the DenseNet CNN are pre-extracted, and loaded in as inputs to the RNN encoder. A second GRU RNN then acts as a decoder to generate words one at a time, based on the previously generated word and the attention weighted encoder time-step features. The attention is a temporal additive scheme [Bahdanau et al., 2015] as seen in Section 1.2.2.

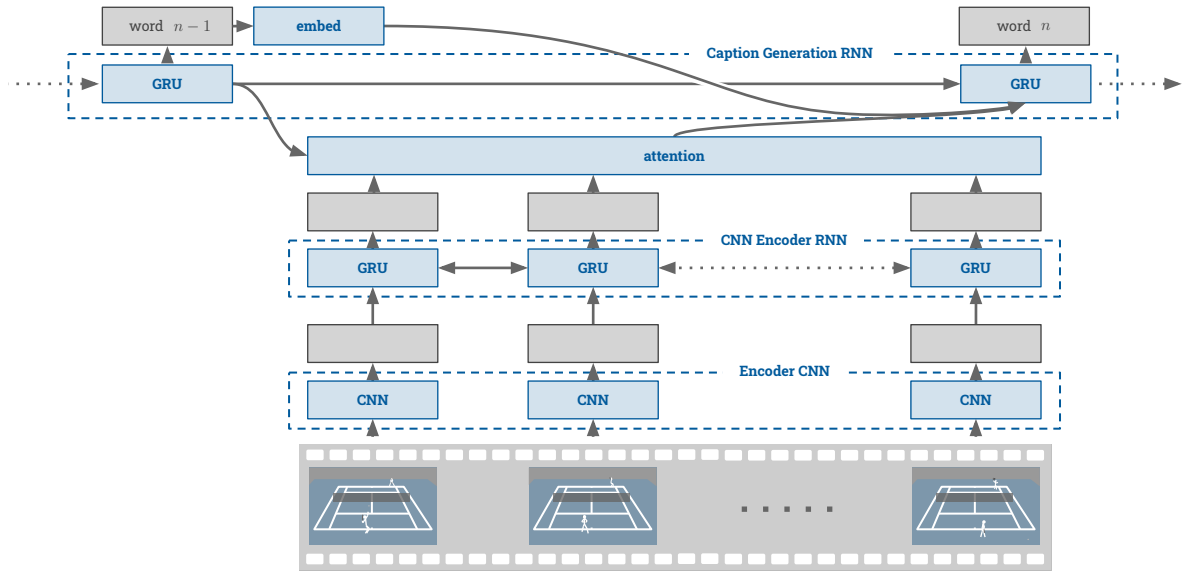


Figure 3.9: A visual representation of the CNN-RNN captioning pipeline. Given a set of frames for a single `point` event, we pass each frame through an encoder CNN, in our case this is our DenseNet-121 model pre-trained on the tennis event classification problem. The output features from the encoder CNN are then passed into a bi-directional GRU RNN to encode each of the temporal features with context from other time-steps in the point (as like was done in Section 3.3.1). These temporally encoded features are then individually weighted with a temporal attention mechanism conditioned on the current caption generator state. A single-time-stepped temporally attended feature is then concatenated with the previous generated word embedding and input through a GRU unit in the caption generation RNN. The caption generation RNN then generates probability distributions for the n^{th} word.

Evaluation

We evaluate the generated captions with the standard captioning metrics BLEU, METEOR, ROUGE and CIDEr (which are described in Section 1.1.5). Table 3.11 presents the metric scores across the test set for varying hidden sizes of the RNN. The results show that using a larger hidden size of 256 is more beneficial than smaller sizes of 128 and 64.

We also investigate the impact of using a CNN which is not pre-trained on the tennis events data, but is rather fine-tuned from ImageNet pre-training during the caption generation model training. This fully end-to-end training gives potential for the CNN to learn useful weights for aiding directly in the captioning. Conversely, not having the network pre-trained on the fine-

Hidden Size	B1	B2	B3	B4	MT	RG	Cr
64	42.0	27.4	19.4	14.0	20.5	35.6	73.7
128	44.9	28.1	19.2	13.9	22.2	37.5	73.1
256	46.7	30.7	22.1	16.4	22.6	43.9	**96.4**

Table 3.11: Experimental results of the commentary caption generation with standard caption metrics. This table presents the captioning metric scores for BLEU1-4 (B1, B2, B3, B4), METEOR (MT), ROUGE-L (RG) and CIDEr (Cr) for varying hidden size values for the RNN. Results show that the larger size of 256 is most effective. It is likely that sizes larger than 256 would further improve performance, however at the time of writing such network sizes were not computationally feasible.

grained tennis event detection data may result in a lack of domain knowledge for the tennis task. Due to computing resource constraints we only utilise a ResNet-18 CNN [He et al., 2016] (in place of the DenseNet-121 model), and sample only every 25th frame from the `point` input clip. We expect that the ResNet-18 having similar event classification accuracy as the DenseNet-121 model (as shown in Table 3.5), that results would be fairly similar as if we were to utilise features extracted from a pre-trained ResNet-18.

Table 3.12 presents the results for the end-to-end model and shows mixed results for the varying hidden sizes and relatively similar results to those in Table 3.11. Despite the similar metric scores, visual inspection shows that these models completely fail, only being able to generate a handful of different caption outputs. For example, the 128 dimensional model (best in this case) only produces one of the three following captions across the entire test set — '*double fault*', '*good serve in the middle np returns a ls return short rally np hits a rs winner down the line*', '*cannon serve is an ace*'. This highlights two important points — firstly, that the captioning models are reliant on the determinative abilities of a specifically trained tennis event detector, and secondly that the standard metrics don't present a good representation of performance for this particular type of data.

Hidden Size	B.1	B.2	B.3	B.4	M.	R.	C.
64	43.2	28.8	21.0	15.8	20.8	35.7	78.1
128	42.3	33.6	25.2	19.7	21.8	33.8	**84.8**
256	42.0	25.7	17.5	12.6	20.3	35.2	61.6

Table 3.12: Experimental results of the commentary caption generation with an end-to-end architecture. This table again presents the captioning metric scores for BLEU1-4 (B1, B2, B3, B4), METEOR (MT), ROUGE-L (RG) and CIDEr (Cr) for varying hidden size values for the RNN. We find in this case the middle hidden size (128) performs best, however it still doesn't achieve the highest score of the tennis pre-trained DenseNet-121 model (84.8 versus 96.4).

Considering the aforementioned second point, these metrics may provide numerical baselines for future models, however they don't provide great insight to how correct the generated commentaries actually are. Therefore, in Table 3.13, we also randomly select and compare ground truth (GT) and predicted (PR) captions from the best performing model (DenseNet-121 with hidden size of 256).

		Caption Ground Truths (GT) and Predictions (PR)
01	GT	*'high kick serve fp returns a ls return short rally fp cross-court rs lands out-side the court'*
	PR	*'fp serves a good one np delivers a rs return fp sends a ls out of the court'*
02	GT	*'good serve aimed at t np only reaches to it hitting the return long'*
	PR	*'fp arrows a good serve at t np is unable to return it'*
03	GT	*'good serve in the middle np returns a quick ls return short rally np cross-court fails to clear the net in the middle'*
	PR	*'good serve in the middle np crafts a ls return fp cross-court ls fails to land inside the court'*
04	GT	*'fp serves a high kick serve np returns a quick ls return brief rally np rs catches the net'*
	PR	*'fine serve placed out wide np returns a ls return short rally fp strokes a rs cross-court winner'*
05	GT	*'quick serve np crafts a rs return fp goes for a ls down the line but catches the net'*
	PR	*'fine serve np shoots a rs return winner'*
06	GT	*'double fault'*
	PR	*'double fault'*
07	GT	*'good serve np generates a rs return fp then returns one into the net'*
	PR	*'good serve in the middle np returns a ls return fp cross-court rs catches the net'*
08	GT	*'fp serves a high kick serve np delivers a high ls return fp produces a ls winner coming to net'*
	PR	*'fp serves a good one np returns a quick rs return fp struggles to keep a cross-court rs in a rally'*
09	GT	*'quick serve np returns a quick rs return fp ls is unable to clear the net'*
	PR	*'nice serve by fp np faces difficulty in returning it'*
10	GT	*'fp aims a high kick serve np returns a ls return fp hits a rs cross-court winner'*
	PR	*'fp serves a good one np returns a quick rs return fp struggles to keep a cross-court rs in a rally'*
11	GT	*'good serve in the middle np crafts a ls return short rally np hits a rs cross-court but it fails to clear the net'*
	PR	*'good serve in the middle np crafts a ls return fp cross-court ls fails to land inside the court'*
12	GT	*'fp hits a bodyline serve np has no answer to it'*
	PR	*'fp arrows a good serve at t np is unable to return it'*
13	GT	*'double fault'*
	PR	*'good serve aimed at t np only reaches to it'*
14	GT	*'double fault'*
	PR	*'double fault'*
15	GT	*'fp aims a high kick serve np crafts a ls return good rally fp sends a rs cross-court out of the court'*
	PR	*'good serve in the middle np crafts a ls return fp cross-court ls fails to land inside the court'*
16	GT	*'fp arrows a bodyline serve np struggles with it'*
	PR	*'fp arrows a good serve at t np is unable to return it'*

Table 3.13: Tennis dataset caption prediction examples. Some randomly sampled examples of ground truth (GT) captions and their predictions (PR). Underlined parts of the captions present the errors and misalignments.

Examining the predicted captions, the main failure cases are the incorrect uses of '*np*' and '*fp*' or '*ls*' and '*rs*' which may be a result of these words being placed close together in the embedding space. In these cases, where words are often used in similar parts of captions but mean completely opposite ideas, relying too heavily on the word embeddings may incorrectly flip crucial concepts. Other errors are related to generating text suggesting the ball lands out rather than hitting the net (`03`, `11`), or a serve is an ace rather than a double fault (`13`). Such errors are bound to occur as deciphering the differences between them require specifc knowledge of ball bound position which isn't explicitly captured in our dataset. Furthermore, in some circumstances (`12`, `16`), captions are repeated for relatively similar samples, despite there being other ways of describing it, suggesting a lack of generation diversity ability.

3.4 Summary

In this chapter we have shifted our focus from generalised video understanding to understanding and interpretation within a specific domain. Focusing on a fine-grained problem allows us to get a sense of the abilities and usefulness of modern video classification, detection and captioning models for real world problems. For numerous different reasons mentioned at the start of the chapter, including because sports analysis is a growing area of interest in the commercial sector, we decided to focus on the particular sport of tennis.

As data underpins all machine learning models, we introduced a novel Tennis dataset that was manually and thoroughly annotated with temporal events (Section 3.2.1) and caption descriptions (Section 3.2.2). To the best of our knowledge this was the first fine-grained dataset to include event and caption annotations in a single dataset. The annotation tool we implemented is versatile and general purpose, allowing it to be utilised for other temporal based event annotation tasks, and is made available for free online (see Section 3.2.1). The tennis dataset consists of over 4000 individual events, across 5 full match videos obtained from YouTube. Furthermore for the 746 individual points, a descriptive commentary style annotation was added, as well as a further 10817 non-grounded captions for improved language modelling. Although our dataset is relatively small in comparison to other modern datasets, it is manually annotated by experts and hence is much more accurate and free of noisy annotations. This property makes it useful for evaluating models that attempt to learn from small amounts of noise-free data.

Using our dataset we examined the abilities of a number of standard modern temporal event detection and video captioning architectures for the tasks of serve and hit detection, and point commentary generation. After experimenting with a set of CNN based frame classifiers (Table 3.5) we found that the DenseNet-121 architecture striked a good balance between performance and efficiency. We uncovered that much of the misclassification that occured was based around visually similar event types, such as the various types of *serve* events — `in`, `fault` and `let` (Figure 3.6). Furthermore, we also investigated the impact of utilising optical flow inputs, finding a performance boost for many of the event classes utilising a two-stream RGB and OF model design (Table 3.8).

In order to accommodate for, and help overcome frame-wise classification errors, a couple of temporal modeling schemes were incorporated and tested. Firstly, max and mean pooling across varying temporal lengths had mixed results, with small improvements over the frame-wise baseline for temporal lengths similar to event lengths (Table 3.9). Secondly, a GRU RNN followed by max pooling showed a similar trend — however with slightly more improvement due to the dynamic nature of the RNN model enriching the features prior to temproral pooling (Table 3.10).

Finally, we utilised a sequence-to-sequence RNN based caption generation model for generating commentary style captions for each `point` event. As a pre-processing step, the tennis vocabulary, consisting of 250 words, was embedded into a 100 dimensional space with a Skip-

gram embedding model (Section 3.3.2). Utilising the DenseNet CNN, which was pre-trained on the tennis event data, as a feature extractor we were able to generate relatively reliable captions. Some errors arose from nearby word embeddings getting mixed up, such as `np` and `fp` (Table 3.13). We point out that such errors are significant for the specific fine-grained tennis case, as knowing which player is attributed to which actions in the captions is key to successful captioning in this case. Enforcing distances between such words or other means of better determining such binary differences is likely needed, we leave this for future study. Furthermore, we also investigated the use of a smaller ResNet model for end-to-end learning using just the captions to jointly fine-tune the visual CNN from ImageNet pre-training. We found that this model, despite similar scores with the standard captioning metrics, was unable to generate useful captions (Table 3.12). This highlights the necessity for a specific tennis trained event detection model for the captioning model to rely on, especially when the amount of data is low, which is often the case with fine-grained datasets requiring more expert annotation.

Overall we believe that our dataset and experiments are a good start for research in one example domain of where machine learning techniques could have a direct impact on a real world applications. In comparison to other datasets, both recent fine-grained ones and more-so general ones, ours is relatively small — limited by the necessity to perform manual annotation. Future studies could focus on extending our dataset to be larger, or to include player and ball location annotations. In relation to the event detection and captioning models, these architectures are relatively standard and simplified, acting as a baseline for future works. There is plenty of scope to design much more tailored solutions to both the shot detection and commentary generation pipelines for the sport of tennis.

Summary & Future Directions

This thesis has taken a broad look at generalised and domain specific video understanding with standardised neural network based machine learning techniques. Chapter 1 focused on video captioning as a means of determining understanding, Chapter 2 examined object detection in video, and Chapter 3 considers event detection and captioning for a domain specific dataset.

With Chapter 1 we introduced the video captioning problem and discussed the varying approaches used to tackle the problem. We investigated the abilities of the more recently introduced Transformer Networks for caption generation in-place of the older LSTM Recurrent Neural Networks, finding for the two main captioning datasets MSVD and MSR-VTT that the TN is not as effective for this task. We believe that with more data and more insightful input features that the TN may become more effective as they are more reliant on the visual features compared to the RNN which is more focused on the current captioning state. After examining the training procedures we found that overfitting model predictions significantly outperform human annotators. Following this we presented a discussion on whats an appropriate means of determining human performance in relation to the current metrics. Furthermore, with a more thorough analysis of human versus model predicted captions, we found that the main challenge for captioning models with the current metrics is correctly identifying and generating specific nouns and verbs in the output caption. Experiments investigating the effect of different input features showed that the most crucial input features were salient spatial region features, which often represent specific objects, further highlighting the importance of providing good support for specific concept recognition. The natural diversity of language and broad range of concepts in the current datasets results in most concepts appearing rarely, prompting very few training examples for most concepts, something which hinders these data-hungry machine learning models. We believe that improving the identification of specific concepts, with a focus on handling concept rarity, with support from specific spatio-temporally localised visual features from exterior broad-class detectors is likely a promising research direction.

