THE UNIVERSITY *of* ADELAIDE

Methods for Understanding and Improving Deep Learning Classification Models

by

Zhibin Liao

A thesis submitted in fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering, Computer and Mathematical Sciences

School of Computer Science

August 2017

Contents

List of Figures iii

List of Tables iv

Abstract v

Declaration vii

Publications viii

Acknowledgements ix

1 Introduction **1**

1.1 Current State-of-the-art in Training Deep Networks 2

1.2 Motivation 6

1.3 Research contribution of this thesis 8

1.4 Thesis outline 9

2 Literature Review **11**

2.1 Machine Learning 11

2.1.1 Deep Learning 12

2.2 Nonlinear Classification on Deep Learning Features 13

2.2.1 Background 14

2.3 Piecewise Linear Activation Units 16

2.3.1 Background 17

2.4 Multiple-size Features in Deep Learning 19

2.4.1 Background 19

2.5 Measuring the Performance of Residual Networks 20

2.5.1 Background 21

2.6 Conclusion 24

3 Methodology **25**

3.1 Overview 25

3.2 Datasets 26

3.3 Non-greedy Hierarchical Classification 27

3.3.1 Complexity Analysis 32
3.4 Batch-normalised Deep Learning with Piecewise Linear Activation Units 32
3.4.1 Piecewise Linear Activation Units 33
3.4.2 Batch Normalisation Units 36
3.4.3 Maxout Network in Maxout Network Model 37
3.5 Competitive Multiple-size Convolution Activation Units 39
3.5.1 Competitive Convolution of Multiple-size Filters Prevents Undesirable Filter Convergence and Filter Co-adaptation 42
3.6 Characterising the Training of Deep Residual Networks with Approximate Fisher Information Matrix 45
3.6.1 Definition of the Measures 45
3.6.2 Dynamic Sampling 49
3.6.3 Dynamic Stochastic Depth Rate 49
3.7 Conclusion 50

4 The Use of Deep Learning Features in a Hierarchical Classifier Learned with the Minimization of a Non-greedy Loss Function that Delays Gratification 51

5 On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units 58

6 A Deep Convolutional Neural Network Module that Promotes Competition of Multiple-size Filters 68

7 Approximate Fisher Information Matrix to Characterise the Training of Deep Residual Networks 99

8 Conclusion and Future Works 112
8.1 Summary of Contributions 112
8.2 Future Works 114

Bibliography 117

List of Figures

1.1 The importance of normalisation in Maxout activation function 7

2.1 Piecewise linear activation functions 17

3.1 The visualised binary tree inference procedure 28
3.2 Comparison between the proposed loss function and the "greedy" loss function 29
3.3 Shape of the proposed loss function 30
3.4 The visualised linear region division on a toy problem 35
3.5 The proposed MIM model 38
3.6 Comparison of the proposed competitive networks and the reference inception style network 41
3.7 Filter co-adaptation shown by testing the transferability of different type of networks 44
3.8 An example of using the proposed measures on the standard ResNet . . 46

List of Tables

1.1 A subset of experiment results on the Pascal VOC 2007 task 6

3.1 Mean orthogonality measure between pairs of filters 43

Abstract

Recently proposed deep learning systems can achieve superior performance with respect to methods based on hand-crafted features on a broad range of tasks, not limited to the object recognition/detection tasks, but also on medical image analysis and game control applications. These advances can be credited in part to the rapid development of computation hardware, and the availability of large-scale public datasets. The training process of deep learning models is a challenging task because of the large number of parameters involved, which requires large annotated training sets. A number of recent works have tried to explain the behaviour of deep learning models during training and testing, but the whole field still has limited understanding of the functionality of deep learning models.

In this thesis, we aim to develop methods that allow for a better understanding of the behaviour of deep learning models. With such methods, we attempt to improve the performance of deep learning models in several applications and reveal promising directions to explore with empirical evidence.

Our first method is a novel nonlinear hierarchical classifier that uses off-the-shelf convolutional neural network (CNN) features. This nonlinear classifier is a tree-structured classifier that uses linear classifier as tree nodes. Experiments suggest that our proposed nonlinear hierarchical classifier achieves better results than the linear classifiers.

In our second method, we use Maxout activation function to replace the common rectified linear unit (ReLU) function to increase the model capacity of deep learning models. We found that it can lead to an ill-conditioned training problem, given that the input data is generally not properly normalised. We show how to mitigate this problem by incorporating Batch Normalisation. This method allows us to build a deep learning model that surpassed the performance of several state-of-the-art methods.

In the third method, we explore the possibility of introducing multiple-size features into deep learning models. Our design includes up to four different filter sizes to provide different spatial pattern candidates, and a max pooling function that selects the maximum response to represent the unit's output. As an outcome of this work, we combine the multiple-size filters and the Batch-normalised Maxout activation unit from the second work to achieve the automatic spatial pattern selection within the activation unit. The result of this research shows significant improvements over the state-of-the-art on five publicly available computer vision datasets, including the ImageNet 2012 dataset.

Finally, we propose two novel measurements derived from the eigenvalues of the approximate empirical Fisher matrix which can be efficiently calculated within the stochastic gradient descent (SGD) iteration. These measurements can be obtained efficiently even for the recent state-of-the-art deep residual networks. We show how to use these measurements to help select training hyper-parameters such as mini-batch size, model structure, learning rate and stochastic depth rate. By using these tools, we discover a new way to schedule the dynamic sampling and dynamic stochastic depth, which leads to performance improvements of deep learning models. We show the proposed training approach reaches competitive classification results in CIFAR-10 and CIFAR-100 datasets with models that have significantly lower capacity compare to the current state-of-the-art in the field.

Declaration

Signed: ____________ ______________________

Date: ____

Publications

This thesis is based on the content of the following peer-reviewed conference papers, journal article, and unpublished manuscript:

- Zhibin Liao, and Gustavo Carneiro. The use of deep learning features in a hierarchical classifier learned with the minimization of a non-greedy loss function that delays gratification. In Proceedings of *the IEEE International Conference on Image Processing* (ICIP), 2015.
(DOI: 10.1109/ICIP.2015.7351666)

- Zhibin Liao, and Gustavo Carneiro. On the importance of normalisation layers in deep learning with piecewise linear activation units. In Proceedings of *the IEEE Winter Conference on Applications of Computer Vision* (WACV). 2016.
(DOI: 10.1109/WACV.2016.7477624)

- Zhibin Liao, and Gustavo Carneiro. A deep convolutional neural network module that promotes competition of multiple-size filters. Published in *Pattern Recognition* (PR), 2017.
(DOI: 10.1016/j.patcog.2017.05.024)

- Zhibin Liao, Gustavo Carneiro, Tom Drummond, and Ian Reid. Approximate fisher information matrix to characterise the training of deep residual networks. Unpublished manuscript.

Acknowledgements

First of all, I would like to express my sincere gratitude to many people who helped me during my PhD study.

I would especially like to thank my principal supervisor Assoc. Prof. Gustavo Carneiro for taking me under his visionary guidance and mentorship, who gave me freedom to explore creative ideas and guided me to cultivate them into innovative research. I would also like to thank Assoc. Prof. Gustavo Carneiro for kindly supervising my Honours study, which motivated me to pursue the PhD.

I would like to thank my co-supervisor Assoc. Prof. Tat-Jun Chin for his help and advice. I'm also grateful to the discussion with Dr. Anders Eriksson, who inspired me to look into the piecewise linear function. I would also like to extend my gratefulness to Prof. Wojciech Chojnacki for helping me with mathematical problems. I treasured many helpful discussions and other fun experiences outside research with my friends at the University of Adelaide. Many thanks to Dr. Zhi Lu, Dr. Neeraj Dhungel, Gabriel Maicas, Rafael Felix, and Toan Minh Tran.

I have been fortunate to be a member of ARC Centre of Excellence for Robotic Vision (ACRV), where I enjoyed a friendly working environment and had access to various resources. I'm sincerely grateful to deputy director Prof. Ian Reid for helping me get the centre's support to finance my study and to upgrade my GPU station. I would also like to extend my thanks to chief investigator Assoc. Prof. Stephen Gould for hosting my visit to ANU and to chief investigator Prof. Tom Drummond for hosting my visit to Monash University, from whom I learned a lot in both visits.

I would like to thank the graduate office of the University of Adelaide for providing me with the Adelaide Graduate Research Scholarships (AGRS). Finally, I'd like to thank the staff and colleges of the Australian Centre of Visual Technologies (ACVT) and the School of Computer Science for professional and ethical working environment.

At last, I would like to thank my family, especially my parents Bingxing Liao and Lili Wang, for their love and support. I would also like to thank my friends Allen Wu and Jasmine Chen, Dr. Zhi Lu and Emma Chen, Dr. William Liu, Dr. Kelly Gao, and Zhipeng Cai for making my life in Adelaide feel like at home. I would like to thank my girlfriend Diya Wang for her support and understanding.

Dedicated to my parents for their unconditional love and constant support.

Chapter 1

Introduction

One of the main challenges in computer vision tasks is how to effectively extract useful visual information from an input image or a video. Many efforts have been devoted to extract such visual information with carefully hand-crafted feature extraction processes [1–7]. However, these hand-crafted features are usually designed for solving specific tasks, which means that they may show generalisation issues extending them to different tasks. Machine learning provides an alternative path through the automatic design of features based on training a model that is adapted to a particular classification/detection/segmentation task.

Current advanced machine learning systems are mostly based on deep learning models. Deep learning has attracted huge attention lately primarily because of its superior performance [8] on large-scale image classification and object detection challenges [9]. As a matter of fact, deep learning methods are rapidly replacing more traditional shallow learning methods, such as Support Vector Machine (SVM) [10] and Boosting [11]. Over the years, deeper and wider models containing millions of parameters have been proposed, setting new limits for the current status of machine intelligence. Some of these deep learning systems have already surpassed human performance on some specific tasks [12].

A deep learning model is a learning system trained in an end-to-end manner by modelling the features and classifier simultaneously with the minimisation of a single objective function. In computer vision (CV), a deep learning model is commonly implemented as a Convolutional Neural Network (CNN), which consists of several stacks of convolution and activation layers, followed by one or more fully-connected layers and an objective expression layer, and trained with the stochastic gradient descend (SGD) optimisation method.

Recent work [13] shows that deep learning models can be easily transferred between different types of computer vision applications–for instance, a classification model pre-trained to solve visual classification problems can be fine-tuned (i.e., fine-tuning adapts the model parameters slightly from pre-learned parameter values) to solve detection and segmentation tasks [14], depth estimation problems [15], or image feature extraction tasks [16]. The popularity of deep learning can also be credited to the public availability of open-sourced deep learning libraries. Some examples of such libraries are: Caffe [17], Torch [18], Theano [19], TensorFlow [20], and MatConvNet [21].

Even though deep learning is popular at the moment, it also faces a few limitations. First, the training of a deep learning model is computationally expensive due to the large number of parameters present in the model (typically deep learning models have more than 10^5 parameters). A number of models have been proposed in recent years [22–25], where the performance is positively correlated with the model size but at a sub-linear rate. Second, a number of works have been proposed to improve the training of deep learning models, leading to a large number of model structure/components, optimisation methods, hyper-parameters, and regularisation techniques, whose effects may be co-variational and difficult to test in isolation. A simple example is that the convergence rate of a model is subject to a change in the learning rate, but training with different mini-batch sizes can cause a different convergence rate with the same learning rate, which is observed in our work in Chapter 7). Third, in the current status of deep learning, a limited amount of tools [26] can be used to analyse the training of deep learning models, in addition to monitor the objective function, where traditional learning analysis tools (e.g., the use of condition number of the Jacobian/Hessian to check the numerical stability of a function) were designed for small and convex problems, which are challenging to be used in the analysis of deep learning models.

1.1 Current State-of-the-art in Training Deep Networks

Deep learning is a broad concept, where any application that involves the use of neural networks containing a relatively large number of hidden layers (more than five [27]) can be classified as a deep learning work. Recent work [28] also points out that deep neural networks are not the only type of deep learning model. For instance, it is possible to train a deep decision forest as a deep learning model. The term deep neural network model can carry similar, but different names in the literature; for instance, the following names are used interchangeably: (very) deep networks, deep nets, and deep (learning) models.

This section summarises the recent major developments on training deep networks:

- **Increasing the depth of the model:** this trend can be observed through a series of works, starting from the Alex-net model [22], which has only 8 layers, to VGG-CNN [23], which contains 19 layers, to GoogLeNet [29], which has over 30 layers, and to the recently proposed residual networks (ResNet) [24], which contains more than 1000 layers. Meanwhile, the number of model parameters is also being pushed to the limit of the hardware capacity, leading to the necessity of multi-GPU computation. Nevertheless, recent work [25] argues that widening (in terms of increasing the number of filters in each layer) the ResNet is a more effective way of improving performance than increasing the model depth.

- **Using small size convolutional filter:** the Alex-net model [22] has 5 convolution layers, where the first layer uses 11×11 convolutional filters, the second layer uses 5×5 filters, and the remaining three layers use 3×3 filters. The OverFeat model [30] uses 7×7 convolutional filters on the first layer. The Network-in-Network (NIN) model [31] implements a micro multiple-layer perception (MLP) network to substitute a single convolution layer, where the MLP network consists a stack of 3×3 and 1×1 convolutional filters. Similarly, the VGG-CNN [23] also uses stacked 3×3 convolution layers to substitute a single layer of larger size filters, resulting in a network with only 1×1 and 3×3 filters. The reasons for using small size filters are the following: 1) a stack of small size filters is able to match the receptive field size of a large size filter but it can incorporate more nonlinear activation functions between filter layers to make the transformation more discriminative; and 2) a stack of small size filters uses a smaller number of parameters compared to a single layer of large size filter, which can help the regularisation of the training process. At the moment, most of the proposed deep networks in the literature [24, 25, 32–37] use only small size filters.

- **New architectures:** FitNet [32] is a deep network that can be trained more efficiently by the guidance of a shallower teacher network. The Highway network [33] has an implementation that allows the input signal to bypass several layers, where the access to this data flow is controlled by a sub-network. The residual network (ResNet) [24] model is similar to the Highway network, where the main difference is that the information flow (i.e., the residual connection) is not controlled by a sub-network. Several studies have tried to explain the behaviour of ResNet and the Highway networks. One view is that the ResNet behaves like an ensemble of relatively shallow networks [38]. Another point of view is the iterative approach, which characterises that the residual connection is a way of forcing the learning blocks to refine the same feature rather than computing new features [39]. The ResNet model has been explored in several papers [34–37].

- **Stochastic training:** the Dropout [40] mechanism randomly omits a portion of features from intermediate network layers to prevent feature co-adaptation, which improves the generalisation of the model. The DropConnect [41] mechanism randomly drops model weights instead of features to achieve a better regularisation. Stochastic drop scheme has been used in combination with ResNet to train ResNet with stochastic depth [42]. A integration of dropout and stochastic depth to train ResNet can be found in [43].

- **Initialisation:** the vanishing/exploding gradient problem is one of the major challenges in training deep networks. On one extreme, if the gradient vanishes in the middle of the chain rule, then the early network layers are trained at a very slow rate, which degrades the model's learning ability; on the other extreme, the gradient can also grow indefinitely during the chain rule, leading to convergence issues. Carefully initialisation schemes of weights such as "Xavier" [44] and "He" [24] can alleviate the gradient problem by regularising the scale of the weights to maintain adequate feature variance throughout the layer transitions.

- **Normalisation:** apart from the initialisation, normalisation can ensure that intermediate features are well-regulated during training. The aforementioned Alex-net uses bio-inspired local response normalisation [22]. The current widely adopted Batch Normalisation (BN) [45] computes a channel-wise normalisation based on the statistics computed from a mini-batch of training samples. BN is empirically evaluated to be effective at improving convergence rate of training and to be helpful at regularising network layers, so its use can be found in many of the recently proposed deep learning models [24, 25]. In addition, BN can reduce the model sensitivity to the initialisation method. The Layer Normalisation [46] resolves the reliance on mini-batch statistics of BN by imposing all channels in a layer to share the same normalisation terms.

- **Activation function:** the first generation of deep neural networks (e.g., LeNet [47]) used sigmoid and hyperbolic tangent activation functions. These activation functions work in a small range of input values, which can cause vanishing gradient problems because the gradient beyond the active region of the function is pushed to zero. The rectified linear unit (ReLU) [48] simply transforms the negative activation values to zero, while linearly transferring the positive values. ReLU is efficient to compute and can ease the vanishing gradient problem, so it is widely used to train deep neural networks [49]. Several variants of ReLU have been proposed, which includes Leaky-ReLU (LReLU) [50], Parametric-ReLU (PReLU) [51], and Exponential Linear unit (ELU) [52], where the common goal of these methods is to relax the constraint of ReLU to allow the existence of small negative activation values.

  Another type of activation function is piecewise linear activation function, such as the Maxout unit [53] and the Local-winner-take-all unit (LWTA) unit [54]. Compared to the ReLU family [48, 50–52], the main difference lies in the number of linear regions formed by each unit, where the ReLU functions can form exactly two regions, and the piecewise linear function can form a custom number of regions depending on the input linear pieces. The piecewise linear activation function can increase the nonlinear capacity of a computation layer in the deep model.

- **Training algorithm and training analysis:** the very deep networks are generally trained with the stochastic gradient descent (SGD) method [55], or any of its variants [56–59]. The popularity of SGD methods lies in a tolerable computation cost with acceptable convergence rate, where the second-order methods are infeasible in the context of training very deep networks. The common characteristic of the SGD variants [56–59] is to rescale the gradient estimation to avoid saddle points and to have some degree of resistance to near-singular curvature of the energy landscape. A number of recent works are devoted to the characterisation of the functionality of SGD optimisation in very deep networks, where the major goal is to investigate the convergence property of deep networks in terms of how the training proceeds around local minima and saddle points in the energy landscape [60–64]. The exact Hessian of a small network has also been studied in [65]. Furthermore, a sensitivity measurement of loss energy landscape is proposed in [26] to associate mini-batch size to the sharpness of local minima.

TABLE 1.1: A selected subset of the experimental results from our first work (See Table 1 of Chapter 4 for the complete results), where we explore the performance of a number of nonlinear classifiers in comparison with linear SVM using the off-the-shelf CNN features and tested on the Pascal VOC 2007 [66] object recognition task. MAP refers to the mean Average Precision (AP) measurement over a range of object classes.

Method	MAP
OverFeat-feature + Nonlinear SVM (RBF kernel)	73.42
OverFeat-feature + Linear SVM	74.26
OverFeat-feature + PBT [67]	71.60
OverFeat-feature + Our nonlinear classifier	**74.43**
VGG-CNN-feature + Nonlinear SVM (RBF kernel)	80.73
VGG-CNN-feature + Linear SVM	80.54
VGG-CNN-feature + PBT [67]	77.52
VGG-CNN-feature + Our nonlinear classifier	**81.05**

1.2 Motivation

This thesis proposes several methods that contribute to advance the understanding of deep networks and improve the art of training deep networks with the specific goal of solving the visual classification problem. The primary motivation of focusing on the visual classification problem comes from the popularity of reusing the pre-trained Imagenet [9] classification model for: 1) extracting off-the-shelf features from images [16]; and 2) initialising the parameters of a new learning model for other image datasets and/or applications [13]. One common observation about pre-trained models is that the classification performance on the original ImageNet task is strongly tied to the performance of the target task. For instance, in Table 1.1, we show the result of using CNN features, from the OverFeat [30] and VGG-CNN [8] models, and different classifiers on the Pascal VOC 2007 [66] object recognition task. The observation is that the VGG-CNN [8] feature is more informative than the OverFeat [30] feature to solve the VOC task. This relationship is consistent with the performance on ImageNet [9], where the classification error of the OverFeat model is 14.7% and VGG-CNN model is 13.1% (measured with the top-5 classification error on the validation set). A conclusion that can be made from the discussion above is that improving the classification performance of a model can also make this model more useful to solve other computer vision problems.

We notice in previous works [8, 16, 30, 68–71] that CNN features are commonly used with linear classifiers, such as linear SVM [10] or softmax classifier, where the nonlinear

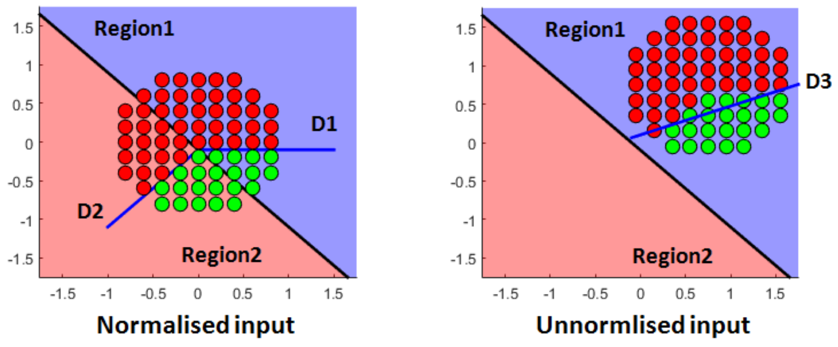


FIGURE 1.1: A rank-2 ($k = 2$) Maxout [53] processing two different kind of inputs: normalised (left hand side) and unnormalised (right hand side). In both cases, the coloured point cloud shows an identical two class nonlinear classification problem. The normalised input in the left hand side figure can ensure that the samples are more evenly distributed across the activation regions (Region 1 and Region 2), where the decision boundary in each region(D1 in Region 1, and D2 in Region 2) can solve a reduced linearly separable problem. On the other hand, the unnormalised input samples in the right hand side figure are placed in just one of the activation regions, resulting in a sub-optimal decision boundary D3.

classifiers have not been explored. This issue may have happened due to the large dimensionality of the CNN features, where the high computational cost of nonlinear classifiers cannot be ignored even if it can achieve better performance in principle. This motivated our first work (in Chapter 4) to search for a low cost nonlinear classifier that can offer larger capacity than linear classifiers.

In our second work, we propose the combination of Batch Normalisation (BN) [45] and Maxout [53] activation function. During our preliminarily work, we found that piecewise linear activation functions (e.g., ReLU [48] or Maxout [53]) can degenerate into a linear activation function when some of the linear pieces (regions) are inactive (i.e., no inputs are available in those regions), leading to an ill-conditioned training and a reduction of model capacity. We studied the possibility of fixing this issue by normalising training samples so that each linear piece remains activate, which motivated us to search for a normalisation method in our second work. The importance of the use of normalisation in the Maxout activation function is illustrated in Figure 1.1.

In our third work, we explore the use of multiple-size filters in deep CNNs in contrast to the state-of-the-art trend of using only small fixed-size filters [24, 25, 33–37]. The motivation to explore multiple-size filters is to improve the robustness of deep convolution neural networks (CNNs) to the scale changes of object appearances. Our work is inspired by the GoogLeNet [29] inception module, which shows outstanding results in the ImageNet ILSVRC 2014 [9] challenge. One particular design of the inception module is the concatenation of multiple-size features, which leads to a massive growth of the number of channels of the output tensor. The consequence of this is that the next module has to increase the number of parameters to process the large input tensor. Therefore, the inception module implements a channel reduction sub-module [31] in order to reduce the size of input tensor before extracting multiple-size features. This issue motivated us to study alternative methods to integrate the multiple-size filters in CNNs.

The understanding of training procedures for deep learning training is difficult because of the large number of parameters and many empirically selected hyper-parameters involved in the training. One possible way to improve the understanding is the development of useful measurements that can be used to provide reliable guidelines on hyper-parameter selection. In our fourth work, we introduce a novel methodology for characterising the SGD [55] training of ResNet [24] with respect to four types of hyper-parameters by examining the eigenvalues of the approximate Fisher matrix. Our motivation to look into the eigenvalues of this matrix primarily comes from [26], where a numerical experiment is proposed to measure the convergence of neural networks by sampling the largest eigenvalue of the objective function within a small neighbourhood.

1.3 Research contribution of this thesis

In general, this thesis contributes to the field with a better understanding of the training of deep learning models, and reveal promising directions to explore with empirical evidence.

Our first method is a novel nonlinear hierarchical classifier designed to work with off-the-shelf CNN features [16]. This nonlinear classifier is a tree-structured classifier that uses logistic regression classifiers in the internal nodes of the tree and linear SVM [10] as the leaf classifiers. The main novelty of our approach is the loss function to learn the hierarchical classifier, which minimizes the classification error in a non-greedy way and at the same time delays hard classification to nodes further down the tree. Experiments suggest that our proposed nonlinear hierarchical classifier achieves better results than the linear SVM classifier and a group of nonlinear classifiers including kernel SVMs.

Our second work is a continuation of our first work, where we reformulate our nonlinear classifier as a piecewise linear activation function to be integrated into deep models to achieve end-to-end training. This activation function is able to express a richer set of nonlinearities than the commonly used ReLU [48] activation function. During the development of this method, we found that our proposed activation unit can be replaced by a Maxout [53] unit that can be trained faster with a comparable performance. Nevertheless, the increasing of capacity can introduce ill-conditioning given that the training data is not properly normalised. We fix this issue with a Batch Normalisation [45] unit inside the Maxout unit, resulting in a better pre-conditioning of the unit. This method allows us to build a deep learning model that surpassed the performance of several state-of-the-art methods.

In our third work, we propose a new module for deep CNNs composed of convolutional filters of up to four filter sizes that are joined by a Maxout activation unit (i.e., channel-wise max pooling), which can promote the competition amongst these filters. In addition, the competition promoted by the max pooling does not incur the channel growth issue as in the Inception Module [29]. The result of this work shows significant improvements over the state-of-the-art on five commonly used computer vision datasets, including the large-scale ImageNet [9] dataset.

Finally, we propose two novel measurements derived from the eigenvalues of the approximate empirical Fisher matrix which can be efficiently calculated within the SGD [55] iteration. These measurements can be obtained efficiently even for the recent deep residual network (ResNet) [24] models. We show how to use these measurements to help select training hyper-parameters such as mini-batch size, model structure, learning rate and stochastic depth rate. By using our measurements to analyse the ResNet training, we discover a new way to schedule the dynamic sampling and dynamic stochastic depth which leads to improved performance while maintaining a moderate model size (in terms of the number of parameters). We show the proposed training approach reaches competitive classification results in CIFAR-10 [72] and CIFAR-100 [72] datasets to other models that have significantly larger capacity.

1.4 Thesis outline

In Chapter 2, we first give an overview of machine learning and deep learning, followed by a review of the related literature with respect to our proposed methods. In Chapter 3, we give a detailed explanation to our proposed methods. In Chapter 4, we show the application of our nonlinear hierarchical classifier using off-the-shelf CNN features. In

Chapter 5, we explain the importance of normalisation in piecewise linear activation units, especially in the Maxout unit. In Chapter 6, we discuss the use of multiple-size filters in CNNs that allow the extraction of multiple-size features. Two measures to characterise the training of ResNets with the use of the approximate Fisher matrix are presented in Chapter 7. Finally in Chapter 8, we conclude our thesis and discuss possible future works.

Chapter 2

Literature Review

In this chapter, we review the main techniques and advancements in training deep networks. We start with a brief overview about the field of machine learning and deep learning in Section 2.1. We conduct a survey of the advantages and disadvantages of various classification techniques in Section 2.2, particularly targeting their use with convolutional neural networks (CNN) features. In Section 2.3, we show the importance of normalisation in the piecewise linear activation unit in order to motivate our method in Section 3.4. In Section 2.4, we exploit the gap in the use of multiple-size filters in current deep learning models. Finally, we show a number of works in Section 2.5 that aim to explain the training of deep learning models and present their advantages and disadvantages.

2.1 Machine Learning

The term *Machine Learning* was first coined in 1959 [73], and it is a field of study in the computer science discipline that strives to endue computers with human-like learning ability without explicit programming [73]. Almost six decades later, machine learning systems are able to beat human competitors on a number of challenges, such as the recent victory of the AlphaGo Go program against world top-ranked human players [74].

Machine learning falls into three major fields: supervised learning, unsupervised learning, and reinforcement learning.

- **Supervised learning** is the most common problem explored in machine learning, where the goal is to build a statistical relationship between the input and a given target from a number of labelled training data. One of the primary applications of supervised learning is the classification task, which is the main focus of this thesis. A well-known challenge in supervised learning is that the learning method can overfit the training set. However, the generalisation ability of the model can always be improved by learning from a large annotated training set. In this sense, the recent successes achieved by many deep learning applications can be credited to the availability of databases with massive training data, such as ImageNet [9] and WordNet [75].

  Unsupervised learning differs from the supervised learning because it aims to estimate the distribution of an unlabelled training data. Early applications of unsupervised learning concentrated on statistical data analysis, such as clustering (e.g., K-means [76]) or signal separation (e.g., PCA [77]). In addition, unsupervised feature learning can estimate a feature embedding through the modelling of a transformation that maps the input data to itself (e.g., Auto-encoder [78, 79]). A recent application of unsupervised feature learning is the generative adversarial network [80], which can generate realistic synthetic images that are perceptually similar to the original training data.

  Reinforcement learning consists of a learning process that attempts to interact with a dynamic environment through a series of actions with the objective of maximising a pre-defined reward. Recent reinforcement learning works involve game control [74, 81], image generation [82] and object recognition [83, 84] domains.

2.1.1 Deep Learning

Deep learning is a broad concept, where any application that involves the use of neural networks containing a relatively large number of hidden layers (more than five [27]) can be classified as a deep learning work. The origin of using neural networks in visual recognition can be traced back to Rosenblatt's Mark I perceptron system [85] in 1957. The neural connectively pattern found by Hubel and Wiesel [86] inspired the early developments of the CNNs, from Neocognitron [87] to LeNet-5 [47], leading to the popularity of neural networks at the end of the last century. In particular, the development of the CNNs [47, 88–90] were not possible without the introduction of the back-propagation algorithm [91].

A CNN model is a high-capacity multiple layer nonlinear classifier, which is commonly viewed as a convolutional feature learning model followed by a simple linear classifier. An important assumption is that the feature learning model (i.e., the convolutional filters) is to be learned from the training data. However, due to relatively small sizes of common object recognition datasets until 2010, CNN models were hard to train, and therefore neglected by the computer vision community. At the same time, the Support Vector Machine (SVM) [10] was favoured by the visual recognition field, which led to a considerable interest in hand-crafting relevant image features. Therefore, the advancing of image understanding in the first decade of the 21st century can be credited to the development of hand-crafted image representation, from earlier SIFT [1], HOG [2], bag-of-word presentations [3–5] to the more recent Fisher Vector [6] and deformable part models [7]. A common characteristic of these representations is that they are carefully designed low-level local image-patch based descriptors. It is arguable that the hand-crafted feature descriptors are functionally comparable to the convolutional filters in the first layers of a CNN [70],

With the rise of GPU computation and the appearance of large-scale datasets [9, 75], most of the current decade has witnessed a stunning performance improvement of deep CNNs in a wide range of visual tasks [30, 68–71]. Such resurgent popularity of CNN has happened after the Alex-Net CNN model [22] won the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [9], showing significantly better results than the competing methods. More recent deep CNN models commonly have between ten [8, 23, 25, 33, 34] to a thousand convolution layers [24], involving millions of parameters. It can be expected that the development of computation hardware may allow much deeper models to be designed.

2.2 Nonlinear Classification on Deep Learning Features

The current mainstream methodology to classify a new image dataset is based on the transferring of a pre-trained CNN model [16], where the features extracted from this model are then used to represent the images from the new dataset, and this representation is then used to train a linear SVM [10] or a softmax classifier. While the linear classifiers can be efficiently trained and tested, the problem is that their low capacity offers limited accuracy. On the other hand, the shallow high-capacity classifiers, such as Boosting [11] and kernel SVMs [10], can handle more difficult classification problems than linear classifiers but at the cost of a high runtime complexity, a large memory footprint, and the risk of overfitting. It may be possible to achieve a good balance between these two extremes with hierarchical nonlinear classifiers that are built with low capacity classifiers. In this section, we first survey the works that use CNN features with linear classifiers, followed by a discussion of the advantages and disadvantages of shallow and hierarchical nonlinear classifiers in order to motivate our proposal in Section 3.3.

2.2.1 Background

The use of CNN as a feature extractor machine has been explored by many works [8, 16, 30, 68, 70]. Oquab et al. [70] show that the image representation learned by CNN from a large-scale dataset can be transferred to other vision tasks containing smaller amount of training data and produce state-of-the-art results. In this method, the parameters of the convolution layers of a pre-trained model are retained, while the parameters of the (fully-connected) classification layers are reinitialised. This reinitialised model is then trained on the target dataset, which shows better performance than training a new model from scratch. Razavian et al. [16] point out that the CNN mid-level activations can be used off the shelf, where only a simple classifier needs to be trained on the target task to obtain state-of-the-art performance. Chatfield et al. [8] show that the off-the-shelf CNN features from the VGG-CNN model (a much deeper model) can produce much more accurate classification result than the previous bag-of-features [3–5] and Fisher vector representation [6]. Two characteristics of the CNN features may explain why linear classifier is favoured in the above works. The first characteristic is that the CNN model is a combination of a complex feature extraction model and a simple classification model. The feature model is trained with a large amount of image data from the ImageNet dataset [9], making it less prone to overfitting. This results in a sparse localisation of data points in the feature space, which is suitable to be used with linear classifiers. The second characteristic of CNN features is that the dimensionality of the CNN features is usually very high (i.e., 4096 dimensions), causing the nonlinear classifiers to be less attractive due to high memory and runtime complexities. Nevertheless, Yosinski et al. [13] have pointed out that if the dissimilarity between the source and target tasks is large, the performance benefits of transferring features decrease. In this case, our hypothesis is that the transferred features are no longer linearly separable. Therefore, a nonlinear classifier could be used to address the nonlinearity caused by the cross task dissimilarity.

We begin the introduction of nonlinear classifiers with the following models: Boosting [11] and kernel SVMs [10]. The Boosting [11] classifier is based on the principle that a strong classifier can be made by a combination of weak classifiers, where individual weak classifiers perform better than random guess [92]. The commonly used AdaBoost [11] method iteratively adds weak classifiers to build a strong classifier, where each new classifier focuses on correcting the mistakes made by the strong classifier from the previous iteration. Assuming that the training of each weak classifier, e.g., a single level decision tree, involves the estimation and comparison of low capacity classifiers in all possible feature spaces, where the feature space has D dimensions and N training samples are available, the complexity of training each weak classifier is $O(DN)$. On the other hand, if one uses all features to build a decision tree for each weak classifier, then the runtime complexity of the full training process is $O(D^2N)$ and the testing runtime complexity is $O(D)$ for each weak classifier. Therefore, we can see that the main obstacles in the use of the Boosting classifier with CNN features is the runtime complexity that can grow quadratically with the feature dimensionality. Kernel SVM [10] projects all the training samples to an infinite-dimensional kernel space, which defines the nonlinear transformation. Two commonly used kernel functions are the polynomial function and the radial basis function. In the kernel space, a linear SVM classifier is built to fulfil the classification task. Due to the computational cost of the kernel, the training has runtime complexity $O(DN^2)$ and a testing runtime complexity $O(DN)$–this means that the most prominent constraint to train a nonlinear SVM with CNN features is the number of images in the training set. For a linear SVM, the training runtime complexity is $O(DN)$ while testing is $O(D)$. It is important to note that, overfitting may happen to both types of classifiers given their large capacities, which can be alleviated with regularisation techniques (e.g., L1/L2 regularisation).