Chapter 2 naturally follows on from conclusions drawn in the first chapter, and considered the current methodologies for the problem of object detection in video. As current video based object detection models are all based on, and underpinned by, image based detection frameworks we firstly discussed the key frameworks and datasets utilised for object detection in images. We then considered three of these architectures to base our extensions on, deciding on the YOLO network due to its beneficial efficiency and performance combination. We investigated the use

of the YOLO framework on the video object detection dataset ImageNet-VID in a framewise capacity. We found that pre-training on any of the image based datasets is beneficial, and that greater intra-class diversity in the training data was favourable for a more generalised and successful detector. Looking to overcome challenges with noisy framewise results, we sought to modify the YOLO architecture as to allow it to handle temporal chunks of frames. We performed experimental analysis over a broad range of standardised modifications, finding promise in a number of the techniques for overcoming errornous framewise detections. Particularly our hierarchical 3D was most beneficial with minimal addition of parameters to the unmodified YOLO model. Most notably, we found that we could use temporal convolutions to improve features at various stages of the pre-trained DarkNet backbone, despite it being frozen. Even with the utilisation of the highly efficient YOLO detector the architectures were still incredibly computationally expensive and cumbersome to train and evaluate, with some taking months to train on ImageNet-VID. We consider efficiency one of the biggest, if not the biggest, challenge facing video based algorithms — both in terms of their uptake for real-world applications, and in terms of their research accessibility. Work focused on the efficiency of use and training of such models is vital for growth in the video object detection problem domain. Concluding the second chapter we also highlighted the lack of class diversity in the main video based detection datasets, and suggested that they would be unsuitable for captioning for this reason. As a step towards improving class diversity we constructed a hierarchical class tree which links the ImageNet-VID dataset with the three main image based detection datasets. Although this improved the coverage of nouns seen in the video captioning datasets MSVD and MSR-VTT, we found training on such a large dataset to be excessively lengthy, and ultimately non-beneficial to classification accuracy. Looking forward we suggest different approaches for improving class diversity, with focus on model improvements rather than increased data. We believe learning more from less is the most interesting direction of further research.

In Chapter 3 we shifted our focus to interpreting video from the fine-grained domain of tennis, looking to determine the applicability and fruition of some modern machine learning based techniques on such a specialised domain. We focused on two problems - serve and hit detection, which can be considered a temporal event classification and detection problem, and commentary generation, which can be considered a video captioning problem. We introduced a new tennis dataset which is manually annotated with temporally dense and contextually hierarchical events, each having specific individual attributes, and a set of commentary style captions. Limited by time and human resources, our dataset is relatively small compared to other fine-grained and general video datasets, which could be a performance limitation for the standard data-hungry machine learning models. However, despite its size its manual expert annotation means there is practically no noise or errors in the annotations, which is often the case for larger datasets. This makes our dataset useful for problems where learning from small amounts of error-free data is key. We made our dataset, along with the features and original videos, publicly available. Utilising the dataset, we explored a number of standardised models that have seen success for generalised event detection and video captioning, investigating performance when trained on our fine-grained tennis data. Despite the simplicity and lack of specialisation to

the tennis domain, we found that both the event detection pipelines and captioning models we examined showed promise for adapting to the tennis domain. These methods were intended as baselines for future models, which may consider the specific domain in their design and implementation.

A Appendices

A About the Appendices

These appendices are split into the following sections:

- **About the Appendices** — You are here.
- **Background** — Covers general theoretical background knowledge to aid in the understanding of concepts, models, and processes.
- **Code** — Covers the implementation details about the software utilised within and written for this thesis.
- **Datasets** — Covers information about the datasets mentioned and used within this thesis.
- **Extended Tables** — Includes some more extensive tables that were summarised in the main thesis text.

B Background

This section provides some further background information, aimed to help a novice understand the methods used in the main thesis body. Herein concepts such as deep learning and neural networks are introduced.

B.1 Deep Learning

In the past few years the term **"deep learning"** has not only become a *buzz-word* within the field of computer science, but also in broader society, as more and more of such technologies affect people's day-to-day lives. ***Deep learning is a categorisation of a particular type of method*** which is a sub-category of **machine learning (ML)** methods, which also include things like *decision tree learning*, *reinforcement learning*, *clustering*, *bayesian networks*, etc. Deep learning methods focus on ***learning data representations*** *(learning)* via a set of ***many sequential operations*** *(deeply)*. We direct readers who want to know more about deep learning to the **Dive into Deep Learning** book [Zhang et al., 2019a] (d2l.ai), which covers all of the current core techniques and principles, some of which we will touch upon in this appendix section.

When we say **learning**, we are referring to learning a model's f parameters such that the model can transform data from one form to another: $\mathrm{y} = f(\mathrm{x})$. For example, for image classification the input x might be an image of a cat and the desired output y is the label `cat`. For neural networks and deep learning we use a **data-driven** learning strategy, where the model learns by processing many example samples. Generally, with each input example, we also know the desired output in some form of label or annotation (x, y). When we teach the network with data samples where we know the label it's called **supervised learning**. We can also have some but not all labels — **semi-supervised learning**, or no labels — **unsupervised learning**. This data driven learning strategy works very effectively, and is the reason why problem focused datasets are required. A **dataset** is a set of these example samples for a particular problem. Within the community there are a number of standard datasets commonly used — see the ones of interest in this work in Appendix D. Many datasets split their samples into three exclusive groups, a **training** split, and **validation** split and a **testing** split.

The process of passing in all the samples to teach the model is called **training**, or the **training phase**. We also have a **testing phase** to determine how good our models are. Neural networks are quite good at learning about the data they are given in the training phase, and so it's important that we test with data that isn't seen during the training. This allows us to see how well our model actually performs on, or generalises to, new and unseen input data. With this idea in mind, it's also important to have a **validation phase** during the training phase. The validation phase is like a mini-testing phase to ensure the model isn't *over-fitting* to the training data. **Over-fitting** is when the model gets very proficient at transforming all of the training samples, but poor on new unseen samples. So during training we give the model a set of data not used in the training dataset and perform a mini-evaluation to ensure it can still perform well on unseen

data. The validation data is separate from the normal testing data, since we chose our best performing models based on their validation accuracy, which would provide a bias if the validation data was the testing data.

The training process involves taking an example (x, y) from the training set, predicting with the current model $\hat{\mathrm{y}} = f(\mathrm{x})$, and comparing it to the desired output y. We have some measure, called the **loss**, of how wrong our prediction $\hat{\mathrm{y}}$ is compared to the desired output y. Then using a method called **backpropogation**, the loss is used to slightly adjust the model's parameters in a way that lowers this loss for this example. This process is repeated many many times for every example in the training set, making one **epoch**. Over time, with many epochs, if the model is successfully learning, the loss will decrease and the model will better fit the both the training and validation data. Figure A.1 presents a visualisation of the training, validation and evaluation processes as well as more general extended deep learning pipeline.

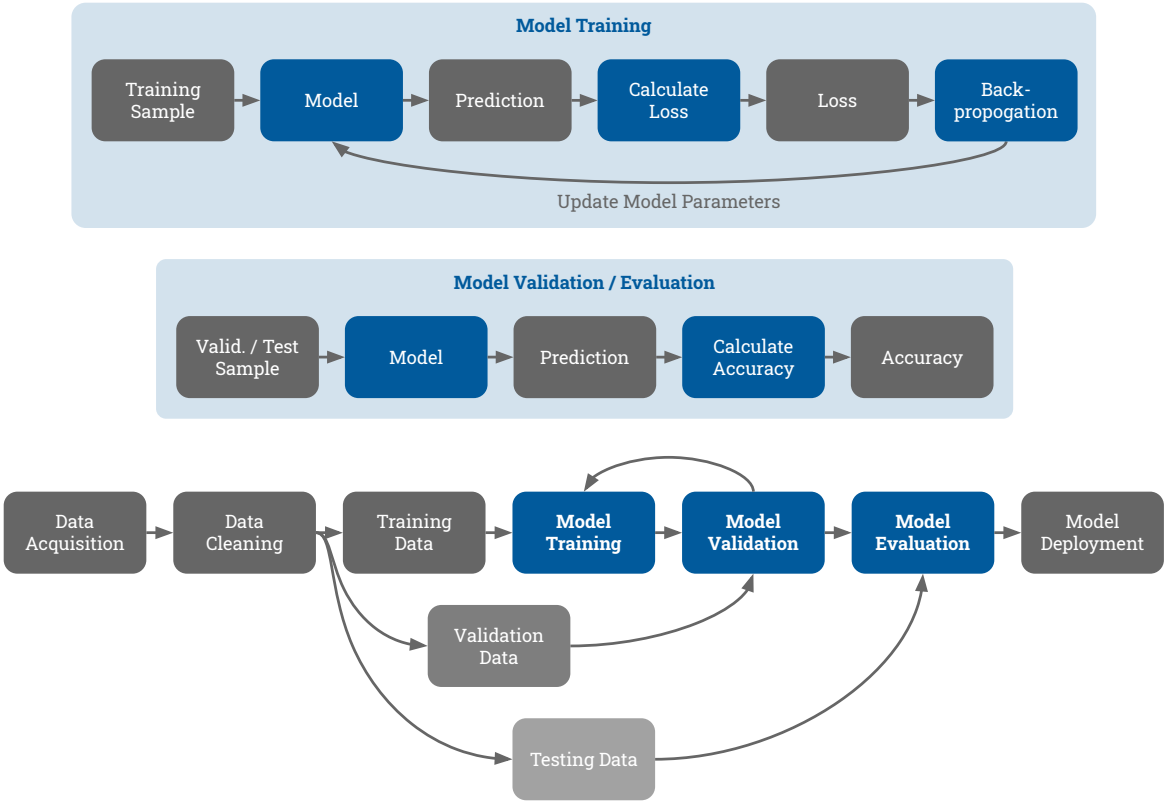
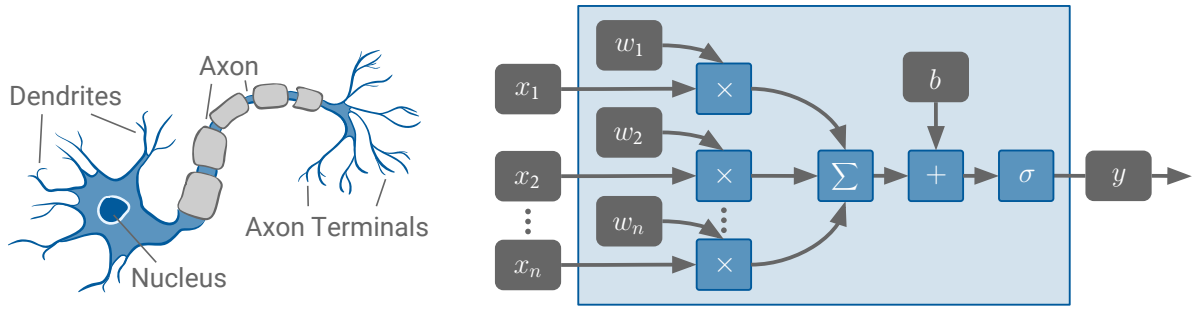


Figure A.1: The training, validation and evaluation processes with the full deep learning pipeline. Starting from the top, the training process consists of taking a training sample x, passing it through the model to make a prediction $\hat{\mathrm{y}}$, using the prediction and ground truth y to calculate the loss, which is then used by the backpropagation algorithm to update the model's f parameters. The validation and evaluation process is similar, passing in a validation or test sample to the model to make a prediction, which is then compared to the ground truth to determine how accurate the model is. Considering the full pipeline (at the bottom), it's necessary to perform some data acquisition, including potentially labelling the data, then some data cleaning is performed before the data is split into the training, validation and testing splits. Each of these splits are then used to train and test the model using the processes previously described. Training and validation are iteratively repeating processes where the validation is performed during training, generally at the end of an epoch. Based on the validation outcomes, at some point the training will cease, and the model can be evaluated on the test data before being deployed for real world use.

B.2 Artificial Neural Networks

The basis and ideas for **Artificial Neural Network (ANN)** models were introduced over half a century ago in works [McCulloch and Pitts, 1943] (introduced threshold logic) and [Hebb, 1961] (introduced Hebbian learning). ANNs are often considered as a computational model that is inspired by, and mimics the behaviour of, the human brain, a biological neural network.

At the core of both artificial and biological networks is the **neuron** — also termed node or unit for ANNs. Neurons take inputs from either other neurons or external sources, and calculate an output based on the inputs. Figure A.2 presents the artificial neuron design alongside that of the biological neuron.



$$y = \sigma\left(\sum_n w_n x_n + b\right) \quad \text{or in vector multiplication form} \quad y = \sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$$

Figure A.2: The biological neuron, its artificial counterpart, and the mathematical representation. The biological neuron receives input signals via its *dendrites*, the size of the dendrites determine the strength of each input signal. These input signals are accumulated in the *nucleus* and if a threshold is reached, the neuron pulses a signal down its *axon* to its axon terminals that join to dendrites of other neurons. The artificial neuron works in a similar way, with **inputs** $\mathbf{x} = [x_1, ..., x_n]$ being the input signals (each is just a number). The **weights** $\mathbf{w} = [w_1, ..., w_n]$ act like the dendrite size, weighting each the input signals (again each is just a number, and there is one weight per input). The inputs are multiplied by their weights before being accumulated (by **summing**). A **bias** b is added to the result and acts like the threshold that the sum needs to reach to generate an output signal (again, the bias is also just a number). This result is then passed through an **activation function** σ, which determines the exact output value y, introducing non-linearity to the neuron.

Activation functions, also called non-linearities, are an integral part of neural networks. Without non-linear functions the depth of a model wouldn't improve its approximation power, since linear functions sequentially applied just result in a linear function. Some of the most commonly used activation functions are *sigmoid*, *tanh*, *ReLU* and *Leaky-ReLU* (Figure A.3).

So far we have covered the workings of a single neuron, but their real power comes when they are joined together into a network or neurons. Figure A.4 presents a three layer *fully connected* (or *dense*) network, or **Multi-Layer Perceptron (MLP)**. It's fully connected because the output of every neuron is passed as input to every neuron in the following layer.

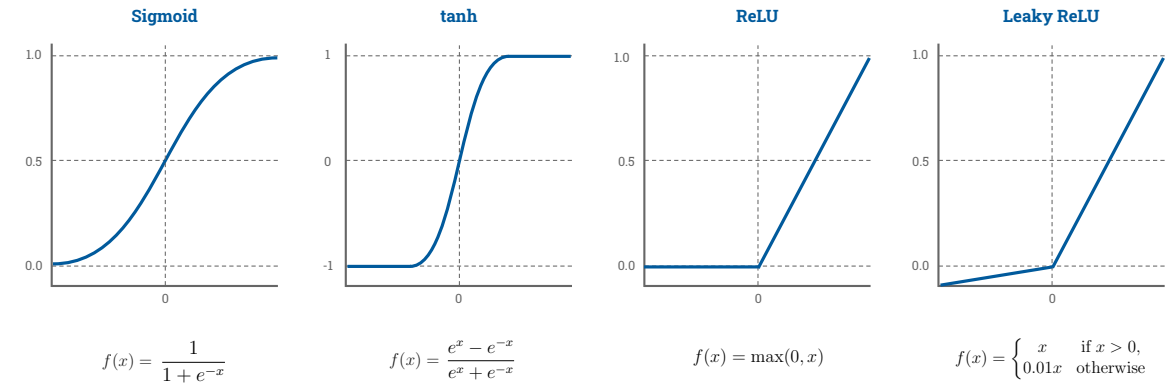


Figure A.3: Activation function graphs and their formal definitions. The *sigmoid* function is often used for generating probability distributions for multiple class problems. The *tanh* function is commonly utilised in Recurrent Neural Networks. The *ReLU* function is the most common activation function utilised between layers, with L-ReLU being a more recent version.

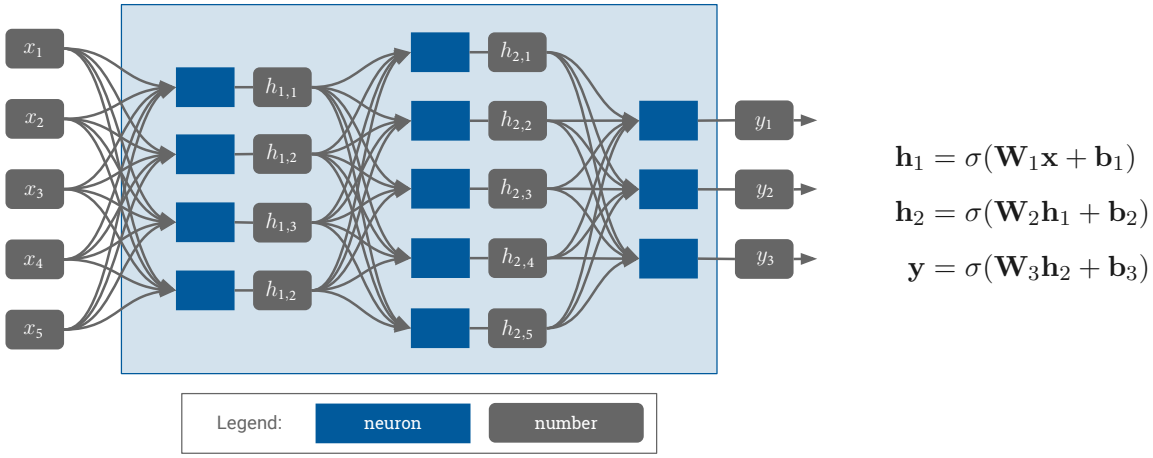


Figure A.4: A simple 3-layer artificial neural network — visually and mathematically. The blue boxes are the neurons and the grey boxes are inputs and outputs for each neuron. The outputs of the layers prior to the final layer, are called the model's **hidden states** $\mathbf{h}$. Each layer has multiple neurons, each with their own weight vectors, and bias scalars — so for each layer there is a weight matrix $\mathbf{W}_l$ and biases vector $\mathbf{b}_l$. These weight matrices and bias vectors are what are **learned** by the model in the training process.

B.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) models, specifically 2D CNNs, are the neural networks widely adopted to process image data. They were introduced in the form used today in [LeCun et al., 1989] for processing handwritten digits. Due to the nature of image data generally containing regular recurring patterns within spatially localised areas, spatially localised convolutional operations are useful for interpreting images.

A **convolutional operation** compares part of an input (in our case an image) to a **filter** made up of **kernels**. Filters were originally hand crafted by researchers and engineers, purposely designed for identifying specific patterns to solve specific problems — for example filters to pick up corners and edges. With the aid of data driven deep learning methods, in CNNs, the filters are no longer hand crafted but rather learnt from the data. Figure A.5 presents a convolution operation on a three channel input image of size 6×6, as well as a more general example with more input channels and filters, resulting in a deeper output feature volume.

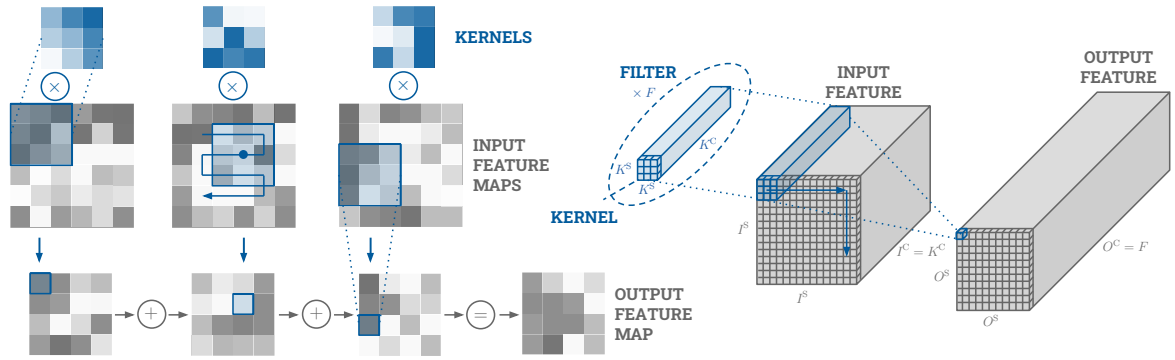


Figure A.5: A visual representation of the 2D convolutional operation. On the left we present an in-depth look at the 2D convolutional operation on a $6 \times 6 \times 3$ input volume. For each of the 3 input channels there is a corresponding kernel (in this case a 3×3 kernel). The kernels are each slid across their corresponding input channels with the overlapping pixels being multiplied together and then summed to generate the output values per pixel position per output channel. The individual output channels are then summed to produce the final output feature map of size $4 \times 4 \times 1$. On the right we present a more general case where a $K^{\mathrm{S}} \times K^{\mathrm{S}} \times K^{\mathrm{C}}$ filter is applied to an entire $I^{\mathrm{S}} \times I^{\mathrm{S}} \times I^{\mathrm{C}}$ input feature volume. Although we only show one, in this case there would be many F filters that would each determine a single channel in the $O^{\mathrm{S}} \times O^{\mathrm{S}} \times O^{\mathrm{C}}$ output volume.

Mathematically, the 2D convolutional operation for a single output volume channel pixel (at position (x, y)) can be defined as:

$$o_{x,y} = \sigma\left(b + \sum_{p=1}^{K^{\mathrm{S}}} \sum_{q=1}^{K^{\mathrm{S}}} \sum_{m=1}^{K^{\mathrm{C}}} w_{p,q,m} i_{\left(x+p-1-\left\lfloor \frac{K^{\mathrm{S}}}{2} \right\rfloor\right),\left(t+q-1-\left\lfloor \frac{K^{\mathrm{S}}}{2} \right\rfloor\right),m}\right)$$

where σ is the activation function, b is the bias, m indexes over the channels of the kernel K^{C}, $w_{p,q,m}$ is the value at position (p, q, m) of the kernel of size $K^{\mathrm{S}} \times K^{\mathrm{S}} \times K^{\mathrm{C}}$, and i is a value in the input volume.

A CNN consists of multiple convolutional layers, which each perform convolution operations using many different filters over an entire image or an intermediate feature map. A *simplified* example of a common CNN architecture is shown in Figure A.6.

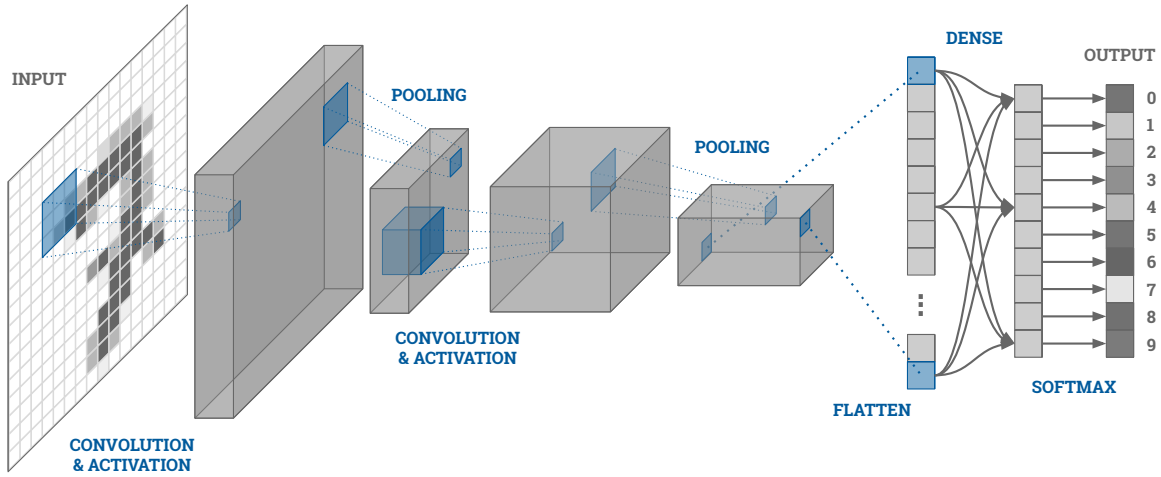


Figure A.6: A visual representation of a simplified Convolutional Neural Network (CNN) architecture. Here we present a very simplified model — modern models are much more sophisticated with many many more layers and special interconncections between layers. In this case, the model consists of two convolutional layers, each followed by a *pooling* layer. The point the pooling layers is to lower the spatial dimensions of the volumes to reduce the number of convolutional operations as the network gets deeper and more channels become beneficial. Pooling layers therefore don't change the number of channels and hold no parameters that need to be learnt. Pooling layers generally take the maximum value (max-pooling) of the feature, although other pooling strategies exist. At the end of the network there is a flattening operation, dense layer and softmax operation. These are normally included on image classification CNNs as a way of going from a spatial feature map to a category label. However, more recent networks have replaced the flatten and dense layers with a convolutional layer consisting of C filters (where C is the number of classification classes) and kernel size of 1×1.

C Code & Commands

Historically, a significant problem with works related to neural networks and deep learning is **reproducibility**. Many papers don't include official code, those that do often provide broken or poorly documented code. Even with working code it can still be difficult to reproduce results due to different hardware and software conditions. Furthermore, many state-of-the-art results require extensive computing resources to obtain, resources which are unavailable to many researchers. The reproducibility problem is well articulated on wired.com[1].

We believe in good software practices and public release to help enable reproducibility and ensure accessibility for future works. To this end, we include this appendix section to provide more details on our code and experiments. This section of the appendix discusses documented code for all results as well as thoroughly explaining experimental conditions that led to the results.

Code implementations for this thesis utilise the community standard - **Python**. All code is available online on my **Github**[2] under numerous projects:

- Chapter 1 - github.com/HaydenFaulkner/Attributes_SVO_Video_Captioning[3]
- Chapter 2 - github.com/HaydenFaulkner/VideoYOLO[4]
- Chapter 3 - github.com/HaydenFaulkner/Tennis[5]

With the recent boom of interest in neural network, a number of deep learning libraries have become available. Currently the three most utilised are:

- Tensorflow[6] (and Keras[7])
- PyTorch[8]
- MXNet[9] (including GluonCV[10] and GluonNLP[11])

Throughout the course of this PhD all three of these libraries were investigated and utilised in some form or another. The libraries have each evolved over the years, changing in popularity and usage amongst researchers, each having various pros and cons. For Chapter 1 we utilise **PyTorch**, while for Chapter 2 and Chapter 3 we utilise **MXNet**.

[1] https://www.wired.com/story/artificial-intelligence-confronts-reproducibility-crisis/
[2] https://github.com/HaydenFaulkner
[3] https://github.com/HaydenFaulkner/Attributes_SVO_Video_Captioning
[4] https://github.com/HaydenFaulkner/VideoYOLO
[5] https://github.com/HaydenFaulkner/Tennis
[6] https://www.tensorflow.org
[7] https://keras.io/
[8] https://pytorch.org/
[9] http://mxnet.incubator.apache.org/
[10] https://gluon-cv.mxnet.io/
[11] http://gluon-nlp.mxnet.io/

C.1 Grounded Captioning

Table 1.5 (page 54) and **Table 1.6** (page 54) present noun and verb occurrence statistics for MSVD and MSR-VTT. Simply running the `dataset_stats.py` script in the `misc` directory generates a print out of the statistics:

```
python misc/dataset_stats.py
```

Table 1.7 (page 64) presents the captioning performance of an LSTM RNN when using specific features for the MSVD and MSR-VTT datasets. Training and testing is carried out with the `train.py` script:

```
python train.py --dataset msvd --captioner_type lstm --model_id lstm_1
    --batch_size 8 --test_batch_size 8 --max_epochs 100 --input_features imrc
```

```
python train.py --dataset msrvtt --captioner_type lstm --model_id lstm_1
    --batch_size 8 --test_batch_size 4 --max_epochs 200 --input_features imrc
```

with `--input_features` being used to specify what features to include. Options include `i` for the image feature, `m` for the motion feature, `r` for the region features and `c` for the classification feature (`c` only used in MSR-VTT experiments).

Note the differences above when training and testing with MSVD vs with MSR-VTT. From now on we will only use MSVD commands as examples, however MSR-VTT experiments are the same except for changes to the `--dataset`, `--test_batch_size` ***and*** `--max_epochs` ***arguments***.

Table 1.8 (page 71) presents the captioning performance of different captioner implementations. Training and testing is carried out with:

```
python train.py --dataset msvd --captioner_type transformer --model_id
    transformer_1 --batch_size 8 --test_batch_size 8 --max_epochs 100
```

with `--captioner_type` being used to specify what captioner to use. Options are `lstm`, `gru` or `transformer`.

Table 1.9 (page 74) presents the captioning performance of including and excluding a feature encoder TN. Training and testing is carried out with:

```
python train.py --dataset msvd --captioner_type lstm --model_id lstm_1_enc
    --batch_size 8 --test_batch_size 8 --max_epochs 100 --input_encoder_layers 1
    --input_encoder_heads 1 --input_encoder_size 512
```

with `--input_encoder_layers`, `--input_encoder_heads` and `--input_encoder_size` specifying the input encoder parameters.

Table 1.10 (page 74) presents the captioning performance of various TN captioner sizes. Training and testing is carried out with:

```
python train.py --dataset msvd --captioner_type transformer --model_id
    transformer_1_4_8_256 --batch_size 8 --test_batch_size 8 --max_epochs 100
    --captioner_layers 4 --captioner_heads 8 --captioner_size 256
```

with `--captioner_layers`, `--captioner_heads` and `--captioner_size` specifying the TN captioner parameters.

Table 1.11 (page 75), **Table 1.12** (page 76) and **Table 1.13** (page 81) present the captioning performance of human annotators. Human evaluations are carried out with the `human_evaluation.py` script in the `misc` directory:

```
python misc/human_evaluation.py
```

Figure 1.29 (page 80) and **Figure 1.30** (page 81) present the TP, FP, and FN rates of particular nouns and verbs in the MSVD and MSR-VTT datasets. These stats are calculated with the `deeper_analysis.py` script in the `misc` directory:

```
python misc/deeper_analysis.py
```

Table 1.14 (page 89) and **Table 1.15** (page 89) present the captioning performance of various grounding methodologies for MSVD and MSR-VTT respectively. Training and testing is carried out with:

```
train.py --dataset msvd --grounder_type niuc --captioner_type lstm --model_id
    lstm_1_niuc --batch_size 8 --test_batch_size 8 --max_epochs 100 --concepts_h5
     sl_top_concepts --num_concepts 5
```

```
train.py --dataset msvd --grounder_type nioc --captioner_type lstm --model_id
    lstm_1_nioc_svo --batch_size 8 --test_batch_size 8 --max_epochs 100
    --concepts_h5 sequencelabel --num_concepts 3
```

with `--grounder_type` specifying the particular grounding method we want to use - `niuc` for the **single** method, `nioc` for the **multi** method or `ioc` for the **TN** method. `--concepts` specifies the particular input concepts we are using as ground truth (either `sl_top_concepts` for the top 5 attributes or `sequencelabel` for the SVO triplets) and `--num_concepts` specifies the particular number of concepts used.