We emphasise that the issue with shallow nonlinear classifiers is the fact that they have a potentially large capacity that may result in training overfitting. To address this issue, many works have proposed the use of hierarchical classifiers that distribute the nonlinear capacity into several steps. The boosted cascade classifier [93] implements a degenerate binary tree with each node being a Boosting classifier. The nodes that appear early in the cascade filter out easily distinguishable cases with a small subset of features, and delay the hard cases to nodes that show later in the cascade. The advantage of the boosted cascade is a quick training and testing processes, especially when the image dataset has imbalanced positive and negative training sample distributions. However, the disadvantage of the boosted cascade is that it requires manual setup to control the false-positive ratio and accuracy for each cascade level. The probabilistic boosting tree

(PBT) [67] is a full binary tree structured classifier that also uses a Boosting classifier in each node. Nevertheless, PBT handles hard cases (in terms of the probability of classification) by sending them to both the positive and negative branches for re-classification. Note that both methods use Boosting classifiers as tree nodes, hampering their generalisation abilities and increasing the training and testing costs. The discriminative learning of relaxed hierarchy (DLRH) [94] is another binary tree classifier, where each node has a nonlinear SVM classifier. This method is specifically designed for handling multi-class classification problems. The gist of this method is based on a class-wise relaxation, where the tree nodes recursively divide separable classes into two branches, and the remaining classes that could not be properly classified are carried over to both branches until the multi-class problem is reduced to a set of binary classification problems and the classification only happens at the leaf nodes. The intermediate nodes of DLRH are not able to produce classification results directly like the cascading classifier and PBT. The common characteristic amongst the above approaches is the use of high capacity decision rules at the tree nodes, where the shallow classifiers have high runtime complexity as mentioned above, which makes the hierarchical classifiers not ideal to be used with the CNN features. We also point out that the common weakness of the above methods is that any mistakes made by one of the early nodes in the tree cannot be recovered by the nodes down the tree.

We describe the proposal of our nonlinear classifier in Section. 3.3, which is designed to address the high runtime complexity issue of the hierarchical classifiers with the use of linear classifiers as tree nodes, i.e., we use linear logistic regression classifiers as intermediate tree nodes and linear SVMs as leaf nodes. Moreover, the tree nodes in our classifier are trained by a loss function that minimises the classification error in a non-greedy way, which postpones hard classification problems to further down the tree. Furthermore, our training method is an iterative approach, which is able to re-configure the tree structure based on the classification performance.

2.3 Piecewise Linear Activation Units

Activation functions of deep learning models have been developed in order to introduce nonlinearities to the model and to handle vanishing gradient problems. Currently, piecewise linear functions [27, 48, 50, 51, 53, 54, 95] have been widely used as activation units in state-of-the-art deep learning models. It has been shown in [27, 96] that the multiple layers of piecewise linear activation units can increase model capacity by

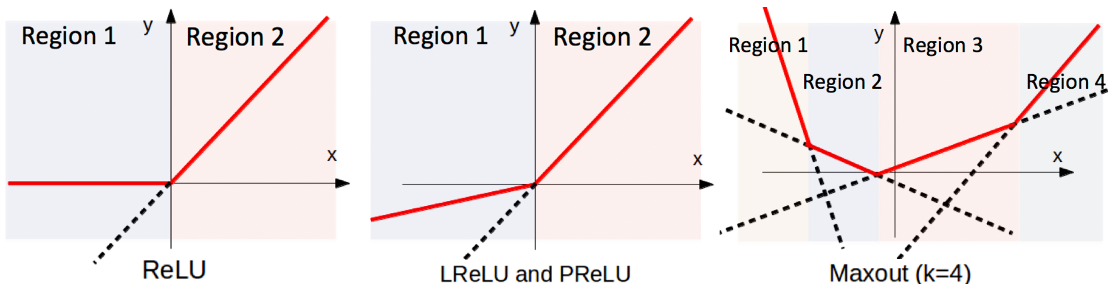


FIGURE 2.1: Piecewise linear activation functions: ReLU [48], LReLU [50], PReLU [51], and a rank-4 ($k = 4$) Maxout [53].

breaking the input space into an exponential number of activation regions. However, the main issue is that the exponential model capacity may be reduced if many of these activation regions are inactive during training and inference, which wastes model capacity. Moreover, this issue also leads to ill-conditioned training because the reduced piecewise linear activation functions act like linear activation functions that can either diminish or amplify the gradient during back-propagation. In this section, we first convey the importance of the model capacity issue, followed by a discussion of current methodologies that deal with this issue to motivate our proposal in Section 3.4.

2.3.1 Background

The piecewise linear activation units introduce model nonlinearities through a division of input space into several linear regions, where a linear decision boundary is placed within each region. It has been found in [27] that a multiple layer composition of piecewise linear units allows an exponential division of the input space with respect to the number of the layers. Another common trait of the piecewise linear units found by [96] is that they promote local competition amongst the activation regions within the unit to activate for each training sample. Therefore, the modelling power of deep learning methods that rely on piecewise linear units can be described as the ability to activate different linear regions (sub-networks) for different patterns and also to be able to activate similar regions to similar patterns. In addition, these sub-networks share their parameters, so even though these sub-networks are formed and trained with a small number of training samples, they are not prone to overfitting, resulting in an implicit regularisation of the training process [27, 96].

To efficiently utilise the model capacity, an important assumption is that a large proportion of the sub-networks should be utilised by the data samples during training and inference, while each data sample should active relatively small portions of the network. This means that in a piecewise linear unit, each linear activation region should contain sufficiently large number of training points. For example, the most widely used rectifier linear unit (ReLU) [48], and variants (Leaky-ReLU [50], Parametric-ReLU [51], and Randomised Leaky-ReLU [95]) maintain two set of points, one locating on the negative side and another in the positive side of the activation function domain (marked as region 1 covering the negative side and region 2 on the positive side in ReLU and {P,L}ReLU cases in Figure 2.1). In addition, in the Maxout [53] and the Local-winner-takes-all (LWTA) units, k sets exist to cover the k regions of the activation function domain (see the Maxout case in Figure 2.1). If the training samples do not fill the domain covered by these k regions, a piecewise linear activation unit is degenerated to a linear function, localising all points in one region. A large proportion of activation units in a learning model acting like linear units reduces the ability to form an exponential number of activation regions in the input space, hence reducing the model capacity. Meanwhile, the degeneration of the piecewise linear unit can also lead to ill-conditioned training. Taking the ReLU unit as an example, let us assume that the negative region of the activation domain contains all points, causing this unit being inactive (i.e., the "dying ReLU" problem [50]) and the training gradient to vanish; or if we assume that the positive region contains all points, resulting in a linear shifting of training point distribution that affects the magnitude of the gradient as all training points produce a positive gradient, which can potentially become too large. Either way, the aforementioned issue can lead to an ill-conditioned training

The "dying ReLU" issue has been addressed by the variants [50, 51, 95] to allow a weak activation in the negative region so that a small non-zero gradient pushes ReLU out of the inactive state. Batch Normalisation [45] implements a normalisation in between the linear function and the ReLU activation, which can help keeping a balanced point distribution for the two activation regions of ReLU. The same issue has been acknowledged by Goodfellow et al. [53] for the Maxout unit, where the dropout [40] unit has been proposed to regularise the input to the linear pieces. Nevertheless, dropout may keep input points closer to the origin but cannot guarantee a more balanced distribution of the points in terms of the activation function domain. Furthermore, dropout is a regularisation technique that does not pre-condition the model. In conclusion, these issues mentioned above remain with dropout. These problems can be addressed by allowing a more balanced distribution at the input of the piecewise linear units. In this sense, we believe that the Batch Normalisation unit should be able to pre-condition the Maxout activation unit, which leads to our proposal of batch-normalised Maxout activation unit described in Section 3.4.

2.4 Multiple-size Features in Deep Learning

State-of-the-art CNNs commonly use convolution layers containing filters of small size for reducing computation and regularising training [24, 25, 33–37], at the cost of losing the ability to discover multi-scale features within the same layer. The use of multi-scale features in CNNs has been studied before [15, 29, 97–99], but it is still a challenging topic. In this section, we introduce the current methodologies that use multi-scale features.

2.4.1 Background

The scale of the feature is an important factor in designing traditional hand-crafted image representation, where the common framework is to extract features from multiple scales that are combined by a pooling operation. On the other hand, the generic CNN architecture is not designed to take the scale of features into account, especially the current state-of-the-art networks that rely on uniform convolutional filter sizes which potentially reduce the ability of the model to find features at various scales. In order to add multi-scale information, Eigen et al. [15] implement a multi-modal network, where each model use different architecture that involves the use of different filter sizes, number of layers, and pooling operations. Another approach is to use the spatial pooling architecture to extract multi-scale information from feature maps, such as the Spatial Pyramid Pooling [98] and the Multi-scale Orderless Pooling [97]. The common problem with the above methods is that the multi-scale feature is hand-designed, and is usually extracted at a quite coarse scale.

A more relevant approach is the use of multiple-size filters to discover the multi-scale features. Based on a neuroscience model [100, 101], Serre et al. [102] proposed the use a group of multi-size Gabor filters to handle features at multiple scales, followed by a max-pooling operation. The recent GoogLeNet [29] model implements the inception module, which concatenates a set of multiple-size features. One extension from [102] to [29] is a much deeper model with learn-able filters, where the model in [102] has only two layers while the GoogLeNet architecture has 22 layers. However, the concatenation design of the inception module can cause a massive growth in the number of output channels of the module, resulting in an undesirable over-parametrisation of the module. This issue has been acknowledged by Szegedy et al. [29] with the use of additional 1×1 convolutions (originally used as a layer that reduces computation cost, allowing for an increase in the number of layers of the Network-in-Network (NIN) model [31]) before the multiple-size layers to reduce the input dimensionality of the filters.

It should be pointed out that the basic structure of the Maxout unit [53] (i.e., max-pooling a group of convolution features, mentioned in Section 2.3) resembles the aforementioned neuroscience-inspired hierarchical model [102], where the main difference is that the Maxout unit does not take the multi-size filters into account. Therefore, we follow this intuition to integrate these two methods in one CNN module: the multiple-size Maxout unit. As a variation of Maxout unit, this module is also a competitive piecewise linear activation unit (described in Section 2.3), where we adapt the same technique to solve the model capacity and ill-conditioned training issues that potentially could happen to this unit. That is, we use Batch Normalisation to normalise the output of the the multiple-size filter responses. We call this module Competitive Multiple-size Convolution activation unit, where the details of this unit are described in Section 3.5.

2.5 Measuring the Performance of Residual Networks

The deep residual networks (ResNets) [24, 42, 103] are showing extremely accurate performance over a broad range of visual classification tasks. ResNets (similarly to most deep learning models) are commonly trained with the stochastic gradient descent (SGD) method [55], or any of its variants [56–59]. The reason behind the wide use of SGD methods is that they can produce robust training results in terms of good convergence and generalisation, and at the same time requires a relatively low run-time and memory costs. However, it is commonly the case that behind the success of the ResNet or any of the state-of-the-art deep learning models is a set of carefully selected generic training hyper-parameters, such as the mini-batch size, learning rate, model structure, and a set of model parameters, such as the stochastic depth drop rate (introduced by [42] for regularising the ResNet). However, current methods available to train deep learning models present no reliable guidelines on how to select these hyper-parameters mostly because classical training analyses need to look at the second order information from the Hessian, which is computationally hard to be obtained from state-of-the-art deep learning models, such as the ResNet model. In this section, we first introduce the SGD [55] methods, followed by an overview of the second order (approximated) Hessian methods [55, 104–107] and the recent popular scaled gradient iterations methods [56–59]. Finally, we discuss approaches [26, 60–63, 65] that use numerical experiments to measure key aspects of SGD training.

2.5.1 Background

SGD [55] is a common first-order iterative optimisation method that has wide usage in deep neural networks training. One of the main goals of SGD is to find a good balance between the stochastic approach (i.e., using one random training sample to estimate the gradient in each iteration, where the speed is faster but gradient estimation is inaccurate) and the batch approach (i.e., using the full training set in each iteration, which is slow but accurate) to provide a favourable trade-off with respect to per-iteration costs and expected per-iteration improvement in minimising an objective function. The popularity of SGD in deep learning lies in the tolerable computation cost with acceptable convergence speed, where second order methods are less-favoured due to the large number of parameters in deep learning models.

Second-order methods are designed to improve the convergence speed of first order methods by re-scaling the gradient vector in order to address the high nonlinearity and ill-conditioning issues of the objective function. In particular, Newton's method uses the inverse of the Hessian matrix for rescaling the gradient vector. This operation has complexity $O(N^3)$ and the storage of Hessian is $O(N^2)$ (where N is the number of model parameters, which is usually between $O(10^6)$ and $O(10^7)$ for modern deep learning models), making the method infeasible. Besides, the Hessian must be positive definite for Newton's method, which is not a reasonable assumption for the training of deep learning models.

In order to avoid computational overload, several approximate second-order methods have been developed. For example, the Hessian-free conjugate gradient (CG) [108] is based on the fact it only needs to compute Hessian-vector products, which avoids the explicit computation of the Hessian and can be efficiently calculated with the $\mathcal{R}$-operator [109] at a comparable cost to a gradient evaluation. This Hessian-free method has been successfully applied to train auto-encoder networks [110]. However, as pointed in [55], the Hessian-vector product is at least as expensive as a gradient evaluation, so the advantage of the accurate CG gradient estimation may be shrouded by the per-iteration overhead, rendering such method uncompetitive compared to the SGD method or limited memory BFGS (L-BFGS) [105] method. In [111], a stochastic Hessian-free CG method has been proposed to reduce the computation by using a smaller sample size. Nevertheless, Hessian-free CG methods have only been applied to medium size networks, which means that the effect on large scale models remains unclear.

Quasi-Newton methods (e.g., the BFGS [55, 104]) take an alternative route and approximate the inversion of the Hessian with only the parameter and gradient displacements in the past gradient iterations. Nevertheless, the computation of this approximation matrix is also infeasible in large optimisation problems, where the L-BFGS [105] method is proposed to reduce the memory usage. However, the L-BFGS method is not widely used in the deep learning context due to two main reasons: 1) similarly to the Hessian-free CG, the accurate gradient estimation cannot justify the high per-iteration cost; 2) small sampling size can help reduce the cost but leads to poor Hessian approximation [112, 113].

The (Generalised) Gauss-Newton method [106, 107] approximates Hessian with the Gauss-Newton matrix. The approximation of the Gauss-Newton matrix requires the multiplication of the Jacobian matrix to its transpose, whose row elements have already been computed as a per-sample gradient during the SG iteration, which makes it practically free to obtain. However, this is not always true in training deep learning networks with back-propagation, where the mini-batch gradient is implicitly averaged by underling deep learning libraries. Another approximate second-order method is the natural gradient method [114], which uses the inverse of the Fisher information matrix to make the search quicker in the parameters that have less effect on the decision function [55]. Note that on a multinomial logistic regression model, the Fisher information matrix is essentially equivalent to the Gauss-Newton matrix [55]. Since the Fisher information matrix is computationally expensive to compute, it is common to compute the empirical Fisher matrix using a subset of the training samples [115, 116]. The (Generalised) Gauss-Newton methods and natural gradient method also share the same $O(N^2)$ memory complexity as the other Hessian approximation methods do.

Without estimating the second-order curvature, some methods can avoid saddle points and perhaps have some degree of resistance to near-singular curvature [55]. For instance, AdaGrad [57] accumulates the square of the gradients of past iterations to rescale each element of the gradient, so that parameters that have been infrequently updated are allowed to have large updates, and frequently updated parameters can only have small updates. Similarly, RMSProp [56] normalises the gradient by the magnitude of recent gradients. Furthermore, Adadelta [58] and Adam [59] improve over AdaGrad [57] by taking more careful gradient re-scaling schemes. Even though these methods do not use the second order information, they have been reported to work well in training deep neural networks and the additional computation cost is negligible. However, these methods depend on the assumption that the parameters should pace evenly along the searching directions even if the actual local information may suggest otherwise. When training very deep ResNets, several reports show that these methods do not necessarily outperform the vanilla SGD [117, 118].

Given the difficulty of using second-order methods in training deep learning models, there has been some interest in the implementation of approaches that can characterise the functionality of SGD optimisation. Choromanska et al. [60] use the spin-glass model to evaluate fully-connected networks and suggest that large size networks (in terms of the number of units in hidden layer) contain many local minima that are equivalent in terms of test performance. A theoretical analysis that use of the spin-glass model to evaluate residual networks can be found in [61]. Furthermore, Lee et al. [63] show that SGD converges to a local minimiser rather than a saddle point (with models that are randomly initialised). Soudry and Carmon [62] provide theoretical guarantees that local minima in multilayer neural networks loss functions have zero training error. We emphasise that the above works characterise the loss function in terms of their local minima, which is interesting but not useful in providing a helpful guideline for characterising the training procedure from beginning to end. Furthermore, some of these analyses [60, 62] conduct experiments on small size networks, so the SGD behaviour and the shape of the loss function during the training of very deep networks is still unclear.

Recently, some methods been developed to characteristic neural network training by looking at the second order information of the eigenvalues of the Hessian. The exact Hessian eigenvalues of a two-layer network has been studied [65] in order to assess the complexity of the training problem and whether the system is over-parameterised. This work suggests that the Hessian of the loss function of the network is very singular, which argues methods that assume non-singular Hessian are not to be used without proper modification [65]. This finding is interesting but the methodology is infeasible to be extended to modern ResNets due to the high computational complexity of processing the Hessian. Goodfellow et al. [64] proposed a linear subspace experiment to characterise the training path taken by the SGD optimisation, which shows that state-of-the-art models do not encounter significant obstacles (local minima, saddle points, etc.) during the training. This method provides valuable insight and can be applied to models with high complexity, but as Goodfellow et al. [64] state in their work, this result is obtained only on well-performing models, so it is not possible to say whether SGD never encounters obstacles during training or SGD works well when it does not encounter these structures. In [26], a new sensitivity measurement of energy landscape is used to provide empirical evidence to support the argument that training with large mini-batch size converges to sharp minima, which in turn leads to poor generalisation. In contrast, small mini-batch size converges to flat minima, but performance degenerates due to noise in the gradient estimation. Though it is very relevant to our proposal, this work [26] focuses only on mini-batch size. In Section 3.6, we show that our proposed measures are able to characterise SGD optimisation with respect to not only mini-batch size but also to model structure, learning rate and stochastic depth rate.

2.6 Conclusion

The methodologies described in the sub-sections above cover four topics in deep learning: nonlinear classification using CNN features, piecewise linear activation function, multi-scale feature learning, and network training analysis. In general, even though deep learning has been widely used and has achieved stunning results in many areas, our understanding of it still has many gaps. The nonlinear classification using CNN features is rarely explored due to the complexity of nonlinear classifiers and the dimensionality of the CNN features. One of the main issues of the piecewise linear activation function is to maintain a balanced distribution of training samples across the activation domain to fully utilise the exponential capacity of the model. Furthermore, current state-of-the-art models focus on the implementation of deep and wide models, but they neglect many other aspects, such as the multi-scale nature of the learned features. Also the training of deep learning models does not have reliable guidelines, leaving the training process to be evaluated only based on the objective function and the classification performance. Our proposed methodologies target the above issues, and we show the performance of our methods are competitive with the state-of-the-art methods on publicly available and widely used datasets.

Chapter 3

Methodology

3.1 Overview

This chapter introduces the techniques that we use to develop our proposed methods. Section 3.3 is devoted to the explanation of our nonlinear tree structured classifier, which is designed to handle large dimensional input features such as the CNN features. This classifier is built with linear classifiers and trained with a novel non-greedy loss function that delays hard classification problems to further down the tree. In Section 3.4, we show an extension of the Network-in-Network (NIN) [31] model by replacing the ReLU [48] activation unit with Maxout [53] activation unit to increase the model capacity, and also placing a Batch Normalisation unit [45] before the Maxout unit to maintain the model capacity and to pre-condition the model. This model is named the Maxout Network in Maxout Network (MIM). We adopt the use of multiple-size filters in the aforementioned batch-normalised Maxout activation unit in Section 3.5 to allow capturing multiple-size features in a single convolutional module. We call this new CNN module the Competitive Multiple-size Convolution module. Finally in Section 3.6, we introduce two measures: the cumulative sum of the condition number and the cumulative sum of the Laplacian of the approximate Fisher matrix, which can be efficiently calculated during the training of state-of-the-art ResNet [24] model in order to characterise the behaviour of the training of hyper-parameters.

3.2 Datasets

In this section, we define the datasets used to evaluate the methods proposed in this thesis.

CIFAR-10 [72] is a publicly available benchmark that is used to evaluate the performance of the classification methods. It consists of 60000 32×32 RGB images, where 50000 images are for training and the rest 10000 for testing. The CIFAR-10 images are sourced from 10 visual object categories including six animals species and four types of vehicles, where each category has 5000 training images and 1000 testing images. The specification of CIFAR-100 [72] is an extension of the CIFAR-10 dataset, except the CIFAR-100 dataset has 100 classes and grouped in 20 super-classes, where each class has 500 training images and 100 testing images. By convention, the classification performance is measured on the 10 classes for CIFAR-10 and 100 classes for CIFAR-100. In both CIFAR-10 and 100 datasets, the visual objects are well-centred in the images. We show measures and performance evaluations of deep learning models that are trained with the above datasets to give empirical support of our arguments and methodologies. In Section 3.5, we use CIFAR-10 trained CNN models to illustrate the similarity between convolutional filters within competitive units. We also use CIFAR-10 pre-trained models to test the transferability of the model by fine-tunning on the CIFAR-100 task. In Section 3.6, we show an example of using the proposed two measures to characterise the training procedure of ResNets by evaluating the CIFAR-10 task.

The MNIST[47] dataset is a hand-written digit recognition dataset which contains 60000 training and 10000 testing 28×28 gray-scale images. The Street View House Number (SVHN) [119] dataset is a real-world house number plate digit recognition dataset with over 600000 32×32 RGB images, partitioned into training ($\sim$73000 images), testing ($\sim$26000 images) and extra ($\sim$530000 images) sets. The SVHN images keep the digit of the interest at the centre and the neighbouring distracting digits on the same house number plate. Both digit recognition datasets are used to provide additional evaluation to our proposed batch-normalised Maxout activation unit (proposed in Section 3.4 and evaluated in Chapter 5) and Competitive Multiple-size Convolution module (CMSC) (proposed in Section 3.5 and evaluated in Chapter 6).

The PASCAL VOC 2007 [66] (VOC07) is another publicly available visual classification benchmark with the goal of recognising objects from 20 visual classes (including people, animal, vehicle and indoor objects) in high resolution realistic images, in contrast to the aforementioned small pre-segmented MNIST, CIFAR and SVHN images.

Furthermore, these images are in arbitrary sizes and the objects of interest may appear in any location on the image. In Chapter 4, we show the classification performance of our nonlinear classifier (proposed in Section 3.3) on two types of CNN features generated from VOC07 images.

The ImageNet Large-Scale Visual Recognition Challenge [9] 2012 (ILSVR12) dataset contains more than 1 million training images and 50000 validation images, catalogued into 1000 object classes, where the style of the images is similar to VOC07. In addition to evaluate top-1 accuracy (i.e., evaluating the prediction with highest confidence score against the ground truth label), the classification performance is also measured by top-5 accuracy (i.e., the prediction is assumed to be correct if one of the top five predictions contains the ground truth). We use the ILSVR12 dataset to evaluate the performance of our (Competitive Multiple-size Convolution) CMSC module on large-scale dataset in Chapter 6.

3.3 Non-greedy Hierarchical Classification

Assume that an image is represented by a feature vector $\mathbf{x} \in \mathbb{R}^D$, the image label is represented by the variable $y \in \{-1, 1\}$, and the binary tree classifier has one root node that classifies samples using a hidden variable $b \in \{-1, 1\}$ (which indicates the left child by -1 and right by $+1$). Then the inference is defined by the following problem:

$$y^* = \arg \max_{y \in \{-1,1\}} \sum_{b \in \{-1,1\}} P(y|\mathbf{x}, b, \theta_b) P(b|\mathbf{x}, \theta_1), \tag{3.1}$$

where $\theta_b, \theta_1 \in \mathbb{R}^D$ denote the classifier parameters of the leaf and root nodes, respectively. Figure 3.1-(a) shows the tree structure of this inference. This inference can be extended to a tree with three levels (see Figure 3.1-(b)), as follows:

$$P(y|\mathbf{x}, \Theta) = \sum_{b^{(1)}} \sum_{b^{(2)}} P(y|\mathbf{x}, b^{(1)}, b^{(2)}, \theta_{b^{(1,2)}}) P(b^{(2)}|\mathbf{x}, b^{(1)}, \theta_{b^{(1)}}) P(b^{(1)}|\mathbf{x}, \theta_1), \tag{3.2}$$

where $b^{(1)}, b^{(2)} \in \{-1, 1\}$, and we assume that the classifier in the root node is represented by the variable θ_1, its left child by $\theta_{b^{(1)}=-1}$ and right child by $\theta_{b^{(1)}=+1}$. The extension to higher trees is then trivial.

For the learning procedure, assume the availability of a training set $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^M$ and a validation set $\mathcal{V} = \{(\mathbf{x}_i, y_i)\}_{i=1}^Q$. The description of the learning methodology is

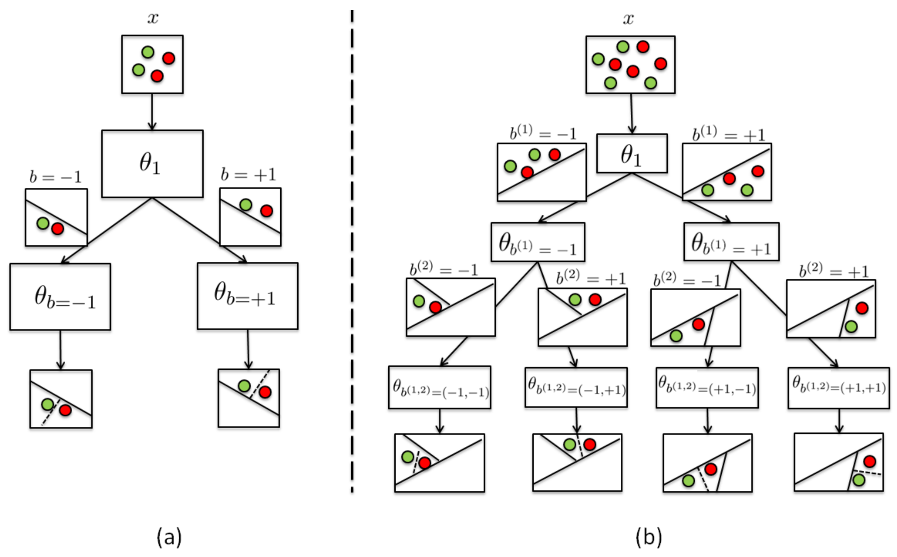


FIGURE 3.1: A two-level binary tree classifier (a) described by (3.1), and a three-level tree classifier (b) described by (3.2), where the root and intermediate nodes depend on hidden variable b that geometrically divides the space (straight lines), and the leaf classifiers produce label y that classify the points (dotted lines). The coloured points represent the binary classification problem in both cases.

clearer if we assume the tree has only one parent node (the extension to larger tree is again trivial), where the ideal learning process uses the training set $\mathcal{T}$ as follows:

$$\{\theta_b^*, \theta_1^*\} = \arg \max_{\{\theta_b, \theta_1\}} \prod_{i=1}^{M} \sum_{b_i} P(y_i|\mathbf{x}_i, b_i, \theta_b) P(b_i|\mathbf{x}, \theta_1), \tag{3.3}$$

where we assume that the training samples are i.i.d. and $b_i \in \{-1, 1\}$. Equation 3.3 involves the maximisation of two functions, which are hard to be optimised jointly, so we break this optimisation into the following iterative algorithm containing two alternating stages, where we assume that the parameters $\theta_b^{(t-1)}$ and $\theta_1^{(t-1)}$ are known at stage (t):

$$\begin{aligned} \theta_1^{(t)} &= \arg\max_{\theta_1} \prod_{i=1}^{M} \sum_{b_i} P(y_i|\mathbf{x}_i, b_i, \theta_b^{(t-1)}) P(b_i|\mathbf{x}_i, \theta_1), \\ \theta_b^{(t)} &= \arg\max_{\theta_b} \prod_{i=1}^{M} \sum_{b_i} P(y_i|\mathbf{x}_i, b_i, \theta_b) P(b_i|\mathbf{x}_i, \theta_1^{(t)}). \end{aligned} \tag{3.4}$$

Essentially, this learning procedure involves the division of the training samples into two clusters, one representing the left child samples (labelled as $b = -1$) and the other, the right child samples (labelled as $b = +1$). After this division is performed, the parameter θ_1 of the classifier $P(b|\mathbf{x}, \theta_1)$ is estimated, and then, the classifier associated

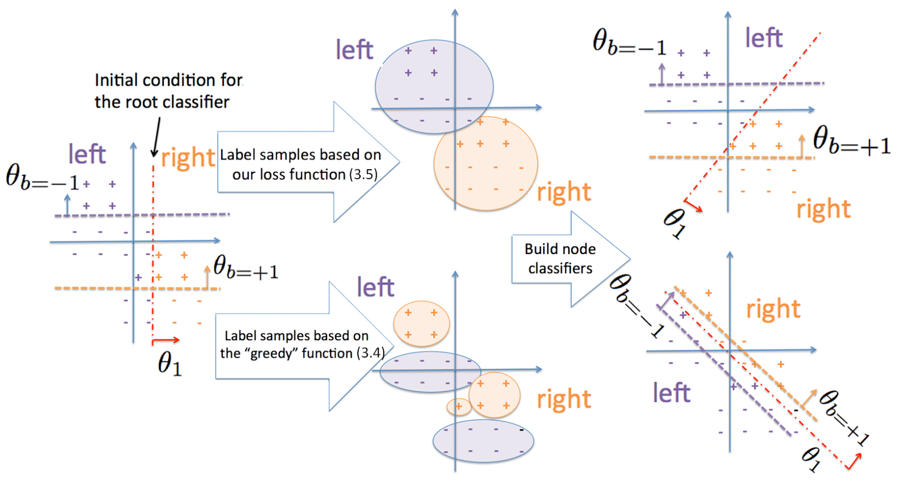

Initial condition for the root classifier
left
right
$\theta_{b=-1}$
$\theta_{b=+1}$
θ_1
Label samples based on our loss function (3.5)
Label samples based on the "greedy" function (3.4)
Build node classifiers
left
right

FIGURE 3.2: Comparison between the learning of the root node classifier parameter θ_1 (in a binary classification problem, with labels '+' and '-') using our proposed loss function in (3.5) in comparison to the "greedy" loss function in (3.4).

with each child can have its parameter θ_b estimated based on the samples belonging to that child (which is decided based on $P(b|\mathbf{x}, \theta_1)$). We consider this training process to be greedy because we maximise the classification probability $P(y|\mathbf{x}, b, \theta_b)$ even when splitting the training points to the left or right children. Our main contribution in this method is the proposal of a non-greedy loss function used for the estimation of θ_1 in (3.4), which also delays hard classification problems.

The motivation for our proposed loss function is based on the graphs shown in Figure 3.2. Assuming that both $P(y|\mathbf{x}, b, \theta_b)$ and $P(b|\mathbf{x}, \theta_1)$ are represented by linear classifiers (which means that θ_b and θ_1 are vectors denoting the normal vectors of the learned hyperplane), notice that if we try to maximise $P(b|\mathbf{x}, \theta_1)P(y|\mathbf{x}, b, \theta_b)$ when estimating θ_1, using the depicted initial guess (graph on left), we will greedily label the '+' points with '+1' (right child) and the '-' points with '-1', which generates a very difficulty learning problem in the next iteration of the algorithm (graph on the bottom of Figure 3.2). On the other hand, our loss function has a shape depicted by Figure 3.3, which means that

1. if the classifiers in both children generate a correct classification (or both are incorrect), pick the one (with its respective child label) with the closest hyperplane (which allows the hyperplanes assigned with geometrical close training samples in the next iteration);
2. if the classifier of only one child is correct, pick the correct one as long as it is not farther from the margin than the incorrect one (which gives a chance for the

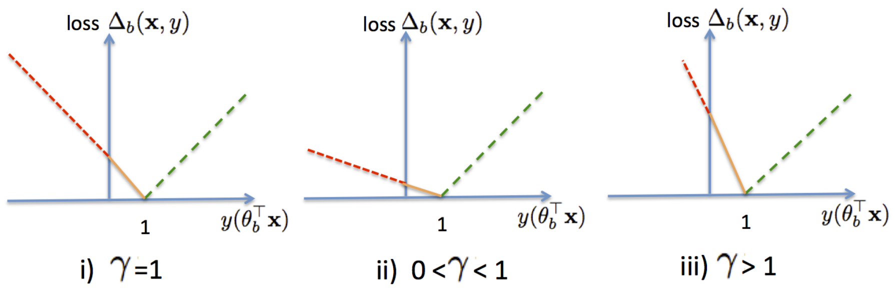


FIGURE 3.3: Shape of the proposed loss function (3.5), where dotted red shows the region where classification is incorrect, solid orange displays the margin region of Hinge loss, and dashed green depicts correct classification.

incorrect training samples to be re-classified by the geometrical closer hyperplane in the next iteration).

This means that in addition to performing a division of the training samples without trying to greedily minimise the classification error (item (1) above), we also delay the hard classification problems to later stages of the binary tree [67, 94], as depicted in top part of Figure 3.2. The loss function depicted in Figure 3.3 is defined by:

$$f(b_i; \mathbf{x}_i, y_i) = \sum_b \Delta_b(\mathbf{x}_i, y_i)\delta(b_i - b), \tag{3.5}$$

with $\delta(.)$ denoting the Dirac delta function and

$$\Delta_b(\mathbf{x}, y) = \gamma \max(0, 1 - y(\theta_b^\top \mathbf{x})) + \max(0, y(\theta_b^\top \mathbf{x}) - 1), \tag{3.6}$$

where $\gamma \in \mathbb{R}^+$ is a scalar that weighs the relative importance of the two terms in (3.6), and θ_b denotes the parameter of a linear SVM classifier learned for each child node b using the training samples $\mathbf{x}_i, y_i$ where $b_i = b$.