Table 1.16 (page 93) and **Table 1.17** (page 93) present the captioning performance of various grounding methodologies with decoupling for MSVD and MSR-VTT respectively. Training and testing is carried out with:

```
train.py --dataset msvd --grounder_type niuc --captioner_type lstm --model_id
    lstm_1_niuc_dec --batch_size 8 --test_batch_size 8 --max_epochs 100
    --concepts_h5 sl_top_concepts --num_concepts 5  --decouple 1
```

with `--decouple` used as a flag specifying whether to use the decoupled methodology or not.

C.2 Concept Detection & Localisation

Table 2.2 (page 110) presented baseline results for different object detection implementations including Faster R-CNN, YOLO and SSD. These experiments were carried out with the default training scripts available on mxnet.io[12] along with the mAP scores. The scripts were modified to run a single epoch of training and validation on Pascal VOC. The number of parameters were provided by using the `net.summary()` function. Memory usage was visually monitored with `nvtop` (github.com/Syllo/nvtop[13]) and the maximum values recorded across the epoch.

Table 2.5 (page 115) presented YOLO mAP baselines on Pascal VOC, MS-COCO, ImageNet-DET and ImageNet-VID. For the all experiments we train YOLO from scratch with an ImageNet pre-trained DarkNet backbone that is fine-tuned during training. Training is carried out across four GPUs with a batch size of 64 using SGD with a learning rate of 0.001, momentum of 0.9 and weight decay of 0.0005.

The Pascal VOC model is trained over 200 epochs, with 4 warm-up epochs and a learning rate decay of 0.1 on epochs 160 and 180. It is trained on the ***trainval 2007 + 2012*** split and evaluated on the ***test 2007*** split. Training and evaluation is done with the `train_yolov3.py` and `detect_yolov3.py` scripts:

```
python train_yolov3.py --dataset voc --gpus 0,1,2,3 --save_prefix 0001
    --num_workers 16 --warmup_epochs 4 --syncbn
```

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0001/
    yolo3_darknet53_voc_best.params --metrics voc,coco --dataset voc
    --save_prefix 0001
```

The MS-COCO model is trained over 280 epochs, with 2 warm-up epochs and a learning rate decay of 0.1 on epochs 220 and 250. It is trained on the ***train 2017*** spilt and evaluated on the ***val 2017*** split. Similarly, the experiments are performed with:

```
python train_yolov3.py --dataset coco --gpus 0,1,2,3 --save_prefix 0003
    --num_workers 16 --lr_decay_epoch 220,250 --epochs 280 --warmup_epochs 2
    --syncbn
```

[12] https://gluon-cv.mxnet.io/model_zoo/detection.html
[13] https://github.com/Syllo/nvtop

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0003/
    yolo3_darknet53_coco_best.params --metrics voc,coco --dataset coco
    --save_prefix 0003
```

The ImageNet-DET model is trained over 140 epochs, with 3 warm-up epochs and a learning rate decay of 0.1 on epochs 100 and 120. It is trained on the ***train*** spilt and evaluated on the ***val*** split. Performed with:

```
python train_yolov3.py --dataset det --gpus 0,1,2,3 --save_prefix 0002
    --num_workers 16 --epochs 140 --warmup_epochs 3 --lr_decay_epoch 100,120
    --syncbn
```

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0002/
    yolo3_darknet53_det_best.params --metrics voc,coco --dataset det
    --save_prefix 0002
```

The ImageNet-VID model is trained over 80 epochs, with 2 warm-up epochs and a learning rate decay of 0.1 on epochs 50 and 70. It is trained on the ***train 2017*** spilt and evaluated on the ***val 2017*** split. Performed with:

```
python train_yolov3.py --dataset vid --gpus 0,1,2,3 --save_prefix 0035
    --num_workers 16 --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2 --every
     25 --syncbn
```

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0035/
    yolo3_darknet53_vid_best.params --metrics voc,coco,vid --dataset vid
    --save_prefix 0035
```

To **resume** any of the models, in the case they stop training before reaching their final epoch you can use the `--start_epoch` and `--resume` arguments like:

```
python train_yolov3.py --dataset vid --gpus 0,1,2,3 --save_prefix 0035
    --num_workers 16 --lr_decay_epoch 50,70 --epochs 80 --every 25 --start_epoch
    -1 --resume models/experiments/0035/ --syncbn
```

Table 2.6 (page 116) presented results of using the same trained models from previously but evaluating them all on the ImageNet-VID dataset (***val 2017*** split). Since the categories are different between the datasets we ignore any categories that aren't in both datasets. For the Pascal VOC trained model, evaluation is performed with:

```
python detect_yolov3.py --trained_on voc --batch_size 1 --model_path models/0001/
    yolo3_darknet53_voc_best.params --metrics vid --dataset vid --save_prefix
    0001
```

where `--trained_on` is set according to the particular data the model was trained on.

Table 2.7 (page 119) presented results for fine-tuning YOLO models on the ImageNet-VID dataset, after each being pre-trained on either Pascal VOC, MS-COCO, or ImageNet-DET. For all models we fine-tune over 40 epochs, with 2 warm-up epochs, a learning rate of 0.001 decaying by 0.1 at epochs 20 and 30. For training and evaluation we use (modifying `--trained_on`, `--save_prefix` and `--resume` for each dataset:

```
python train_yolov3.py --dataset vid --trained_on voc --gpus 0,1,2,3
    --save_prefix 0011 --batch_size 16 --num_workers 16 --lr_decay_epoch 20,30
    --epochs 40 --warmup_epochs 2 --every 25 --resume models/experiments/0001/
    yolo3_darknet53_voc_best.params --syncbn
```

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0011/
    yolo3_darknet53_vid_best.params --metrics vid --dataset vid --save_prefix
    0011
```

Table 2.10 (page 127) presented the comparison between YOLO with and without a frozen DarkNet backbone. The models are trained and evaluated on ImageNet-VID, with and without the argument `--freeze_base`:

```
python train_yolov3.py --freeze_base --batch_size 16 --dataset vid --gpus 0,1,2,3
     --save_prefix 0030 --num_workers 16 --lr_decay_epoch 50,70 --epochs 80
    --warmup_epochs 2 --every 25 --syncbn
```

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0030/
    yolo3_darknet53_vid_best.params --metrics vid --dataset vid --save_prefix
    0030
```

For the following experiments we use the main settings as above unless specified differently (with the exception of `--save_prefix` which changes for every model).

Table 2.11 (page 129) presented the comparison between different pooling strategies. The arguments `--window`, `--k_join_type` (`mean` or `max`), and `--k_join_pos` (`early` or `late`) are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0032 --num_workers 16  --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 16 --window 3,25 --k_join_type mean
    --k_join_pos early
```

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0032/
    yolo3_darknet53_vid_best.params --metrics vid --dataset vid --save_prefix
    0032 --window 3,25 --k_join_type mean --k_join_pos early
```

Table 2.12 (page 130) presented the comparison between different concatenation strategies.

The arguments `--window`, `--k_join_type`, and `--k_join_pos` are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0041 --num_workers 16  --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 16 --window 3,25 --k_join_type cat
    --k_join_pos early
```

Table 2.13 (page 132) presented the comparison between feature correlation strategies. The arguments `--window`, `--corr_d`, and `--corr_pos` (`early` or `late`) are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0051 --num_workers 16  --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 16 --window 3,25 --corr_d 4 --corr_pos early
```

Table 2.14 (page 133) presented the comparison between different three dimensional convolutional strategies. The arguments `--window`, `--block_conv_type` (`3` for 3D or `21` for (2+1)D), and `--k_join_pos` are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0061 --num_workers 16  --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 16 --window 3,1 --block_conv_type 3
    --k_join_pos late
```

Table 2.17 (page 139) presented the results with the use of pre-trained motion feature streams. The arguments `--window` and `--motion_stream` (`flownet` or `r21d`) are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0081 --num_workers 16 --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 16 --window 3,1 --motion_stream flownet
```

Table 2.18 (page 140) presented the results with the use of a hierarchical structure to the 2D DarkNet structure. The arguments `--window`, `--hier`, `--h_join_type` (`max` or `conv`) and `--new_model` are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0107 --num_workers 16 --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 8 --window 9,1 --hier 3,3,1,1,1
    --h_join_type conv --new_model
```

Table 2.19 (page 142) presented the results with the use of a hierarchical DarkNet with 3D convolutions. The arguments `--window` and `--temp` are used to specify these experiments:

```
python train_yolov3.py --freeze_base --dataset vid --gpus 0,1,2,3 --save_prefix
    0139 --num_workers 16 --lr_decay_epoch 50,70 --epochs 80 --warmup_epochs 2
    --every 25 --syncbn --batch_size 16 --window 5,25 --temp
```

Table 2.20 (page 143) presented a summary of results for most experiments as well as class agnostic evaluations. The arguments `--model_agnostic` and `--metric_agnostic` are used to specify these experiments:

```
python detect_yolov3.py --batch_size 1 --model_path models/experiments/0032/
    yolo3_darknet53_vid_best.params --metrics vid --dataset vid --save_prefix
    0032 --window 3,25 --k_join_type mean --k_join_pos early --model_agnostic
    --metric_agnostic
```

C.3 Fine-Grained Understanding

Table 3.3 (page 164) presents class split statistics for our tennis dataset. Simply running the `dataset.py` script generates a print out of the statistics:

```
python dataset.py
```

Table 3.5 (page 167) presents the framewise classification performance of different CNN methods on the tennis dataset. Training and testing is carried out with the `train.py` script:

```
python train.py --backbone DenseNet121 --model_id 0006 --split_id 01 --batch_size
    64 --epochs 20
```

with `--backbone` being used to specify the particular CNN architecture to use.

Table 3.8 (page 171) presents the framewise classification performance of different optical flow methods on the tennis dataset. Training and testing is carried out with:

```
python train.py --backbone DenseNet121 --model_id 0010 --split_id 01 --batch_size
    64 --epochs 20 --flow twos
```

with `--flow` being used to specify the particular optical flow strategy to use, options being `sixc` for six channel inputs, `only` for no RGB and `twos` for two-stream.

Table 3.9 (page 172) presents the framewise classification performance of different temporal pooling strategies. Training and testing is carried out with:

```
python train.py --backbone DenseNet121 --model_id 0028 --split_id 01 --batch_size
    64 --epochs 20 --window 15 --temp_pool mean
```

with `--window` specifying the number of consecutive input frames and `--temp_pool` specifying the particular pooling strategy to use, options being `mean`, `max` or `gru`.

Table 3.10 (page 173) presents the framewise classification performance of different temporal pooling strategies. Training and testing is carried out with:

```
python train.py --backbone DenseNet121 --model_id 0042 --split_id 01 --batch_size
    64 --epochs 20 --window 30 --temp_pool gru
```

Figure 3.8 (page 174) presents the tennis dataset's vocabulary embedding. Training is carried out with the `train_embeddings.py` script:

```
python train_embeddings.py
```

Table 3.11 (page 176) presents the captioning performance of a GRU RNN on the tennis dataset. Training and testing is carried out with the `train_gnmt.py` script:

```
python train_gnmt.py --model_id 0102 --split_id 01 --batch_size 128 --epochs 40
    --num_hidden 256 --cell_type gru
```

with `--num_hidden` specifies the hidden dimension of the RNN and `--cell_type` specifies the cell type (either `gru` or `lstm`).

D Datasets

Datasets are a vital part of modern day computer vision and machine learning techniques, with all modern systems relying on data to learn. The community of datasets is expansive, complex and quickly evolving, so all datasets mentioned within this thesis are summarised in Table A.1.

Image Dataset	Year	Cls.	Det.		Seg.	Cap.	Rel.	FG	Extends	Citation	Homepage
			S	T							
Pascal VOC	2007		✓							[Everingham et al., 2010]	host.robots.ox.ac.uk
ImageNet-CLS-LOC	2009	✓								[Russakovsky et al., 2015]	image-net.org
UIUC-PS	2010					✓			Pascal VOC	[Rashtchian et al., 2010]	vision.cs.uiuc.edu
Flickr 8k	2013					✓				[Hodosh et al., 2013]	
MS-COCO	2014		✓		✓	✓				[Lin et al., 2014]	cocodataset.org
Flickr 30k	2014					✓				[Young et al., 2014]	shannon.cs.illinois.edu
Flickr 30k Entities	2015		✓						Flickr 30k	[Plummer et al., 2015]	github.com
ImageNet-DET	2015		✓							[Russakovsky et al., 2015]	image-net.org
Open Images	2017	✓	✓		✓		✓			[Kuznetsova et al., 2020]	storage.googleapis.com
Visual Genome	2017		✓			✓	✓			[Krishna et al., 2017b]	visualgenome.org

Video Dataset	Year	Cls.	Det.		Seg.	Cap.	Rel.	FG	Extends	Citation	Homepage
			S	T							
MSVD	2011					✓				[Chen and Dolan, 2011]	microsoft.com
HMBD-51	2011	✓								[Kuehne et al., 2011]	serre-lab.clps.brown.edu
UCF101	2012	✓								[Soomro et al., 2012]	crcv.ucf.edu
MPII-Cooking	2012					✓		✓		[Rohrbach et al., 2012a]	mpi-inf.mpg.de
MPII-Composites	2012					✓		✓	MPII-Cooking	[Rohrbach et al., 2012b]	mpi-inf.mpg.de
J-HMDB	2013	✓								[Jhuang et al., 2013]	jhmdb.is.tue.mpg.de
TACoS	2013					✓		✓	MPII-Composites	[Regneri et al., 2013]	
TACoS M-L	2014					✓		✓	TACoS	[Rohrbach et al., 2014]	mpi-inf.mpg.de
Sports-1M	2014	✓								[Karpathy et al., 2014]	cs.stanford.edu
ActivityNet	2015	✓		✓						[Caba Heilbron et al., 2015]	activity-net.org
MPII-MD	2015					✓				[Rohrbach et al., 2015b]	mpi-inf.mpg.de
M-VAD	2015					✓				[Torabi et al., 2015]	mila.quebec
ImageNet-VID	2015		✓							[Russakovsky et al., 2015]	image-net.org
MSR-VTT	2016					✓				[Xu et al., 2016]	microsoft.com
NCAA Basketball	2016							✓		[Ramanathan et al., 2016]	appspot.com
MERL Shopping	2016							✓		[Singh et al., 2016]	merl.com
MPII-Cooking 2	2016					✓		✓	MPII-Composites	[Rohrbach et al., 2016]	mpi-inf.mpg.de
YouTube-8M	2016	✓								[Abu-El-Haija et al., 2016]	research.google.com
Charades	2016			✓		✓				[Sigurdsson et al., 2016]	allenai.org
Kinetics	2017	✓								[Kay et al., 2017]	deepmind.com
THUMOS	2017	✓		✓						[Idrees et al., 2017]	thumos.info
ActivityNet Captions	2017	✓		✓		✓			ActivityNet	[Krishna et al., 2017a]	cs.stanford.edu
MultiTHUMOS	2018	✓		✓					THUMOS	[Yeung et al., 2018]	ai.stanford.edu
ActivityNet Entities	2019	✓	✓	✓		✓			ActivityNet	[Zhou et al., 2019]	github.com
YouTube-8M Seg.	2019	✓		✓					YouTube-8M	[Abu-El-Haija et al., 2016]	research.google.com

Table A.1: Datasets summary. A summary of all of the datasets mentioned in this work, with details such as release year, applications (**Cls**: Classification, **Det**: Detection, **S**: Spatial, **T**: Temporal, **Seg**: Segmentation, **Rel**: Visual Relationships, **FG**: Fine-Grained), extension of, introductory paper, and web page link.

E Extended Tables

Combined Set Coverage

Table A.2 and Table A.3 are the full MSVD and MSR-VTT concept coverage tables which were summarised in Table 2.25 (Section 2.3.3) on page 153.

Missing Concepts

Table A.4 and Table A.5 are the full MSVD and MSR-VTT missing concepts tables which where summarised in Table 2.26 (Section 2.3.3) on page 154.

Class	Ext	Chd	Tot	% set
LEVEL 0				
ROOT	0	1075	1075	89.58
LEVEL 1				
artifact	0	669	669	55.75
food	243	126	301	25.08
living thing	0	877	877	73.08
LEVEL 2				
clothing	2	19	21	1.75
sports equipment	0	110	110	9.17
furniture	1	136	136	11.33
appliance	3	53	54	4.50
tool	5	10	14	1.17
scissors	9	0	9	0.75
street sign	0	0	0	0.00
musical instrument	0	86	86	7.17
toiletry	0	23	23	1.92
tableware	0	149	149	12.42
fireplug	0	0	0	0.00
parking meter	0	0	0	0.00
bag	20	20	20	1.67
vehicle	17	118	120	10.00
kitchen utensil	0	36	36	3.00
rubber eraser	0	0	0	0.00
medical instrument	0	1	1	0.08
teddy	1	0	1	0.08
electronic device	0	50	50	4.17
milk can	0	0	0	0.00
traffic light	0	0	0	0.00
binder	0	0	0	0.00
book	5	0	5	0.42
rule	0	0	0	0.00
stretcher	1	0	1	0.08
pencil box	0	0	0	0.00
pot	69	0	69	5.75
pencil sharpener	0	0	0	0.00
vegetable	37	36	61	5.08
guacamole	0	0	0	0.00
baked goods	0	0	0	0.00
trifle	0	0	0	0.00
ice lolly	0	0	0	0.00
hotdog	0	0	0	0.00
hamburger	8	0	8	0.67
burrito	0	0	0	0.00
pizza	18	0	18	1.50
fruit	28	35	52	4.33
animal	81	282	297	24.75
person	703	0	703	58.58
plant	14	0	14	1.17

Class	E	C	T	% set
LEVEL 3				
necktie	0	0	0	0.00
helmet	4	0	4	0.33
hat	6	0	6	0.50
diaper	1	0	1	0.08
brassiere	0	0	0	0.00
miniskirt	0	0	0	0.00
sunglasses	8	0	8	0.67
swimsuit	1	0	1	0.08
bathing cap	0	0	0	0.00
ball	51	13	53	4.42
baseball equip.	0	0	0	0.00
golf equipment	0	0	0	0.00
dumbbell	1	0	1	0.08
gymnastic app.	0	0	0	0.00
puck	0	0	0	0.00
punching bag	0	0	0	0.00
bow	5	0	5	0.42
board	46	8	49	4.08
ski	2	0	2	0.17
sport kite	0	0	0	0.00
tennis racket	0	0	0	0.00
frisbee	2	0	2	0.17
bookcase	0	0	0	0.00
bed	37	0	37	3.08
toilet	7	0	7	0.58
lamp	1	0	1	0.08
baby bed	0	0	0	0.00
washbasin	3	0	3	0.25
seat	4	50	54	4.50
table	49	0	49	4.08
file	4	0	4	0.33
hand blower	0	0	0	0.00
stove	38	0	38	3.17
washer	0	0	0	0.00
waffle iron	0	0	0	0.00
toaster	2	0	2	0.17
dishwasher	0	0	0	0.00
coffee maker	0	0	0	0.00
oven	7	0	7	0.58
refrigerator	4	0	4	0.33
vacuum	3	0	3	0.25
microwave	3	0	3	0.25
nail	5	0	5	0.42
hammer	3	0	3	0.25
screwdriver	2	0	2	0.17
ax	1	0	1	0.08
power drill	0	0	0	0.00

Class	E	C	T	% set
LEVEL 3 (cont.)				
chain saw	0	0	0	0.00
percussion inst.	0	6	6	0.50
wind inst.	0	11	11	0.92
stringed inst.	0	57	57	4.75
keyboard inst.	0	27	27	2.25
makeup	12	1	12	1.00
soap dispenser	0	0	0	0.00
cream	11	0	11	0.92
hair spray	0	0	0	0.00
perfume	0	0	0	0.00
toothbrush	1	0	1	0.08
glass	50	0	50	4.17
bowl	108	0	108	9.00
bowl	108	0	108	9.00
bottle	21	0	21	1.75
platter	2	0	2	0.17
mug	1	0	1	0.08
vase	1	0	1	0.08
pitcher	5	0	5	0.42
bag	20	20	20	1.67
purse	2	0	2	0.17
plastic bag	0	0	0	0.00
backpack	3	0	3	0.25
baggage	1	0	1	0.08
motorcycle	19	0	19	1.58
train	4	0	4	0.33
snowmobile	0	0	0	0.00
boat	12	0	12	1.00
unicycle	0	0	0	0.00
cart	2	0	2	0.17
bicycle	21	0	21	1.75
car	52	0	52	4.33
bus	7	0	7	0.58
airplane	13	0	13	1.08
snowplow	0	0	0	0.00
truck	10	0	10	0.83
ladle	1	0	1	0.08
cutlery	0	28	28	2.33
can opener	0	0	0	0.00
frying pan	0	0	0	0.00
plate rack	0	0	0	0.00
spatula	6	0	6	0.50
saltshaker	0	0	0	0.00
strainer	2	0	2	0.17
corkscrew	0	0	0	0.00
stethoscope	1	0	1	0.08
neck brace	0	0	0	0.00

Continues next page

Class	E	C	T	% set
LEVEL 3 (cont.)				
syringe	0	0	0	0.00
crutch	0	0	0	0.00
band aid	0	0	0	0.00
telephone	15	0	15	1.25
ipod	1	0	1	0.08
laptop	5	0	5	0.42
electric fan	0	0	0	0.00
computer keyboard	0	0	0	0.00
display	0	0	0	0.00
mouse	11	0	11	0.92
microphone	19	0	19	1.58
remote control	0	0	0	0.00
printer	0	0	0	0.00
tape player	0	0	0	0.00
digital clock	0	0	0	0.00
broccoli	6	0	6	0.50
cucumber	14	0	14	1.17
mushroom	3	0	3	0.25
head cabbage	0	0	0	0.00
artichoke	0	0	0	0.00
bell pepper	0	0	0	0.00
carrot	13	0	13	1.08
pretzel	0	0	0	0.00
doughnut	0	0	0	0.00
bagel	0	0	0	0.00
pineapple	5	0	5	0.42
lemon	4	0	4	0.33
fig	1	0	1	0.08
strawberry	3	0	3	0.25
apple	10	0	10	0.83
banana	4	0	4	0.33
orange	15	0	15	1.25
pomegranate	0	0	0	0.00
mammal	5	255	255	21.25
reptile	0	9	9	0.75
insect	4	3	5	0.42
centipede	0	0	0	0.00
amphibian	0	5	5	0.42
crustacean	0	1	1	0.08
jellyfish	0	0	0	0.00
aquatic vertebrate	0	0	0	0.00
bird	12	0	12	1.00
snail	0	0	0	0.00
starfish	0	0	0	0.00
arachnid	0	0	0	0.00
houseplant	0	0	0	0.00
LEVEL 4				
windsor tie	0	0	0	0.00
bow tie	0	0	0	0.00
cowboy hat	0	0	0	0.00

Class	E	C	T	% set
LEVEL 4 (cont.)				
maillot	0	0	0	0.00
swimming trunks	0	0	0	0.00
golf ball	0	0	0	0.00
volleyball	0	0	0	0.00
basketball	8	0	8	0.67
soccer ball	0	0	0	0.00
baseball	5	0	5	0.42
croquet ball	0	0	0	0.00
ping-pong ball	0	0	0	0.00
rugby ball	0	0	0	0.00
tennis ball	0	0	0	0.00
baseball glove	0	0	0	0.00
baseball bat	0	0	0	0.00
golfcart	0	0	0	0.00
balance beam	0	0	0	0.00
horizontal bar	0	0	0	0.00
snowboard	1	0	1	0.08
skateboard	7	0	7	0.58
aquaplane	0	0	0	0.00
bench	9	0	9	0.75
sofa	20	0	20	1.67
chair	26	0	26	2.17
dining-room table	0	0	0	0.00
drum	6	0	6	0.50
chime	0	0	0	0.00
maraca	0	0	0	0.00
cornet	0	0	0	0.00
harmonica	0	0	0	0.00
oboe	0	0	0	0.00
flute	10	0	10	0.83
sax	1	0	1	0.08
trombone	0	0	0	0.00
french horn	0	0	0	0.00
banjo	1	0	1	0.08
harp	1	0	1	0.08
guitar	53	0	53	4.42
violin	9	0	9	0.75
cello	1	0	1	0.08
piano	26	0	26	2.17
accordion	2	0	2	0.17
lipstick	1	0	1	0.08
face powder	0	0	0	0.00
wineglass	0	0	0	0.00
beaker	0	0	0	0.00
water bottle	0	0	0	0.00
wine bottle	0	0	0	0.00
fork	5	0	5	0.42
table knife	0	0	0	0.00
spoon	24	0	24	2.00
armadillo	0	0	0	0.00

Class	E	C	T	% set
LEVEL 4 (cont.)				
giant panda	0	0	0	0.00
aquatic mammal	0	6	6	0.50
monkey	37	0	37	3.08
elephant	6	0	6	0.50
rodent	2	9	11	0.92
bear	18	0	18	1.50
sheep	1	0	1	0.08
giraffe	2	0	2	0.17
canine	1	150	150	12.50
feline	0	17	17	1.42
hippopotamus	1	0	1	0.08
horse	27	0	27	2.25
koala	0	0	0	0.00
rabbit	7	0	7	0.58
swine	0	0	0	0.00
lesser panda	0	0	0	0.00
cattle	2	0	2	0.17
antelope	1	0	1	0.08
cow	5	0	5	0.42
skunk	0	0	0	0.00
zebra	4	0	4	0.33
camel	0	0	0	0.00
otter	2	0	2	0.17
snake	6	0	6	0.50
lizard	0	0	0	0.00
turtle	3	0	3	0.25
bee	1	0	1	0.08
butterfly	1	0	1	0.08
ant	1	0	1	0.08
dragonfly	1	0	1	0.08
ladybug	0	0	0	0.00
frog	5	0	5	0.42
isopod	0	0	0	0.00
lobster	1	0	1	0.08
ray	0	0	0	0.00
goldfish	0	0	0	0.00
tick	0	0	0	0.00
scorpion	0	0	0	0.00
LEVEL 5				
seal	1	0	1	0.08
whale	5	0	5	0.42
hamster	5	0	5	0.42
porcupine	1	0	1	0.08
squirrel	5	0	5	0.42
fox	6	0	6	0.50
dog	147	0	147	12.25
tiger	15	0	15	1.25
domestic cat	0	0	0	0.00
lion	10	0	10	0.83

Table A.2: Concept overlap between class tree and MSVD — (full). Full version of left side of Table 2.25 on page 153.

Class	Ext	Chd	Tot	% set
LEVEL 0				
ROOT	0	8116	8116	81.16
LEVEL 1				
artifact	1	5522	5522	55.22
food	1102	373	1312	13.12
living thing	0	5802	5802	58.02
LEVEL 2				
clothing	202	393	569	5.69
sports equip.	0	951	951	9.51
furniture	20	1367	1375	13.75
appliance	8	242	250	2.50
tool	57	46	99	0.99
scissors	14	0	14	0.14
street sign	0	0	0	0.00
musical inst.	0	358	358	3.58
toiletry	0	285	285	2.85
tableware	0	718	718	7.18
fireplug	0	0	0	0.00
parking meter	0	0	0	0.00
bag	143	168	168	1.68
vehicle	441	1518	1553	15.53
kitchen uten.	0	199	199	1.99
rubber eraser	0	0	0	0.00
medical inst.	0	7	7	0.07
teddy	3	0	3	0.03
electronic dev.	0	614	614	6.14
milk can	0	0	0	0.00
traffic light	0	0	0	0.00
binder	1	0	1	0.01
book	134	0	134	1.34
rule	2	0	2	0.02
stretcher	3	0	3	0.03
pencil box	0	0	0	0.00
pot	230	0	230	2.30
pencil sharp.	0	0	0	0.00
vegetable	29	46	72	0.72
guacamole	1	0	1	0.01
baked goods	0	6	6	0.06
trifle	0	0	0	0.00
ice lolly	0	0	0	0.00
hotdog	1	0	1	0.01
hamburger	10	0	10	0.10
burrito	3	0	3	0.03
pizza	19	0	19	0.19
fruit	46	235	268	2.68
animal	217	905	977	9.77
person	5272	0	5272	52.72
plant	50	0	50	0.50

Class	E	C	T	% set
LEVEL 3				
necktie	3	0	3	0.03
helmet	81	0	81	0.81
hat	235	0	235	2.35
diaper	6	0	6	0.06
brassiere	0	0	0	0.00
miniskirt	1	0	1	0.01
sunglasses	69	0	69	0.69
swimsuit	15	0	15	0.15
bathing cap	0	0	0	0.00
ball	553	341	674	6.74
baseball eq.	0	0	0	0.00
golf equip.	0	0	0	0.00
dumbbell	0	0	0	0.00
gymnastic ap.	0	0	0	0.00
puck	2	0	2	0.02
punching bag	0	0	0	0.00
bow	48	0	48	0.48
board	224	12	231	2.31
ski	18	0	18	0.18
sport kite	0	0	0	0.00
tennis racket	0	0	0	0.00
frisbee	2	0	2	0.02
bookcase	6	0	6	0.06
bed	218	0	218	2.18
toilet	15	0	15	0.15
lamp	5	0	5	0.05
baby bed	0	0	0	0.00
washbasin	1	0	1	0.01
seat	97	385	474	4.74
table	741	0	741	7.41
file	23	0	23	0.23
hand blower	0	0	0	0.00
stove	181	0	181	1.81
washer	2	0	2	0.02
waffle iron	0	0	0	0.00
toaster	4	0	4	0.04
dishwasher	0	0	0	0.00
coffee maker	0	0	0	0.00
oven	51	0	51	0.51
refrigerator	11	0	11	0.11
vacuum	5	0	5	0.05
microwave	11	0	11	0.11
nail	18	0	18	0.18
hammer	22	0	22	0.22
screwdriver	4	0	4	0.04
ax	4	0	4	0.04
power drill	0	0	0	0.00

Class	E	C	T	% set
LEVEL 3 (cont.)				
chain saw	0	0	0	0.00
percussion in.	0	21	21	0.21
wind inst.	0	8	8	0.08
stringed inst.	0	262	262	2.62
keyboard inst.	0	82	82	0.82
makeup	183	39	200	2.00
soap dispen.	0	0	0	0.00
cream	114	0	114	1.14
hair spray	0	0	0	0.00
perfume	4	0	4	0.04
toothbrush	1	0	1	0.01
glass	282	6	284	2.84
bowl	392	0	392	3.92
bowl	392	0	392	3.92
bottle	96	0	96	0.96
platter	10	0	10	0.10
mug	14	0	14	0.14
vase	4	0	4	0.04
pitcher	25	0	25	0.25
bag	143	168	168	1.68
purse	29	0	29	0.29
plastic bag	0	0	0	0.00
backpack	37	0	37	0.37
baggage	0	0	0	0.00
motorcycle	112	0	112	1.12
train	61	0	61	0.61
snowmobile	2	0	2	0.02
boat	134	0	134	1.34
unicycle	1	0	1	0.01
cart	37	0	37	0.37
bicycle	52	0	52	0.52
car	1004	0	1004	10.04
bus	78	0	78	0.78
airplane	123	0	123	1.23
snowplow	0	0	0	0.00
truck	168	0	168	1.68
ladle	7	0	7	0.07
cutlery	0	177	177	1.77
can opener	0	0	0	0.00
frying pan	0	0	0	0.00
plate rack	0	0	0	0.00
spatula	25	0	25	0.25
saltshaker	0	0	0	0.00
strainer	7	0	7	0.07
corkscrew	0	0	0	0.00
stethoscope	2	0	2	0.02
neck brace	0	0	0	0.00

Continues next page

Class	E	C	T	% set
LEVEL 3 (cont.)				
syringe	5	0	5	0.05
crutch	0	0	0	0.00
band aid	0	0	0	0.00
telephone	27	0	27	0.27
ipod	4	0	4	0.04
laptop	89	0	89	0.89
electric fan	0	0	0	0.00
computer key.	0	0	0	0.00
display	167	0	167	1.67
mouse	76	0	76	0.76
microphone	265	0	265	2.65
remote control	0	0	0	0.00
printer	4	0	4	0.04
tape player	0	0	0	0.00
digital clock	0	0	0	0.00
broccoli	10	0	10	0.10
cucumber	11	0	11	0.11
mushroom	11	0	11	0.11
head cabbage	0	0	0	0.00
artichoke	0	0	0	0.00
bell pepper	0	0	0	0.00
carrot	18	0	18	0.18
pretzel	1	0	1	0.01
doughnut	4	0	4	0.04
bagel	2	0	2	0.02
pineapple	3	0	3	0.03
lemon	13	0	13	0.13
fig	0	0	0	0.00
strawberry	2	0	2	0.02
apple	39	0	39	0.39
banana	9	0	9	0.09
orange	176	0	176	1.76
pomegranate	0	0	0	0.00
mammal	3	716	717	7.17
reptile	3	45	46	0.46