The proposed learning algorithm iterated steps 1-3 below (until convergence), assuming that we have the estimated value for $\theta_b^{(t-1)}$:

1). For each training sample $i \in \{1, ..., M\}$, determine its label b_i with:

$$b_i^{(t)} = \arg \max_{b \in \{-1,+1\}} f(b_i; \mathbf{x}_i, y_i), \tag{3.7}$$

defined in (3.5) and using $\theta_b^{(t-1)}$;

2). Estimate $\theta_1^{(t)}$, as follows:

$$\begin{aligned}\theta_1^{(t)} &= \arg\max_{\theta_1} \prod_{i=1}^{M} P(b_i^{(t)}|\mathbf{x}_i, \theta_1), \\ &= \arg\min_{\theta_1} \frac{1}{2}\|\theta_1\|^2 + \lambda \sum_{i=1}^{M} \log(1+\exp(-b_i^{(t)}\theta_1^\top \mathbf{x}_i)),\end{aligned} \tag{3.8}$$

which is a logistic regression classifier;
3). Estimate $\theta_b^{(t)}$, with:

$$\begin{aligned}\theta_b^{(t)} &= \arg\max_{\theta_b} \prod_{i=1}^{M} P(y_i|\mathbf{x}_i, b_i^{(t)}, \theta_b)\delta(b_i^{(t)} - b) \\ &= \arg\min_{\theta_b} \frac{1}{2}\|\theta_b\|^2 + \lambda \sum_{i=1}^{M} \max(0, 1 - y_i(\theta_b^\top \mathbf{x}_i)),\end{aligned} \tag{3.9}$$

for $b \in \{-1, +1\}$, which is the definition for the linear SVM classifier (note that we use the soft margin training that allows for non-separable problems).

The initialisation of this algorithm is achieved with the K-means clustering (with two clusters) considering the set $\{\mathbf{x}_i | y_i = +1, (\mathbf{x}_i, y_i) \in \mathcal{T} \bigcup \mathcal{V}\}$. We run this clustering 20 times and pick the labels b_i for the case that minimises the loss in (3.5), and then we run from step (2) of the algorithm above. The stopping criterion is also based on the computation of the loss in (3.5), where when the loss difference between two iterations is smaller than a threshold ϵ, then we stop iterating.

Finally, there is also a model selection problem involved in this method concerning the structure of the binary tree, where after convergence, we verify a condition to determine if we will backtrack to the previous structure or keep the current tree structure and thus continue to grow the tree. The initial tree contains only the root node presented as a linear SVM classifier (i.e., this is the original classifier found in previous works [8]), and the condition that we use for model selection is the classification accuracy (e.g., mean average precision) measured in the validation set $\mathcal{V}$ using the latest trained tree. This procedure can determine whether the expansion of the tree results in overfitting of the training set $\mathcal{T}$, and the backtracking to be previous tree restores the generalisation ability of the whole classifier.

Compared to the greedy objective function in (3.4), our method can reduce the chance that θ_1 undesirably takes over the classification responsibility, leading to the triviality of training θ_b by the assigned proportion of training samples (i.e., the assigned proportion of training data mostly belongs to one class). This is illustrated in the example depicted at the bottom of Figure 3.2 – the "greedy" loss function, where the red dotted line θ_1 divides two clusters that aligns with the ground truth label. However, we argue that the performance of our method may not surpass the original objective in (3.3) if the nonlinearity in the data distribution is minimal or the volume of training samples is not sufficiently large, in which situation the high capacity classifiers prone to overfit the training set.

3.3.1 Complexity Analysis

For each node expansion, we train three separate linear classifiers per iteration of our algorithm. Therefore, we need K iterations in a tree with N nodes, where $N \in [2^h - 1, 2^{h+1} - 1]$, with h =maximum tree depth, so the training complexity of our method is $O(3KNDM)$ (recall that D is the feature size and M is the training set size), so this means that compared to the linear classifier, our training is $3KN$ slower. Fortunately, we can control the values for K and N by constraining the number of iterations and the tree depth, and in general we have $KN << D, M$, which means that our training is significantly faster than typical shallow nonlinear classifiers (see Section 2.2.1). The testing complexity is essentially based on running h linear classifiers, which means that the running time complexity is $O(hD)$, which is just marginally larger than a single linear classifier given that $h \leq 3$, typically.

3.4 Batch-normalised Deep Learning with Piecewise Linear Activation Units

In this section, we first explain the piecewise linear activation units, followed by an introduction of how the Batch Normalisation [45] unit works. We describe the proposed MIM model at the end the this section, including its training and inference procedures.

The nomenclature adopted in this section is the same as the one introduced by Montufar et al. [27], where a feedforward neural network is defined by the function $F_{net} : \mathbb{R}^{n_0} \to \mathbb{R}^{\text{out}}$:

$$F_{net}(\mathbf{x}, \theta) = f_{\text{out}} \circ g_L \circ h_L \circ f_L \circ ... \circ g_1 \circ h_1 \circ f_1(\mathbf{x}), \tag{3.10}$$

where $f(.)$ represents a preactivation function, $h_l(.)$ represents a normalisation function, $g_l(.)$ is a nonlinear activation function, $f_{out}(.)$ denotes a preactivation function followed by a softmax activation function, and the parameter θ is formed by the input weight matrices $\mathbf{W}_l \in \mathbb{R}^{k.n_l \times n_{l-1}}$ and bias vectors $\mathbf{b}_l \in \mathbb{R}^{k.n_l}$ of $f(.)$ and normalisation parameters γ_l and β_l in (3.12) of $g(.)$ for each layer $l \in \{1, ..., L\}$. The preactivation function is defined by $f_l(\mathbf{x}_{l-1}) = \mathbf{W}_l\mathbf{x}_{l-1} + \mathbf{b}_l$, where the output of the $(l-1)^{th}$ layer is $\mathbf{x}_l = [\mathbf{x}_{l,1}, ..., \mathbf{x}_{l,n_l}]$, denoting the activations $\mathbf{x}_{l,i}$ of the units $i \in \{1, ..., n_l\}$ from layer l. This output is computed from the activations of the preceding layer by $\mathbf{x}_l = g_l(h_l(f_l(\mathbf{x}_{l-1})))$. Also note that $\mathbf{f}_l = [\mathbf{f}_{l,1}, ..., \mathbf{f}_{l,n_l}]$ is an array of n_l preactivation vectors $\mathbf{f}_{l,i} \in \mathbb{R}^k$, which after normalisation, results in an array of n_l normalised vectors $\mathbf{h}_{l,i} \in \mathbb{R}^k$ produced by $h_{l,i}(f_{l,i}(\mathbf{x}_{l-1}))$, and the activation of the i^{th} unit in the l^{th} layer is represented by $\mathbf{x}_{l,i} = g_{l,i}(h_{l,i}(f_{l,i}(\mathbf{x}_{l-1})))$.

3.4.1 Piecewise Linear Activation Units

By dropping the layer index l to facilitate the nomenclature, the recently proposed piecewise linear activation units ReLU [48], LReLU [50], PReLU [51], and Maxout [53] are represented as follows [27]:

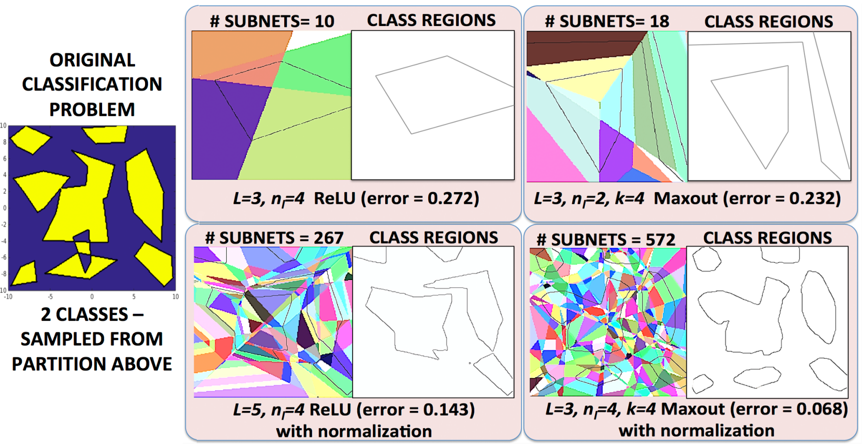
$$\begin{aligned} &\text{ReLU:} && g_i(\mathbf{h}_i) = \max\{0, \mathbf{h}_i\}, \\ &\text{LReLU or PReLU:} && g_i(\mathbf{h}_i) = \max\{\alpha.\mathbf{h}_i, \mathbf{h}_i\}, \\ &\text{Maxout:} && g_i(\mathbf{h}_i) = \max\{\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,k}\}. \end{aligned} \tag{3.11}$$

where $\mathbf{h}_i \in \mathbb{R}$ and $k = 1$ for ReLU, LReLU [50], and PReLU [51], α is represented by a small constant in LReLU, but a learnable model parameter in PReLU, k denotes the number of regions of the Maxout activation function, and $\mathbf{h}_i = [\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,k}] \in \mathbb{R}^k$. Note the difference between a Maxout function and a max-pooling function in the context of CNN is that the Maxout is effectively an element-wise maximum operation over k feature maps, which is also known as “channel-wise” pooling; on the other hand, the max-pooling refers to select the maximum value over a small receptive window on individual feature map, and commonly used to reduce the size of feature maps. It should also be noted that a Maxout activation function with $k > 1$ increases the number of parameters.

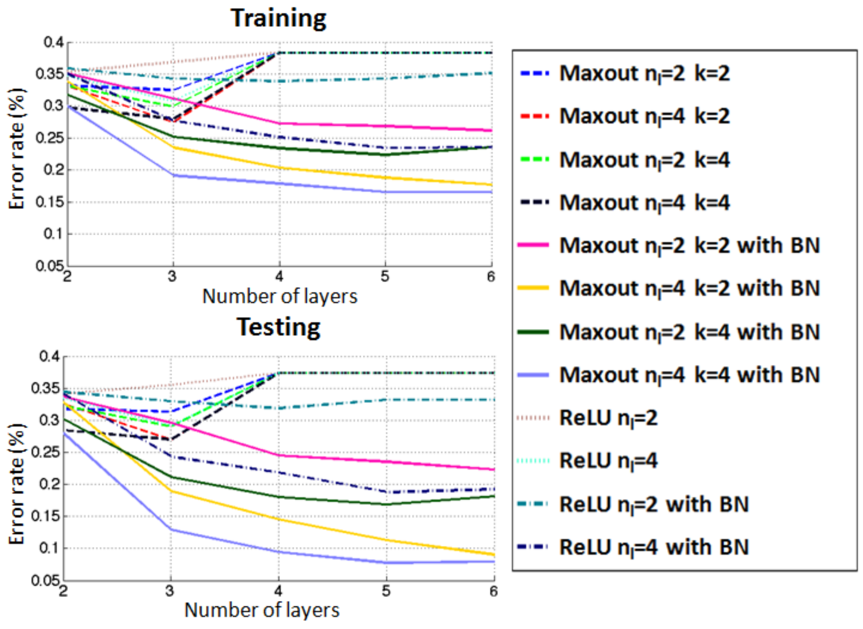
According to Montufar et al. [27], the network structure is defined by the input dimensionality n_0, the number of layers L and the width n_l of each layer. A linear region of the function $F : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^m$ is a maximal connected subset of $\mathbb{R}^{n_0}$. Note from (3.11) that rectifier units have two behaviour types: 1) constant 0 (ReLU) or linear (LReLU or PReLU) with a small slope when the input is negative; and 2) linear with slope 1 when input is positive. These two behaviours are separated by a hyperplane (see Figure 2.1) and the set of all hyperplanes within a rectifier layer forms a hyperplane arrangement, which split the input space into several linear regions. A multi-layer network that uses rectifier linear units with n_0 inputs and L hidden layers with $n \geq n_0$ nodes can compute functions that have $\Omega\left((n/n_0)^{L-1} n^{n_0}\right)$ linear regions, and a multi-layer network that uses Maxout activation units with L layers of width n_0 and rank k can compute functions that have $k^{L-1} k^{n_0}$ linear regions [27]. These results indicate that multi-layer networks with Maxout and rectifier linear units can compute functions with a number of linear regions that grows exponentially with the number of layers [27].

To visualise the linear regions, we show a 2-D synthetic binary classification problem in Figure 3.4, where these linear regions can be observed as the coloured polygons in Figure 3.4-(a) and the number of linear regions is denoted by "# SUBNETS". The specification of this toy problem is listed below. First, the samples are drawn (12K for training and 2K for testing) using a uniform distribution between $[-10, 10]$ (in each dimension) from the partition shown in Figure 3.4-(a) (leftmost image), with the colors blue and yellow indicating the class labels. We train a multiple-layer perceptron (MLP) with varying number of nodes per layer $n_l \in \{2, 4\}$ and varying number of layers $L \in \{2, 3, 4, 5, 6\}$, and it is possible to place two types of piecewise linear activation functions after each layer: ReLU [48] and Maxout [53], where for Maxout we can vary the number of regions $k \in \{2, 4\}$ (e.g., Figure 2.1 shows a Maxout with 4 regions). Training is based on back propagation [91] using mini-batches of size 100, learning rate of 0.0005 for 20 epochs then 0.0001 for another 20 epochs, momentum of 0.9 and weight decay of 0.0001, where we run five times the training (with different training and test samples) and report the mean train and test errors of the best architecture of all configurations (with respect to the combination of n_l and L). Finally, the MLP weights are initialised with Normal distribution scaled by 0.01 for all layers.

The training process of networks containing piecewise linear activation units uses a divide and conquer strategy where $\frac{\partial \ell}{\partial \mathbf{W}_l}$ moves the classification boundary for layer l according to the loss function ℓ with respect to the points in its current linear region (similarly for the bias term $\mathbf{b}_l$), and $\frac{\partial \ell}{\partial \mathbf{x}_{l-1}}$ moves the offending points (i.e., points being erroneously classified) away from their current linear regions. Dividing the data points into an exponentially large number of linear regions is advantageous because the training algorithm can focus on minimizing the loss for each one of these regions almost


ORIGINAL CLASSIFICATION PROBLEM
2 CLASSES – SAMPLED FROM PARTITION ABOVE
# SUBNETS= 10
CLASS REGIONS
L=3, n_l=4 ReLU (error = 0.272)
# SUBNETS= 18
CLASS REGIONS
L=3, n_l=2, k=4 Maxout (error = 0.232)
# SUBNETS = 267
CLASS REGIONS
L=5, n_l=4 ReLU (error = 0.143) with normalization
# SUBNETS = 572
CLASS REGIONS
L=3, n_l=4, k=4 Maxout (error = 0.068) with normalization

a) Original classification problem (left) with the liner regions found by each model (represented by the color of each subnet) and classification division of the original space (class regions).



b) Train and test error as a function of the number of layers (L), number of nodes per layer (n_l), piecewise linear activation function, number of regions in the activation function (k), and the use of normalisation (with or without BN).

FIGURE 3.4: Toy problem with the division of the space into linear regions and classification profile produced by each model (a), and a quantitative comparison between models (b).

independently of others - this is why we say it uses a divide and conquer algorithm. We also say that it is an almost independent training of each linear region because the training parameters for each region are shared with all other regions, and this helps the regularisation of the training process. However, the initialisation of this training process is critical because if the data points are not evenly distributed at the beginning, then all these points may lie in only one of the regions of the piecewise linear unit. This will drive the learning of the classification boundary for that specific linear region, where the loss will be minimised for all those points in that region, and the boundary for the other linear regions will be trained less effectively with much fewer points. This means that even the points with relatively high loss will remain in that initial region because the other regions have been ineffectively trained, and consequently may have a larger loss. This issue is very clear with the use of Maxout units, where in the extreme case, only one of the k regions is active, which means that the Maxout unit will behave as a simple linear unit. If a large amount of Maxout units behave as linear units, then this will reduce the ability of these networks to compute functions that have an exponential number of linear regions, and consequently decrease the capacity of the model.

3.4.2 Batch Normalisation Units

In order to force the initialisation to distribute the data points evenly in the domain of the piecewise activation functions, such that a large proportion of the k regions is used, we propose the use of Batch Normalisation by Ioffe and Szegedy's [45]. This normalisation has been proposed because of the difficulty in initializing the network parameters and setting the value for the learning rate, and also because the inputs for each layer are affected by the parameters of the previous layers. These issues lead to a complicated learning problem, where the input distribution for each layer changes continuously - an issue that is called covariate shift [45]. The main contribution of this Batch Normalisation is the introduction of a simple *feature-wise* centring and normalisation to make it have mean zero and variance one, which is followed by a scaling unit and a shifting unit that restores the representation power of the feature by shifting and scaling the normalised value. For instance, assuming that the input to the normalisation unit is $\mathbf{f} = [\mathbf{f}_1, ..., \mathbf{f}_{n_l}]$, where $\mathbf{f}_i \in \mathbb{R}^k$, the BN unit consists of two stages:

$$\begin{array}{ll} \text{Normalisation:} & \hat{\mathbf{f}}_{i,k} = \frac{\mathbf{f}_{i,k} - E[\mathbf{f}_{i,k}]}{\sqrt{\text{Var}[\mathbf{f}_{i,k}]}} \\ \text{Scale and shift:} & \mathbf{h}_{i,k} = \gamma_i \hat{\mathbf{f}}_{i,k} + \beta_i \end{array}, \tag{3.12}$$

where the shift and scale parameters $\{\gamma_i, \beta_i\}$ are new network parameters that participate in the training procedure [45]. Another important point is that the BN unit does not process each training sample independently, but it uses both the training sample and other samples in a mini-batch.

We illustrate the importance of using normalisation in piecewise linear unit in the aforementioned toy problem, by adding the option of placing a Batch Normalisation unit [45] before each activation function and show the distribution of linear regions (the two bottom images) in contrast to the region distribution of the unnormalised unit (the two top images) in Figure 3.4-(a). Analysing the mean train and test error in Figure 3.4-(b), we first notice that all models have good generalisation capability, which is a characteristic already identified for deep networks that use piecewise linear activation units [27, 96]. Looking at the curves for the networks with 2 and 3 layers, where all models seem to be trained properly (i.e., they are pre-conditioned), the models containing Batch Normalisation units (denoted by ”with BN”) produce the smallest train and test errors, indicating the higher capacity of these models. Beyond 3 layers, the models that do not use the Batch Normalisation units become ill-conditioned, producing errors of 0.39, which effectively means that all points are classified as one of the binary classes. In general, Batch Normalisation allows the use of Maxout in deeper MLPs that contain more nodes per layer, and the Maxout function contains more regions (i.e., larger k). The best result (in terms of mean test and train error) is achieved with an MLP of 5 or more layers, where each layer contains 4 nodes and Maxout has 4 regions (test error saturates at 0.07). The best results with ReLU are also achieved with Batch Normalisation, using a large number of layers (5 or more), and 4 nodes per layer, but notice that the smallest ReLU errors (around 0.19 on test set) are significantly higher than the Maxout ones, indicating that Maxout has larger capacity. The images in Figure 3.4-(a) show the division of the input space (into linear regions) used to train the sub-networks within the MLP model (we show the best performing models of ReLU with and without normalisation and Maxout with and without normalisation), where it is worth noticing that the best Maxout model (bottom-right image) produces a very large number of linear regions, which generate class regions that are similar to the original classification problem. The input space division, used to train the sub-networks, are generated by clustering the training points that produce the same activation pattern from all nodes and layers of the MLP. We also run these same experiments using dropout (of 0.2), and the relative results are similar to the ones presented in Figure 3.4-(b), but the test errors with dropout are around $2\times$ larger, which indicate that dropout does not pre-condition the model (i.e., the models that do not have the Batch Normalisation units still become ill-conditioned when having 3 or more layers), nor does it balance the input data for the activation units (i.e., the capacity of the model does not increase with dropout).

3.4.3 Maxout Network in Maxout Network Model

As mentioned in Section 3.4.1, the number of linear regions that networks with piecewise linear activation unit can have grows exponentially with the number of layers, so it

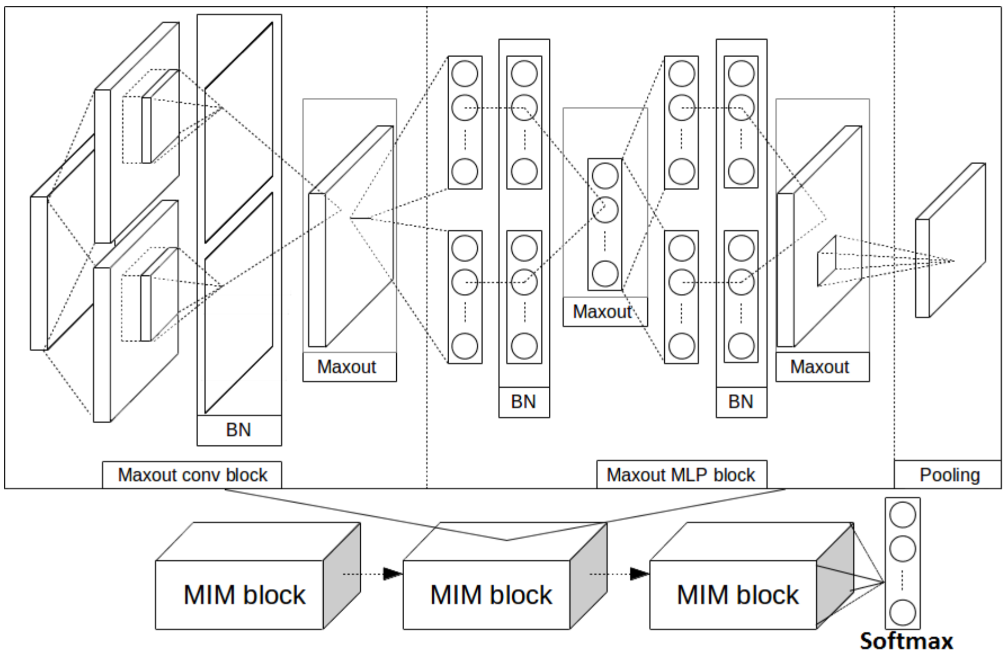


FIGURE 3.5: The MIM model is based on the NIN [31] model. This model contains three blocks that have nearly identical architectures, with small differences in terms of the number of filters and stride in convolution layers. The first two blocks use max-pooling and the third block uses average pooling.

is important to add as many layers as possible in order to increase the ability of the network to estimate complex functions. For this reason, we extend the recently proposed Network in Network (NIN) [31] model, which is based on a CNN that uses a multi-layer perceptron (MLP) as its activation layer (this layer is called the Mlpconv layer). In its original formulation, the NIN model introduces the Mlpconv with ReLU activation after each convolution layer, and replaces the fully connected layers for classification in CNN (usually present at the end of the whole network) by a spatial average of the feature maps from the last Mlpconv layer, which is fed into a softmax layer. In particular, we extend the NIN model by replacing the ReLU activation after each convolution layer of the Mlpconv by a Maxout activation unit, which has the potential to increase even further the model capacity. In addition, we also add the BN unit before the Maxout units. These two contributions form our proposed model, we give it a simple name Maxout Network in Maxout Network Model (MIM), which is depicted in Figure 3.5. Finally, we include a dropout layer [40] between MIM blocks for regularizing the model.

3.5 Competitive Multiple-size Convolution Activation Units

In this section, we introduce the proposed Competitive Multiple-size Convolution activation unit to capture multiple-size features within one convolution unit.

Let us assume that an image is represented by $\mathbf{x} : \Omega \rightarrow \mathbb{R}$, where Ω denotes the image lattice, and that an image patch of size $(2k-1) \times (2k-1)$ (for $k \in \{1, 2, ..., K\}$) centred at position $i \in \Omega$ is represented by $\mathbf{x}_{i+\phi(k-1)}$, where $\phi(j) = \{-j, \ldots, j\}$. The models being proposed in this section follow the structure of the NIN model [31], and is in general defined as follows:

$$F_{net}(\mathbf{x}, \theta) = f_{out} \circ F_L \circ ... \circ F_2 \circ F_1(\mathbf{x}), \tag{3.13}$$

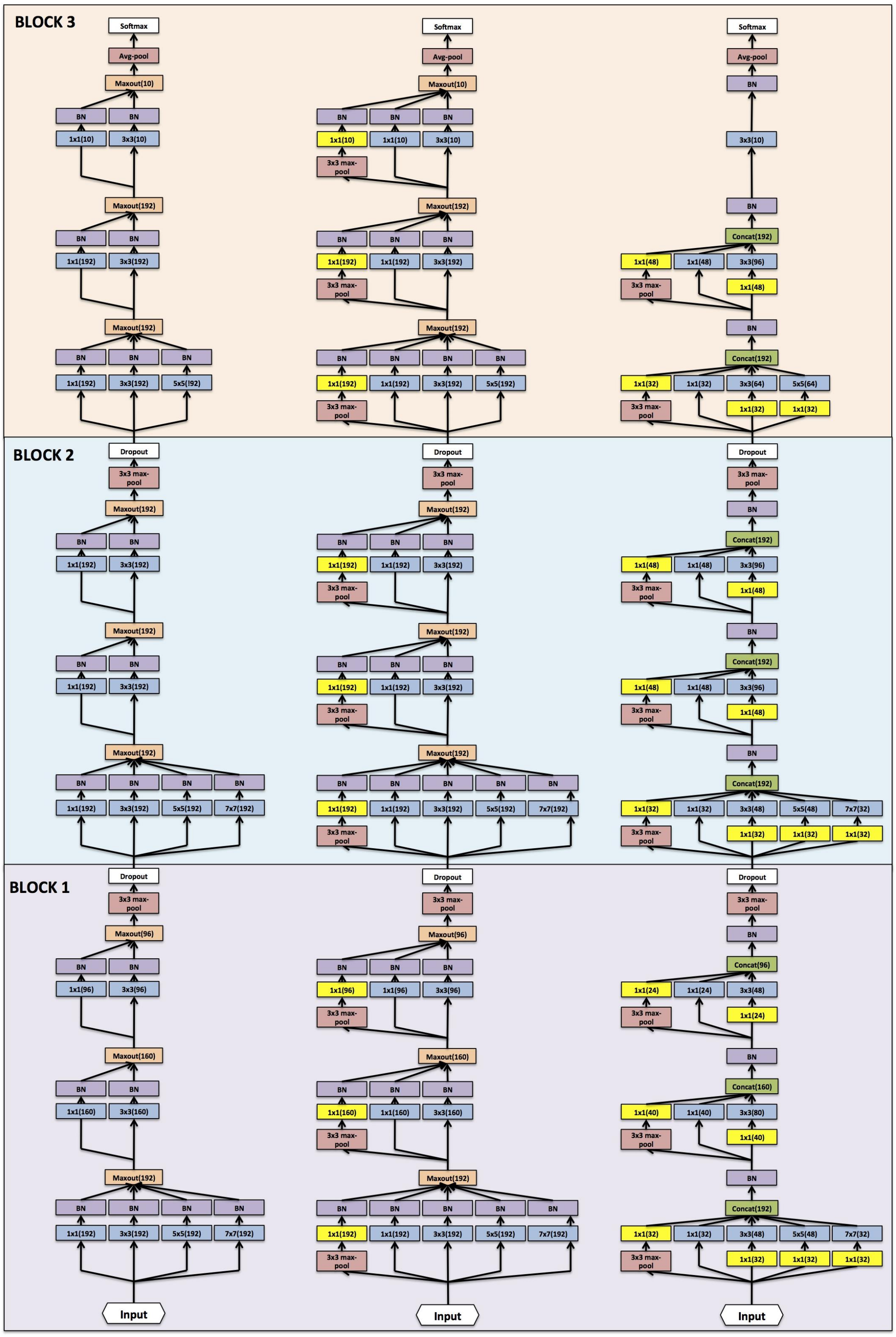
where $\circ$ denotes the composition operator, θ represents all the CNN parameters (i.e., weights and biases), $f_{out}(.)$ denotes an averaging pooling unit followed by a softmax activation function [31], the network blocks are represented by $l \in \{1, ..., L\}$, with each block containing a composition of n_l modules with $F_l(\mathbf{x}) = F_{l,n_l} \circ ... \circ F_{l,2} \circ F_{l,1}(\mathbf{x})$, and $F_{l,i}(\mathbf{x}) = g_{l,i}(h_{l,i}(f_{l,i}(\mathbf{x})))$ encapsulates the preactivation $f(.)$, normalisation $g(.)$ and nonlinear activation function $h(.)$ (defined in (3.10)) of a network layer. Each module $F_{l,n}(.)$ at a particular position $i \in \Omega$ of the input data for block l is defined by:

$$\begin{aligned} F_{l,n}(\mathbf{x}_i) = \sigma\big(&\mathrm{BN}_{\gamma_1,\beta_1}(\mathbf{W}_1^\top \mathbf{x}_i),\ \mathrm{BN}_{\gamma_3,\beta_3}(\mathbf{W}_3^\top \mathbf{x}_{i+\phi(1)}), ..., \\ &\mathrm{BN}_{\gamma_{2k-1},\beta_{2k-1}}(\mathbf{W}_{2k-1}^\top \mathbf{x}_{i+\phi(k-1)}), \\ &\mathrm{BN}_{\gamma_p,\beta_p}(\mathbf{W}_1^\top p_{3\times3}(\mathbf{x}_{i+\phi(1)}))\big), \end{aligned} \tag{3.14}$$

where $\sigma(.)$ represents the Maxout activation function [53] (see (3.11)), the convolutional filters of the module are represented by the weight matrices $\mathbf{W}_{2k-1}$ for $k \in \{1, ..., K_l\}$ (i.e., filters of size $2k-1 \times 2k-1 \times \#filters$, with $\#filters$ denoting the number of 2-D filters present in $\mathbf{W}$), which means that each module n in block l has K_l different filter sizes and $\#filters$ different filters, $\mathrm{BN}_{\gamma,\beta}$ represent the Batch Normalisation transformation with scaling and shifting parameters [45] (see Section 3.4.2), and $p_{3\times3}(\mathbf{x}_{i+\phi(1)})$ represents a max-pooling operator on the 3×3 subset of the input data for layer l centred at $i \in \Omega$, i.e.$\mathbf{x}_{i+\phi(1)}$.

Using the CNN module defined in (3.14), our proposed models differ mainly in the presence or absence of the node with the max-pooling operator within the module (i.e., the node represented by $\mathrm{BN}_{\gamma_p,\beta_p}(\mathbf{W}_1^\top p_{3\times3}(\mathbf{x}_{i+\phi(1)}))$. When the module does not contain such node, it is called **Competitive Multiple-size Convolution** (see Figure 3.6-(a)), but when the module has the max-pooling node, then we call it **Competitive Inception** (see Figure 3.6-(b)) because of its similarity to the original inception module [29]. The original inception module is also implemented for comparison purposes (see Figure 3.6-(c)), and we call this model the **Inception Style**, which is similar to (3.13) and (3.14) but with the following differences: 1) the function $\sigma(.)$ in (3.14) denotes the concatenation of the input parameters; 2) a 1×1 convolution is applied to the input $\mathbf{x}$ before a second round of convolutions with filter sizes larger than or equal to 3×3; and 3) a ReLU activation function [48] is present after each convolutional layer.

An overview of all models with the structural parameters is displayed in Figure 3.6. Note that all models are inspired by NIN [31], GoogLeNet [29], and our proposed MIM in Section 3.4. In particular, we replace the original 5×5 convolutional layers of MIM by multiple-size filters of sizes 1×1, 3×3, 5×5, and 7×7. For the inception style model, we ensure that the number of output units in each module is the same as for the competitive inception and competitive multiple-size convolution, and we also use a 3×3 max-pooling path in each module, as used in the original inception module [29]. Another important point is that in general, when designing the inception style network, we follow the suggestion by Szegedy et al. [29] and include a relatively larger number of 3×3 and 5×5 filters in each module, compared to filters of other sizes (e.g., 1×1 and 7×7). An important distinction between the original GoogLeNet [29] and the inception style network in Figure 3.6-(c) is the fact that we replace the fully connected layer in the last layer by a single 3×3 convolution node in the last module, followed by an average pooling and a softmax unit, similarly to the NIN model [31]. We propose this modification to limit the number of training parameters (with the removal of the fully connected layer) and to avoid the concatenation of the nodes from different paths (i.e., max-pooling, 1×1 convolutional filter, and etc.) into a number of channels that is equal to the number of classes (i.e., each channel is averaged into a single node, which is used by a single softmax unit), where the concatenation would imply that some of the paths would be directly linked to a subset of the classes. Therefore, the last two layers of block 3 of all architectures in Figure 3.6 contain an average pooling and a softmax unit to ensure a fair comparison amongst these networks.



(a) Competitive Multiple-size Convolution (b) Competitive Inception (c) Inception style

FIGURE 3.6: The proposed competitive multiple-size convolution (a) and competitive inception (b) networks, together with the reference inception style network (c). In these three models, we ensure that the output of each layer has the same number of units. Also note that the inception style model uses ReLU [48] after all convolutional layers, the number of filters per convolutional node is represented by the number in brackets, and these models assume a 10-class classification problem.

3.5.1 Competitive Convolution of Multiple-size Filters Prevents Undesirable Filter Convergence and Filter Co-adaptation

The main reason being explored in the field to justify the use of competitive activation units [48, 53, 54, 120] is the fact that they build a network formed by multiple underlying sub-networks [96]. More clearly, given that these activation units consist of piece-wise linear functions, it has been shown that the composition of several layers containing such units, divide the input space in a number of regions that is exponentially proportional to the number of network layers [27], where sub-networks will be trained with the samples that fall into one of these regions, and as a result become specialised to the problem in that particular region [96], where overfitting can be avoided because these sub-networks must share their parameters with one another [96]. It is worth noting that these regions can only be formed if the underlying convolutional filters do not converge to similar features, otherwise all input training samples will fall into only one region of the competitive unit, which degenerates into a simple linear transform, preventing the formation of the sub-networks.

A straightforward solution to avoid such undesirable convergence can be achieved by limiting the number of training samples in a mini-batch during stochastic gradient descent. The small batches allow the generation of "noisy" gradients during training which can activate a smaller proportion of sub-networks, making those sub-networks more specialised to "remember" unique patterns in each training iteration. This means that different linear pieces of the activation unit can be fitted, allowing the formation of an exponentially large number of regions. On the other hand, the use of a large mini-batch size generates stable gradients and "opens" many Maxout gates, leading to a large proportion of sub-networks being active and trained at the same iteration, which promotes co-adaptation. However, the drawback of this approach lies in the determination of the "right" number of samples per mini-batch. A mini-batch size that is too small leads to poor convergence, and if it is too large, then it may not allow the formation of many sub-networks. In Section 3.4, we propose a solution to this problem based on the use of BNU [45] that distributes the training samples evenly over the regions formed by the competitive unit, allowing the training to use different sets of training points for each region of the competitive unit, resulting in the formation of an exponential number of sub-networks. However, there is still a potential problem with that proposal, which is that the underlying convolutional filters are trained using feature spaces of the same size (i.e., the underlying filters are of fixed size), which can induce the filters to converge to similar regions of the feature space, also preventing the formation of the sub-networks.