insect	8	20	28	0.28
centipede	0	0	0	0.00
amphibian	0	22	22	0.22
crustacean	0	7	7	0.07
jellyfish	1	0	1	0.01
aquatic vert.	0	13	13	0.13
bird	87	0	87	0.87
snail	7	0	7	0.07
starfish	4	0	4	0.04
arachnid	0	9	9	0.09
houseplant	0	0	0	0.00
LEVEL 4				
windsor tie	0	0	0	0.00
bow tie	0	0	0	0.00
cowboy hat	0	0	0	0.00

Class	E	C	T	% set
LEVEL 4 (cont.)				
maillot	0	0	0	0.00
swimming trunks	0	0	0	0.00
golf ball	0	0	0	0.00
volleyball	32	0	32	0.32
basketball	204	0	204	2.04
soccer ball	0	0	0	0.00
baseball	127	0	127	1.27
croquet ball	0	0	0	0.00
ping-pong ball	0	0	0	0.00
rugby ball	0	0	0	0.00
tennis ball	0	0	0	0.00
baseball glove	0	0	0	0.00
baseball bat	0	0	0	0.00
golfcart	0	0	0	0.00
balance beam	0	0	0	0.00
horizontal bar	0	0	0	0.00
snowboard	3	0	3	0.03
skateboard	10	0	10	0.10
aquaplane	0	0	0	0.00
bench	47	0	47	0.47
sofa	70	0	70	0.70
chair	288	0	288	2.88
dining-room table	0	0	0	0.00
drum	21	0	21	0.21
chime	0	0	0	0.00
maraca	0	0	0	0.00
cornet	0	0	0	0.00
harmonica	1	0	1	0.01
oboe	0	0	0	0.00
flute	6	0	6	0.06
sax	1	0	1	0.01
trombone	0	0	0	0.00
french horn	0	0	0	0.00
banjo	4	0	4	0.04
harp	1	0	1	0.01
guitar	250	0	250	2.50
violin	14	0	14	0.14
cello	1	0	1	0.01
piano	80	0	80	0.80
accordion	2	0	2	0.02
lipstick	39	0	39	0.39
face powder	0	0	0	0.00
wineglass	0	0	0	0.00
beaker	6	0	6	0.06
water bottle	0	0	0	0.00
wine bottle	0	0	0	0.00
fork	14	0	14	0.14
table knife	0	0	0	0.00
spoon	165	0	165	1.65
armadillo	1	0	1	0.01

Class	E	C	T	% set
LEVEL 4 (cont.)				
giant panda	0	0	0	0.00
aquatic mammal	0	15	15	0.15
monkey	71	0	71	0.71
elephant	23	0	23	0.23
rodent	17	38	42	0.42
bear	48	0	48	0.48
sheep	9	0	9	0.09
giraffe	2	0	2	0.02
canine	4	290	292	2.92
feline	4	57	59	0.59
hippopotamus	2	0	2	0.02
horse	160	0	160	1.60
koala	1	0	1	0.01
rabbit	25	0	25	0.25
swine	0	0	0	0.00
lesser panda	0	0	0	0.00
cattle	2	0	2	0.02
antelope	6	0	6	0.06
cow	24	0	24	0.24
skunk	6	0	6	0.06
zebra	6	0	6	0.06
camel	9	0	9	0.09
otter	2	0	2	0.02
snake	24	0	24	0.24
lizard	10	0	10	0.10
turtle	13	0	13	0.13
bee	8	0	8	0.08
butterfly	10	0	10	0.10
ant	3	0	3	0.03
dragonfly	0	0	0	0.00
ladybug	0	0	0	0.00
frog	22	0	22	0.22
isopod	0	0	0	0.00
lobster	7	0	7	0.07
ray	8	0	8	0.08
goldfish	5	0	5	0.05
tick	2	0	2	0.02
scorpion	7	0	7	0.07
LEVEL 5				
seal	5	0	5	0.05
whale	10	0	10	0.10
hamster	30	0	30	0.30
porcupine	2	0	2	0.02
squirrel	6	0	6	0.06
fox	35	0	35	0.35
dog	259	0	259	2.59
tiger	41	0	41	0.41
domestic cat	0	0	0	0.00
lion	28	0	28	0.28

Table A.3: Concept overlap between class tree and MSR-VTT — (full). Full version of right side of Table 2.25 on page 153.

Noun	Count	% set	Noun	#	%	Noun	#	%	Noun	#	%
man	791	65.92	girls	59	4.92	jumps	35	2.92	keyboard	24	2.00
someone	500	41.67	movie	58	4.83	cut	35	2.92	fun	24	2.00
woman	460	38.33	ingredients	57	4.75	skin	34	2.83	children	24	2.00
something	437	36.42	field	57	4.75	potato	34	2.83	spices	23	1.92
lady	386	32.17	couple	56	4.67	making	34	2.83	potatoe	23	1.92
girl	266	22.17	persons	55	4.58	body	34	2.83	kitten	23	1.92
women	260	21.67	instrument	54	4.50	stunts	33	2.75	edge	23	1.92
men	249	20.75	house	54	4.50	ride	33	2.75	butter	23	1.92
boy	245	20.42	stage	53	4.42	plastic	33	2.75	sushi	22	1.83
kichen	232	19.33	recipe	53	4.42	paper	33	2.75	stick	22	1.83
guy	218	18.17	rice	50	4.17	potatoes	32	2.67	sings	22	1.83
video	194	16.17	head	49	4.08	peoples	32	2.67	seasonings	22	1.83
chef	194	16.17	back	49	4.08	oil	32	2.67	race	22	1.83
people	173	14.42	mixture	48	4.00	bread	32	2.67	metal	22	1.83
s	171	14.25	animals	48	4.00	wall	31	2.58	mans	22	1.83
pieces	154	12.83	way	47	3.92	ladies	31	2.58	leaves	22	1.83
playing	152	12.67	guys	47	3.92	home	31	2.58	dances	22	1.83
knife	133	11.08	face	47	3.92	garden	31	2.58	cycle	22	1.83
water	131	10.92	egg	47	3.92	fish	31	2.58	cheese	22	1.83
cook	124	10.33	cuts	47	3.92	container	31	2.58	band	22	1.83
piece	121	10.08	liquid	46	3.83	walks	30	2.50	vessel	21	1.75
dish	114	9.50	half	46	3.83	walking	30	2.50	toddler	21	1.75
song	111	9.25	dance	46	3.83	thing	30	2.50	round	21	1.75
music	110	9.17	i	45	3.75	football	30	2.50	pours	21	1.75
pan	102	8.50	grass	45	3.75	dogs	30	2.50	players	21	1.75
vegetables	98	8.17	skillet	44	3.67	curry	30	2.50	lake	21	1.75
ground	97	8.08	play	44	3.67	talks	29	2.42	gun	21	1.75
kid	92	7.67	kids	44	3.67	sauce	29	2.42	foot	21	1.75
group	91	7.58	game	44	3.67	plate	29	2.42	flour	21	1.75
sort	90	7.50	camera	44	3.67	pet	29	2.42	fire	21	1.75
baby	90	7.50	cute	43	3.58	wood	27	2.25	counter	21	1.75
hand	89	7.42	area	43	3.58	strips	27	2.25	beef	21	1.75
clip	87	7.25	frying	42	3.50	place	27	2.25	beach	21	1.75
road	85	7.08	chicken	42	3.50	machine	27	2.25	background	21	1.75
cat	82	6.83	air	42	3.50	lot	27	2.25	yard	20	1.67
slices	79	6.58	scene	41	3.42	dirt	27	2.25	trick	20	1.67
room	78	6.50	onion	41	3.42	dancing	27	2.25	tree	20	1.67
kitchen	76	6.33	anyone	41	3.42	style	26	2.17	things	20	1.67
side	74	6.17	film	40	3.33	hill	26	2.17	slice	20	1.67
front	74	6.17	singing	39	3.25	finger	26	2.17	shirt	20	1.67
kind	72	6.00	onions	38	3.17	actor	26	2.17	practice	20	1.67
cooking	70	5.83	eggs	38	3.17	tv	25	2.08	picture	20	1.67
floor	69	5.75	bike	38	3.17	stunt	25	2.08	path	20	1.67
meat	67	5.58	forest	37	3.08	shot	25	2.08	motor	20	1.67
child	67	5.58	mouth	36	3.00	player	25	2.08	jack	20	1.67
boys	67	5.58	item	36	3.00	paws	25	2.08	ingrediants	20	1.67
show	65	5.42	box	36	3.00	jump	25	2.08	foods	20	1.67
hands	62	5.17	tricks	35	2.92	sheet	24	2.00	fingers	20	1.67
street	61	5.08	toy	35	2.92	rider	24	2.00	watches	19	1.58
somebody	61	5.08	sea	35	2.92	puppy	24	2.00	time	19	1.58
top	59	4.92	plays	35	2.92	legs	24	2.00	tamil	19	1.58

Table A.4: The most common nouns from MSVD which are not part of the combined dataset class tree — (full). Full version of left side of Table 2.26 on page 154.

Noun	Count	% set	Noun	#	%	Noun	#	%	Noun	#	%
man	6707	67.07	road	660	6.60	channel	376	3.76	studio	248	2.48
video	4737	47.37	clips	654	6.54	band	376	3.76	narrator	246	2.46
people	4043	40.43	water	652	6.52	cars	375	3.75	outdoors	239	2.39
woman	3977	39.77	crowd	645	6.45	tshirt	374	3.74	friend	239	2.39
someone	3035	30.35	child	632	6.32	face	372	3.72	action	238	2.38
show	2520	25.20	sports	623	6.23	building	372	3.72	items	236	2.36
something	2487	24.87	play	622	6.22	reporter	371	3.71	type	234	2.34
guy	2370	23.70	children	617	6.17	boys	354	3.54	machine	234	2.34
girl	2351	23.51	kid	610	6.10	head	351	3.51	fun	234	2.34
men	2127	21.27	street	606	6.06	others	346	3.46	court	234	2.34
clip	2074	20.74	couple	593	5.93	lot	341	3.41	track	232	2.32
screen	1971	19.71	players	585	5.85	segment	339	3.39	videos	231	2.31
talks	1965	19.65	scenes	576	5.76	area	339	3.39	night	231	2.31
group	1949	19.49	dish	570	5.70	city	337	3.37	speech	229	2.29
women	1807	18.07	singing	568	5.68	competition	331	3.31	shots	228	2.28
game	1758	17.58	animation	561	5.61	team	330	3.30	set	227	2.27
music	1679	16.79	bunch	539	5.39	review	327	3.27	presentation	226	2.26
scene	1632	16.32	interview	525	5.25	animals	327	3.27	country	226	2.26
front	1561	15.61	field	525	5.25	view	324	3.24	series	223	2.23
lady	1514	15.14	voice	521	5.21	features	321	3.21	minecraft	223	2.23
camera	1510	15.10	sings	513	5.13	way	313	3.13	cloth	220	2.20
s	1399	13.99	pictures	509	5.09	speaks	313	3.13	door	218	2.18
tv	1392	13.92	hands	500	5.00	glasses	312	3.12	grass	217	2.17
shirt	1318	13.18	suit	499	4.99	part	311	3.11	cook	217	2.17
room	1274	12.74	side	499	4.99	baby	311	3.11	box	216	2.16
song	1269	12.69	top	496	4.96	paper	310	3.10	trees	215	2.15
boy	1265	12.65	picture	488	4.88	performance	302	3.02	cartoons	215	2.15
movie	1227	12.27	persons	476	4.76	blonde	302	3.02	female	212	2.12
dress	1181	11.81	things	474	4.74	pan	298	2.98	demo	212	2.12
cartoon	1083	10.83	program	470	4.70	singer	294	2.94	stadium	211	2.11
stage	1002	10.02	place	455	4.55	demonstration	294	2.94	montage	211	2.11
kids	871	8.71	conversation	455	4.55	trailer	293	2.93	commentary	211	2.11
characters	861	8.61	recipe	454	4.54	jacket	293	2.93	performs	210	2.10
character	849	8.49	floor	450	4.50	image	293	2.93	beach	210	2.10
color	808	8.08	house	445	4.45	family	293	2.93	judges	209	2.09
background	808	8.08	dance	443	4.43	friends	289	2.89	fashion	208	2.08
television	789	7.89	match	434	4.34	wall	287	2.87	toys	207	2.07
playing	787	7.87	speaking	432	4.32	dancing	287	2.87	rock	207	2.07
footage	758	7.58	home	426	4.26	pink	286	2.86	words	206	2.06
plays	757	7.57	games	425	4.25	world	284	2.84	pieces	206	2.06
audience	754	7.54	film	421	4.21	sits	284	2.84	science	202	2.02
girls	749	7.49	time	416	4.16	language	283	2.83	item	201	2.01
kitchen	741	7.41	cooking	416	4.16	instructions	281	2.81	gun	201	2.01
hand	724	7.24	piece	413	4.13	event	280	2.80	shows	200	2.00
hair	721	7.21	male	413	4.13	work	277	2.77	back	199	1.99
guys	708	7.08	images	399	3.99	story	276	2.76	sky	198	1.98
news	698	6.98	gameplay	390	3.90	race	274	2.74	ladies	198	1.98
talk	691	6.91	blue	389	3.89	coat	274	2.74	beauty	198	1.98
player	687	6.87	ingredients	387	3.87	toy	273	2.73	slideshow	197	1.97
ground	672	6.72	advertisement	385	3.85	body	273	2.73	end	197	1.97
computer	662	6.62	chef	384	3.84	school	270	2.70	plate	195	1.95

Table A.5: The most common nouns from MSR-VTT which are not part of the combined dataset class tree — (full). Full version of right side of Table 2.26 on page 154.

References

[Aafaq et al., 2019a] Aafaq, N., Akhtar, N., Liu, W., Gilani, S. Z., and Mian, A. (2019a). **Spatio-Temporal Dynamics and Semantic Attribute Enriched Visual Encoding for Video Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 12487–12496. 37, 39, 41, 51, 62

[Aafaq et al., 2019b] Aafaq, N., Mian, A., Liu, W., Gilani, S. Z., and Shah, M. (2019b). **Video Description: A Survey of Methods, Datasets, and Evaluation Metrics**. *ACM Computing Surveys (CSUR)*, 52(6):1–37. 34

[Abu-El-Haija et al., 2016] Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., and Vijayanarasimhan, S. (2016). **YouTube-8M: A Large-Scale Video Classification Benchmark**. *arXiv preprint arXiv:1609.08675*. 202

[Anderson et al., 2016] Anderson, P., Fernando, B., Johnson, M., and Gould, S. (2016). **SPICE: Semantic Propositional Image Caption Evaluation**. In *Proceedings of the European Conference on Computer Vision*, pages 382–398. 59

[Anderson et al., 2018] Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., and Zhang, L. (2018). **Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6077–6086. 83

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). **Neural Machine Translation by Jointly Learning to Align and Translate**. In *Proceedings of the International Conference on Learning Representations*. 51, 66, 175

[Ballas et al., 2016] Ballas, N., Yao, L., Pal, C., and Courville, A. (2016). **Delving deeper into convolutional networks for learning video representations**. In *Proceedings of the International Conference on Learning Representations*. 29, 44

[Banerjee and Lavie, 2005] Banerjee, S. and Lavie, A. (2005). **METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments**. In *Proceedings of the Association for Computational Linguistics*, pages 65–72. 56

[Baraldi et al., 2017] Baraldi, L., Grana, C., and Cucchiara, R. (2017). **Hierarchical Boundary-Aware Neural Encoder for Video Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1657–1666. 37, 39, 44

[Bertasius et al., 2018] Bertasius, G., Torresani, L., and Shi, J. (2018). **Object Detection in Video with Spatiotemporal Sampling Networks**. In *Proceedings of the European Conference on Computer Vision*, pages 331–346. 30, 125

[Caba Heilbron et al., 2015] Caba Heilbron, F., Escorcia, V., Ghanem, B., and Carlos Niebles, J. (2015). **ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 961–970. 36, 38, 52, 202

[Carreira and Zisserman, 2017] Carreira, J. and Zisserman, A. (2017). **Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6299–6308. 39, 134

[Chen and Dolan, 2011] Chen, D. L. and Dolan, W. B. (2011). **Collecting Highly Parallel Data for Paraphrase Evaluation**. In *Proceedings of the Association for Computational Linguistics*, pages 190–200. 52, 202

[Chen et al., 2018a] Chen, K., Wang, J., Yang, S., Zhang, X., Xiong, Y., Change Loy, C., and Lin, D. (2018a). **Optimizing Video Object Detection via a Scale-Time Lattice**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 7814–7823. 30, 124

[Chen et al., 2019] Chen, S., Yao, T., and Jiang, Y.-G. (2019). **Deep Learning for Video Captioning: A Review.** In *IJCAI*, pages 6283–6290. 29

[Chen et al., 2020] Chen, Y., Cao, Y., Hu, H., and Wang, L. (2020). **Memory Enhanced Global-Local Aggregation for Video Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 10337–10346. 30, 125

[Chen et al., 2018b] Chen, Y., Wang, S., Zhang, W., and Huang, Q. (2018b). **Less Is More: Picking Informative Frames for Video Captioning**. In *Proceedings of the European Conference on Computer Vision*, pages 358–373. 37

[Cho et al., 2014] Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). **Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation**. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734. Association for Computational Linguistics. 51

[Chollet, 2017] Chollet, F. (2017). **Xception: Deep learning with Depthwise Separable Convolutions**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1251–1258. 133

[Dai et al., 2016] Dai, J., Li, Y., He, K., and Sun, J. (2016). **R-FCN: Object Detection via Region-based Fully Convolutional Networks**. In *Advances in Neural Information Processing*, pages 379–387. 30, 41, 100, 123

[Dai et al., 2017] Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., and Wei, Y. (2017). **Deformable Convolutional Networks**. In *Proceedings of the International Conference on Computer Vision*, pages 764–773. 125

[Dalal et al., 2006] Dalal, N., Triggs, B., and Schmid, C. (2006). **Human Detection Using Oriented Histograms of Flow and Appearance**. In *Proceedings of the European Conference on Computer Vision*, pages 428–441. 39

[Deng et al., 2019a] Deng, H., Hua, Y., Song, T., Zhang, Z., Xue, Z., Ma, R., Robertson, N., and Guan, H. (2019a). **Object Guided External Memory Network for Video Object Detection**. In *Proceedings of the International Conference on Computer Vision*, pages 6678–6687. 30, 125