TABLE 3.1: Mean orthogonality measure between pairs of filters within competitive multiple-size convolution modules (a), and filters from competitive single-size convolution modules (b), gathered from the first competitive layer of the first two blocks of the models trained on CIFAR-10.

Multiple-size (Block 1)				
	1x1	3x3	5x5	7x7
1x1	1	0.12	0.15	0.13
3x3	0.12	1	0.21	0.12
5x5	0.15	0.21	1	0.12
7x7	0.13	0.12	0.11	1

Multiple-size (Block 2)				
	1x1	3x3	5x5	7x7
1x1	1	0.05	0.03	0.02
3x3	0.05	1	0.05	0.04
5x5	0.03	0.05	1	0.07
7x7	0.02	0.04	0.07	1

a) Mean orthogonality measure between filters within competitive multiple-size convolution modules

Single-size (Block 1)				
	7x7	7x7	7x7	7x7
7x7	1	0.22	0.22	0.21
7x7	0.22	1	0.19	0.20
7x7	0.22	0.19	1	0.23
7x7	0.21	0.20	0.23	1

Single-size (Block 2)				
	7x7	7x7	7x7	7x7
7x7	1	0.09	0.10	0.09
7x7	0.09	1	0.09	0.09
7x7	0.10	0.09	1	0.09
7x7	0.09	0.09	0.09	1

b) Mean orthogonality measure between filters from competitive single-size convolution modules

Our proposed module that promotes competition amongst filters of multiple sizes represents a way to prevent the undesirable convergence mentioned in Section 3.4. To evaluate this hypothesis, we examine the similarity between convolutional filters within each competitive unit by empirically showing that the multiple sizes of the convolutional filters within a competitive unit promotes the learning of different features. This demonstration is based on measuring the orthogonality between filters within the same competitive unit, where filters that have similar responses will have orthogonality measures closer to one, and different responses will lead to orthogonality measures close to zero. The orthogonality between two filters, represented by $\mathbf{w}_1$ and $\mathbf{w}_2$, is measured by $\frac{|\mathbf{w}_1^\top \mathbf{w}_2|}{\|\mathbf{w}_1\|_2 \|\mathbf{w}_2\|_2}$, where zero padding along the filter border is used in order to allow the measure of orthogonality between filters of different sizes. We show the mean value of the orthogonality measures between pairs of single-size and multiple-size filters in a competitive unit in Table 3.1, where results are gathered from the first competitive layer of the first two blocks of the models trained on CIFAR-10 (see Chapter 6 for competitive single-size network architecture). The results from Table 3.1 provide evidence that the filter responses from the competitive multiple-size modules are more orthogonal to each other than the responses from the competitive single-size module.

Our second hypothesis is that the competition amongst multiple-size filters within a convolutional module prevents co-adaptation throughout the CNN model. The evidence for such hypothesis is displayed in Figure 3.7, which shows Yosinski et al.'s test [13] that assesses the transferability of CNN filters. The idea of this experiment is that the CNN

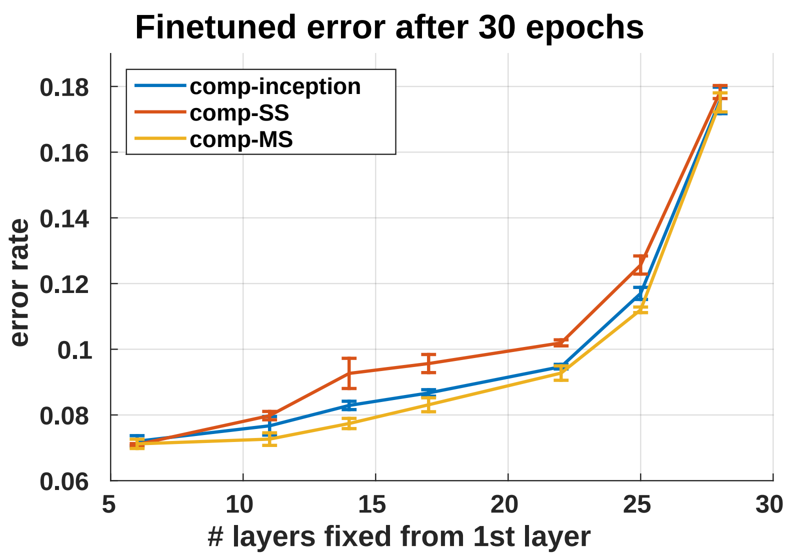


FIGURE 3.7: Co-adaptation experiment that tests the transferability of the filters learned for one dataset (CIFAR-10) to another (CIFAR-100) [13] for Competitive inception (**comp-inception**), Single-size (**comp-SS**), and Multiple-size (**comp-MS**) networks. The vertical axis shows the testing error and the horizontal axis represents the index of the first layer of the CNN to be re-trained for the new dataset, where the layers below are fixed during the fine-tuning process that lasts 30 epochs.

with more co-adapted filters will produce larger testing errors when the higher level layers are removed because it is unlikely that the re-trained layers will be able to discover the complex co-adaptations present in the removed layers. To conduct this experiment, we train a model using one dataset (CIFAR-100), remove a subset of the higher level convolutional layers (e.g., from layers 7 to the last layer, where the indexing used in the horizontal axis of Figure 3.6 is consistent with the first two competitive/inception layers to remove), and re-train only these layers for the CIFAR-10 dataset, while keeping the remaining lower level layers fixed (i.e., we do not re-train these lower layers). Figure 3.7 shows that the competitive modules with filters of multiple sizes have the lowest testing error in all evaluated configurations, when compared to the single size competitive module and the competitive inception module. This result supports our hypothesis that competition amongst filters of multiple sizes in a CNN module represents a way to prevent co-adaptation.

3.6 Characterising the Training of Deep Residual Networks with Approximate Fisher Information Matrix

In this section, we introduce the proposed characterisation of SGD training, which is based on the cumulative sum of the condition number C_K and the cumulative sum of the (weighted) eigenvalues L_K of the Fisher matrix. We first show empirically that C_K and L_K enable a consistent characterisation of various models trained with different mini-batch sizes, model structure, learning rate and stochastic depth rate. Figure 3.8-(a),(b) show how the C_K and L_K values computed at the last epoch for the training and testing sets of CIFAR-10 [72] vary for different types of ResNet models (notice the **skip**$\{\mathbf{1}, ..., \mathbf{12}\}$), different mini-batch sizes (see **size**$\{\mathbf{8}, ..., \mathbf{512}\}$) and different stochastic drop rates (**batch-drop**$\{\mathbf{0}.\mathbf{1}, ..., \mathbf{0}.\mathbf{9}\}$). In addition, these two measures are also shown in Figure 3.8-(c) to be stable when computed during the first epochs, allowing the training mechanism to be reliably characterised early on in the training process, relative to other models, saving precious training cycles. These measures also suggest new training procedures that dynamically increases the mini-batch size or decreases the stochastic depth rate, allowing the training procedure to navigate in this landscape of C_K and L_K measures. The dynamic increase of mini-batch size approach has been suggested before [26, 55], but we are not aware of previous implementations.

3.6.1 Definition of the Measures

Let us assume the availability of a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{|\mathcal{D}|}$, where the i^{th} image $\mathbf{x}_i : \Omega \to \mathbb{R}$ (Ω denotes image lattice) is annotated with the label $y_i \in \{1, ..., C\}$, with C denoting the number of classes. This dataset is divided into two mutually exclusive training set $\mathcal{T} \in \mathcal{D}$ and testing set $\mathcal{S} \in \mathcal{D}$.

The ResNet model [24] and its stochastic depth extension [42] are defined by a concatenation of residual blocks, with each block defined by:

$$r_l(\mathbf{v}_l) = b_l \times R_l(\mathbf{v}_l, \mathbf{W}_l) + \mathbf{v}_l, \tag{3.15}$$

where $l \in \{1, ..., L\}$ indexes the residual blocks, $\mathbf{W}_l$ denotes the parameters for the l^{th} block, $\mathbf{v}_l$ is the input, with the image input of the model being represented by $\mathbf{v}_1 =$

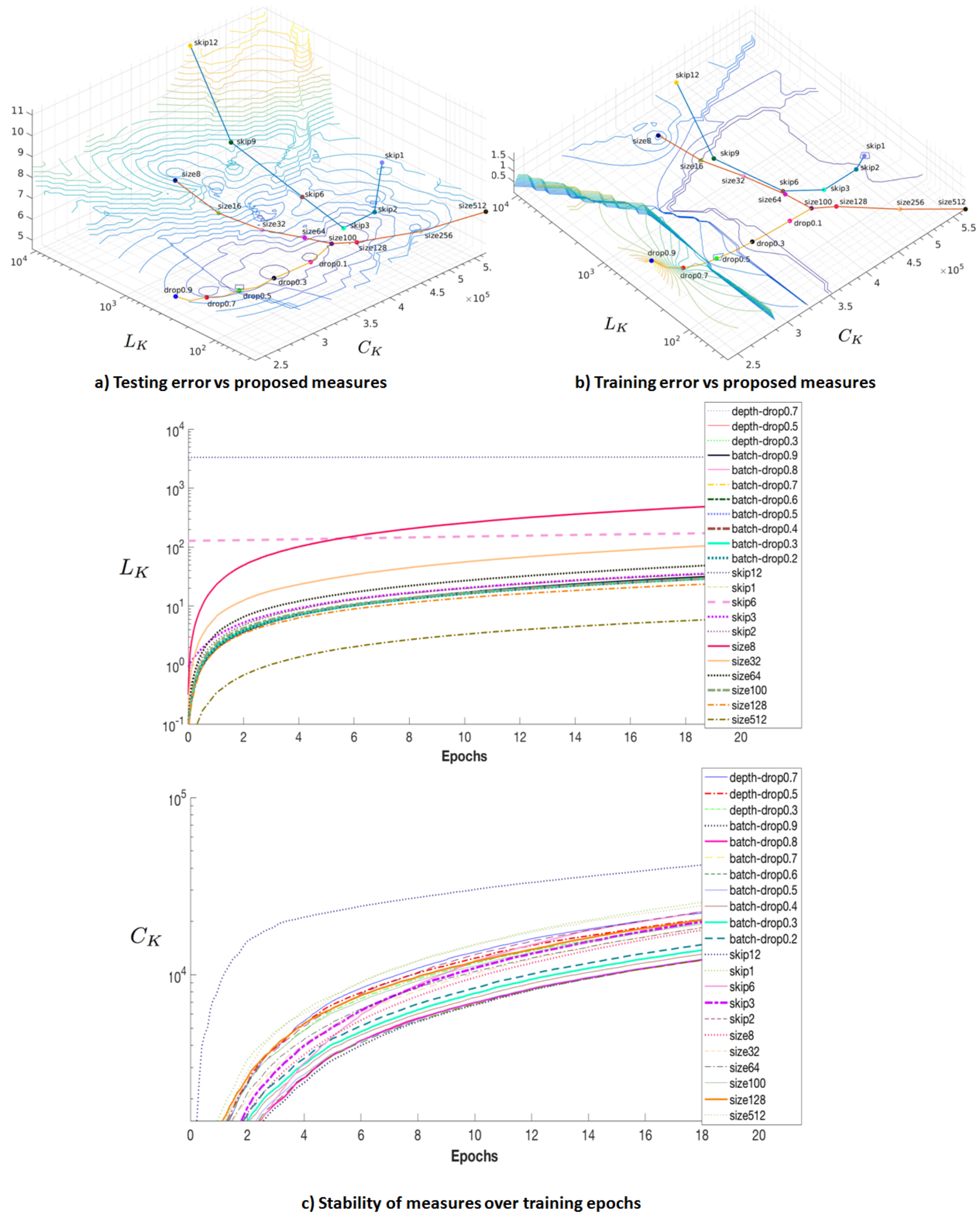


FIGURE 3.8: Testing (a) and training (b) errors on CIFAR-10 as a function of the proposed measures: cumulative sum of the condition number C_K and of the summed (weighted) eigenvalues L_K of the Fisher matrix at the final training epoch. The labels represent the training method used: 1) size$\{8, ..., 512\}$ denotes the mini-batch size for training the standard ResNet, 2) batch_drop$\{0.1, ..., 0.9\}$ represents the stochastic depth rate with a fixed mini-batch size of 100, and 3) skip$\{1, ..., 12\}$ indicates different ResNet structures with residual connections involving a large or small number of blocks. The graph in (c) shows the stability of C_K and L_K as a function of training epochs.

$\mathbf{x}$, $R_l(\mathbf{v}_l, \mathbf{W}_l)$ represents a residual unit containing a sequence of linear and nonlinear transforms [48], and Batch Normalisation [45], and $b_l \in \{0, 1\}$ denotes a Bernoulli random variable indicating if the l^{th} block is active ($b_l = 1$) or inactive ($b_l = 0$) [42] (note that in the original ResNet model [24], $b_l = 1$ for all blocks). The full model is then defined by:

$$F_{net}(\mathbf{x}, \theta) = f_{out} \circ r_L \circ ... \circ r_1(\mathbf{x}), \tag{3.16}$$

where $\circ$ represents the composition operator, $\theta \in \mathbb{R}^P$ denotes all model parameters $\{\mathbf{W}_1, ...\mathbf{W}_L\} \bigcup \mathbf{W}_{out}$, and $f_{out}(.)$ is a linear transform parameterised by weights $\mathbf{W}_{out}$ with a softmax activation function that outputs a value in $[0, 1]^C$ indicating the probability of selecting each of the C classes. The training of the model in (3.16) minimises the multi-class cross entropy loss $\ell(.)$ on the training set $\mathcal{T}$, as follows:

$$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \ell\left(y_i, F_{net}(\mathbf{x}_i, \theta)\right)). \tag{3.17}$$

The SGD training minimises the loss in (3.17) by iteratively taking the following step:

$$\theta_{k+1} = \theta_k - \frac{\alpha_k}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla \ell(y_i, F_{net}(\mathbf{x}_i, \theta_k)), \tag{3.18}$$

where $\mathcal{B}_k$ is the mini-batch for the k^{th} iteration of the minimisation process. As noted by Keskar et al. [26], the shape of the loss function can be characterised by the spectrum of the $\nabla^2 \ell(y_i, F_{net}(\mathbf{x}_i, \theta_k))$, where a significant number of large positive eigenvalues tend to make the training process generalise less well, and numerous small eigenvalues are likely to lead to better generalisation. Given that the computation of the spectrum of $\nabla^2 \ell(y_i, F_{net}(\mathbf{x}_i, \theta_k))$ is infeasible, we must resort to the use of methods that can reliably approximate such spectrum, such as the Fisher matrix [121, 122], defined by

$$\mathbf{F}_k = \left(\nabla \ell(y_{i \in \mathcal{B}_k}, F_{net}(\mathbf{x}_{i \in \mathcal{B}_k}, \theta_k)) \nabla \ell(y_{i \in \mathcal{B}_k}, F_{net}(\mathbf{x}_{i \in \mathcal{B}_k}, \theta_k))^\top\right), \tag{3.19}$$

where $\mathbf{F}_k \in \mathbb{R}^{P \times P}$.

The calculation of $\mathbf{F}_k$ in (3.19) depends on the Jacobian $\mathbf{J}_k = \nabla \ell(y_{i \in \mathcal{B}_k}, F_{net}(\mathbf{x}_{i \in \mathcal{B}_k}, \theta_k))$, with $\mathbf{J}_k \in \mathbb{R}^{P \times |\mathcal{B}_k|}$. Given that $\mathbf{F}_k = \mathbf{J}_j \mathbf{J}_j^\top \in \mathbb{R}^{P \times P}$ scales with $P = O(10^6)$ and that we are only interested in the spectrum of $\mathbf{F}_k$, we compute instead $\widetilde{\mathbf{F}}_k = \mathbf{J}_j^\top \mathbf{J}_j \in \mathbb{R}^{|\mathcal{B}_k| \times |\mathcal{B}_k|}$ that scales with the mini-batch size $|\mathcal{B}_k| = O(10^2)$. Note that from $\widetilde{\mathbf{F}}_k$ we can compute the largest $|\mathcal{B}_k|$ non-zero eigenvalues of $\mathbf{F}_k$ by using the Cholesky decomposition [123], represented by the set $\mathcal{E}_k$.

The **first measure** we propose is the **cumulative sum of the condition number of $\widetilde{\mathbf{F}}_k$** , defined by

$$C_K = \sum_{k=1}^{K} c_k, \tag{3.20}$$

where K denotes the epoch number, and $c_k = \left(\frac{\max(\mathcal{E}_k)}{\min(\mathcal{E}_k)}\right)^{\frac{1}{2}}$ represents the condition number of $\widetilde{\mathbf{F}}_k$. This measure is used to describe the ill-conditioning of the gradient updates accumulated during the training process.

The **second measure** is the **cumulative sum of the square root of the trace of $\widetilde{\mathbf{F}}_k$, weighted by** $\alpha_k^2/|\mathcal{B}_k|^2$:

$$L_K = \sum_{k \in K} \left(\frac{\alpha_k^2}{|\mathcal{B}_k|^2} \operatorname{Tr}\left(\widetilde{\mathbf{F}}_k\right) \right)^{\frac{1}{2}}, \tag{3.21}$$

where $\operatorname{Tr}(.)$ defines the trace operator, and $\operatorname{Tr}\left(\widetilde{\mathbf{F}}_k\right)$ approximates the Laplacian, defined by $\operatorname{Tr}\left(\nabla^2 \ell(y_i, F_{net}(\mathbf{x}_i, \theta_k))\right)$ that sums the eigenvalues of the Hessian. The sum of such eigenvalues is generally associated with the steepness of the energy function landscape. Note that the weighting in (3.21) is reasonable because it is the same factor used in the SGD update rule (3.18), and it proved to be useful for measuring training convergence and generalisation.

Different ResNet models, learning rates, mini-batch sizes and stochastic depth rates are observed to have relatively robust values for C_K and L_K, as displayed in Figure 3.8-(c). This means that models and training procedures can be reliably characterised early on in the training process, which can significantly speed up the assessment of new approaches. For instance, if a reference ResNet model produces a good result, and we know its C_K and L_K values for various epochs, then new models must navigate close to this reference model – see for example in Figure 3.8 that **skip2** and **skip3** models produce good convergence and generalisation, so new models must try to navigate close enough to them. In addition, new models that show very distinctive growth in C_K and L_K values with respect to the reference model are unlikely to produce competitive result – see for example in Fig. 3.8 that **skip6**, **skip12**, **size8**, and **size512**, which eventually produce uncompetitive performance, can be identified and stopped as early as 4 epochs to save the precious computation.

3.6.2 Dynamic Sampling

Dynamic sampling [26, 55] is a method that is believed to improve the convergence rate of SGD by reducing the noise of the gradient estimation with a gradual increase of the mini-batch size over the training process (this method has been suggested before, but we are not aware of previous implementations). It extends SGD by replacing the fixed size mini-batches $\mathcal{B}_k$ in (3.18) with a variable size mini-batch. The general idea of this method [26, 55] is that the initial noisy gradient estimations from small mini-batches explore a relatively flat energy landscape without falling into sharp local minima. The increase of mini-batch sizes over the training procedure provides a more robust gradient estimation on a sharper energy landscape that is supposed to be in a region of the space with better generalisation properties.

The most important point that dynamic sampling showed with respect to our proposed measures C_K, L_K in (3.20),(3.21) is that it broke the stability observed in Figure 3.8. In general, we note that the application of dynamic sampling allowed the curves to move from the region with the original batch size to the region of the final batch size.

3.6.3 Dynamic Stochastic Depth Rate

The stochastic depth ResNet [42] drops a random number of residual units at each epoch k, with probability p_l to drop residual blocks (through b_l in (3.15)) towards the last layer L of the model, as follows:

$$p_l = 1 - \frac{l}{L}(1 - p_L), \tag{3.22}$$

where p_L is a hyper-parameter that represents the drop probability of the L^{th} residual block. The proposed dynamic stochastic depth rate follows similar intuition as dynamic sampling of Section 3.6.2, where the runtime drop probability p_L is reduced as a function of training epoch k. In effect, the initial gradient estimations with large p_L are noisy, leading to the exploration of flat energy landscapes, and in later epochs, smaller p_L produce robust gradient estimations with a sharper energy landscape.

3.7 Conclusion

In this chapter, we presented the proposed methodologies for four fundamental deep learning problems: nonlinear classification using CNN features, normalisation of piecewise linear activation units, multiple-size features and CNN with Maxout units, and training characterisation of residual networks. The main goal of these methods is to improve the performance of deep learning classifiers, especially the CNN classifier, on challenging visual classification problems.

Chapter 4

The Use of Deep Learning Features in a Hierarchical Classifier Learned with the Minimization of a Non-greedy Loss Function that Delays Gratification

---

Zhibin Liao **Gustavo Carneiro**
ARC Centre of Excellence for Robotic Vision
University of Adelaide, Australia

The work contained in this chapter has been published in
the IEEE International Conference on Image Processing (ICIP), 2015.

The final publication is available at: DOI: 10.1109/ICIP.2015.7351666

Zhibin Liao, and Gustavo Carneiro. The Use of Deep Learning Features in a Hierarchical Classifier Learned with the Minimization of a Non-greedy Loss Function that Delays Gratification. In Proceedings of *the IEEE International Conference on Image Processing* (ICIP), 2015.

Statement of Authorship

Title of Paper	The Use of Deep Learning Features in a Hierarchical Classifier Learned with the Minimization of a Non-greedy Loss Function that Delays Gratification
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Zhibin Liao and Gustavo Carneiro. "The use of deep learning features in a hierarchical classifier learned with the minimization of a non-greedy loss function that delays gratification". Image Processing (ICIP), 2015 IEEE International Conference on. IEEE, 2015: 4540-4544.

Principal Author

Name of Principal Author (Candidate)	Zhibin Liao		
Contribution to the Paper	- Wrote all the coding and did all the experiments - Built the conceptual idea - Wrote and refined the manuscript		
Overall percentage (%)	50%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature	[illegible]	Date	19th May 2017

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Gustavo Carneiro		
Contribution to the Paper	- Built the conceptual idea - Wrote and refined the manuscript - Supervised the development of this work		
Signature		Date	22-05-17

THE USE OF DEEP LEARNING FEATURES IN A HIERARCHICAL CLASSIFIER LEARNED WITH THE MINIMIZATION OF A NON-GREEDY LOSS FUNCTION THAT DELAYS GRATIFICATION

Zhibin Liao *Gustavo Carneiro*

ARC Centre of Excellence for Robotic Vision, University of Adelaide, Australia

ABSTRACT

Recently, we have observed the traditional feature representations are being rapidly replaced by the deep learning representations, which produce significantly more accurate classification results when used together with the linear classifiers. However, it is widely known that non-linear classifiers can generally provide more accurate classification but at a higher computational cost involved in their training and testing procedures. In this paper, we propose a new efficient and accurate non-linear hierarchical classification method that uses the aforementioned deep learning representations. In essence, our classifier is based on a binary tree, where each node is represented by a linear classifier trained using a loss function that minimizes the classification error in a non-greedy way, in addition to postponing hard classification problems to further down the tree. In comparison with linear classifiers, our training process increases only marginally the training and testing time complexities, while showing competitive classification accuracy results. In addition, our method is shown to generalize better than shallow non-linear classifiers. Empirical validation shows that the proposed classifier produces more accurate classification results when compared to several linear and non-linear classifiers on Pascal VOC07 database.

1. INTRODUCTION

Ever since Krizhevsky and Hinton [1] published the outstanding classification results on ImageNet [2], the attention of the computer vision community has shifted from the bag of features [3] and Fisher vector [4] representations to the deep learning representation provided by convolutional neural networks [5, 6]. Recent results [7, 8, 9, 10, 11, 12] show that convolutional neural networks (CNN), a high-capacity multi-layer non-linear classification framework, can produce the most accurate classification results in several databases in the field. It is generally believed the convolutional layers of CNN are responsible for generating well-separated image representations. For this reason, simple linear support vector machine (SVM) and softmax [1] classifier are used to classify the CNN image representations. More complex classifiers are with higher capacity, such as non-linear SVM [13, 14] or boosting [15], but the main limitations are the high training and testing time complexities and the risk of overfitting the training data. Hierarchical models can achieve a good trade off between generalization and training and testing complexities, and for this reason it has been explored in computer vision in the past, such as with the probabilistic boosting tree (PBT) [16] and the discriminative learning of relaxed hierarchy (DLRH) [17]. In this paper, we propose a new hierarchical classifier to be used with high dimensional feature vectors (e.g., CNN features). The main novelty of our proposal is the loss function to learn this hierarchical classifier, which minimizes the classification error in a non-greedy way and at the same time delays hard classification problems to nodes further down the tree. Our main objective with this new training process of hierarchical classifiers is to reach a good trade-off between generalization and complexities, particularly when compared with linear [1, 7] , shallow non-linear [13, 15, 14] and hierarchical classifiers [17, 16] previoulsly proposed in the field. We test our method on the Pascal VOC07 [18, 19] database, and show that we produce better classification results and have less training test complexities, compared to other linear and non-linear classifiers using the deep learning features [7, 12].

This research was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016)

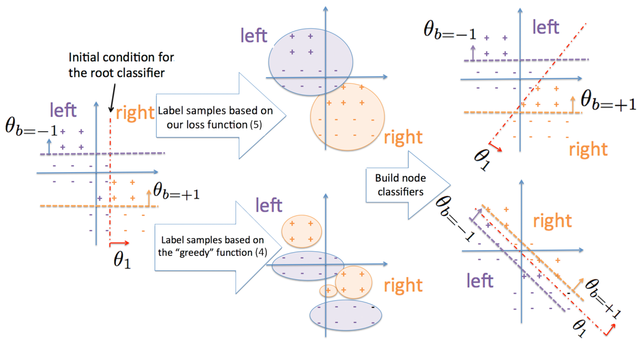
2. LITERATURE REVIEW

In this section, we briefly describe the related work in the field, by first introducing the mid-level features extracted from convolutional neural nets, then by discussing the classifiers that are (or can) be used with such feature vectors.

Deep learning methods have been present in the field for several years [20, 21], but their use as feature generators has been considered only after the outstanding result obtained on ImageNet [1]. Using millions of training images, a large number of nodes and hidden layers and an effective implementation, Krizhevsky and Hinton [1] produced a CNN model that integrates feature extraction and classification functionalities into an easy-to-use framework. In addition to this, Razavian et al. [11] have found that the CNN mid-layer activations trained with ImageNet provide powerful off-the-shelf image representations for other databases. This results have been confirmed by Chatfield et al. [3], who showed that much deeper CNN mid-layer activations produce much more accurate classification results than the previous bag of features [3] and Fisher vector [4] representations.

An interesting observation about the use of these CNN features is that the classifier is usually based on linear SVM [13] or softmax [1], which are relatively low-capacity models that are efficiently trained and tested, but offer limited accuracy depending on the distribution of the training set samples. In general shallow high-capacity classifiers can handle more difficult classification problems better than linear classifiers, but at a usually higher running time computational cost and the risk of overfitting. On the other hand, hierarchical classifiers that use low capacity classifiers in each node, usually achieve a good trade-off between complexity and accuracy.

Boosting [15] and non-linear SVM [13] are typical examples of high-capacity shallow classifiers that can deal with highly complex classification problems. The former uses a quite large number of features and build simple linear (weak) classifier in each one of these feature spaces, which are then combined to form a strong clas-



Initial condition for the root classifier
left
right
$\theta_{b=-1}$
$\theta_{b=+1}$
θ_1
Label samples based on our loss function (5)
Label samples based on the "greedy" function (4)
Build node classifiers
left
right

Fig. 1: Comparison between the learning of the root node classifier parameter θ_1 (in a binary classification problem, with labels '+' and '-') using our proposed loss function in (5) in comparison to the "greedy" loss function in (4).

sifier. In general, the training of each weak classifier involves the estimation and comparison of low capacity classifiers in all possible feature spaces, which means that at least this training has complexity $O(DN)$, where D is the size of the feature space and N is the training set size. Considering that potentially, one can use the whole feature space, the complexity of the full training process is around $O(D^2N)$ and the testing complexity is $O(D)$. Non-linear SVM uses the kernel trick to project all training samples in the kernel space and builds a linear classifier there, which is a training process that has complexity around $O(DN^2)$. The testing complexity of non-linear SVM is $O(ND)$. In addition to the complex training and testing processes, both methods can overfit the training data given their large capacity.

The issues presented above have been realized in several works that propose the use of hierarchical classifiers instead of their shallow counterparts. The boosted cascade classifier [22] is a good example of a hierarchical classifier, based on a degenerate binary tree with a boosting classifier on each node, where the training involved in each node minimizes a loss function that greedily estimates the best decision boundary, but delays the decision about hard classification problems. The probabilistic Boosting Tree (PBT), proposed by Tu [16], is a full binary tree structured classifier that has a boosting classifier in each node, which minimizes a similar loss function. One important issue that affects the cascade and PBT classifiers is that each node uses a high capacity boosting classifier that can overfit the training data, hampering their generalization abilities and incrementing the training and testing time complexities. A more relevant work to our proposal is the discriminative learning of relaxed hierarchy (DLRH) [17], which consists of a binary tree, where each interior node is trained using a loss function that delays hard classification problems, but at the same time greedily reduces the classification error. This greedy error reduction can again overfit the training data.

3. METHODOLOGY

Assume that an image is represented by a feature vector $\mathbf{x} \in \mathbb{R}^D$, the image label is represented by the variable $y \in \{-1, 1\}$, and the binary tree classifier has one root node that classifies samples using a hidden variable $b \in \{-1, 1\}$ (which indicates the left child by -1 and right by $+1$).

The inference is defined by the following problem:

$$y^* = \arg \max_{y \in \{-1,1\}} \sum_{b \in \{-1,1\}} P(y|\mathbf{x}, b, \theta_b) P(b|\mathbf{x}, \theta_1), \qquad (1)$$

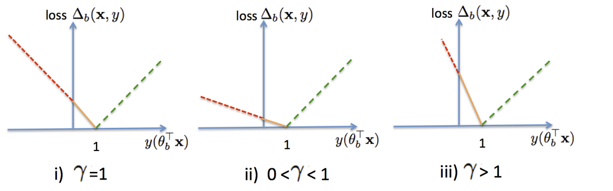


Fig. 2: Shape of the proposed loss function (5), where dotted red shows the region where classification is incorrect, solid orange displays the margin region, and dashed green depicts correct classification.

where $\theta_b, \theta_1 \in \mathbb{R}^D$ denote the classifier parameters of the leaf and root nodes, respectively. This inference is easily extended to a tree with three levels, as follows:

$$\begin{aligned} P(y|\mathbf{x}, \Theta) = & \sum_{b^{(1)}} \sum_{b^{(2)}} \sum_{b^{(3)}} P(y|\mathbf{x}, b^{(1)}, b^{(2)}, \theta_{b^{(1,2)}}) \\ & P(y|\mathbf{x}, b^{(1)}, b^{(3)}, \theta_{b^{(1,3)}}) P(b^{(2)}|\mathbf{x}, b^{(1)}, \theta_2) \\ & P(b^{(3)}|\mathbf{x}, b^{(1)}, \theta_3) P(b^{(1)}|\mathbf{x}, \theta_1), \end{aligned} \qquad (2)$$

where $b^{(1)}, b^{(2)}, b^{(3)} \in \{-1, 1\}$, and we assume that the classifier in the root node is represented by the variable $b^{(1)}$, its left child by $b^{(2)}$ and right child by $b^{(3)}$. The extension to higher trees is then trivial.

For the learning procedure, assume the availability of a training set $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^M$ and a validation set $\mathcal{V} = \{(\mathbf{x}_i, y_i)\}_{i=1}^Q$. The description of the learning methodology is clearer if we assume the tree has only one parent node (the extension to larger tree is again trivial), where the ideal learning process uses the training set $\mathcal{T}$ as follows:

$$\{\theta_b^*, \theta_1^*\} = \arg \max_{\{\theta_b, \theta_1\}} \prod_{i=1}^M \sum_{b_i} P(y_i|\mathbf{x}_i, b_i, \theta_b) P(b_i|\mathbf{x}, \theta_1), \qquad (3)$$

where we assume that the training samples are i.i.d. and $b_i \in \{-1, 1\}$. Eq. 3 involves the maximization of two functions, which are hard to be optimized jointly, so we break this optimization into the following iterative algorithm containing two alternating stages, where we assume that the parameters $\theta_b^{(t-1)}$ and $\theta_1^{(t-1)}$ are known at stage (t):

$$\begin{aligned} \theta_1^{(t)} &= \arg \max_{\theta_1} \prod_{i=1}^M \sum_{b_i} P(y_i|\mathbf{x}_i, b_i, \theta_b^{(t-1)}) P(b_i|\mathbf{x}, \theta_1), \\ \theta_b^{(t)} &= \arg \max_{\theta_b} \prod_{i=1}^M \sum_{b_i} P(y_i|\mathbf{x}_i, b_i, \theta_b) P(b_i|\mathbf{x}, \theta_1^{(t)}). \end{aligned} \qquad (4)$$

Essentially, this learning procedure involves the division of the training samples into two clusters, one representing the left child samples (labeled as $b = -1$) and the other, the right child samples (labeled as $b = +1$). After this division is performed, the parameter θ_1 of the classifier $P(b|\mathbf{x}, \theta_1)$ is estimated, and then, the classifier associated with each child can have its parameter θ_b estimated based on the samples belonging to that child (which is decided based on $P(b|\mathbf{x}, \theta_1)$). We consider this training process to be greedy because we maximize the classification probability $P(y|\mathbf{x}, b, \theta_b)$ even when splitting the training points to the left or right children. Our main contribution in this paper is the proposal of a non-greedy loss function used for the estimation of θ_1 in (4), which also delays hard classification problems.