[Deng et al., 2019b] Deng, J., Pan, Y., Yao, T., Zhou, W., Li, H., and Mei, T. (2019b). **Relation Distillation Networks for Video Object Detection**. In *Proceedings of the International Conference on Computer Vision*, pages 7023–7032. 30, 125

[Donahue et al., 2015] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). **Long-term Recurrent Convolutional Networks for Visual Recognition and Description**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 2625–2634. 51, 127, 136

[Dosovitskiy et al., 2015] Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., and Brox, T. (2015). **FlowNet: Learning Optical Flow with Convolutional Networks**. In *Proceedings of the International Conference on Computer Vision*, pages 2758–2766. 123, 124, 128, 130, 134, 169

[Duan et al., 2019] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., and Tian, Q. (2019). **CenterNet: Keypoint Triplets for Object Detection**. In *Proceedings of the International Conference on Computer Vision*, pages 6569–6578. 104

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). **The Pascal Visual Object Classes (VOC) Challenge**. *International Journal of Computer Vision*, 88(2):303–338. 104, 202

[Feichtenhofer et al., 2016] Feichtenhofer, C., Pinz, A., and Zisserman, A. (2016). **Convolutional Two-Stream Network Fusion for Video Action Recognition**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1933–1941. 169

[Feichtenhofer et al., 2017] Feichtenhofer, C., Pinz, A., and Zisserman, A. (2017). **Detect to Track and Track to Detect**. In *Proceedings of the International Conference on Computer Vision*, pages 3038–3046. 30, 124, 130

[Gan et al., 2017] Gan, Z., Gan, C., He, X., Pu, Y., Tran, K., Gao, J., Carin, L., and Deng, L. (2017). **Semantic Compositional Networks for Visual Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5630–5639. 37, 39, 45, 51, 83

[Ghiasi et al., 2019] Ghiasi, G., Lin, T.-Y., and Le, Q. V. (2019). **NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 7036–7045. 101

[Girshick, 2015] Girshick, R. (2015). **Fast R-CNN**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1440–1448. 100

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). **Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 580–587. 99

[Gkioxari and Malik, 2015] Gkioxari, G. and Malik, J. (2015). **Finding Action Tubes**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 759–768. 39, 124

[Han et al., 2016] Han, W., Khorrami, P., Paine, T. L., Ramachandran, P., Babaeizadeh, M., Shi, H., Li, J., Yan, S., and Huang, T. S. (2016). **Seq-NMS for Video Object Detection**. *arXiv preprint arXiv:1602.08465*. 122

[Hara et al., 2018] Hara, K., Kataoka, H., and Satoh, Y. (2018). **Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?** In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6546–6555. 39

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). **Mask R-CNN**. In *Proceedings of the International Conference on Computer Vision*, pages 2961–2969. 41

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). **Deep Residual Learning for Image Recognition**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 770–778. 29, 37, 41, 61, 62, 136, 166, 176

[Hebb, 1961] Hebb, D. O. (1961). ***The Organization of Behavior: A Neuropsychological Theory***. Psychology Press. 189

[Herdade et al., 2019] Herdade, S., Kappeler, A., Boakye, K., and Soares, J. (2019). **Image Captioning: Transforming Objects into Words**. In *Advances in Neural Information Processing*, pages 11137–11147. 51

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). **Long Short-Term Memory**. *Neural Computation*, 9(8):1735–1780. 51

[Hodosh et al., 2013] Hodosh, M., Young, P., and Hockenmaier, J. (2013). **Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics**. *Journal of Artificial Intelligence Research*, 47:853–899. 202

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**. *arXiv preprint arXiv:1704.04861*. 133, 166

[Hu et al., 2018] Hu, H., Gu, J., Zhang, Z., Dai, J., and Wei, Y. (2018). **Relation Networks for Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3588–3597. 104, 125

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). **Densely Connected Convolutional Networks**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4700–4708. 166

[Idrees et al., 2017] Idrees, H., Zamir, A. R., Jiang, Y.-G., Gorban, A., Laptev, I., Sukthankar, R., and Shah, M. (2017). **The THUMOS Challenge on Action Recognition for Videos "in the wild"**. *Computer Vision and Image Understanding*, 155:1–23. 202

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**. In *International Conference on Machine Learning*, pages 448–456. PMLR. 39

[Jhuang et al., 2013] Jhuang, H., Gall, J., Zuffi, S., Schmid, C., and Black, M. J. (2013). **Towards Understanding Action Recognition**. In *Proceedings of the International Conference on Computer Vision*, pages 3192–3199. 202

[Ji et al., 2013] Ji, S., Xu, W., Yang, M., and Yu, K. (2013). **3D Convolutional Neural Networks for Human Action Recognition**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231. 132

[Jiao et al., 2021] Jiao, L., Zhang, R., Liu, F., Yang, S., Hou, B., Li, L., and Tang, X. (2021). **New Generation Deep Learning for Video Object Detection: A Survey**. *IEEE Transactions on Neural Networks and Learning Systems*. 98

[Kang et al., 2017] Kang, K., Li, H., Xiao, T., Ouyang, W., Yan, J., Liu, X., and Wang, X. (2017). **Object Detection in Videos with Tubelet Proposal Networks**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 727–735. 123

[Kang et al., 2018] Kang, K., Li, H., Yan, J., Zeng, X., Yang, B., Xiao, T., Zhang, C., Wang, Z., Wang, R., Wang, X., et al. (2018). **T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos**. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907. 123

[Kang et al., 2016] Kang, K., Ouyang, W., Li, H., and Wang, X. (2016). **Object Detection from Video Tubelets with Convolutional Neural Networks**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 817–825. 122, 123

[Karpathy et al., 2014] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). **Large-scale Video Classification with Convolutional Neural Networks**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1725–1732. 29, 39, 62, 132, 137, 202

[Kay et al., 2017] Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., et al. (2017). **The Kinetics Human Action Video Dataset**. *arXiv preprint arXiv:1705.06950*. 39, 202

[Krishna et al., 2017a] Krishna, R., Hata, K., Ren, F., Fei-Fei, L., and Carlos Niebles, J. (2017a). **Dense-Captioning Events in Videos**. In *Proceedings of the International Conference on Computer Vision*, pages 706–715. 202

[Krishna et al., 2017b] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. (2017b). **Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations**. *International Journal of Computer Vision*, 123(1):32–73. 36, 40, 106, 202

[Krizhevsky et al., 2017] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). **ImageNet Classification with Deep Convolutional Neural Networks**. *Communications of the ACM*, 60(6):84–90. 37

[Kuehne et al., 2011] Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. (2011). **HMDB: A Large Video Database for Human Motion Recognition**. In *Proceedings of the International Conference on Computer Vision*, pages 2556–2563. IEEE. 202

[Kuznetsova et al., 2020] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., et al. (2020). **The Open Images Dataset v4**. *International Journal of Computer Vision*, pages 1–26. 105, 202

[Law and Deng, 2018] Law, H. and Deng, J. (2018). **CornerNet: Detecting Objects as Paired Keypoints**. In *Proceedings of the European Conference on Computer Vision*, pages 734–750. 104

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). **Backpropagation Applied to Handwritten Zip Code Recognition**. *Neural Computation*, 1(4):541–551. 191

[Li et al., 2018] Li, D., Qiu, Z., Dai, Q., Yao, T., and Mei, T. (2018). **Recurrent Tubelet Proposal and Recognition Networks for Action Detection**. In *Proceedings of the European Conference on Computer Vision*, pages 303–318. 30

[Lin, 2004] Lin, C.-Y. (2004). **ROUGE: A Package for Automatic Evaluation of Summaries**. *Proceedings of the Association for Computational Linguistics*. 57

[Lin et al., 2017] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). **Feature Pyramid Networks for Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 2117–2125. 101

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). **Microsoft COCO: Common Objects in Context**. In *Proceedings of the European Conference on Computer Vision*. 40, 62, 104, 202

[Liu et al., 2020] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2020). **Deep Learning for Generic Object Detection: A Survey**. *International Journal of Computer Vision*, 128(2):261–318. 98

[Liu and Zhu, 2018] Liu, M. and Zhu, M. (2018). **Mobile Video Object Detection with Temporally-Aware Feature Maps**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5686–5695. 124

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). **SSD: Single Shot Multibox Detector**. In *Proceedings of the European Conference on Computer Vision*, pages 21–37. Springer. 30, 101, 124

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). **RoBERTa: A Robustly Optimized BERT Pretraining Approach**. *arXiv preprint arXiv:1907.11692*. 70

[Lu et al., 2018] Lu, J., Yang, J., Batra, D., and Parikh, D. (2018). **Neural Baby Talk**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 7219–7228. 83

[Ma et al., 2018] Ma, C.-Y., Kadav, A., Melvin, I., Kira, Z., AlRegib, G., and Peter Graf, H. (2018). **Attend and Interact: Higher-Order Object Interactions for Video Understanding**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6790–6800. 29, 41, 46, 51

[Mansimov et al., 2015] Mansimov, E., Srivastava, N., and Salakhutdinov, R. (2015). **Initialization Strategies of Spatio-Temporal Convolutional Neural Networks**. *arXiv preprint arXiv:1503.07274*. 134

[Marvasti-Zadeh et al., 2021] Marvasti-Zadeh, S. M., Cheng, L., Ghanei-Yakhdan, H., and Kasaei, S. (2021). **Deep Learning for Visual Tracking: A Comprehensive Survey**. *IEEE Transactions on Intelligent Transportation Systems*. 98

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). **A Logical Calculus of the Ideas Immanent in Nervous Activity**. *The Bulletin of Mathematical Biophysics*, 5(4):115–133. 189

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). **Distributed Representations of Words and Phrases and their Compositionality**. In *Advances in Neural Information Processing*, pages 3111–3119. 50, 174

[Miller, 1995] Miller, G. A. (1995). **WordNet: A Lexical Database for English**. *Communications of the ACM*, 38(11):39–41. 106

[Miller et al., 1990] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. J. (1990). **Introduction to WordNet: An On-line Lexical Database**. *International Journal of Lexicography*, 3(4):235–244. 146

[Ning et al., 2017] Ning, G., Zhang, Z., Huang, C., Ren, X., Wang, H., Cai, C., and He, Z. (2017). **Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking**. In *International Symposium on Circuits and Systems*, pages 1–4. IEEE. 30, 124

[Pan et al., 2020] Pan, B., Cai, H., Huang, D.-A., Lee, K.-H., Gaidon, A., Adeli, E., and Niebles, J. C. (2020). **Spatio-Temporal Graph for Video Captioning with Knowledge Distillation**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 10870–10879. 37, 39, 41, 61

[Pan et al., 2016a] Pan, P., Xu, Z., Yang, Y., Wu, F., and Zhuang, Y. (2016a). **Hierarchical Recurrent Neural Encoder for Video Representation with Application to Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1029–1038. 29, 43, 51

[Pan et al., 2016b] Pan, Y., Mei, T., Yao, T., Li, H., and Rui, Y. (2016b). **Jointly Modeling Embedding and Translation to Bridge Video and Language**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4594–4602. 29, 37, 39, 48, 51

[Pan et al., 2017] Pan, Y., Yao, T., Li, H., and Mei, T. (2017). **Video Captioning with Transferred Semantic Attributes**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6504–6512. 29, 37, 39, 45, 51, 83

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). **BLEU: a Method for Automatic Evaluation of Machine Translation**. In *Proceedings of the Association for Computational Linguistics*, pages 311–318. 54

[Pareek and Thakkar, 2021] Pareek, P. and Thakkar, A. (2021). **A Survey on Video-Based Human Action Recognition: Recent Updates, Datasets, Challenges, and Applications**. *Artificial Intelligence Review*, 54(3):2259–2322. 98

[Pei et al., 2019] Pei, W., Zhang, J., Wang, X., Ke, L., Shen, X., and Tai, Y.-W. (2019). **Memory-Attended Recurrent Network for Video Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 8347–8356. 29, 37, 39, 48, 49

[Peng and Schmid, 2016] Peng, X. and Schmid, C. (2016). **Multi-region two-stream R-CNN for action detection**. In *Proceedings of the European Conference on Computer Vision*, pages 744–759. Springer. 169

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). **GloVe: Global Vectors for Word Representation**. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. 50

[Plummer et al., 2015] Plummer, B. A., Wang, L., Cervantes, C. M., Caicedo, J. C., Hockenmaier, J., and Lazebnik, S. (2015). **Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models**. In *Proceedings of the International Conference on Computer Vision*, pages 2641–2649. 53, 105, 202

[Qiu et al., 2017] Qiu, Z., Yao, T., and Mei, T. (2017). **Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks**. In *Proceedings of the International Conference on Computer Vision*, pages 5533–5541. 133

[Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). **Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer**. *Journal of Machine Learning Research*, 21:1–67. 70