The motivation for our proposed loss function is based on the graphs shown in Fig. 1. Assuming that both $P(y|\mathbf{x}, b, \theta_b)$ and $P(b|\mathbf{x}, \theta_1)$ are represented by linear classifiers (which means that

θ_b and θ_1 are vectors denoting the normal vectors of the learned hyperplane), notice that if we try to maximize $P(b|\mathbf{x},\theta_1)P(y|\mathbf{x},b,\theta_b)$ when estimating θ_1, using the depicted initial guess (graph on left), we will greedily label the '+' points with '+1' (right child) and the '-' points with '-1', which generates a very difficulty learning problem in the next iteration of the algorithm (graph on the bottom of Fig. 1). On the other hand, our loss function has a shape depicted by Fig. 2, which means that

1. if the classifiers in both children generate a correct classification (or both are incorrect), pick the one (with its respective child label) with the closest hyperplane,
2. if the classifier of only one child is correct, pick the correct one as long as it is not farther from the margin than the incorrect one.

This means that in addition to performing a division of the training samples without trying to greedily minimize the classification error (item (1) above), we also delay the hard classification problems to later stages of the binary tree [17, 16], as depicted in top part of Fig. 1. The loss function depicted in Fig. 2 is defined by:

$$f(b_i;\mathbf{x}_i,y_i) = \sum_b \Delta_b(\mathbf{x}_i,y_i)\delta(b_i - b), \quad (5)$$

with $\delta(.)$ denoting the Dirac delta function and

$$\Delta_b(\mathbf{x},y) = \gamma\max(0, 1 - y(\theta_b^\top\mathbf{x})) + \max(0, y(\theta_b^\top\mathbf{x}) - 1), \quad (6)$$

where $\gamma \in \mathbb{R}^+$ is a scalar that weighs the relative importance of the two terms in (6), and θ_b denotes the parameter of a linear SVM classifier learned for each child node b using the training samples $\mathbf{x}_i, y_i$ where $b_i = b$.

The proposed learning algorithm iterated steps 1-3 below (until convergence), assuming that we have the estimated value for $\theta_b^{(t-1)}$:

1). For each training sample $i \in \{1,...,M\}$, determine its label b_i with:

$$b_i^{(t)} = \arg\max_{b\in\{-1,+1\}} f(b_i;\mathbf{x}_i,y_i), \quad (7)$$

defined in (5) and using $\theta_b^{(t-1)}$;

2). Estimate $\theta_1^{(t)}$, as follows:

$$\begin{aligned}\theta_1^{(t)} &= \arg\max_{\theta_1}\prod_{i=1}^{M} P(b_i^{(t)}|\mathbf{x},\theta_1),\\ &= \arg\min_{\theta_1}\frac{1}{2}\|\theta_1\|^2 + \lambda\sum_{i=1}^{M}\log(1+\exp(-b_i^{(t)}\theta_1^\top\mathbf{x}_i)),\end{aligned} \quad (8)$$

which is a logistic regression classifier;

3). Estimate $\theta_b^{(t)}$, with:

$$\begin{aligned}\theta_b^{(t)} &= \arg\max_{\theta_b}\prod_{i=1}^{M} P(y_i|\mathbf{x}_i,b_i^{(t)},\theta_b)\delta(b_i^{(t)} - b)\\ &= \arg\min_{\theta_b}\frac{1}{2}\|\theta_b\|^2 + \lambda\sum_{i=1}^{M}\max(0, 1 - y_i(\theta_b^\top\mathbf{x}_i)),\end{aligned} \quad (9)$$

for $b \in \{-1,+1\}$, which is the definition for the linear SVM classifier (note that we use the soft margin training that allows for non-separable problems).

The initialization of this algorithm is achieved with the K-means clustering (with two clusters) considering the set $\{\mathbf{x}_i|y_i = +1, (\mathbf{x}_i,y_i) \in \mathcal{T}\bigcup\mathcal{V}\}$. We run this clustering 20 times and pick the labels b_i for the case that minimizes the loss in (5), and then we run from step (2) of the algorithm above. The stopping criterion is also based on the computation of the loss in (5), where when the loss difference between two iterations is smaller than a threshold ϵ, then we stop iterating. Finally, there is also a model selection problem involved in this method concerning the structure of the binary tree, where after convergence, we verify a condition to determine if we will backtrack to the previous structure or keep the current tree structure and thus continue to grow the tree. The initial tree contains only the root node and a linear SVM classifier (i.e., this is the original classifier found in previous works [7]), and the condition that we use for model selection is the classification accuracy (e.g., mean average precision) measured in the validation set $\mathcal{V}$ using the latest trained tree.

3.1. Complexity Analysis

For each node expansion, we train three separate linear classifiers per iteration of our algorithm. Therefore, we need K iterations in a tree with N nodes, where $N \in [2^h - 1, 2^{h+1} - 1]$, with h =maximum tree depth, so the training complexity of our method is $O(3KNDM)$ (recall that D is the feature size and M is the training set size), so this means that compared to the linear classifier, our training is $3KN$ slower. Fortunately, we can control the values for K and N by constraining the number of iterations and the tree depth, and in general we have $KN << D, M$, which means that our training is significantly faster than typical shallow non-linear classifiers (see Sec. 2). The testing complexity is essentially based on running h linear classifiers, which means that the running time complexity is $O(hD)$, which is just marginally larger than a single linear classifier given that $h \le 3$, typically.

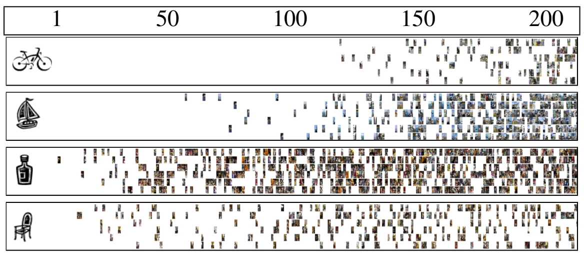


Fig. 3: Sorted Pascal VOC07 [19] first 200 retrieval results by using the VGG features [7]. Note that only the incorrectly classified images are shown, so the sparsity pattern of the correct retrieval can be analyzed by noticing the white regions of the image. For each presented class (see icon on the left [7]), each row is the result of the methods PBT [16], DLRH [17], Non-linear SVM (3-poly), Non-linear SVM (RBF), Linear SVM, and our method, respectively.

4. EXPERIMENTS

We quantitatively compare our proposed hierarchical classifier method with linear and non-linear SVM classifiers (RBF and 3rd-degree polynomial kernels), and also with the following hierarchical classification methods: PBT [16] and DLRH [17]. For the compared methods except the DLRH [17] method, which learns a single model for all classes jointly, we adopt the one-against-all (OVA) strategy for training each class. We re-implemented the PBT method exactly as described by Tu [16], and use the publicly available training method by Gao and Koller [17] available from their web page to train the DLRH. The LIBLINEAR [23] and LIBSVM [24] are used for training linear and non-linear SVMs respectively. The implementation of our method involves setting the values for the maximum number of iterations to $K = 1$ and the maximum number of tree nodes to $N = 3$ (see Sec. 3.1). Also, the weight γ in (5) and (6)

Table 1: AP results on Pascal VOC07 [19] using the **OverFeat** [12] and **VGG** [7] features. The best results per class are highlighted.

	Classifier	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	MAP
OverFeat features [12]	Non-linear SVM (RBF)	87.59	78.64	**87.72**	78.76	**46.61**	70.11	85.34	**82.44**	**63.81**	**59.61**	66.37	77.27	77.82	**80.92**	89.11	47.06	70.77	61.46	**88.41**	68.66	73.42
	Non-linear SVM (3-poly)	85.26	79.46	82.31	81.30	42.55	72.18	83.31	80.87	59.41	58.84	**67.26**	76.91	79.01	78.22	88.88	54.17	71.13	63.49	87.21	71.08	73.14
	DLRH [17]	86.69	78.33	82.94	82.39	37.00	69.52	85.26	81.21	59.08	52.37	65.54	78.56	79.00	79.28	88.33	53.17	68.72	60.99	86.22	68.35	72.15
	PBT [16]	85.44	78.40	82.86	80.54	40.48	69.38	84.41	78.01	55.68	59.35	63.94	75.46	81.11	77.41	89.36	49.72	**71.96**	56.20	85.85	66.35	71.60
	Linear SVM	**88.89**	79.72	84.35	82.02	44.63	73.26	**85.98**	81.76	61.47	57.49	67.08	77.99	81.20	78.92	90.55	**55.71**	71.43	**63.57**	87.13	**72.10**	74.26
	Our Method	**88.89**	**80.63**	84.32	**82.75**	43.25	**73.55**	85.66	81.76	61.47	59.51	67.20	**79.03**	**81.20**	78.92	**90.72**	**55.71**	71.46	**63.57**	86.96	**72.10**	**74.43**

	Classifier	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	MAP
VGG features [7]	Non-linear SVM (RBF)	90.76	**87.33**	89.79	86.47	51.83	**81.99**	**88.29**	87.89	64.01	**77.47**	76.82	**86.64**	88.13	**85.80**	92.19	60.82	82.36	70.15	90.39	75.38	80.73
	Non-linear SVM (3-poly)	90.39	84.91	88.15	86.06	53.47	79.16	85.40	86.46	62.85	73.91	**78.42**	84.57	84.69	81.16	89.63	59.66	81.82	70.49	90.83	73.50	79.28
	DLRH [17]	**93.13**	85.19	88.99	86.69	51.28	78.28	87.67	87.69	60.72	71.58	72.63	85.92	85.64	84.00	90.19	58.21	80.56	70.80	91.41	71.81	79.12
	PBT [16]	91.16	84.58	87.96	82.70	47.68	77.30	86.02	86.68	57.77	74.75	69.18	82.28	86.89	82.32	91.20	53.90	79.49	66.30	91.19	70.97	77.52
	Linear SVM	91.44	86.31	89.54	85.93	53.16	79.74	87.74	87.93	64.79	75.97	78.05	85.14	**88.20**	83.83	92.33	60.16	**82.52**	71.59	91.84	74.57	80.54
	Our Method	91.90	86.80	**89.88**	**86.59**	**53.59**	80.47	87.87	**88.75**	**66.50**	75.53	77.87	85.88	87.83	84.46	**92.70**	**62.18**	82.22	**72.65**	**91.94**	**75.38**	**81.05**

that controls the shape of loss function is set at $\gamma = 1$ (we reached this value by varying $\gamma \in [0.1, 100]$ and noticing little change in the classification results).

We evaluate the performance of the methods on the Pascal VOC07 [19] database, which contains 9963 images of 20 visual classes. The performance of each class is measured by average precision (AP), which is the standard ranking efficiency measurement. The mean AP (MAP) of all 20 classes is also listed to show the average performance. The parameter C of the SVM training are cross-validated from the standard training-validation split of the database. The running time of the training and testing processes are obtained using a computer with the following configuration: Intel(R) Xeon(R) CPU @ 2.70GHz, with 8GB of memory.

The features are extracted from two publicly available CNN models: OverFeat [12] and VGG [7]. Both models were trained on the ImageNet ILSVRC12 [25] database, which contains 1.2 million images with 1000 classes. For all experiments, the extracted CNN features are $L2$ normalized before being used as the input features for the classifiers. The OverFeat [12] feature is extracted from the first fully connected layer (layer 22) of its 'accurate' architecture, which has 4096 dimensions and is the fully connected layer before the rectification layer. No data augmentations were used to increase the training volume. For the VGG [12], Chatfield et al. offer several pre-trained models and data augmentation choices. We choose the 2048-dimensional feature model with the augmentation choice 'ss' (both training and testing are max-pooled as one sample from the ten samples) option because this combination is reported with the top results in [7]. In our experiments, we noticed the VGG features are extracted after the rectification layer (note that we consider the fact that these features are extracted after the rectification layer as one of the major differences compared to the OverFeat features).

Table 2: Average running times (in seconds) of the training and testing methods using the VGG features of Table 1 on Pascal VOC07 [19]. The results is reported as the average training time (in seconds) per visual class and the testing time per testing image.

Classifier	Training time/class	Testing time/image
Non-linear SVM (RBF)	36.7	36.4
Non-linear SVM (3-poly)	40.1	34.1
DLRH [17]	16.6	2.5
PBT [16]	1459.3	0.5
Linear SVM	1.4	0.6
Our method	11.7	0.1

4.1. Results

In Table 1, we show the AP results obtained with the OverFeat features [12] and VGG features [7] on Pascal VOC07 [19] using the studied classifiers. The running time of the training and testing processes of these classifiers listed are shown in Table 2. We show in Fig. 3 the sorted retrieval results, where only the wrong cases are shown in order to assess the sparsity pattern of the detection results for each classifier (i.e., the white regions represent correctly retrieved samples).

5. DISCUSSION AND CONCLUSIONS

By analyzing the results, our methodology provides the best MAP among all considered classifiers in Tables 1. We also notice to notice the non-linear methods provide competitive results for some individual classes. This demonstrates that non-linear SVM methods are relevant, particularly in cases where the database is not large (such as the case with Pascal VOC07). On the other hand, the linear SVM always produce robust average classification results, but rarely presents the best per-class result. This shows that linear SVM can generalize well but may have difficulties dealing with some of the more complicated classification problems. Compared to the other hierarchical methods [17, 16], our approach shows superior accuracy, which, we believe, is related to the use of our proposed loss function for training the hierarchical classifier and the fact that the node classifiers are linear methods (which provides good generalization properties). Analyzing the sparsity pattern of the retrieval results in Fig. 3, we see that our approach tends to have one of the most sparse results among the studied methods. It also shows that our approach takes longer to retrieve the first incorrectly classified images. These two facts confirm the AP results from Tables 1.

In addition, our binary trees can grow up to $N = 3$ nodes limits the number of vector multiplications ($\mathbf{x}^\top \theta$) to 2 times per test image which brings the fastest testing time. For linear SVM, the testing time is longer than our method mostly because of the overheads involved in running LIBLINEAR [23]. In terms of training time, we are significantly faster than the non-linear SVM methods and PBT [16], but comparable to DLRH [17], while the linear SVM classifier has the fastest training time. We believe that our training time presents competitive results because we limit the number of iterations to $K = 1$ and the size of the tree in $N = 3$ nodes (see Sec. 3.1), which make the theoretical training complexity only slightly larger than linear SVM.

We plan to apply this method to other databases [2, 26, 27] and use other CNN features that will become available in the near future. We also plan to increase the difficulty of the current database and verify if the impact of the non-linear methods is more remarkable.

6. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems (NIPS)*, 2012, pp. 1097–1105.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 248–255.

[3] Josef Sivic, Andrew Zisserman, et al., "Video google: A text retrieval approach to object matching in videos.," in *International Conference on Computer Vision (ICCV)*, 2003, vol. 2, pp. 1470–1477.

[4] Florent Perronnin, Jorge Sánchez, and Thomas Mensink, "Improving the fisher kernel for large-scale image classification," *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 143–156, 2010.

[5] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3642–3649.

[6] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[7] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *British Machine Vision Conference (BMVC)*, 2014, pp. 1–12.

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*. ACM, 2014, pp. 675–678.

[10] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1717–1724.

[11] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson, "Cnn features off-the-shelf: An astounding baseline for recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Washington, DC, USA, 2014, CVPRW '14, pp. 512–519, IEEE Computer Society.

[12] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2014, pp. 1–16.

[13] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 609–616.

[15] Yoav Freund and Robert E Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*. Springer, 1995, pp. 23–37.

[16] Zhuowen Tu, "Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2005, vol. 2, pp. 1589–1596.

[17] Tianshi Gao and Daphne Koller, "Discriminative learning of relaxed hierarchy for large-scale visual recognition," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2072–2079.

[18] Mark Everingham, LV Gool, Chris Williams, and Andrew Zisserman, "Pascal visual object classes challenge results," *Available from www. pascal-network. org*, 2005.

[19] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC 2007) Results," The PASCAL Visual Object Classes Challenge 2007 (VOC 2007) Results [Online].

[20] Yann LeCun and Yoshua Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, 1995.

[21] Geoffrey E Hinton and Ruslan R Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[22] Paul Viola and Michael Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2001, vol. 1, pp. 1–1.

[23] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin, "Liblinear: A library for large linear classification," *The Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[24] Chih-Chung Chang and Chih-Jen Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 27, 2011.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "Imagenet large scale visual recognition challenge," 2014.

[26] Ariadna Quattoni and Antonio Torralba, "Recognizing indoor scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 413–420.

[27] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, pp. 3485–3492.

Chapter 5

On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units

Zhibin Liao **Gustavo Carneiro**
ARC Centre of Excellence for Robotic Vision
University of Adelaide, Australia

The work contained in this chapter has been published in
the IEEE Winter Conference on Applications of Computer Vision (WACV), 2016.

The final publication is available at: DOI: 10.1109/WACV.2016.7477624

Zhibin Liao, and Gustavo Carneiro. On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units. In Proceedings of *the IEEE Winter Conference on Applications of Computer Vision* (WACV), 2016.

Statement of Authorship

Title of Paper	On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Zhibin Liao, Gustavo Carneiro. "On the importance of normalisation layers in deep learning with piecewise linear activation units". Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on. IEEE, 2016: 1-8.

Principal Author

Name of Principal Author (Candidate)	Zhibin Liao		
Contribution to the Paper	- Wrote all the coding and did all the experiments - Built the conceptual idea - Wrote and refined the manuscript		
Overall percentage (%)	50%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	19th May 2017

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Gustavo Carneiro		
Contribution to the Paper	- Built the conceptual idea - Wrote and refined the manuscript - Supervised the development of this work		
Signature		Date	22-05-17

On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units

Zhibin Liao Gustavo Carneiro
ARC Centre of Excellence for Robotic Vision
University of Adelaide, Australia
{zhibin.liao,gustavo.carneiro}@adelaide.edu.au

Abstract

Deep feedforward neural networks with piecewise linear activations are currently producing the state-of-the-art results in several public datasets (e.g., CIFAR-10, CIFAR-100, MNIST, and SVHN). The combination of deep learning models and piecewise linear activation functions allows for the estimation of exponentially complex functions with the use of a large number of subnetworks specialized in the classification of similar input examples. During the training process, these subnetworks avoid overfitting with an implicit regularization scheme based on the fact that they must share their parameters with other subnetworks. Using this framework, we have made an empirical observation that can improve even more the performance of such models. We notice that these models assume a balanced initial distribution of data points with respect to the domain of the piecewise linear activation function. If that assumption is violated, then the piecewise linear activation units can degenerate into purely linear activation units, which can result in a significant reduction of their capacity to learn complex functions. Furthermore, as the number of model layers increases, this unbalanced initial distribution makes the model ill-conditioned. Therefore, we propose the introduction of batch normalisation units into deep feedforward neural networks with piecewise linear activations, which drives a more balanced use of these activation units, where each region of the activation function is trained with a relatively large proportion of training samples. Also, this batch normalisation promotes the pre-conditioning of very deep learning models. We show that by introducing maxout and batch normalisation units to the network in network model results in a model that produces classification results that are better than or comparable to the current state of the art in CIFAR-10, CIFAR-100, MNIST, and SVHN datasets.

1. Introduction

The use of piecewise linear activation units in deep learning models [1–6], such as deep convolutional neural network (CNN) [7], has produced models that are showing state-of-the-art results in several public databases (e.g., CIFAR-10 [8], CIFAR-100 [8], MNIST [9] and SVHN [10]). These piecewise linear activation units have been the subject of study by Montufar et al. [1] and by Srivastava et al. [11], and the main conclusions achieved in these works are: 1) the use of a multi-layer composition of piecewise linear activation units allows for an exponential division (in terms of the number of network layers) of the input space [1]; 2) given that the activation units are trained based on a local competition that selects which region of the activation function a training sample will use, "specialized" subnetworks will be formed by the consistency that they respond to similar training samples (i.e., samples lying in one of the regions produced by the exponential division above) [11] and 3) even though subnetworks are formed and trained with a potentially small number of training samples, these models are not prone to overfitting because these subnetworks share their parameters, resulting in an implicit regularization of the training process [1,11].

An assumption made by these works is that a large proportion of the regions of the piecewise linear activation units are active during training and inference. For instance, in the Rectifier Linear Unit (ReLU) [2], Leaky-ReLu (LReLU) [3] and Parametric-ReLU (PReLU) [4], there must be two sets of points: one lying in the negative side and another on the positive side of the activation function domain (see region 1 covering the negative side and region 2 on the positive side in {P,L}ReLU cases of Fig. 1). Moreover, in the Maxout [6] and Local winner takes all (LWTA) [5] activation units, there must be k sets of points - each set lying in one of the k regions of the activation function domain (see Maxout case in Fig. 1). This assumption is of utmost importance because if violated, then the activation units may degenerate into simple linear functions that are not capable of exponentially

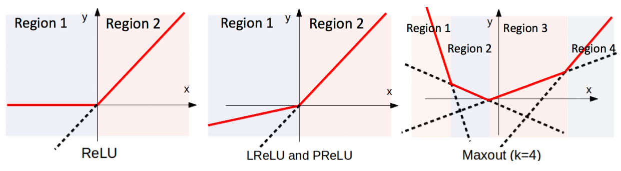


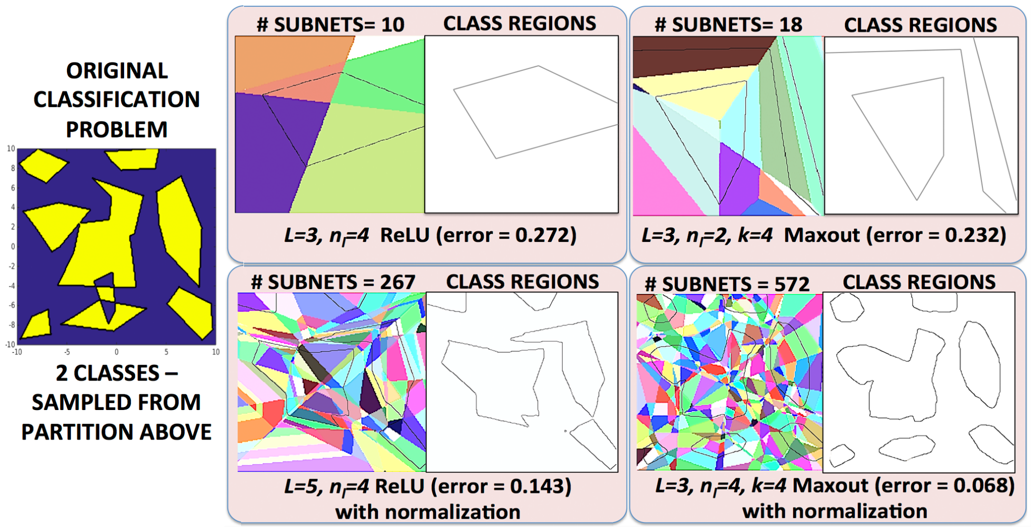
Figure 1. Piecewise linear activation functions: ReLU [2], LReLU [3], PReLU [4], and Maxout [6].

dividing the input space or training the "specialized" subnetworks (i.e., the model capacity is reduced). Moreover, in learning models that have very deep architectures, the violation of this assumption makes the model ill-conditioned, as shown in the toy example below. In this paper, we propose the introduction of batch normalisation units [12] before the piecewise linear activation units to guarantee that the input data is evenly distributed with respect to the activation function domain, which results in a more balanced use of all the regions of the piecewise linear activation units and pre-conditions the model. Note that Goodfellow et al. [6] have acknowledged this assumption and proposed the use of dropout [13] to regularize the training process, but dropout cannot guarantee a more balanced distribution of the input data in terms of the activation function domain. Furthermore, dropout is a regularization technique that does not help pre-condition the model. Therefore, the issues that we have identified remains with dropout.

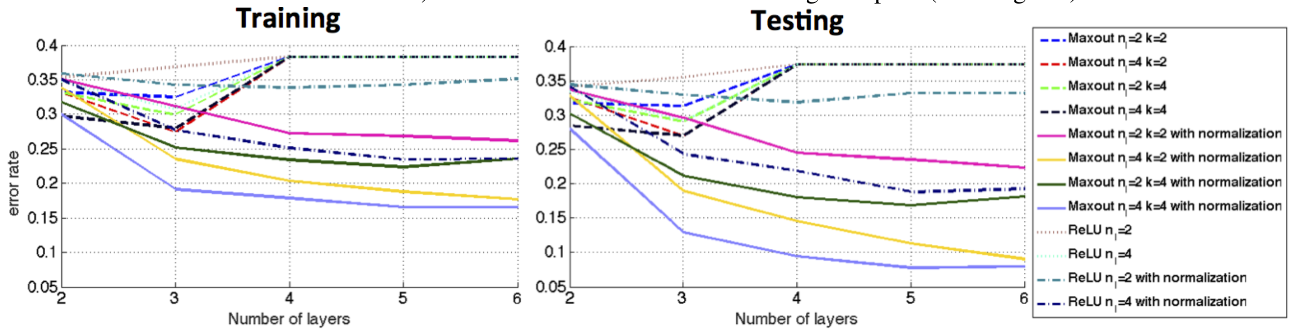
In order to motivate our observation and the model being proposed in this paper, we show a toy problem that illustrates well our points. Assume that we have a 2-D binary problem, where samples are drawn (12K for training and 2K for testing) using a uniform distribution between $[-10, 10]$ (in each dimension) from the partition shown in Fig. 2-(a) (leftmost image), with the colors blue and yellow indicating the class labels. We train a multi-layer perceptron (MLP) with varying number of nodes per layer $n_l \in \{2, 4\}$ and varying number of layers $L \in \{2, 3, 4, 5, 6\}$, and it is possible to place two types of piecewise linear activation functions after each layer: ReLU [2] and maxout [6], where for maxout we can vary the number of regions $k \in \{2, 4\}$ (e.g., Fig. 1 shows a maxout with 4 regions). Also, before each activation function, we have the option of placing a batch normalisation unit [12]. Training is based on backpropagation [14] using mini-batches of size 100, learning rate of 0.0005 for 20 epochs then 0.0001 for another 20 epochs, momentum of 0.9 and weight decay of 0.0001, where we run five times the training (with different training and test samples) and report the mean train and test errors. Finally, the MLP weights are initialized with Normal distribution scaled by 0.01 for all layers.

Analysing the mean train and test error in Fig. 2-(b), we first notice that all models have good generalization capability, which is a characteristic already identified for deep networks that use piecewise linear activation units [1,11]. Looking at the curves for the networks with 2 and 3 layers, where all models seem to be trained properly (i.e., they are pre-conditioned), the models containing batch normalisation units (denoted by "with normalisation") produce the smallest train and test errors, indicating the higher capacity of these models. Beyond 3 layers, the models that do not use the batch normalisation units become ill-conditioned, producing errors of 0.39, which effectively means that all points are classified as one of the binary classes. In general, batch normalisation allows the use of maxout in deeper MLPs that contain more nodes per layer, and the maxout function contains more regions (i.e., larger k). The best result (in terms of mean test and train error) is achieved with an MLP of 5 or more layers, where each layer contains 4 nodes and maxout has 4 regions (test error saturates at 0.07). The best results with ReLU are also achieved with batch normalisation, using a large number of layers (5 or more), and 4 nodes per layer, but notice that the smallest ReLU errors (around 0.19 on test set) are significantly higher than the maxout ones, indicating that maxout has larger capacity. The images in Fig. 2-(a) show the division of the input space (into linear regions) used to train the subnetworks within the MLP model (we show the best performing models of ReLU with and without normalisation and maxout with and without normalisation), where it is worth noticing that the best maxout model (bottom-right image) produces a very large number of linear regions, which generate class regions that are similar to the original classification problem. The input space division, used to train the subnetworks, are generated by clustering the training points that produce the same activation pattern from all nodes and layers of the MLP. We also run these same experiments using dropout (of 0.2), and the relative results are similar to the ones presented in Fig. 2-(b), but the test errors with dropout are around $2\times$ larger, which indicate that dropout does not pre-condition the model (i.e., the models that do not have the batch normalisation units still become ill-conditioned when having 3 or more layers), nor does it balance the input data for the activation units (i.e., the capacity of the model does not increase with dropout).

This toy experiment motivates us to propose a model that: 1) contains a large number of layers and nodes per layer, 2) uses maxout activation function [6], and 3) uses a batch normalisation unit [12] before each maxout layer. More specifically, we extend the *Network in Network* (NIN) model [15], where we replace the original ReLU units by batch normalisation units followed by maxout units. Replacing ReLU by maxout has the potential to increase the capacity of the model [6], and as mentioned before, the use of batch normalisation units will guarantee a more balanced distribution of this input data for those maxout units, which increases the model capacity and pre-conditions the


ORIGINAL CLASSIFICATION PROBLEM
2 CLASSES – SAMPLED FROM PARTITION ABOVE
# SUBNETS= 10
CLASS REGIONS
L=3, n_l=4 ReLU (error = 0.272)
# SUBNETS= 18
CLASS REGIONS
L=3, n_l=2, k=4 Maxout (error = 0.232)
# SUBNETS = 267
CLASS REGIONS
L=5, n_l=4 ReLU (error = 0.143) with normalization
# SUBNETS = 572
CLASS REGIONS
L=3, n_l=4, k=4 Maxout (error = 0.068) with normalization

a) Original classification problem (left) with the liner regions found by each model (represented by the color of each subnet) and classification division of the original space (class regions).



b) Train and test error as a function of the number of layers, number of nodes per layer, piecewise linear activation function, number of regions in the activation function, and the use of normalisation

Figure 2. Toy problem with the division of the space into linear regions and classification profile produced by each model (a), and a quantitative comparison between models (b).

model. We call our proposed model the maxout network in maxout network (MIM) - see Fig. 3. We assess the performance of our model on the following datasets: CIFAR-10 [8], CIFAR-100 [8], MNIST [9] and SVHN [10]. We first show empirically the improvements achieved with the introduction of maxout and batch normalisation units to the NIN model [15], forming our proposed MIM model, then we show a study on how this model provides a better pre-conditioning for the proposed deep learning model, and finally we show the final classification results on the datasets above, which are compared to the state of the art and demonstrated to be the best in the field in two of these datasets (CIFAR-10, CIFAR-100) and competitive on MNIST and SVHN.

2. Batch Normalised Deep Learning with Piecewise Linear Activation Units

In this section, we first explain the piecewise linear activation units, followed by an introduction of how the batch normalisation unit works and a presentation of the proposed MIM model, including its training and inference procedures.

The nomenclature adopted in this section is the same as the one introduced by Montufar et al. [1], where a feedforward neural network is defined by the function $F : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{\text{out}}$:

$$F(\mathbf{x}, \theta) = f_{\text{out}} \circ g_L \circ h_L \circ f_L \circ ... \circ g_1 \circ h_1 \circ f_1(\mathbf{x}), \quad (1)$$

where $f(.)$ represents a preactivation function, the parameter θ is formed by the input weight matrices $\mathbf{W}_l \in \mathbb{R}^{k.n_l \times n_{l-1}}$, bias vectors $\mathbf{b}_l \in \mathbb{R}^{k.n_l}$ and normalisation pa-

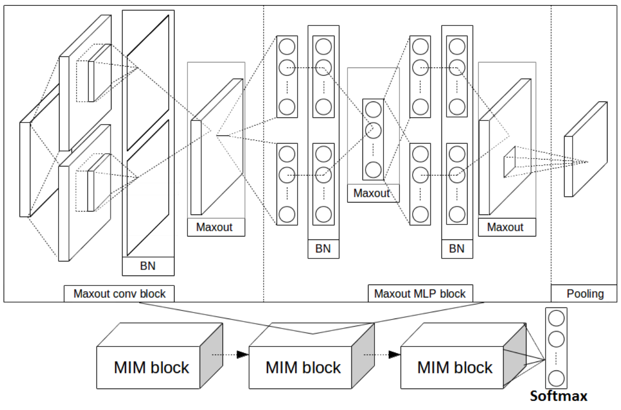


Figure 3. Proposed MIM model. The MIM model is based on the NIN [15] model. This model contains three blocks that have nearly identical architectures, with small differences in terms of the number of filters and stride in convolution layers. The first two blocks use max pooling and the third block uses average pooling.

rameters γ_l and β_l in (3) for each layer $l \in \{1, ..., L\}$, $h_l(.)$ represents a normalisation function, and $g_l(.)$ is a non-linear activation function. The preactivation function is defined by $f_l(\mathbf{x}_{l-1}) = \mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l$, where the output of the $(l-1)^{th}$ layer is $\mathbf{x}_l = [\mathbf{x}_{l,1}, ..., \mathbf{x}_{l,n_l}]$, denoting the activations $\mathbf{x}_{l,i}$ of the units $i \in \{1, ..., n_l\}$ from layer l. This output is computed from the activations of the preceding layer by $\mathbf{x}_l = g_l(h_l(f_l(\mathbf{x}_{l-1})))$. Also note that $\mathbf{f}_l = [\mathbf{f}_{l,1}, ..., \mathbf{f}_{l,n_l}]$ is an array of n_l preactivation vectors $\mathbf{f}_{l,i} \in \mathbb{R}^k$, which after normalisation, results in an array of n_l normalised vectors $\mathbf{h}_{l,i} \in \mathbb{R}^k$ produced by $h_{l,i}(f_{l,i}(\mathbf{x}_{l-1}))$, and the activation of the i^{th} unit in the l^{th} layer is represented by $\mathbf{x}_{l,i} = g_{l,i}(h_{l,i}(f_{l,i}(\mathbf{x}_{l-1})))$.

2.1. Piecewise Linear Activation Units

By dropping the layer index l to facilitate the nomenclature, the recently proposed piecewise linear activation units ReLU [2], LReLU [3], PReLU [4], and Maxout [6] are represented as follows [1]:

$$\begin{array}{ll} \text{ReLU:} & g_i(\mathbf{h}_i) = \max\{0, \mathbf{h}_i\}, \\ \text{LReLU or PReLU:} & g_i(\mathbf{h}_i) = \max\{\alpha.\mathbf{h}_i, \mathbf{h}_i\}, \\ \text{Maxout:} & g_i(\mathbf{h}_i) = \max\{\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,k}\}. \end{array} \tag{2}$$

where $\mathbf{h}_i \in \mathbb{R}$ and $k = 1$ for ReLU, LReLU [3], and PReLU [4], α is represented by a small constant in LReLU, but a learnable model parameter in PReLU, k denotes the number of regions of the maxout activation function, and $\mathbf{h}_i = [\mathbf{h}_{i,1}, ..., \mathbf{h}_{i,k}] \in \mathbb{R}^k$.

According to Montufar et al. [1], the network structure is defined by the input dimensionality n_0, the number of layers L and the width n_l of each layer. A linear region of the function $F : \mathbb{R}^{n_0} \to \mathbb{R}^m$ is a maximal connected subset of $\mathbb{R}^{n_0}$. Note from (2) that rectifier units have two behaviour types: 1) constant 0 (ReLU) or linear (LReLU or PReLU) with a small slope when the input is negative; and 2) linear with slope 1 when input is positive. These two behaviours are separated by a hyperplane (see Fig. 1) and the set of all hyperplanes within a rectifier layer forms a hyperplane arrangement, which split the input space into several linear regions. A multi-layer network that uses rectifier linear units with n_0 inputs and L hidden layers with $n \geq n_0$ nodes can compute functions that have $\Omega\left((n/n_0)^{L-1} n^{n_0}\right)$ linear regions, and a multi-layer network that uses maxout activation units with L layers of width n_0 and rank k can compute functions that have $k^{L-1} k^{n_0}$ linear regions [1]. These results indicate that multi-layer networks with maxout and rectifier linear units can compute functions with a number of linear regions that grows exponentially with the number of layers [1]. Note that these linear regions can be observed as the colored polygons in Fig. 2-(a), where the number of linear regions is denoted by ”# SUBNETS”.