[Ramanathan et al., 2016] Ramanathan, V., Huang, J., Abu-El-Haija, S., Gorban, A., Murphy, K., and Fei-Fei, L. (2016). **Detecting events and key actors in multi-person videos**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3043–3053. 32, 159, 160, 202

[Rashtchian et al., 2010] Rashtchian, C., Young, P., Hodosh, M., and Hockenmaier, J. (2010). **Collecting Image Annotations using Amazon's Mechanical Turk**. In *Proceedings of the Association for Computational Linguistics*, pages 139–147. 202

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). **You Only Look Once: Unified, Real-Time Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 779–788. 102

[Redmon and Farhadi, 2017] Redmon, J. and Farhadi, A. (2017). **YOLO9000: Better, Faster, Stronger**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 7263–7271. 29, 41, 102, 108, 124, 144, 146

[Redmon and Farhadi, 2018] Redmon, J. and Farhadi, A. (2018). **Yolov3: An Incremental Improvement**. *arXiv preprint arXiv:1804.02767*. 30, 102, 108, 110, 146

[Regneri et al., 2013] Regneri, M., Rohrbach, M., Wetzel, D., Thater, S., Schiele, B., and Pinkal, M. (2013). **Grounding Action Descriptions in Videos**. *Proceedings of the Association for Computational Linguistics*, 1:25–36. 32, 159, 202

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks**. In *Advances in Neural Information Processing*, pages 91–99. 29, 30, 41, 62, 100

[Rohrbach et al., 2014] Rohrbach, A., Rohrbach, M., Qiu, W., Friedrich, A., Pinkal, M., and Schiele, B. (2014). **Coherent Multi-Sentence Video Description with Variable Level of Detail**. In *German Conference on Pattern Recognition*, pages 184–195. Springer. 32, 159, 202

[Rohrbach et al., 2015a] Rohrbach, A., Rohrbach, M., and Schiele, B. (2015a). **The Long-Short Story of Movie Description**. In *Proceedings of the German Conference on Pattern Recognition*, pages 209–221. 51

[Rohrbach et al., 2015b] Rohrbach, A., Rohrbach, M., Tandon, N., and Schiele, B. (2015b). **A Dataset for Movie Description**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3202–3212. 53, 202

[Rohrbach et al., 2012a] Rohrbach, M., Amin, S., Andriluka, M., and Schiele, B. (2012a). **A Database for Fine Grained Activity Detection of Cooking Activities**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1194–1201. IEEE. 32, 159, 202

[Rohrbach et al., 2012b] Rohrbach, M., Regneri, M., Andriluka, M., Amin, S., Pinkal, M., and Schiele, B. (2012b). **Script Data for Attribute-based Recognition of Composite Activities**. In *Proceedings of the European Conference on Computer Vision*, pages 144–157. Springer. 32, 159, 202

[Rohrbach et al., 2016] Rohrbach, M., Rohrbach, A., Regneri, M., Amin, S., Andriluka, M., Pinkal, M., and Schiele, B. (2016). **Recognizing Fine-Grained and Composite Activities using Hand-Centric Features and Script Data**. *International Journal of Computer Vision*, 119(3):346–373. 32, 159, 160, 202

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). **ImageNet Large Scale Visual Recognition Challenge**. *International Journal of Computer Vision*, 115(3):211–252. 36, 40, 61, 102, 104, 106, 110, 202

[Sigurdsson et al., 2016] Sigurdsson, G. A., Varol, G., Wang, X., Farhadi, A., Laptev, I., and Gupta, A. (2016). **Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding**. In *Proceedings of the European Conference on Computer Vision*, pages 510–526. Springer. 202

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). **Two-Stream Convolutional Networks for Action Recognition in Videos**. In *Advances in Neural Information Processing*, pages 568–576. 37, 136, 169

[Singh et al., 2018] Singh, B., Li, H., Sharma, A., and Davis, L. S. (2018). **R-FCN-3000 at 30fps: Decoupling Detection and Classification**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1081–1090. 101

[Singh et al., 2016] Singh, B., Marks, T. K., Jones, M., Tuzel, O., and Shao, M. (2016). **A Multi-Stream Bi-Directional Recurrent Neural Network for Fine-Grained Action Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1961–1970. 32, 159, 160, 202

[Soomro et al., 2012] Soomro, K., Zamir, A. R., and Shah, M. (2012). **UCF101: A Dataset of 101 Human Actions Classes from Videos in the Wild**. *arXiv preprint arXiv:1212.0402*. 38, 202

[Sukhbaatar et al., 2015] Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). **End-To-End Memory Networks**. *Advances in Neural Information Processing*, 28:2440–2448. 48

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). **Sequence to Sequence Learning with Neural Networks**. *Advances in Neural Information Processing*, 27:3104–3112. 51

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2017). **Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning**. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 29, 37, 61

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). **Going Deeper with Convolutions**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1–9. 37

[Szegedy et al., 2013] Szegedy, C., Toshev, A., and Erhan, D. (2013). **Deep Neural Networks for Object Detection**. In *Advances in Neural Information Processing*, pages 2553–2561. 122, 160

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). **Rethinking the Inception Architecture for Computer Vision**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 2818–2826. 37

[Tan et al., 2020] Tan, M., Pang, R., and Le, Q. V. (2020). **EfficientDet: Scalable and Efficient Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 10781–10790. 101

[Tian et al., 2019] Tian, Z., Shen, C., Chen, H., and He, T. (2019). **FCOS: Fully Convolutional One-Stage Object Detection**. In *Proceedings of the International Conference on Computer Vision*, pages 9627–9636. 104

[Torabi et al., 2015] Torabi, A., Pal, C., Larochelle, H., and Courville, A. (2015). **Using Descriptive Video Services to Create a Large Data Source for Video Annotation Research**. *arXiv preprint arXiv:1503.01070*. 53, 202

[Toutanova et al., 2003] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). **Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network**. In *Proceedings of the Association for Computational Linguistics*, pages 173–180. 45

[Tran et al., 2015] Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). **Learning Spatiotemporal Features with 3D Convolutional Networks**. In *Proceedings of the International Conference on Computer Vision*, pages 4489–4497. 39, 62, 132

[Tran et al., 2018] Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M. (2018). **A Closer Look at Spatiotemporal Convolutions for Action Recognition**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6450–6459. 133, 134

[Uijlings et al., 2013] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). **Selective Search for Object Recognition**. *International Journal of Computer Vision*, 104(2):154–171. 99, 122

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). **Attention is All you Need**. In *Advances in Neural Information Processing*, pages 5998–6008. 44, 51

[Vedantam et al., 2015] Vedantam, R., Lawrence Zitnick, C., and Parikh, D. (2015). **CIDEr: Consensus-based Image Description Evaluation**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4566–4575. 58

[Veenman et al., 2001] Veenman, C. J., Reinders, M. J., and Backer, E. (2001). **Resolving Motion Correspondence for Densely Moving Points**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(1):54–72. 160

[Venugopalan et al., 2015a] Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015a). **Sequence to Sequence - Video to Text**. In *Proceedings of the International Conference on Computer Vision*, pages 4534–4542. 29, 37, 39, 43, 51

[Venugopalan et al., 2015b] Venugopalan, S., Xu, H., Donahue, J., Rohrbach, M., Mooney, R., and Saenko, K. (2015b). **Translating Videos to Natural Language Using Deep Recurrent Neural Networks**. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*. 29, 37, 41, 51, 127

[Vinyals et al., 2015] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). **Show and Tell: A Neural Image Caption Generator**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3156–3164. 51

[Wang et al., 2019] Wang, B., Ma, L., Zhang, W., Jiang, W., Wang, J., and Liu, W. (2019). **Controllable Video Captioning with POS Sequence Guidance Based on Gated Fusion Network**. In *Proceedings of the International Conference on Computer Vision*, pages 2641–2650. 29, 37, 39, 45, 51, 83, 85

[Wang et al., 2018a] Wang, B., Ma, L., Zhang, W., and Liu, W. (2018a). **Reconstruction Network for Video Captioning**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7622–7631. 37

[Wang et al., 2009] Wang, H., Ullah, M. M., Klaser, A., Laptev, I., and Schmid, C. (2009). **Evaluation of local spatio-temporal features for action recognition**. In *Proceedings of the British Machine Vision Conference*, pages 124–1. 39

[Wang et al., 2018b] Wang, J., Wang, W., Huang, Y., Wang, L., and Tan, T. (2018b). **M3: Multimodal Memory Modelling for Video Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 7512–7520. 29, 37, 39, 48, 49

[Wang et al., 2015] Wang, L., Ouyang, W., Wang, X., and Lu, H. (2015). **Visual Tracking with Fully Convolutional Networks**. In *Proceedings of the International Conference on Computer Vision*, pages 3119–3127. 122, 123

[Wang et al., 2018c] Wang, S., Zhou, Y., Yan, J., and Deng, Z. (2018c). **Fully Motion-Aware Network for Video Object Detection**. In *Proceedings of the European Conference on Computer Vision)*, pages 542–557. 30, 124

[Weston et al., 2014] Weston, J., Chopra, S., and Bordes, A. (2014). **Memory Networks**. *arXiv preprint arXiv:1410.3916*. 48

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). **Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation**. *arXiv preprint arXiv:1609.08144*. 175

[Xie et al., 2017] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). **Aggregated Residual Transformations for Deep Neural Networks**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1492–1500. 29, 37, 41

[Xu et al., 2016] Xu, J., Mei, T., Yao, T., and Rui, Y. (2016). **MSR-VTT: A Large Video Description Dataset for Bridging Video and Language**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5288–5296. 52, 202

[Yao et al., 2015] Yao, L., Torabi, A., Cho, K., Ballas, N., Pal, C., Larochelle, H., and Courville, A. (2015). **Describing Videos by Exploiting Temporal Structure**. In *Proceedings of the International Conference on Computer Vision*, pages 4507–4515. 29, 37, 39, 42, 51

[Yeung et al., 2018] Yeung, S., Russakovsky, O., Jin, N., Andriluka, M., Mori, G., and Fei-Fei, L. (2018). **Every Moment Counts: Dense Detailed Labeling of Actions in Complex Videos**. *International Journal of Computer Vision*, 126(2-4):375–389. 202

[You et al., 2016] You, Q., Jin, H., Wang, Z., Fang, C., and Luo, J. (2016). **Image Captioning with Semantic Attention**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4651–4659. 45

[Young et al., 2014] Young, P., Lai, A., Hodosh, M., and Hockenmaier, J. (2014). **From Image Descriptions to Visual Denotations: New Similarity Metrics for Semantic Inference over Event Descriptions**. *Proceedings of the Association for Computational Linguistics*, 2:67–78. 202

[Yu et al., 2019] Yu, J., Li, J., Yu, Z., and Huang, Q. (2019). **Multimodal Transformer With Multi-View Visual Representation for Image Captioning**. *IEEE Transactions on Circuits and Systems for Video Technology*. 29, 51

[Yue-Hei Ng et al., 2015] Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). **Beyond Short Snippets: Deep Networks for Video Classification**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4694–4702. 43, 127, 136

[Zanfir et al., 2016] Zanfir, M., Marinoiu, E., and Sminchisescu, C. (2016). **Spatio-Temporal Attention Models for Grounded Video Captioning**. In *Proceedings of the Asian Conference on Computer Vision*, pages 104–119. Springer. 51, 127

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). **Visualizing and Understanding Convolutional Networks**. In *Proceedings of the European Conference on Computer Vision*, pages 818–833. 101

[Zhang et al., 2019a] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2019a). ***Dive into Deep Learning***. 187

[Zhang and Peng, 2019] Zhang, J. and Peng, Y. (2019). **Object-aware Aggregation with Bidirectional Temporal Graph for Video Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 8327–8336. 37, 41, 51

[Zhang et al., 2019b] Zhang, Z., He, T., Zhang, H., Zhang, Z., Xie, J., and Li, M. (2019b). **Bag of Freebies for Training Object Detection Neural Networks**. *arXiv preprint arXiv:1902.04103*. 108

[Zheng et al., 2020] Zheng, Q., Wang, C., and Tao, D. (2020). **Syntax-Aware Action Targeting for Video Captioning**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 13096–13105. 29, 37, 39, 41, 45, 51, 61, 83, 85, 127

[Zhou et al., 2019] Zhou, L., Kalantidis, Y., Chen, X., Corso, J. J., and Rohrbach, M. (2019). **Grounded Video Description**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 6578–6587. 37, 41, 46, 51, 52, 127, 202

[Zhou et al., 2018] Zhou, L., Zhou, Y., Corso, J. J., Socher, R., and Xiong, C. (2018). **End-to-End Dense Video Captioning with Masked Transformer**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 8739–8748. 29, 37, 39, 44, 51

[Zhu et al., 2018] Zhu, X., Dai, J., Yuan, L., and Wei, Y. (2018). **Towards High Performance Video Object Detection**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 7210–7218. 30, 124

[Zhu et al., 2017a] Zhu, X., Wang, X., Dai, J., Yuan, L., and Wei, Y. (2017a). **Flow-Guided Feature Aggregation for Video Object Detection**. In *Proceedings of the International Conference on Computer Vision*. 30

[Zhu et al., 2017b] Zhu, X., Wang, Y., Dai, J., Yuan, L., and Wei, Y. (2017b). **Flow-Guided Feature Aggregation for Video Object Detection**. In *Proceedings of the International Conference on Computer Vision*, pages 408–417. 107, 123

[Zhu et al., 2017c] Zhu, X., Xiong, Y., Dai, J., Yuan, L., and Wei, Y. (2017c). **Deep Feature Flow for Video Recognition**. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 2349–2358. 30, 123