The training process of networks containing piecewise linear activation units uses a divide and conquer strategy where $\frac{\partial \ell}{\partial \mathbf{W}_l}$ moves the classification boundary for layer l according to the loss function ℓ with respect to the points in its current linear region (similarly for the bias term $\mathbf{b}_l$), and $\frac{\partial \ell}{\partial \mathbf{x}_{l-1}}$ moves the offending points (i.e., points being erroneously classified) away from their current linear regions. Dividing the data points into an exponentially large number of linear regions is advantageous because the training algorithm can focus on minimizing the loss for each one of these regions almost independently of others - this is why we say it uses a divide and conquer algorithm. We also say that it is an almost independent training of each linear region because the training parameters for each region are shared with all other regions, and this helps the regularization of the training process. However, the initialization of this training process is critical because if the data points are not evenly distributed at the beginning, then all these points may lie in only one of the regions of the piecewise linear unit. This will drive the learning of the classification boundary for that specific linear region, where the loss will be minimized for all those points in that region, and the boundary for the other linear regions will be trained less effectively with much fewer points. This means that even the points with relatively high loss will remain in that initial region because the other regions have been ineffectively trained, and consequently may have a larger loss. This issue is very clear with the use of maxout units, where in the extreme case, only one of the k regions is active, which means that the maxout unit will behave as a simple linear unit. If a large amount of maxout units behave as linear units, then this will reduce the ability of these networks to compute functions that have an exponential number of linear regions, and consequently decrease the capacity of the model.

2.2. Batch Normalisation Units

In order to force the initialization to distribute the data points evenly in the domain of the piecewise activation functions, such that a large proportion of the k regions is used, we propose the use of batch normalisation by Ioffe and Szegedy's [12]. This normalisation has been proposed because of the difficulty in initializing the network parameters and setting the value for the learning rate, and also because the inputs for each layer are affected by the parameters of the previous layers. These issues lead to a complicated learning problem, where the input distribution for each layer changes continuously - an issue that is called covariate shift [12]. The main contribution of this batch normalisation is the introduction of a simple *feature-wise* centering and normalisation to make it have mean zero and variance one, which is followed by a batch normalisation (BN) unit that shifts and scales the normalised value. For instance, assuming that the input to the normalisation unit is $\mathbf{f} = [\mathbf{f}_1, ..., \mathbf{f}_{n_l}]$, where $\mathbf{f}_i \in \mathbb{R}^k$, the BN unit consists of two stages:

$$\begin{array}{ll} \text{Normalisation:} & \hat{\mathbf{f}}_{i,k} = \frac{\mathbf{f}_{i,k} - E[\mathbf{f}_{i,k}]}{\sqrt{\text{Var}[\mathbf{f}_{i,k}]}} \\ \text{Scale and shift:} & \mathbf{h}_{i,k} = \gamma_i \hat{\mathbf{f}}_{i,k} + \beta_i \end{array}, \qquad (3)$$

where the shift and scale parameters $\{\gamma_i, \beta_i\}$ are new network parameters that participate in the training procedure [12]. Another important point is that the BN unit does not process each training sample independently, but it uses both the training sample and other samples in a mini-batch.

2.3. Maxout Network in Maxout Network Model

As mentioned in Sec. 2.1, the number of linear regions that networks with piecewise linear activation unit can have grows exponentially with the number of layers, so it is important to add as many layers as possible in order to increase the ability of the network to estimate complex functions. For this reason, we extend the recently proposed Network in Network (NIN) [15] model, which is based on a CNN that uses a multi-layer perceptron (MLP) as its activation layer (this layer is called the Mlpconv layer). In its original formulation, the NIN model introduces the Mlpconv with ReLU activation after each convolution layer, and replaces the fully connected layers for classification in CNN (usually present at the end of the whole network) by a spatial average of the feature maps from the last Mlpconv layer, which is fed into a softmax layer. In particular, we extend the NIN model by replacing the ReLU activation after each convolution layer of the Mlpconv by a maxout activation unit, which has the potential to increase even further the model capacity. In addition, we also add the BN unit before the maxout units. These two contributions form our proposed model, we give it a simple name Maxout Network in Maxout Network Model (MIM), which is depicted in Fig. 3. Finally, we include a dropout layer [13] between MIM blocks for regularizing the model.

3. Experiments

We evaluate our proposed method on four common deep learning benchmarks: CIFAR-10 [8], CIFAR-100 [8], MNIST [9] and SVHN [10]. The CIFAR-10 [8] dataset contains 60000 32x32 RGB images of 10 classes of common visual objects (e.g., animals, vehicles, etc.), where 50000 images are for training and the rest 10000 for testing. The CIFAR-100 [8] is an extension of CIFAR-10, where the difference is that CIFAR-100 has 100 classes with 500 training images and 100 testing images for each class. In both CIFAR-10 and 100, the visual objects are well-centred in the images. The MNIST [9] dataset is a standard benchmark for comparing learning methods. It contains 70000 28x28 grayscale images of numerical digits from 0 to 9, divided as 60000 images for training and 10000 images for testing. Finally, the Street View House Number (SVHN) [10] dataset is a real-world digit dataset with over 600000 32x32 RGB images containing images of house numbers (i.e., digits 0-9). The cropped digits are well-centred and the original aspect ratio is kept, but some distracting digits are present next to the centred digits of interest. The dataset is partitioned into training, test and extra sets, where the extra 530000 images are less difficult samples to be used as extra training data.

For each of these datasets, we validate our algorithm using the same training and validation splitting described by Goodfellow et al. [6] in order to estimate the model hyperparameters. For the reported results, we run 5 training processes, each with different model initializations, and the test results consist of the mean and standard deviation of the errors in these 5 runs. Model initialization is based on randomly producing the MIM weight values using a Normal distribution, which is multiplied by 0.01 in the first layer of the first MIM block and by 0.05 in all remaining layers. Moreover, we do not perform data augmentation for any of these datasets and only compare our MIM model with the state-of-the-art methods that report non data-augmented results. For the implementation, we use the MatConvNet [16] CNN toolbox and run our experiments on a standard PC equipped with Intel i7-4770 CPU and Nvidia GTX TITAN X GPU. Finally, Tab. 1 specifies the details of the proposed MIM architecture used for each dataset.

Below, we first show experiments that demonstrate the performance of the original NIN model [15] with the introduction of maxout and BN units, which comprise our contributions in this paper that form the MIM model. Then, we show s study on how the BN units pre-conditions the NIN model. Finally, we show a comparison between our proposed MIM model against the current state of the art on the aforementioned datasets.

Arch.	m-conv1	m-mlp1	m-conv2	m-mlp2	m-conv3	m-mlp3
CIFAR-10 CIFAR-100 SVHN	5x5x192 stride. 1, pad. 2, k. 2 BN	1x1x160 stride. 1, pad. 0, k. 2 BN ↓ 1x1x96 stride. 1, pad. 0, k. 2 BN 3x-max.pool dropout	5x5x192 stride. 1, pad. 2, k. 2 BN	1x1x192 stride. 1, pad. 0, k. 2 BN ↓ 1x1x192 stride. 1, pad. 0, k. 2 BN 3x-max.pool dropout	3x3x192 stride. 1, pad. 0, k. 2 BN	1x1x160 stride. 1, pad. 0, k. 2 BN ↓ 1x1x10(100) stride. 1, pad. 0, k. 2 BN 8x-avg.pool
MNIST	5x5x128 stride. 1, pad. 2, k. 2 BN	1x1x96 stride. 1, pad. 0, k. 2 BN ↓ 1x1x48 stride. 1, pad. 0, k. 2 BN 3x-max.pool dropout	5x5x128 stride. 1, pad. 2, k. 2 BN	1x1x96 stride. 1, pad. 0, k. 2 BN ↓ 1x1x48 stride. 1, pad. 0, k. 2 B 3x-max.pool dropout	3x3x128 stride. 1, pad. 0, k. 2 BN	1x1x96 stride. 1, pad. 0, k. 2 BN ↓ 1x1x10 stride. 1, pad. 0, k. 2 BN 7x-avg.pool

Table 1. The proposed MIM model architectures used in the experiments. In each maxout-conv unit (m-conv), the convolution kernel is defined by the first row in the block: (height)x(width)x(num of units). The second row of the block contains the information of convolution stride (stride), padding (pad), and maxout rank (k). The third row contains the BN units. Each layer of the maxout-mlp unit (m-mlp) is equivalent to a maxout-conv unit with 1x1 convolution kernel size. A softmax layer is present as the last layer of the model (but not shown in this table). The model on top row is used on CIFAR-10 and 100 and SVHN, while the model on the bottom row is for MNIST.

Method	Test Error (mean $\pm$ standard deviation)
NIN [15]	10.41%
NIN with maxout (without BN)	$10.95 \pm 0.21\%$
NIN with ReLU (with BN)	$9.43 \pm 0.21\%$
MIM (= NIN with maxout and BN - our proposal)	$8.52 \pm 0.20\%$

Table 2. Results on CIFAR-10 of the introduction of maxout and BN units to NIN, which produce our proposed MIM model (last row).

3.1. Introducing Maxout and Batch Normalisation to NIN model

In this section, we use CIFAR-10 to show the contribution provided by each component proposed in this paper. The first row of Tab. 2 shows the published results of NIN [15]. In our first experiment, we replace all ReLU units from the NIN model by the maxout units (with $k = 2$) and run the training and test experiments described above (test results are shown in the second row of Tab. 2). Second, we include the BN units before each ReLU unit in the NIN model and show the test results in the third row of Tab. 2. Finally, we include the maxout and BN units to the NIN model, which effectively forms our proposed MIM model, and test results are displayed in the fourth row of Tab. 2.

3.2. Ill-conditioning Study in Real Datasets

The study of how the BN units pre-conditions the proposed model (on CIFAR-10 and MNIST) is shown in Fig. 4. For this evaluation, we use the combination of NIN and maxout units as the standard model, and ensure that the learning rate is the only varying parameter. We train five distinct models with learning rates in $[10^{-2}, 10^{1}]$ for CIFAR-10, and $[10^{-3}, 10^{1}]$ for MNIST, and plot the error curves with the mean and standard deviation values. From Fig. 4, we can see that without BN, a deep learning model can become ill-conditioned. It is also interesting to see that

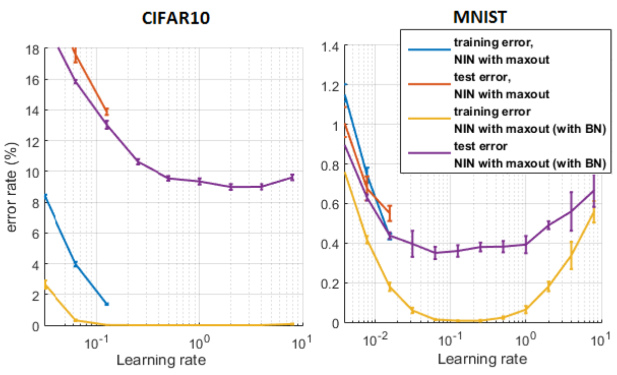


Figure 4. The ill-conditioning, measured by the model's inability to converge as a function of the learning rate. The training error (blue curve) and test error (orange curve) of the models trained without BN stay at the initial error when the learning rate is above a certain value, showing no sign of convergence (the results in terms of these learning rates are therefore omitted).

these un-normalized models give best performance right before the learning rate drives it into the ill-conditioning mode.

3.3. Comparison with the State of the Art

We compare the proposed MIM model (first row of Tab. 1) with Stochastic Pooling [17], Maxout Networks [6], Network in Network [15], Deeply-supervised nets [18], and recurrent CNN [19] on CIFAR-10 [8] and show the results in Tab. 3. The comparison on CIFAR-100 [8] against the same state-of-the-art models above, and also the Tree based Priors [20], is shown in Tab. 4. The performance on MNIST [9] of the MIM model (second row of Tab. 1) is compared against Stochastic Pooling [17], Conv.

Method	Test Error (mean ± standard deviation)
Stochastic Pooling [17]	15.13%
Maxout Networks [6]	11.68%
Network in Network [15]	10.41%
Deeply-supervised nets [18]	9.69%
RCNN-160 [19]	8.69%
MIM (our proposal)	$8.52 \pm 0.20\%$

Table 3. Comparison between MIM and the state-of-the-art methods on CIFAR-10 [8].

Method	Test Error (mean ± standard deviation)
Stochastic Pooling [17]	42.51%
Maxout Networks [6]	38.57%
Tree based Priors [20]	36.85%
Network in Network [15]	35.68%
Deeply-supervised nets [18]	34.57%
RCNN-160 [19]	31.75%
MIM (our proposal)	$29.20 \pm 0.20\%$

Table 4. Comparison between MIM and the state-of-the-art methods on CIFAR-100 [8].

Method	Test Error (mean ± standard deviation)
Stochastic Pooling [17]	0.47%
Conv. Maxout+Dropout [6]	0.47%
Network in Network [15]	0.45%
Deeply-supervised nets [18]	0.39%
MIM (our proposal)	$0.35 \pm 0.03\%$
RCNN-96 [19]	0.31%

Table 5. Comparison between MIM and the state-of-the-art methods on MNIST [9].

Method	Test Error (mean ± standard deviation)
Stochastic Pooling [17]	2.80%
Conv. Maxout+Dropout [6]	2.47%
Network in Network [15]	2.35%
MIM (our proposal)	$1.97 \pm 0.08\%$
Dropconnect [21]	1.94%
Deeply-supervised nets [18]	1.92%
RCNN-192 [19]	1.77%

Table 6. Comparison between MIM and the state-of-the-art methods on SVHN [10].

Maxout+Dropout [6], Network in Network [15], Deeply-supervised nets [18], and recurrent CNN [19] in Tab. 5. It is important to mention that the best result we observed with the MIM model on MNIST over the 5 runs is 0.32%. Finally, our MIM model in the first row of Tab. 1 is compared against the same models above, plus Dropconnect [21] on SVHN [10], and results are displayed in Tab. 6.

4. Discussion and Conclusion

The results in Sec. 3.1 show that the replacement of ReLU by maxout increases the test error on CIFAR-10, similarly to what has been shown in Fig. 2. The introduction of BN with ReLU activation units provide a significant improvement of the test error, and the introduction of BN units before the maxout units produce the smallest error, which happens due to the even input data distribution with respect to the activation function domain, resulting in a more balanced use of all the regions of the maxout units. The study in Sec. 3.2 clearly shows that the introduction of BN units pre-conditions the model, allowing it to use large learning rates and produce more accurate classification. The comparison against the current state of the art (Sec. 3.3) shows that the proposed MIM model produces the best result in the field on CIFAR-10 and CIFAR-100. On MNIST, our best result over five runs is comparable to the best result in the field. Finally, on SVHN, our result is slightly worse than the current best result in the field. An interesting point one can make is with respect to the number of regions k that we set for the MIM maxout units. Note that we set $k = 2$ because we did not notice any significant improvement with bigger values of k, and also because the computational time and memory requirements of the training become intractable.

This paper provides an empirical demonstration that the combination of piecewise linear activation units with BN units provides a powerful framework to be explored in the design of deep learning models. More specifically, our work shows how to guarantee the assumption made in the use of piecewise linear activation units about the balanced distribution of the input data for these units. This empirical evidence can be shown more theoretically in a future work, following the results produced by Montufar, Srivastava and others [1,11].

Acknowledgement

This research was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016)

References

[1] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2924–2932.

[2] Vinod Nair and Geoffrey E Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of International Conference on Machine Learning*, 2010, pp. 807–814.

[3] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013, vol. 30.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[5] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber, "Compete to compute," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 2310–2318.

[6] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio, "Maxout networks.," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013, vol. 28, pp. 1319–1327.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems (NIPS)*, 2012, pp. 1097–1105.

[8] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on Deep Learning and Unsupervised Feature Learning*, 2011, p. 5.

[11] Rupesh Kumar Srivastava, Jonathan Masci, Faustino Gomez, and Jürgen Schmidhuber, "Understanding locally competitive networks," in *International Conference on Learning Representations (ICLR)*, 2015, pp. 1–11.

[12] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, vol. 37, pp. 448–456.

[13] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, pp. 1, 1988.

[15] Min Lin, Qiang Chen, and Shuicheng Yan, "Network in network," in *International Conference on Learning Representations (ICLR)*, 2014, pp. 1–10.

[16] Andrea Vedaldi and Karel Lenc, "Matconvnet: Convolutional neural networks for matlab," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 689–692.

[17] Matthew D Zeiler and Rob Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," in *International Conference on Learning Representations (ICLR)*, 2013, pp. 1–9.

[18] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu, "Deeply-supervised nets," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015, pp. 562–570.

[19] Ming Liang and Xiaolin Hu, "Recurrent convolutional neural network for object recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3367–3375.

[20] Nitish Srivastava and Ruslan R Salakhutdinov, "Discriminative transfer learning with tree-based priors," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 2094–2102.

[21] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013, vol. 28, pp. 1058–1066.

Chapter 6

A Deep Convolutional Neural Network Module that Promotes Competition of Multiple-size Filters

Zhibin Liao   **Gustavo Carneiro**
ARC Centre of Excellence for Robotic Vision
University of Adelaide, Australia

The work contained in this chapter has been published in
Pattern Recognition (PR), 2017.

Zhibin Liao, and Gustavo Carneiro. A Deep Convolutional Neural Network Module that Promotes Competition of Multiple-size Filters. Published in *Pattern Recognition* (PR), 2017.

Statement of Authorship

Title of Paper	A Deep Convolutional Neural Network Module that Promotes Competition of Multiple-size Filters
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Zhibin Liao and Gustavo Carneiro. "A deep convolutional neural network module that promotes competition of multiple-size filters". Pattern Recognition, 2017.

Principal Author

Name of Principal Author (Candidate)	Zhibin Liao		
Contribution to the Paper	- Wrote all the coding and did all the experiments - Built the conceptual idea - Wrote and refined the manuscript		
Overall percentage (%)	50%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	26th May 2017

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Gustavo Carneiro		
Contribution to the Paper	- Built the conceptual idea - Wrote and refined the manuscript - Supervised the development of this work		
Signature		Date	29 May 2017

A Deep Convolutional Neural Network Module that Promotes Competition of Multiple-size Filters

Zhibin Liao Gustavo Carneiro
ARC Centre of Excellence for Robotic Vision
University of Adelaide, Australia

Abstract

We introduce a new deep convolutional neural network (ConvNet) module that promotes competition amongst a set of convolutional filters of multiple sizes. This new module is inspired by the inception module, where we replace the original collaborative pooling stage (consisting of a concatenation of the multiple size filter outputs) by a competitive pooling represented by a maxout activation unit. This extension has the following two objectives: 1) the selection of the maximum response amongst the multiple size filters prevents filter co-adaptation and allows the formation of multiple sub-networks within the same model, which has been shown to facilitate the training of complex learning problems; and 2) the maxout unit reduces the dimensionality of the outputs from the multiple size filters. We show that the use of our proposed module in typical deep ConvNets produces classification results that are competitive with the state-of-the-art results on the following benchmark datasets: MNIST, CIFAR-10, CIFAR-100, SVHN, and ImageNet ILSVRC 2012.

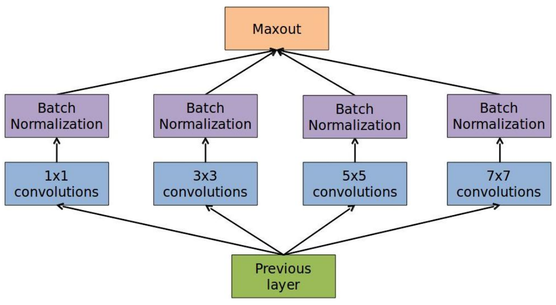
1 Introduction

The use of competitive activation units in deep convolutional neural networks (ConvNets) is generally understood as a way of building one network by the combination of multiple sub-networks, with each one being capable of solving a simpler task when compared to the complexity of the original problem involving the whole dataset [1]. Similar ideas have been explored in the past using multi-layer perceptron models [2], but there is a resurgence in the use of competitive activation units in deep ConvNets [3, 1]. For instance, rectified linear unit (ReLU) [4] promotes a competition between the input sum (usually computed from the output of convolutional layers) and a fixed value of 0, while maxout [5] and local winner-take-all (LWTA) [3] explore an explicit competition amongst the input units. As shown by Srivastava et al. [1], these competitive activation units allow the formation of sub-networks that respond consistently to similar input patterns, which facilitates training [4, 5, 3] and generally produces superior classification results [1].

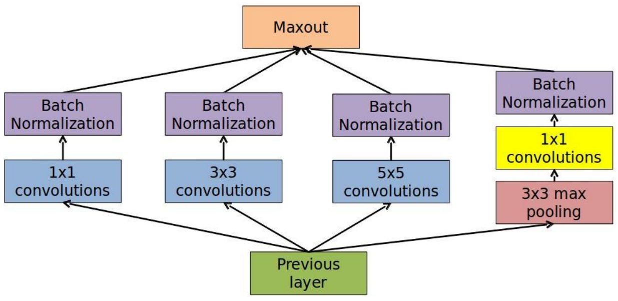
In this paper, we introduce a new module for deep ConvNets composed of several convolutional filters of multiple sizes that are joined by a maxout activation unit, which promotes competition amongst these filters. Our idea has been inspired by the recently proposed inception module [6], which currently produces state-of-the-art results on the ILSVRC 2014 classification and detection challenges [7]. The gist of our proposal is depicted in Fig. 1, where we have the data in the input layer filtered in parallel by a set of convolutional filters of multiple sizes [8, 6, 9]. Then the output of each filter of the convolutional layer passes through a batch normalisation unit (BNU) [10] that weights the importance of each filter size and also pre-conditions the model (note that the pre-conditioning ability of BNUs in ConvNets containing piece-wise linear activation units has been empirically shown in [11]). Finally, the multiple size filter outputs, weighted by BNU, are joined with a maxout unit [5] that reduces the dimensionality of the joint filter outputs and promotes competition amongst the multiple size filters. We empirically show that the competition amongst filters of multiple size prevents filter co-adaptation and allows the formation of multiple sub-networks. Furthermore, we show that the introduction of our proposal module in a typical deep ConvNet produces competitive results in the field for the benchmark datasets MNIST [12], CIFAR-10 [13], CIFAR-100 [13], street view house number (SVHN) [14], and ImageNet ILSVRC 2012 [7].

2 Literature Review

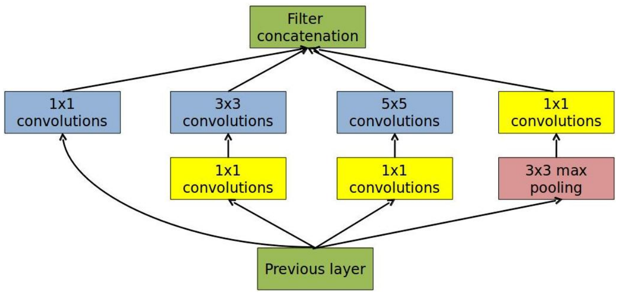
One of the main reasons behind the outstanding performance of deep ConvNets is attributed to the use of competitive activation units in the form of piece-wise linear functions [15, 1], such as ReLU [4], maxout [5] and LWTA [3] (see Fig. 2). In general, these activation functions enable the formation of sub-networks that



(a) Competitive multiple-size convolution module



(b) Competitive Inception module



(c) Original inception module

Figure 1: The proposed deep ConvNet modules are depicted in (a) and (b), where (a) only contains multiple size convolutional filters within each module, while (b) contains the max-pooling path, which resembles the original inception module [6] depicted in (c) for comparison.

respond consistently to similar input patterns [1], dividing the input data points (and more generally the training space) into regions [15], where classifiers and regressors can be learned more effectively given that the sub-problems in each of these regions are simpler than the original problem involving the whole train-

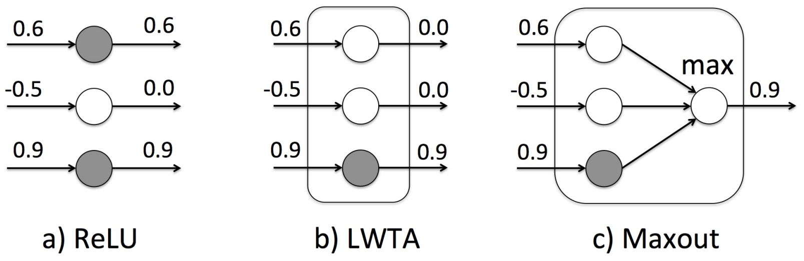


Figure 2: Competitive activation units, where the grey nodes are the active ones, from which errors flow during backpropagation. ReLU [4] (a) is active when the input is bigger than 0, LWTA [3] (b) activates only the node that has the maximum value (setting to zero the other ones), and maxout [5] (c) has only one output containing the maximum value from the input. This figure was adapted from Fig.1 of [1].

ing set. In addition, the joint training of the sub-networks present in such deep ConvNets represents a useful regularisation method [4, 5, 3]. In practice, ReLU, maxout and LWTA allows the division of the input space in exponentially many regions as a function of the number of layers and the number of input nodes to each competitive activation unit, so this means that maxout and LWTA can estimate exponentially complex functions more effectively than ReLU because of the larger number of sub-networks that are jointly trained. An important aspect of deep ConvNets with competitive activation units is the fact that the use of batch normalisation units (BNU) helps not only with respect to the convergence rate [10], but also with the pre-conditioning of the model by promoting an even distribution of the input data points, which results in the maximisation of the number of the regions (and respective sub-networks) produced by the piece-wise linear activation functions [11]. Furthermore, training ConvNets with competitive activation units [1, 11] usually involves the use of dropout [16] that consists of a regularisation method that prevents filter co-adaptation [16], which is a particularly important issue in such models, because filter co-adaptation can lead to a severe reduction in the number of the sub-networks that can be formed during training.

Another aspect of the current research on deep ConvNets is the idea of making the network deeper, which has been shown to improve classification results [17]. However, one of the main ideas being studied in the field is how to increase the depth of a ConvNet without necessarily increasing the complexity of the model parameter space [18, 6]. For the Szegedy et al.'s model [6], this is achieved with the use of 1×1 convolutional filters [19] that are placed before each local filter present in the inception module in order to reduce the input dimensionality of the filter. In Simonyan et al.'s approach [18], the idea is to use a large number of layers with convolutional filters of small size (e.g., 3×3). In this work, we restrict the complexity of the deep ConvNet with the use of maxout activation units, which selects only one of the input nodes, as shown in Fig, 2.

Finally, the use of multiple size filters in deep ConvNets is another important implementation that is increasingly being explored by several researchers [8, 6, 9]. Essentially, multiple size filtering follows a neuroscience model [20] that suggests that the input image data should be processed by filters of different sizes (which can lead to filters of different scales) and then pooled together, so that the deeper processing stages can become more robust to scale changes [6]. We explore this idea in our proposal, as depicted in Fig. 1, but we hypothesise (and show supporting evidence) that the multiple size of the filters prevents their co-adaptation during training, leading to better generalisation. We also hypothesise and show evidence that what is driving this better generalisation is the fact that the multiple sizes of the filters promote the learning of features that are more different from each other within competitive units when compared to the single-size filters.

3 Methodology

Assume that an image is represented by $\mathbf{x} : \Omega \to \mathbb{R}$, where Ω denotes the image lattice, and that an image patch of size $(2k-1) \times (2k-1)$ (for $k \in \{1, 2, ..., K\}$) centred at position $i \in \Omega$ is represented by $\mathbf{x}_{i \pm (k-1)}$. The models being proposed in this paper follow the structure of the NIN model [19], and is in general defined as follows:

$$f(\mathbf{x}, \theta_f) = f_{out} \circ f_L \circ ... \circ f_2 \circ f_1(\mathbf{x}), \tag{1}$$

where $\circ$ denotes the composition operator, θ_f represents all the ConvNet parameters (i.e., weights and biases), $f_{out}(.)$ denotes an averaging pooling unit followed by a softmax activation function [19], and the network has blocks represented by $l \in \{1, ..., L\}$, with each block containing a composition of N_l modules with $f_l(\mathbf{x}) = f_l^{(N_l)} \circ ... \circ f_l^{(2)} \circ f_l^{(1)}(\mathbf{x})$. Each module $f_l^{(n)}(.)$ at a particular position

$i \in \Omega$ of the input data for block l is defined by:

$$\begin{aligned} f_l^{(n)}(\mathbf{x}_i) = \sigma\big(&\mathrm{BN}_{\gamma_1,\beta_1}(\mathbf{W}_1^\top \mathbf{x}_i),\ \mathrm{BN}_{\gamma_3,\beta_3}(\mathbf{W}_3^\top \mathbf{x}_{i\pm1}), ..., \\ &\mathrm{BN}_{\gamma_{2k-1},\beta_{2k-1}}(\mathbf{W}_{2k-1}^\top \mathbf{x}_{i\pm(k-1)}), \\ &\mathrm{BN}_{\gamma_p,\beta_p}(\mathbf{W}_1^\top p_{3\times3}(\mathbf{x}_{i\pm1}))\big), \end{aligned} \tag{2}$$

where $\sigma(.)$ represents the maxout activation function [5]$\sigma(x) = \max_{j\in\{1,2,...,J\}} x_j$, the convolutional filters of the module are represented by the weight matrices $\mathbf{W}_{2k-1}$ for $k \in \{1, ..., K_l\}$ (i.e., filters of size $2k-1 \times 2k-1 \times \#filters$, with $\#filters$ denoting the number of 2-D filters present in $\mathbf{W}$), which means that each module n in block l has K_l different filter sizes and $\#filters$ different filters, $\mathrm{BN}_{\gamma,\beta}$ represent the batch normalization transformation with scaling and shifting parameters [10], and $p_{3\times3}(\mathbf{x}_{i\pm1})$ represents a max pooling operator on the 3×3 subset of the input data for layer l centred at $i \in \Omega$, i.e.$\mathbf{x}_{i\pm1}$. The batch normalisation transformation $\mathrm{BN}_{\gamma,\beta}$ [10] is computed as

$$\begin{aligned} \mu_\mathcal{B} &= \frac{1}{m}\sum_{i=1}^{m} x_i, \\ \sigma_\mathcal{B}^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2, \\ \hat{x}_i &= \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}, \\ \mathrm{BN}_{\gamma,\beta}(x_i) &= \gamma\hat{x}_i + \beta, \end{aligned} \tag{3}$$

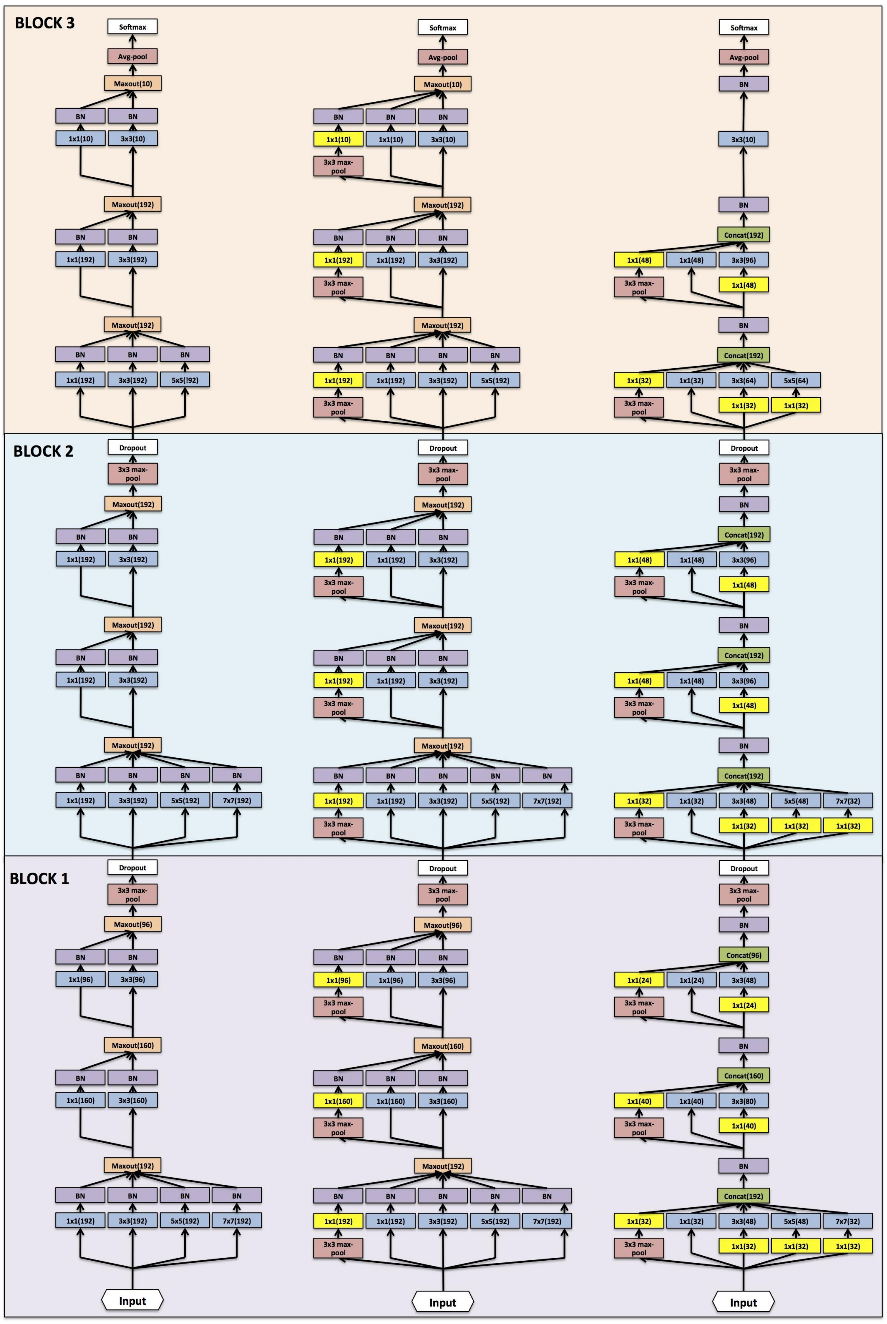
where $\mathcal{B} = \{x_1, ..., x_m\}$ represents a mini-batch of inputs, γ and β are two learnable parameters, and ϵ is a small constant.

Using the ConvNet module defined in (2), our proposed models differ mainly in the presence or absence of the node with the max-pooling operator within the module (i.e., the node represented by $\mathrm{BN}_{\gamma_p,\beta_p}(\mathbf{W}_1^\top p_{3\times3}(\mathbf{x}_{i\pm1}))$). When the module does not contain such node, it is called **Competitive Multiple-size Convolution** (see Fig. 3-(a)), but when the module has the max-pooling node, then we call it **Competitive Inception** (see Fig. 3-(b)) because of its similarity to the original inception module [6]. The original inception module is also implemented for comparison purposes (see Fig. 3-(c)), and we call this model the **Inception Style**, which is similar to (1) and (2) but with the following differences: 1) the function $\sigma(.)$ in (2) denotes the concatenation of the input parameters; 2) a 1×1 convolution is applied to the input $\mathbf{x}$ before a second round of convolutions with filter sizes larger than or equal to 3×3; and 3) a ReLU activation function [4] is present after each convolutional layer.

An overview of all models with the structural parameters is displayed in Fig. 3. Note that all models are inspired by NIN [19], GoogLeNet [6], and MIM [11]. In particular, we replace the original 5×5 convolutional layers of MIM by multiple size filters of sizes 1×1, 3×3, 5×5, and 7×7. For the inception style model, we ensure that the number of output units in each module is the same as for the competitive inception and competitive multiple size convolution, and we also use a 3×3 max-pooling path in each module, as used in the original inception module [6]. Another important point is that in general, when designing the inception style network, we follow the suggestion by Szegedy et al. [6] and include a relatively larger number of 3×3 and 5×5 filters in each module, compared to filters of other sizes (e.g., 1×1 and 7×7). An important distinction between the original GoogLeNet [6] and the inception style network in Fig. 3-(c) is the fact that we replace the fully connected layer in the last layer by a single 3×3 convolution node in the last module, followed by an average pooling and a softmax unit, similarly to the NIN model [19]. We propose this modification to limit the number of training parameters (with the removal of the fully connected layer) and to avoid the concatenation of the nodes from different paths (i.e., maxpooling, 1×1 convolution filter, and etc.) into a number of channels that is equal to the number of classes (i.e., each channel is averaged into a single node, which is used by a single softmax unit), where the concatenation would imply that some of the paths would be directly linked to a subset of the classes. Therefore, the last two layers of block 3 of all architectures in Fig. 3 contain an average pooling and a softmax unit to ensure a fair comparison amongst these networks.

3.1 Competitive Convolution of Multiple Size Filters Prevents Undesirable Filter Convergence and Filter Co-adaptation

The main reason being explored in the field to justify the use of competitive activation units [4, 5, 3] is the fact that they build a network formed by multiple underlying sub-networks [1]. More clearly, given that these activation units consist of piece-wise linear functions, it has been shown that the composition of several layers containing such units, divide the input space in a number of regions that is exponentially proportional to the number of network layers [15], where subnetworks will be trained with the samples that fall into one of these regions, and



(a) Competitive Multiple-size Convolution (b) Competitive Inception (c) Inception style

Figure 3: The proposed competitive multiple-size convolution (a) and competitive inception (b) networks, together with the reference inception style network (c). In these three models, we ensure that the output of each layer has the same number of units. Also note that the inception style model uses ReLU [21] after all convolutional layers, the number of filters per convolutional node is represented by the number in brackets, and these models assume a 10-class classification problem.

as a result become specialised to the problem in that particular region [1], where overfitting can be avoided because these sub-networks must share their parameters with one another [1]. It is worth noting that these regions can only be formed if the underlying convolutional filters do not converge to similar features, otherwise all input training samples will fall into only one region of the competitive unit, which degenerates into a simple linear transform, preventing the formation of the sub-networks.

A straightforward solution to avoid such undesirable convergence can be achieved by limiting the number of training samples in a mini-batch during stochastic gradient descent. These small batches allow the generation of "noisy" gradient directions during training that can activate different maxout gates, so that the different linear pieces of the activation unit can be fitted, allowing the formation of an exponentially large number of regions. However, the drawback of this approach lies in the determination of the "right" number of samples per mini-batch. A mini-batch size that is too small leads to poor convergence, and if it is too large, then it may not allow the formation of many sub-networks. Recently, Liao and Carneiro [11] propose a solution to this problem based on the use of BNU [10] that distributes the training samples evenly over the regions formed by the competitive unit, allowing the training to use different sets of training points for each region of the competitive unit, resulting in the formation of an exponential number of sub-networks. However, there is still a potential problem with that approach [11], which is that the underlying convolutional filters are trained using feature spaces of the same size (i.e., the underlying filters are of fixed size), which can induce the filters to converge to similar regions of the feature space, also preventing the formation of the sub-networks.

Our proposed module that promotes competition amongst filters of multiple sizes represents a way to prevent the undesirable convergence mentioned above [11]. To evaluate this hypothesis, we examine the similarity between convolutional filters within each competitive unit by empirically showing that the multiple sizes of the convolutional filters within a competitive unit promotes the learning of different features. This demonstration is based on measuring the orthogonality between filters within the same competitive unit, where filters that have similar responses will have orthogonality measures closer to one, and different responses will lead to orthogonality measures close to zero. The orthogonality between two filters, represented by $\mathbf{w}_1$ and $\mathbf{w}_2$, is measured by $\frac{|\mathbf{w}_1^\top \mathbf{w}_2|}{\|\mathbf{w}_1\|_2 \|\mathbf{w}_2\|_2}$, where zero padding along the filter border is used in order to allow the measure of orthogonality between filters of different sizes. We show the mean value of the orthogonality measures between pairs of single-size and multiple-size filters in a competitive unit in Table 1, where results are gathered from the first competitive layer of the first two blocks of the models trained on CIFAR-10 (see Sec. 4.1 for competitive single-size network architecture). The results from Table 1 provide evidence that the filter responses from the competitive multiple-size modules are more orthogonal to each other than the responses from the competitive single-size module.

Our second hypothesis is that the competition amongst multiple size filters within a convolution module prevents co-adaptation throughout the ConvNet model. The evidence for such hypothesis is displayed in Fig 4, which shows Yosinski et al.'s test [22] that assesses the transferability of ConvNet filters. The idea of this experiment is that the ConvNet with more co-adapted filters will produce larger testing errors when the higher level layers are removed because it is unlikely that the re-trained layers will be able to discover the complex co-adaptations present in the removed layers. To conduct this experiment, we train a model using one dataset (CIFAR-100), remove a subset of the higher level convolutional layers (e.g., from layers 7 to the last layer, where the indexing used in the horizontal axis of Fig. 3 is consistent with the first two competitive/inception layers to remove), and re-train only these layers for the CIFAR-10 dataset, while keeping the remaining lower level layers fixed (i.e., we do not re-train these lower layers). Fig. 4 shows that the competitive modules with filters of multiple sizes have the lowest testing error in all evaluated configurations, when compared to the single size competitive module and the competitive inception module. This result supports our hypothesis that competition amongst filters of multiple sizes in a ConvNet module represents a way to prevent co-adaptation.

4 Experiments

We quantitatively measure the performance of our proposed models **Competitive Multiple-size Convolution** and **Competitive Inception** on five computer vision benchmark datasets: CIFAR-10[13], CIFAR-100[13], MNIST [12], SVHN [14], and ImageNet ILSVRC 2012 [7]. We first describe the experimental setup, then using CIFAR-10 and MNIST, we show a quantitative analysis (in terms of classification error, number of model parameters and train/test time) of the two proposed models, the Inception Style model presented in Sec. 3, and two additional versions of the proposed models that justify the use of multiple-size filters, explained in Sec. 3.1. Finally, we compare the performance of the proposed Competitive Multiple-size Convolution and Competitive Inception with respect to the current

Multiple-size (Block 1)				
	1x1	3x3	5x5	7x7
1x1	1	0.12	0.15	0.13
3x3	0.12	1	0.21	0.12
5x5	0.15	0.21	1	0.12
7x7	0.13	0.12	0.11	1

Multiple-size (Block 2)				
	1x1	3x3	5x5	7x7
1x1	1	0.05	0.03	0.02
3x3	0.05	1	0.05	0.04
5x5	0.03	0.05	1	0.07
7x7	0.02	0.04	0.07	1

Single-size (Block 1)				
	7x7	7x7	7x7	7x7
7x7	1	0.22	0.22	0.21
7x7	0.22	1	0.19	0.20
7x7	0.22	0.19	1	0.23
7x7	0.21	0.20	0.23	1

Single-size (Block 2)				
	7x7	7x7	7x7	7x7
7x7	1	0.09	0.10	0.09
7x7	0.09	1	0.09	0.09
7x7	0.10	0.09	1	0.09
7x7	0.09	0.09	0.09	1

Table 1: Mean orthogonality measure between pairs of filters within competitive multiple-size convolution modules (up), and filters from competitive single-size convolution modules (down), gathered from the first competitive layer of the first two blocks of the models trained on CIFAR-10.

state-of-the-art results in the four benchmark datasets mentioned above.

The CIFAR-10 [13] dataset contains 60000 images of 10 commonly seen object categories (e.g., animals, vehicles, etc.), where 50000 images are used for training and the rest 10000 for testing, and all 10 categories have equal volume of training and test images. The images of CIFAR-10 consist of 32×32-pixel RGB images, where the objects are well-centered in the middle of the image. The CIFAR-100 [13] dataset extends CIFAR-10 by increasing the number of categories to 100, whereas the total number of images remains the same, so the CIFAR-100 dataset is considered as a harder classification problem than CIFAR-10 since it contains 10 times less images per class and 10 times more categories. The well-known MNIST [12] dataset contains 28×28 grayscale images comprising 10 handwritten digits (from 0 to 9), where the dataset is divided into 60000 images for training and 10000 for testing, but note that the number of images per digit is not uniformly distributed. The Street View House Number (SVHN) [14] is also a digit classification benchmark dataset that contains 600000 32×32 RGB images of printed digits (from 0 to 9) cropped from pictures of house number plates. The cropped images is centered in the digit of interest, but nearby digits and other

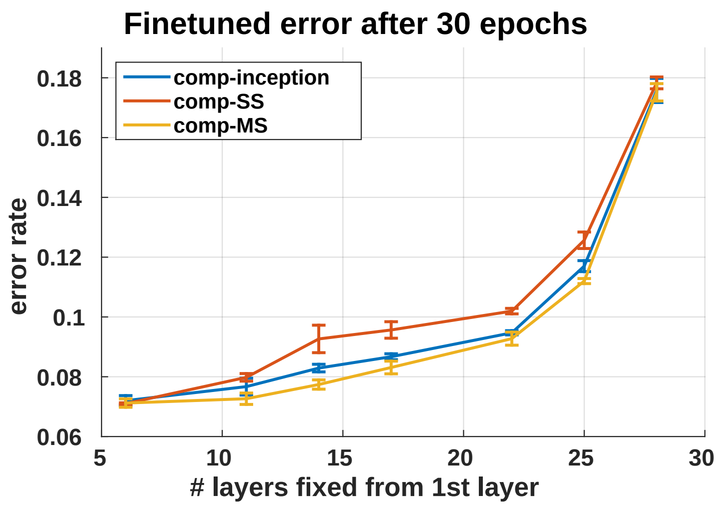


Figure 4: Co-adaptation experiment that tests the transferability of the filters learned for one dataset (CIFAR-10) to another (CIFAR-100) [22] for Competitive inception (**comp-inception**), Single-size (**comp-SS**), and Multiple-size (**comp-MS**) networks. The vertical axis shows the testing error and the horizontal axis represents the index of the first layer of the ConvNet to be re-trained for the new dataset, where the layers below are fixed during the fine-tuning process that lasts 30 epochs.

distractors are kept in the image. SVHN has three sets: training, testing sets and a extra set with 530000 images that are less difficult and can be used for helping with the training process. We do not consider data augmentation on MNIST and SVHN dataset but we show results with and without data augmentation (we only use simple data augmentation, i.e., image translation and mirroring [23, 24]) on CIFAR-10/100. Finally, the ImageNet ILSVRC 2012 [7] dataset contains more than 1 million training images and 50000 validation images of 1000 classes. The performance is evaluated by computing the top1 and top5 (i.e., one of the top 5 predictions is the ground truth label) accuracies on the validation set.

In all these benchmark datasets we minimize the softmax loss function present in the last layer of each model for the respective classification in each dataset, and we report the results as the proportion of misclassified test images. We use an initial learning rate of 0.1 and follow a multiple-step decay to a final learning rate of 0.001 (in 80 epochs for CIFAR-10 and CIFAR-100, 50 epochs for MNIST, and 40 epochs for SVHN). For ImageNet dataset, we decay the final learning rate to 1e-5 in 90 epochs. The stopping criterion is determined by the convergence observed in the error on the validation set. The mini-batch size for CIFAR-10, CIFAR-100, and MNIST datasets is 100, and 128 for SVHN and ImageNet dataset. The momentum and weight decay are set to standard values 0.9 and 0.0005, respectively. For each result reported (except the ImageNet results), we compute the mean and standard deviation of the test error from five separately trained models, where for each model, we use the same training set and parameters (e.g., the learning rate sequence, momentum, etc.), and we change only the random initialization of the filter weights and randomly shuffle the training samples. We also show the best achieved error of the five models as in [25, 23].

We use the GPU-accelerated ConvNet library MatConvNet [26] and Torch-7 [27] based fb.resnet.torch package [28] to perform the experiments specified in this paper. Our experimental environment is a desktop PC equipped with i7-4770 CPU, 24G memory and a 12G GTX TITAN X graphic card. Using this machine, we report the mean training and testing times of our models.

4.1 Model Design Choices

In this section, we show the results from several experiments that show the design choices for our models, where we provide comparisons in terms of their test errors, the number of parameters involved in the training process and the training and testing times. Table 2 shows the results on CIFAR-10 and MNIST for the models **Competitive Multiple-size Convolution**, **Competitive Inception**, and **Inception Style** models, in addition to other models explained below. Note that all models in Table 2 are constrained to have the same numbers of input channels and output channels in each module, and all networks contain three blocks [19], each with three modules (so there is a total of nine modules in each network), as shown in Fig. 3. These models are trained without data augmentation.

We argue that the multiple-size nature of the filters within the competitive module is important to avoid the co-adaptation issue explained in Sec. 3.1. We assess this importance by comparing both the number of parameters and the test error results between the proposed models and the model **Competitive Single-size Convolution**, which has basically the same architecture as the Competitive Multiple-size Convolution model represented in Fig. 3-(a), but with the following changes: the first two blocks contain four sets of 7×7 filters in the first module, and in the second and third modules, two sets of 3×3 filters; and the third block has three filters of size 5×5 in the first module, followed by two modules with two 3×3 filters. Notice that this configuration implies that we replace the multiple-size filters by the filter of the largest size of the module in each node, which is a configuration similar to the recently proposed MIM model [11]. The configuration for the Competitive Single-size Convolution has around two times more parameters than the Competitive Multiple-size Convolution model and takes longer to train, as displayed in Table 2. The idea behind the use of the largest size filters within each module is based on the results obtained from the training of the batch normalisation units of the Competitive Multiple-size Convolution modules, which indicates that the highest weights (represented by γ in (2)) are placed in the largest size filters within each module, as shown in Fig. 5. The classification results of the Competitive Single-size Convolution, shown in Table 2, demonstrate that it is consistently inferior to the Competitive Multiple-size Convolution model.

Another important point that we test in this section is the relevance of dropping connections in a deterministic or stochastic manner when training the competitive convolution modules. In particular, we are interested if the deterministic masking provided by our proposed Competitive Multiple-size Convolution module is more effective at avoiding undesirable filter convergence and filter co-adaptation than the stochastic masking provided by DropConnect [29]. We run a quantitative analysis of the **Competitive DropConnect Single-size Convolution**, where we take the Competitive Single-size Convolution proposed before and randomly drop connections using a rate, which is computed such that it has on average the same number of parameters to learn in each round of training as the Competitive Multiple-size Convolution, but notice that the Competitive DropConnect Single-size Convolution has in fact the same number of parameters as the Competitive Single-size Convolution. Using Fig. 6, we see that the DropConnect rate is 0.57 for the module 1 of blocks 1 and 2 specified in Fig. 3. The results in Table 2 show that it has around two times more parameters, takes longer to train and performs significantly worse than the Competitive Multiple-size Convolution model.

We also notice that the competitive multiple-size module has better generalisation ability amongst the evaluated models, which is shown in Fig. 7, where the competitive multiple-size module has the highest training error in CIFAR-10 and CIFAR-100 (amongst competitive models), but the smallest testing error. Finally, the reported training and testing times in Table 2 show a clear relation between the number of model parameters and those times.

CIFAR-10

Method	No. of Params	Test Error best (mean $\pm$ std dev)	Train Time (h)	Test Time (ms)
Competitive Multiple-size Convolution	4.48 M	6.80 (6.87 $\pm$ 0.05)%	6.4 h	2.7 ms
Competitive Inception	4.69 M	6.67 (7.13 $\pm$ 0.31)%	7.6 h	3.1 ms
Inception Style	0.61 M	8.42 (8.50 $\pm$ 0.06)%	3.9 h	1.5 ms
Competitive Single-size Convolution	9.35 M	6.98 (7.15 $\pm$ 0.12)%	8.0 h	3.2 ms
Competitive DropConnect Single-size Convolution	9.35 M	8.88 (9.12 $\pm$ 0.17)%	7.7 h	3.1 ms

MNIST

Method	No. of Params	Test Error best (mean $\pm$ std dev)	Train Time (h)	Test Time (ms)
Competitive Multiple-size Convolution	1.13 M	0.29 (0.33 $\pm$ 0.04)%	1.5 h	0.8 ms
Competitive Inception	1.19 M	0.38 (0.40 $\pm$ 0.02)%	1.9 h	1.0 ms
Inception Style	0.18 M	0.43 (0.44 $\pm$ 0.01)%	1.4 h	0.7 ms
Competitive Single-size Convolution	2.39 M	0.33 (0.37 $\pm$ 0.03)%	1.7 h	0.9 ms
Competitive DropConnect Single-size Convolution	2.39 M	0.32 (0.35 $\pm$ 0.03)%	1.6 h	0.9 ms

Table 2: Results on CIFAR-10 and MNIST of the proposed models, in addition to the Competitive Single-size Convolution and Competitive DropConnect Single-size Convolution that test our research questions posed in Sec. 3.1.

4.2 Comparison with the State of the Art

We show the performances of the proposed Competitive Multiple-size Convolution (CMSC) and Competitive Inception Convolution models on CIFAR-10, CIFAR-100, MNIST and SVHN, and compare them with the current state of the art in the field. To further explore the usefulness of our module in deeper learning models, we introduce the **CMSC-V2** model which adds one more CMSC layer with filter size up to 5×5 in between the two sequential 3×3 CMSC layers in each block of the original CMSC design (see Fig. 3-a), resulting in a 12-layer model with 7 million parameters.

We now give a brief introduction to the state of the art methods. **Stochastic**

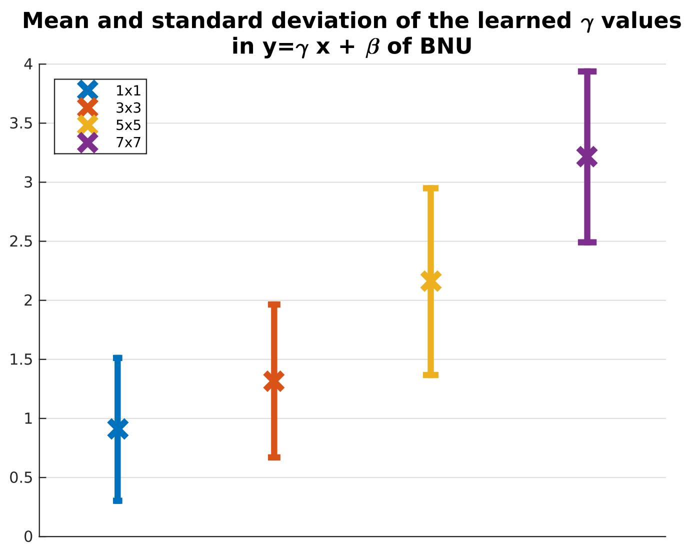
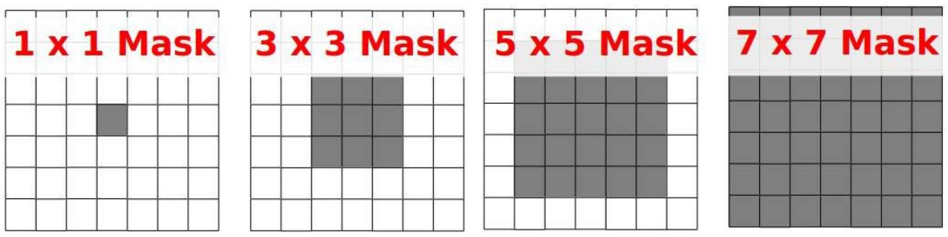
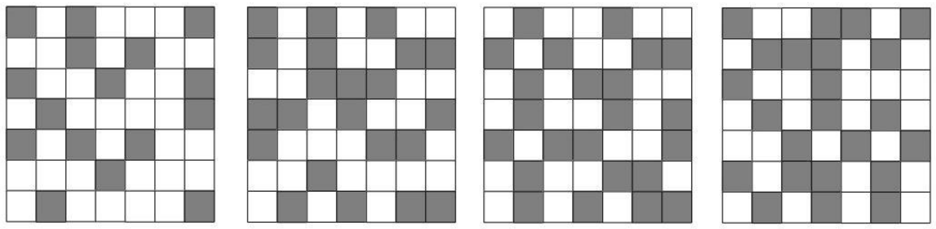


Figure 5: Mean and standard deviation of the learned γ values in the batch normalisation unit of (2) for the Competitive Multiple-size Convolution model on CIFAR-10. This result provides an estimate of the importance placed on each filter by the training procedure.

Pooling [30] proposes a regularization based on a replacement of the deterministic pooling (e.g., max or average pooling) by a stochastic procedure, which randomly selects the activation within each pooling region according to a multinomial distribution, estimated from the activation of the pooling unit. **Spectral Pooling** [31] proposes to perform pooling by truncating the representation in the frequency domain, which preserves considerably more information than other pooling methods and enables flexible choices of pooling output dimensionality. **BinaryConnect** [32] introduces a method to train a DNN with binary weights during the forward and backward propagations, and the Binary Connect method acts as regularizer. **Maxout Networks** [5] introduces a piece-wise linear activation unit that is used together with dropout training [16] and is introduced in Fig. 2-(c). The **Network in Network** (NIN) [19] model consists of the introduction of multilayer perceptrons as activation functions to be placed between convolution layers, and the replacement of a final fully connected layer by average pooling, where the number of output channels represent the final number of classes in the classifica-



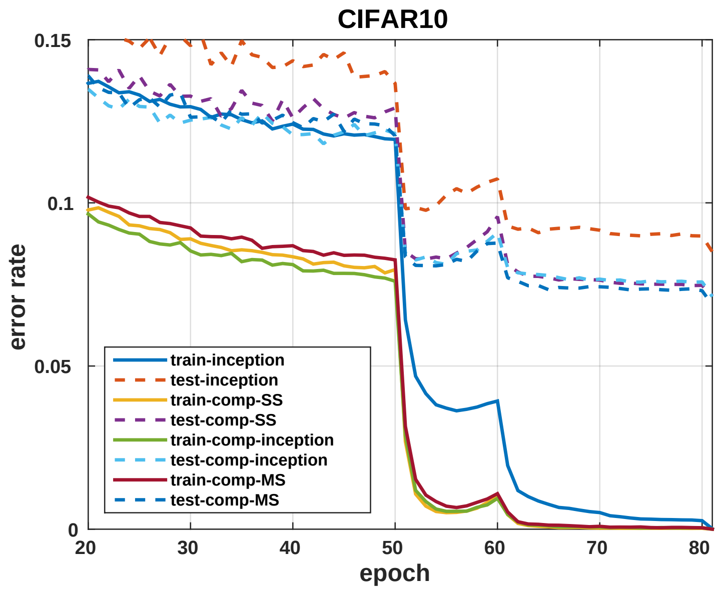
(a) Competitive Multiple-size Convolution Filter Masks



(b) Competitive Single-size DropConnect Convolution Filter Masks

Figure 6: The Competitive Multiple-size Convolution module has filters of size 1×1, 3×3, 5×5, and 7×7, which is equivalent to having four 7×7 filters (with a total of 196 weights) with the masks in (a), where the number of deterministically masked out (or dropped) weights is 112. Using a DropConnect rate of $112/196 \approx 0.57$, a possible set of randomly dropped weights is shown in (b). Note that even though the proportion and number of weights dropped in (a) and (b) are the same, the deterministic or stochastic masking of the weights make a difference in the performance, as explained in the paper.

tion problem. **Deeply-supervised nets** [33] introduce explicit training objectives to all hidden layers, in addition to the back-propagated errors from the last softmax layer. The use of a recurrent structure that replaces the purely feed-forward structure in ConvNets is explored by the model **RCNN** [34]. An extension of the NIN model based on the use of maxout activation function instead of the multilayer perceptron is introduced in the **MIM** model [11], which also shows that the use of batch normalization units are crucial for allowing an effective training of several single-size filters that are joined by maxout units. Clevert et al. [35] introduces the exponential linear unit (**ELU**) which alleviates the vanishing gradient problem and speeds up the learning, leading to better classification performance. **FitNet** [36] learns a student network from the softmax output of a large teacher network or ensemble networks. Residual network (**ResNet**) [23] implements a residual connection to bypass learning blocks, showing competitive results on both CIFAR [13]



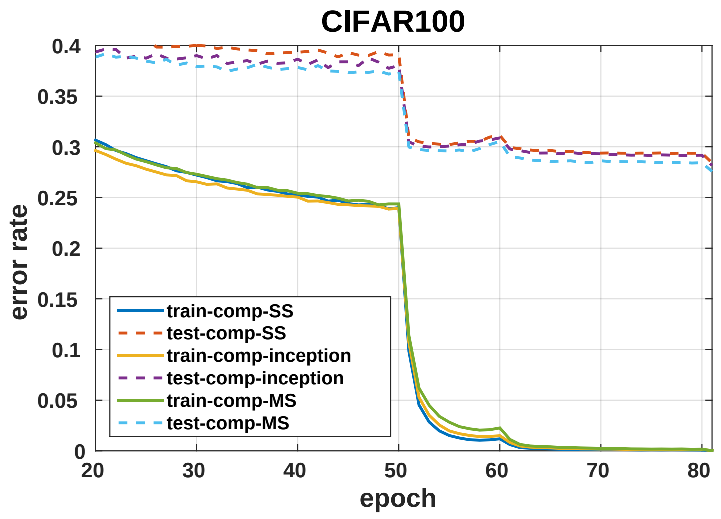


Figure 7: Training and testing errors for CIFAR-10 (up) and CIFAR-100 (down) of the following models: Inception Style (**inception**) , Competitive Single-size Convolution (**comp-SS**), Competitive Inception (**comp-inception**), and Competitive Multiple-size Convolution (**comp-MS**).

Method	CIFAR-10			CIFAR-100	
	No. of Params	noAug	Aug	noAug	Aug
CMSC	4.48 M	6.80 (6.87 ± 0.05)%	6.36 (6.52 ± 0.18)%	26.87 (27.56 ± 0.49)%	25.52 (25.89 ± 0.30)%
CMSC-V2	7.01 M	6.73 (6.80 ± 0.12)%	5.39 (5.61 ± 0.16)%	29.24 (29.63 ± 0.44)%	24.43 (25.37 ± 0.54)%
Competitive Inception	4.69 M	6.67 (7.13 ± 0.31)%	5.98 (6.18 ± 0.21)%	27.73 (28.17 ± 0.25)%	26.30 (26.61 ± 0.38)%
FractalNet-20layers [24]	38.6 M	7.27%	4.68%	29.05%	24.32%
ELU [35]	-	-	6.55%	-	24.28%
FitNet [36]	2.50 M	-	8.39%	-	35.04%
MIM [11]	1.93 M	8.52 ± 0.20%	-	29.20 ± 0.20%	-
Spectral pooling [31]	-	8.60%	-	31.60%	-
RCNN [34]	-	8.69%	7.09%	31.75%	-
Deeply-supervised nets [33]	-	9.69%	7.97%	34.57%	-
Network in Network [19]	0.97 M	10.41%	-	35.68%	-
Maxout Networks [5]	-	11.68%		38.57%	-
ResNets:					
WideResNet [38]	36.5 M	-	4.17%	-	20.50%
ResNet (Stochastic depth) [37]	1.70 M	11.66%	5.25%	37.80%	24.98%
ResNet [23, 37]	1.70 M	13.63%	6.41%	44.74%	27.76%

Table 3: Comparison in terms of classification error between our proposed models (highlighted) and the state-of-the-art methods on CIFAR-10 and CIFAR-100 [13].

datasets and Imagenet [7]. **Stochastic depth** [37] is a method that dynamically drops learning blocks of the ResNet in order to improve the generalisation ability of the ResNet model. **WideResNet** [38] model aims to reduce the depth of the ResNet while increasing the capacity of each residual block. Finally, the **FractalNet** [24] shows a network module with interactive sub-paths, which can be used as an alternative to the residual module to compose a very deep network.

The comparison on CIFAR-10 and CIFAR-100 [13] datasets are shown in Table 3, where we separate the results by whether they are obtained with or without data augmentation. Our proposed methods has the best result among the state-of-the-art methods without data augmentation and show competitive results to the state-of-the-art methods that using data augmentation. Table 4 shows the results on MNIST [12], where it is worth reporting that the best result (over the five trained models) produced by our CMSC model is a test error of 0.29%, which is better than the single result from Liang and Hu [34]. Finally, the comparison on SVHN[14] dataset is shown in Table 5, where our CMSC model is able to achieve the best error of 1.69%.

4.3 ImageNet Evaluation

ImageNet [7] is a large-scale image dataset that is use to benchmark several methods proposed in the field. Naively, our Competitive Multiple-size module includes the use of large size filters, which produce a large number of parameters that do not scale well with the input dimensionality (i.e., the number of channels of input ten-

Method	Test Error
CMSC	0.29 (0.33 ± 0.04)%
RCNN [34]	0.31%
MIM [11]	0.35 ± 0.03%
Deeply-supervised nets [33]	0.39%
Competitive Inception	0.38 (0.40 ± 0.02)%
Network in Network [19]	0.45%
Conv. Maxout+Dropout [5]	0.47%
Stochastic Pooling [30]	0.47%

Table 4: Comparison in terms of classification error between our proposed models (highlighted) and the state-of-the-art methods on MNIST [12].

Method	Test Error
ResNet (Stochastic depth) [37]	1.75%
CMSC	1.69 (1.76 ± 0.07)%
RCNN [34]	1.77%
Competitive Inception	1.70 (1.78 ± 0.09)%
ResNet [23, 37]	1.80%
FractalNet-20layers [24]	1.87%
Deeply-supervised nets [33]	1.92%
Drop-connect [29]	1.94%
MIM [11]	1.97 ± 0.08%
BinaryConnect [32]	2.15%
Network in Network [19]	2.35%
Conv. Maxout+Dropout [5]	2.47%
Stochastic Pooling [30]	2.80%

Table 5: Comparison in terms of classification error between our proposed models (highlighted) and the state-of-the-art methods on SVHN [14].

sor). This means that implementing a similar ConvNet structure to GoogLeNet [6] in terms of the number of layers and number of units per layer would require more than 70M parameters, as opposed to 7M parameters needed by GoogLeNet. To resolve this issue, we adopt the "bottleneck" design from the ResNet [23] model to reduce the dimensionality of the input tensor to keep Competitive Multiple-size Convolution (CMSC) unit computation manageable. Our proposal is to replace the second 3×3 ReLU activated convolution layer [4] with our CMSC module inside the bottleneck module (See Fig. 8) to compose the bottleneck-CMSC module. We replace all the bottleneck units in the original ResNet model with

Method	No. of Params	Top1	Top5
CMSC-ResNet-V1-5x5 (r=0.6)	23.48 M	26.22%	8.06%
CMSC-ResNet-V1-7x7 (r=0.4)	22.59 M	26.76%	8.46%
CMSC-ResNet-V2-5x5 (r=0.9)	22.23 M	25.50%	7.66%
CMSC-ResNet-V2-7x7 (r=0.9)	22.71 M	25.48%	7.59%
Baseline ResNet (r=1.0)[1]	22.78 M	26.63%	8.34%

Table 6: Comparison between the CMSC-ResNet and the baseline ResNet on ILSVRC 2012 validation set, showing the top1 and top5 validation errors by evaluating only the center crop (224x224) of the validation images. All models have 50 layers.

the bottleneck-CMSC units and name it CMSC-ResNet-V1 model. Since a single CMSC module contains more parameters than a 3×3 convolution module, we use $r < 1$ (uniformly used in all modules) to indicate the reduction rate of the number of channels in the input tensor of the CMSC unit (shown at bottom of Fig. 8).

In Table 6 we show that our CMSC-ResNet-V1-5x5 model outperforms the ResNet in a well-controlled experiment setup (i.e., same number of epochs, learning rate schedule, batch size, etc.). Note that the performance of the ResNet model surpasses the performance of GoogLeNet, so that is why we compare our method with ResNet. In the CMSC-ResNet-V1 models, the $k \times k$ suffix means the maximum filter size (i.e., -5×5 means the CMSC module uses 1×1, 3×3, and 5×5 filters) used in each bottleneck-CMSC module. The CMSC-ResNet-V1-5x5 model shows better result than the baseline ResNet model, which indicates the advantage of using the Competitive Multiple-size module in very deep networks. However, we also notice that the CMSC-ResNet-V1-7x7 does not improve the baseline performance of ResNet model, where our hypothesis is that this issue is due to the 60% reduction of activating units in that model. To verify this hypothesis, we increase the number of units (while keeping a similar number of parameters) and introduce the CMSC-ResNet-V2 models, where the large size bottleneck-CMSC units are only used at the bottom of the model and small bottleneck-CMSC units (CMSC with 1×1 and 3×3 filters) at the top of the model, resulting in further improvements. We show that the best model (CMSC-ResNet-V2-7x7) outperforms the baseline ResNet model by 1.15%.

[1]Our ResNet baseline model is trained (from scratch) from the default script provided in [28] with a few modifications to suit our computational environment: 1) we use mini-batches of size 128; and 2) we downsize the images to 256x256 and store them in an LMDB database, which speeds up the training but at the cost of reading the same training image sequence in each epoch. In comparison, the official script [28] uses mini-batches of size 256, full size images and randomised

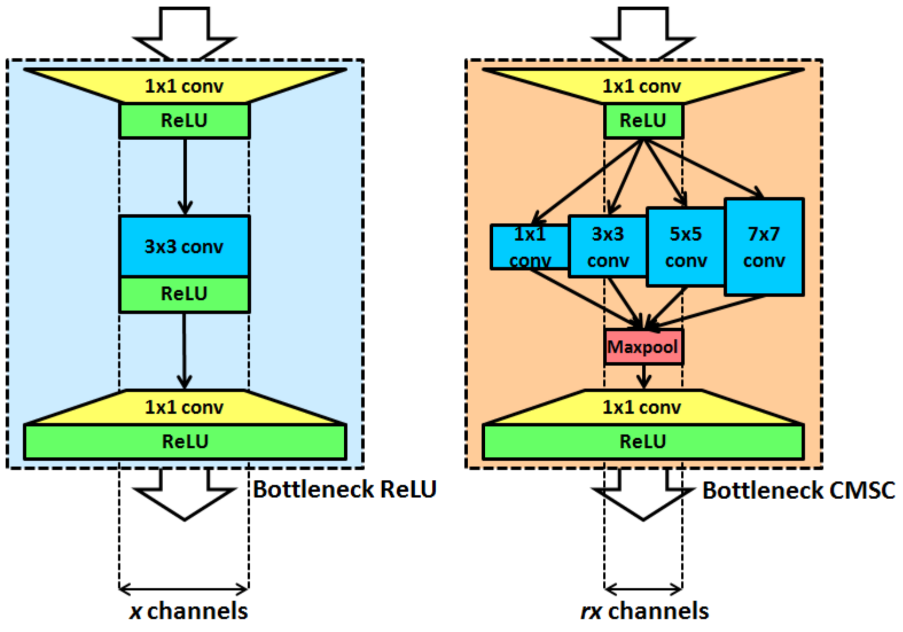


Figure 8: The comparison of the original "bottleneck" ReLU architecture (left) used in ResNet [23] model and our proposed "bottleneck" Competitive Multiple-size Convolution (CMSC) architecture. The width of each computation block illustrates the number of channels of each block, where the first computation block is a 1×1 convolution that is used to reduce the number of input channels for the second computation block. The hyper-parameter r(where $r < 1$) is used to indicate the reduction rate of the number of input channels (as well as the number of CMSC units), compared to the number of input channels/units in the baseline ResNet model. The third computation block is also a 1×1 convolution which restore the number of channels/units.

5 Discussion

In terms of the model design choices in Sec. 4.1, we can see that the proposed Competitive Multiple-size Convolution produces more accurate classification results than the proposed Competitive Inception. Given that the main difference between these two models is the presence of the max-pooling path within each module, we can conclude that this path does not help with the classification ac-

image reads as default.

curacy of the model. The better performance of both models with respect to the Inception Style model can be attributed to the maxout unit that induces competition amongst the underlying filters, which helps more the classification results when compared with the collaborative nature of the Inception module. Also, a comparison with the Competitive Single-size Convolution in Tab. 2 replicates the results from Sec. 3.1, which shows that the proposed Competitive Multiple-size Convolution produces more accurate classification results on the test set. Considering model complexity, it is important to notice that the relation between the number of parameters and training and testing times is not linear, where even though the Inception Style model has $10\times$ fewer parameters, it trains and tests 2 to $1.5\times$ faster than the proposed Competitive Multiple-size Convolution and Competitive Inception models.

The use of deterministic, as opposed to stochastic, mapping also appears to be more effective in avoiding filter co-adaptation given the more accurate classification results of the former mapping. Nevertheless, the reason behind the worse performance of the stochastic mapping may be due to the fact that DropConnect has been designed for the fully connected layers only [29], while our test bed for the comparison is set in the convolutional filters. To be more specific, we think that a fully connected layer usually encapsulates hundreds to thousands of weights for inputs of similar size of dimensions, thus a random dropping on a subset of weight elements can hardly change the distribution of the outputs pattern. However, the convolution filters are of small dimensions, and each of our maxout unit controls 4 to 5 filters at most, so such masking scheme over small weights matrix could result in "catastrophic forgetting" [39] which explains why the Competitive DropConnect Single-size Convolution performs even worse than Competitive Single-size Convolution on CIFAR-10.

The use of 7×7 filters in the competitive module allows us to capture large size spatial patterns in each computation layer. It may be argued that such large size spatial patterns may not be necessary to obtain good classification performance. To explore this point, we test the exclusion of the 7×7 filters from the proposed Competitive Multi-size Convolution model (see Fig. 3-a), resulting in a model of 3.68 M parameters, where the classification result is $7.21 \pm 0.11\%$, which is worse than the one with the 7×7 filters ($6.87 \pm 0.05\%$). We further show an experiment that assesses whether filters of larger size within a competitive module can improve the classification accuracy at the expense of having a larger number of parameters to train. We test the inclusion of two more filters of sizes 9×9 and 11×11 in module 1 of blocks 1 and 2, and two more filter sizes 7×7 and 9×9 in module 1 of block 3 (see Fig. 3-a). The classification result obtained is $7.36 \pm 0.16\%$

on CIFAR-10, and number of model parameters is 13.11 M. These experiments show that decreasing and increasing the number of filters of larger sizes can have different effects in the classification results, where we may conclude that some spatial pattern at 7×7 size is helpful to classify the CIFAR-10 classes. On the other hand, the CMSC-ResNet-V2-7x7 model shows slightly better performance to CMSC-ResNet-V2-5x5 model on ImageNet, which suggests that the 5×5 spatial patterns may be as expressive as the 7×7 patterns for ImageNet. To conclude, the filter sizes to be included in the competitive unit is a task dependent design choice, where the large size filters can be omitted if the performance gain does not justify the increase in computational costs.

An important modification that can be suggested for our proposed Competitive Multiple-size Convolution model is the replacement of the maxout by ReLU activation, where only the largest size filter of each module is kept and all other filters are removed. One can argue that such model is perhaps less complex (in terms of the number of parameters) and probably as accurate as the proposed model. However, the results we obtained with such model on CIFAR-10 show that this model has 3.28 M parameters (i.e., just slightly less complex than the proposed models, as shown in Table 2) and has a classification test error of $8.16 \pm 0.15\%$, which is significantly larger than for our proposed models. On MNIST, this model has 0.81 M parameters and produces a classification error of $0.37 \pm 0.05\%$, which also shows no advantage over the proposed models.

6 Conclusion

In this paper, we show the effectiveness of using competitive units on modules that contain multiple-size filters. We argue that the main reason of the superior classification results of our proposal, compared with the current state of the art in several benchmark datasets, lies in the following points: 1) the deterministic masking implicitly used by the multiple-size filters avoids the issue of filter co-adaptation; 2) the competitive unit that joins the underlying filters and the batch normalization units promote the formation of a large number of sub-networks that are specialized in the classification problem restricted to a small area of the input space and that are regularized by the fact that they are trained together within the same model; and 3) the maxout unit allows the reduction of the number of parameters in the model. It is important to note that such modules can be applied in several types of deep learning networks, and we plan to apply it to other types of models, such as the recurrent neural network [34].

Acknowledgement

This research was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016).

References

[1] R. K. Srivastava, J. Masci, F. Gomez, J. Schmidhuber, Understanding locally competitive networks, in: International Conference on Learning Representations (ICLR), 2015, pp. 1–11.

[2] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton, Adaptive mixtures of local experts, in: Neural computation, Vol. 3, MIT Press, 1991, pp. 79–87.

[3] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, J. Schmidhuber, Compete to compute, in: Advances in Neural Information Processing Systems (NIPS), 2013, pp. 2310–2318.

[4] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Vol. 15, 2011, pp. 315–323.

[5] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, Y. Bengio, Maxout networks., in: Proceedings of the International Conference on Machine Learning (ICML), Vol. 28, 2013, pp. 1319–1327.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, in: International Journal of Computer Vision, Springer, 2014, pp. 1–42.

[8] Y. Gong, L. Wang, R. Guo, S. Lazebnik, Multi-scale orderless pooling of deep convolutional activation features, in: Proceedings of the European Conference on Computer Vision (ECCV), Springer, 2014, pp. 392–407.

[9] S. Zagoruyko, N. Komodakis, Learning to compare image patches via convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 4353–4361.

[10] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the International Conference on Machine Learning (ICML), Vol. 37, 2015, pp. 448–456.

[11] Z. Liao, G. Carneiro, On the importance of normalisation layers in deep learning with piecewise linear activation units, in: IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2016, pp. 1–8.

[12] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[13] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Tech. rep. (2009).

[14] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS workshop on Deep Learning and Unsupervised Feature Learning, 2011, p. 5.

[15] G. F. Montufar, R. Pascanu, K. Cho, Y. Bengio, On the number of linear regions of deep neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2014, pp. 2924–2932.

[16] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., Journal of Machine Learning Research (JMLR) 15 (1) (2014) 1929–1958.

[17] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, V. Shet, Multi-digit number recognition from street view imagery using deep convolutional neural networks, in: International Conference on Learning Representations (ICLR), 2014, pp. 1–13.

[18] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations (ICLR), 2015, pp. 1–14.

[19] M. Lin, Q. Chen, S. Yan, Network in network, in: International Conference on Learning Representations (ICLR), 2014, pp. 1–10.

[20] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, T. Poggio, Robust object recognition with cortex-like mechanisms, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (3).

[21] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of International Conference on Machine Learning, 2010, pp. 807–814.

[22] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, in: Advances in Neural Information Processing Systems (NIPS), 2014, pp. 3320–3328.

[23] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[24] G. Larsson, M. Maire, G. Shakhnarovich, Fractalnet: Ultra-deep neural networks without residuals, in: International Conference on Learning Representations (ICLR), 2016, pp. 1–11.

[25] R. K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in neural information processing systems (NIPS), 2015, pp. 2377–2385.

[26] A. Vedaldi, K. Lenc, Matconvnet: Convolutional neural networks for matlab, in: Proceedings of the 23rd ACM international conference on Multimedia, ACM, 2015, pp. 689–692.

[27] R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: A matlab-like environment for machine learning, in: BigLearn, NIPS Workshop, EPFL-CONF-192376, 2011.

[28] Resnet training in torch, ResNet training in Torch [Online], accessed: 2017-04-25.

[29] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, R. Fergus, Regularization of neural networks using dropconnect, in: Proceedings of the International Conference on Machine Learning (ICML), Vol. 28, 2013, pp. 1058–1066.

[30] M. D. Zeiler, R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, in: International Conference on Learning Representations (ICLR), 2013, pp. 1–9.

[31] O. Rippel, J. Snoek, R. P. Adams, Spectral representations for convolutional neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2015, pp. 2449–2457.

[32] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in Neural Information Processing Systems (NIPS), 2015, pp. 3123–3131.

[33] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, Deeply-supervised nets, in: Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), 2015, pp. 562–570.

[34] M. Liang, X. Hu, Recurrent convolutional neural network for object recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3367–3375.

[35] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), in: International Conference on Learning Representations (ICLR), 2016, pp. 1–14.

[36] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, in: International Conference on Learning Representations (ICLR), 2015, pp. 1–13.

[37] G. Huang, Y. Sun, Z. Liu, D. Sedra, K. Q. Weinberger, Deep networks with stochastic depth, in: Proceedings of the European Conference on Computer Vision (ECCV), Springer, 2016, pp. 646–661.

[38] S. Zagoruyko, N. Komodakis, Wide residual networks, in: British Machine Vision Conference (BMVC), 2016, pp. 1–15.

[39] M. McCloskey, N. J. Cohen, Catastrophic interference in connectionist networks: The sequential learning problem, in: The psychology of learning and motivation, Vol. 24, 1989, p. 92.

Chapter 7

Approximate Fisher Information Matrix to Characterise the Training of Deep Residual Networks

Zhibin Liao[*] **Gustavo Carneiro**[*] **Tom Drummond**[†] **Ian Reid**[*]
ARC Centre of Excellence for Robotic Vision, Australia
[*]University of Adelaide [†]Monash University

The work contained in this chapter is unpublished.

Zhibin Liao, Gustavo Carneiro, Tom Drummond, and Ian Reid. Approximate Fisher Information Matrix to Characterise the Training of Deep Residual Networks. Unpublished manuscript.

Statement of Authorship

Title of Paper	Approximate Fisher Information Matrix to Characterise the Training of Deep Residual Networks
Publication Status	☐ Published ☐ Accepted for Publication ☐ Submitted for Publication [✓ Unpublished and Unsubmitted work written in manuscript style]
Publication Details	Zhibin Liao, Gustavo Carneiro, Tom Drummond, and Ian Reid. "Approximate fisher information matrix to characterise the training of deep residual networks".

Principal Author

Name of Principal Author (Candidate)	Zhibin Liao		
Contribution to the Paper	- Wrote all the coding and did all the experiments - Built the conceptual idea - Wrote and refined the manuscript		
Overall percentage (%)	50%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	19th May 2017

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Gustavo Carneiro		
Contribution to the Paper	- Built the conceptual idea - Wrote and refined the manuscript - Supervised the development of this work		
Signature		Date	22-05-17

Name of Co-Author	Tom Drummond		
Contribution to the Paper	- Helped in the development of the conceptual idea		
Signature		Date	22/05/2017

Name of Co-Author	Ian Reid		
Contribution to the Paper	- Helped in the development of the conceptual idea		
Signature		Date	22/5/17

Approximate Fisher Information Matrix to Characterise the Training of Deep Residual Networks

Zhibin Liao[*] Gustavo Carneiro[*] Tom Drummond[†] Ian Reid[*]
ARC Centre of Excellence for Robotic Vision, Australia
[*]University of Adelaide [†]Monash University
{zhibin.liao, gustavo.carneiro, ian.reid}@adelaide.edu.au, tom.drummond@monash.edu

Abstract

In this paper, we introduce a novel methodology for characterising the performance of deep residual networks (ResNets) with respect to training convergence and generalisation as a function of mini-batch size, model structure, learning rate and stochastic depth rate. This methodology is based on novel measurements derived from the eigenvalues of the approximate Fisher information matrix, which can be efficiently computed even for high capacity ResNet models and can be used to manipulate the parameters of the training process to obtain faster training and improved inference performance. Our proposed measurements are shown to enable a reliable estimation of training convergence and generalisation using just a few training epochs. Furthermore, the proposed measurements also allow us to show that it is possible to manipulate the training process with a new dynamic sampling training approach that continuously increase the mini-batch size over the training process and a novel dynamic stochastic depth rate mechanism that continuously decrease the depth drop rate. We show that this proposed training approach reaches competitive classification results in CIFAR-10, and CIFAR-100 datasets with models of significantly lower capacity that can be trained more efficiently than current state of the art ResNets.

1. Introduction

Deep learning models, especially the recently proposed deep residual networks (ResNets) [12, 13], are achieving extremely accurate classification performance over a broad range of tasks. ResNets (and most of large capacity deep learning models) are generally trained with the stochastic gradient descent (SGD) methods [4], or any of its variants, given that they have shown to produce robust training results (i.e., good convergence and generalisation) at a relatively low computational cost, in terms of run-time and memory complexity. However, a successful SGD training of ResNets depends on a careful selection of mini-batch size, model structure, learning rate and stochastic depth rate, but there are currently no reliable guidelines on how to select these hyper-parameters.

Recently, Keskar et al. [15] proposed numerical experiments to show that large mini-batch size methods converge to sharp minimisers of the objective function, leading to poor generalisation, and small mini-batch size approaches converge to flat minimisers. In particular, Keskar et al. [15] proposed a new sensitivity measurement based on an exploration approach that calculates the largest value of the objective function within a small neighbourhood. Even though very relevant to our work, that paper [15] focuses only on mini-batch size, and it does not elaborate on the dynamic sampling training method – the paper only shows the rough idea of a training algorithm that starts with a small mini-batch and then suddenly switches to a large mini-batch. Other recent works characterise the loss function in terms of their local minima [7, 20, 27, 19], which is interesting but does not provide a helpful guideline for characterising the training procedure. Finally, the distribution of the Hessian eigenvalues has been studied [25] in order to assess the complexity of the training problem and whether the system is over-parameterised, which is interesting but un-feasible to be computed in modern ResNets due to the high computational complexity of processing the Hessian.

In this paper, we introduce a novel methodology for characterising the SGD training of ResNets [12] with respect to mini-batch sizes, model structure, learning rate and stochastic depth rate [13]. These experiments are based on the efficient computation of the eigenvalues of the approximate Fisher information matrix (hereafter, referred to as Fisher matrix) [6, 22]. In general, the eigenvalues of the Fisher matrix can be efficiently computed (in terms of memory and run-time complexities), and they are usually assumed to approximate of the Hessian spectrum [6, 22], which in turn can be used to estimate the objective function shape.

The proposed characterisation of SGD training is based on the cumulative sum of the condition number C_K and the cumulative sum of the (weighted) eigenvalues L_K of the Fisher matrix. We show empirically that C_K and L_K enable a consistent characterisation of various models trained with different mini-batch sizes, model structure, learning

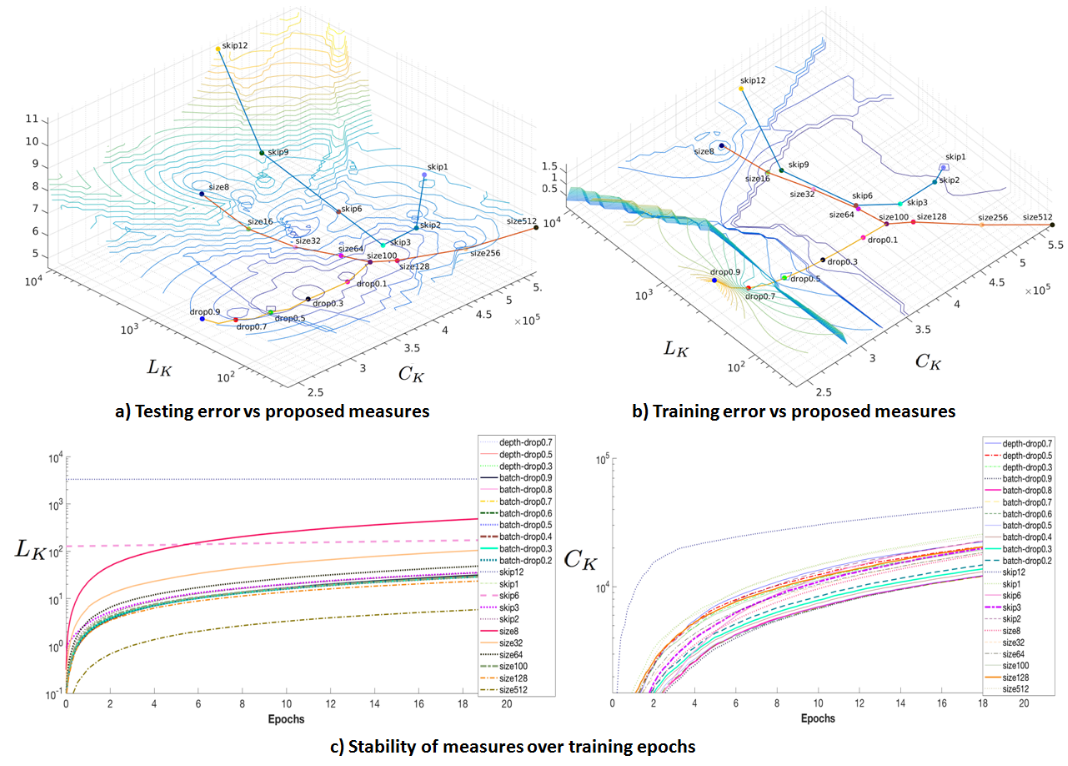


Figure 1. Testing (a) and training (b) errors on CIFAR-10 as a function of the proposed measures: cumulative sum of the condition number C_K and of the summed (weighted) eigenvalues L_K of the Fisher matrix at the final training epoch. The labels represent the training method used: 1) size$\{8, ..., 512\}$ denotes the mini-batch size for training the standard ResNet, 2) batch_drop$\{0.1, ..., 0.9\}$ represents the stochastic depth rate with a fixed mini-batch size of 100, and 3) skip$\{1, ..., 12\}$ indicates different ResNet structures with residual connections involving a large or small number of blocks. The graph in (c) shows the stability of C_K and L_K as a function of training epochs.

rate and stochastic depth rate. Fig. 1(a),(b) shows how the C_K and L_K values computed at the last epoch for the training and testing sets of CIFAR-10 [18] vary for different types of ResNet models (notice the **skip**$\{\mathbf{1}, ..., \mathbf{12}\}$), different mini-batch sizes (see **size**$\{\mathbf{8}, ..., \mathbf{512}\}$) and different stochastic drop rates (**batch-drop**$\{\mathbf{0.1}, ..., \mathbf{0.9}\}$). In addition, these two measures are also shown in Fig. 1(c) to be stable when computed during the first epochs, allowing the training mechanism to be reliably characterised early on in the training process, relative to other models, saving precious training cycles. These measures also suggest new training procedures that dynamically increases the mini-batch size or decreases the stochastic depth rate, allowing the training procedure to navigate in this landscape of C_K and L_K measures. The dynamic increase of mini-batch size approach has been suggested before [15, 4], but we are not aware of previous implementations. This approach has a faster training time and a competitive accuracy result compared to the current state of the art. All our experiments are conducted on CIFAR-10, and CIFAR-100 [18], and our proposed training procedure obtains competitive classification errors using ResNet models with smaller memory requirements, when compared to the current state of the art.

2. Literature review

In this section, we first discuss stochastic gradient descent (SGD) [4], inexact Newton and quasi-Newton methods [10, 4, 5], as well as (generalized) Gauss-Newton methods [3, 26] the natural gradient method [2], and scaled gradient iterations such as RMSprop [28] and AdaGrad [9]. Then we discuss other approaches that rely on numerical experiments to measure key aspects of SGD training [15, 7, 20, 27, 19, 25].

SGD training [4] is a common iterative optimisation method that has wide usage in deep neural networks training. One of the main goals of SGD is to find a good balance between stochastic and batch approaches to provide a favourable trade-off with respect to per-iteration costs and expected per-iteration improvement in minimising the objective function. The popularity of SGD in deep learning lies in the tolerable computation cost with acceptable convergence speed. Second-order methods aim to improve the convergence speed of SGD by re-scaling the gradient vector in order to compensate for the high non-linearity and ill-conditioning of the objective function. In particular, Newton's method uses the inverse of the Hessian matrix for re-scaling the gradient vector. This operation has complexity $O(N^3)$ (where N is the number of model parameters, which is usually between $O(10^6)$ and $O(10^7)$ for modern deep learning models), which makes it infeasible. Besides, the Hessian must be positive definite for Newton's method to work, which is not a good assumption for the training of deep learning models.

In order to avoid the computational cost above, several approximate second-order methods have been developed. For example, the Hessian-free conjugate gradient (CG) [30] is based on the fact it only needs to compute Hessian-vector products, which can be efficiently calculated with the $\mathcal{R}$-operator [24] at a comparable cost to a gradient evaluation. This Hessian-free method has been successfully applied to train neural networks [21, 17]. Quasi-Newton methods (e.g., the BFGS [10, 4]) take an alternative route and approximate the inversion of Hessian with only the parameter and gradient displacements in the past gradient iterations. However, the explicit use of the approximation matrix is also infeasible in large optimisation problems, where the L-BFGS [5] method is proposed to reduce the memory usage. The (Generalized) Gauss-Newton method [3, 26] approximates Hessian with the Gauss-Newton matrix. Another approximate second-order method is the natural gradient method [2] that uses the inverse of the Fisher matrix to make the search quicker in the parameters that have less effect on the decision function [4]. Without estimating the second-order curvature, some methods can avoid saddle points and perhaps have some degree of resistance to near-singular curvature [4]. For instance, AdaGrad [9] keeps an accumulation of the square of the gradients of past iterations to re-scale each element of the gradient, so that parameters that have been infrequently updated are allowed to have large updates, and frequently updated parameters can only have small updates. Similarly, RMSProp [28] normalises the gradient by the magnitude of recent gradients. Furthermore, Adadelta [32] and Adam [16] improve over AdaGrad [9] by taking more careful gradient re-scaling schemes.

Given the issues involved in the development of (approximate) second-order methods, there has been some interest in the implementation of approaches that could characterise the functionality of SGD optimisation. Choromanska et. al [7] use the spin-glass model to evaluate fully-connected networks and suggest that large size networks contain many local minima that are equivalent in terms of test performance. An extension of the use of the spin-glass model to evaluate residual networks can be found in [20]. Furthermore, Lee et al. [19] show that SGD converges to a local minimiser rather than a saddle point (with models that are randomly initialised). Soudry and Carmon [27] provide theoretical guarantees that local minima in multilayer neural networks loss functions have zero training error. In addition, the exact Hessian of the neural network has been found to be singular, suggesting that methods that assume non-singular Hessian are not to be used without proper modification [25]. Goodfellow et. al. [11] found the state-of-the-art neural networks do not encounter significant obstacles (local minima, saddle points, etc.) during the training. In [15], a new sensitivity measurement of energy landscape is used to provide empirical evidence to support the argument that training with large mini-batch size converges to sharp minima, which in turn leads to poor generalisation. In contrast, small mini-batch size converges to flat minima, but performance degenerates due to noise in the gradient estimation. Our paper can be regarded as a new approach to characterise SGD optimisation, where our main contributions are: 1) new efficiently computed measures derived from the Fisher matrix that can be used to explain the training convergence and generalisation of ResNets, and 2) new dynamic sampling and dynamic stochastic depth rate training algorithms.

3. Methodology

In this section, we assume the availability of a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{|\mathcal{D}|}$, where the i^{th} image $\mathbf{x}_i : \Omega \rightarrow \mathbb{R}$ (Ω denotes image lattice) is annotated with the label $y_i \in \{1, ..., C\}$, with C denoting the number of classes. This dataset is divided into two mutually exclusive training set $\mathcal{T} \in \mathcal{D}$ and testing set $\mathcal{S} \in \mathcal{D}$.

The ResNet model [12] and its stochastic depth extension [13] are defined by a concatenation of residual blocks, with each block defined by:

$$r_l(\mathbf{v}_l) = b_l \times f(\mathbf{v}_l, \mathbf{W}_l) + \mathbf{v}_l, \tag{1}$$

where $l \in \{1, ..., L\}$ indexes the residual blocks, $\mathbf{W}_l$ denotes the parameters for the l^{th} block, $\mathbf{v}_l$ is the input, with the image input of the model being represented by $\mathbf{v}_1 = \mathbf{x}$, $f(\mathbf{v}_l, \mathbf{W}_l)$ represents a residual unit containing a sequence of linear and non-linear transforms [23], and batch normalisation [14], and $b_l \in \{0, 1\}$ denotes a Bernoulli random variable indicating if the l^{th} block is active ($b_l = 1$) or inactive ($b_l = 0$) [13] (note that in the original ResNet model [12], $b_l = 1$ for all blocks). The full model is then defined by:

$$f(\mathbf{x}, \theta) = f_{out} \circ r_L \circ ... \circ r_1(\mathbf{x}), \tag{2}$$

where $\circ$ represents the composition operator, $\theta \in \mathbb{R}^P$ denotes all model parameters $\{\mathbf{W}_1, ... \mathbf{W}_L\} \bigcup \mathbf{W}_{out}$, and $f_{out}(.)$ is a linear transform parameterised by weights $\mathbf{W}_{out}$ with a softmax activation function that outputs a value in $[0, 1]^C$ indicating the probability of selecting each of the C classes. The training of the model in (2) minimises the multi-class cross entropy loss $\ell(.)$ on the training set $\mathcal{T}$, as follows:

$$\theta^* = \arg\min_\theta \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \ell\left(y_i, f(\mathbf{x}_i, \theta)\right). \tag{3}$$

The SGD training minimises the loss in (3) by iteratively taking the following step:

$$\theta_{k+1} = \theta_k - \frac{\alpha_k}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla \ell(y_i, f(\mathbf{x}_i, \theta_k)), \qquad (4)$$

where $\mathcal{B}_k$ is the mini-batch for the k^{th} epoch of the minimisation process. As noted by Keskar et al. [15], the shape of the loss function can be characterised by the spectrum of the $\nabla^2 \ell(y_i, f(\mathbf{x}_i, \theta_k))$, where a significant number of large positive eigenvalues tend to make the training process generalise less well, and numerous small eigenvalues are likely to lead to better generalisation. Given that the computation of the spectrum of $\nabla^2 \ell(y_i, f(\mathbf{x}_i, \theta_k))$ is infeasible, we must resort to the use of methods that can reliably approximate such spectrum, such as the Fisher matrix [6, 22], defined by

$$\mathbf{F}_k = \left(\nabla \ell(y_{i \in \mathcal{B}_k}, f(\mathbf{x}_{i \in \mathcal{B}_k}, \theta_k)) \nabla \ell(y_{i \in \mathcal{B}_k}, f(\mathbf{x}_{i \in \mathcal{B}_k}, \theta_k))^\top \right), \qquad (5)$$

where $\mathbf{F}_k \in \mathbb{R}^{P \times P}$.

The calculation of $\mathbf{F}_k$ in (5) depends on the Jacobian $\mathbf{J}_k = \nabla \ell(y_{i \in \mathcal{B}_k}, f(\mathbf{x}_{i \in \mathcal{B}_k}, \theta_k))$, with $\mathbf{J}_k \in \mathbb{R}^{P \times |\mathcal{B}_k|}$. Given that $\mathbf{F}_k = \mathbf{J}_j \mathbf{J}_j^\top \in \mathbb{R}^{P \times P}$ scales with $P = O(10^6)$ and that we are only interested in the spectrum of $\mathbf{F}_k$, we compute instead $\widetilde{\mathbf{F}}_k = \mathbf{J}_j^\top \mathbf{J}_j \in \mathbb{R}^{|\mathcal{B}_k| \times |\mathcal{B}_k|}$ that scales with the mini-batch size $|\mathcal{B}_k| = O(10^2)$. Note that from $\widetilde{\mathbf{F}}_k$ we can compute the largest $|\mathcal{B}_k|$ non-zero eigenvalues of $\mathbf{F}_k$ by using the Cholesky decomposition [29], represented by the set $\mathcal{E}_k$.

The **first measure** we propose is the **cumulative sum of the condition number of $\widetilde{\mathbf{F}}_k$** , defined by

$$C_K = \sum_{k=1}^{K} c_k, \qquad (6)$$

where K denotes the epoch number, and $c_k = \left(\frac{\max(\mathcal{E}_k)}{\min(\mathcal{E}_k)} \right)^{\frac{1}{2}}$ represents the condition number of $\widetilde{\mathbf{F}}_k$. This measure is used to describe the ill-conditioning of the gradient updates accumulated during the training process.

The **second measure** is the **cumulative sum of the square root of the trace of $\widetilde{\mathbf{F}}_k$, weighted by** $\alpha_k^2 / |\mathcal{B}_k|^2$:

$$L_K = \sum_{k \in K} \left(\frac{\alpha_k^2}{|\mathcal{B}_k|^2} \operatorname{Tr}\left(\widetilde{\mathbf{F}}_k \right) \right)^{\frac{1}{2}}, \qquad (7)$$

where $\operatorname{Tr}(.)$ defines the trace operator, and $\operatorname{Tr}\left(\widetilde{\mathbf{F}}_k\right)$ approximates the Laplacian, defined by $\operatorname{Tr}\left(\nabla^2 \ell(y_i, f(\mathbf{x}_i, \theta_k))\right)$ that sums the eigenvalues of the Hessian. The sum of such eigenvalues is generally associated with the steepness of the energy function landscape. Note that the weighting in (7) is reasonable because it is the same factor used in the SGD update rule (4), and it proved to be useful for measuring training convergence and generalisation, as shown later in this paper.

Different ResNet models, learning rates, mini-batch sizes and stochastic depth rates are observed to have relatively robust values for C_K and L_K, as displayed in Fig. 1(c). This means that models and training procedures can be reliably characterised early on in the training process, which can significantly speed up the assessment of new approaches. For instance, if a reference ResNet model produces a good result, and we know its C_K and L_K values for various epochs, then new models must navigate close to this reference model – see for example in Fig. 1 that **skip2** and **skip3** models produce good convergence and generalisation, so new models must try to navigate close enough to them. In addition, new models that show very distinctive growth in C_K and L_K values with respect to the reference model are unlikely to produce competitive result – see for example in Fig. 1 that **skip6**, **skip12**, **size8**, and **size512**, who eventually produce uncompetitive performance, can be identified and stopped as early as 4 epochs to reduce computation.

3.1. Dynamic Sampling

Dynamic sampling [15, 4] is a method that is believed to improve the convergence rate of SGD by reducing the noise of the gradient estimation with a gradual increase of the mini-batch size over the training process (this method has been suggested before, but we are not aware of previous implementations). It extends SGD by replacing the fixed size mini-batches $\mathcal{B}_k$ in (4) with a variable size mini-batch. The general idea of this method [15, 4] is that the initial noisy gradient estimations from small mini-batches explore a relatively flat energy landscape without falling into sharp local minima. The increase of mini-batch sizes over the training procedure provides a more robust gradient estimation on a sharper energy landscape that is supposed to be in a region of the space with better generalisation properties.

The most important point that dynamic sampling showed with respect to our proposed measures C_K, L_K in (6),(7) is that it broke the stability observed in Fig. 1. In general, we note that the application of dynamic sampling allowed the curves to move from the region with the original batch size to the region of the final batch size.

3.2. Dynamic Stochastic Depth Rate

The stochastic depth ResNet [13] drops a random number of residual units at each epoch k, with probability p_l to drop residual blocks towards the last layer L of the model, as follows:

$$p_l = 1 - \frac{l}{L}(1 - p_L), \qquad (8)$$

where p_L is a hyper-parameter that represents the drop probability of the L^{th} residual block. The proposed dynamic stochastic depth rate follows similar intuition as dynamic sampling of Sec. 3.1, where the runtime drop probability p_L is reduced as a function of training epoch k. In effect, the initial gradient estimations with large p_L are noisy, leading to the exploration of flat energy landscapes, and in later epochs, smaller p_L produce robust gradient estimations with a sharper energy landscape.

4. Experiments

The experiments are carried out on two commonly evaluated benchmark datasets: CIFAR-10 [18], and CIFAR-100 [18]. CIFAR-10 and CIFAR-100 datasets contain 60000 32×32-pixel coloured images, where 50000 are used for training and 10000 for testing.

We use a 110-layer ResNet model [12], including 54 residual units, formed by the following operators in order: 3×3 convolution, batch normalisation [14], ReLU [23], 3×3 convolution, and batch normalisation. This residual unit empirically shows better performance than previously proposed residual units (also observed in [1] in parallel to our own work). We use the simplest skip connection with no trainable parameters. For SGD, we use 0.9 for momentum, and the learning rate decay is performed in multiple steps: the initial learning rate is 0.1, which decays by $1/10$ at the 161^{st} epoch, and by another $1/10$ at the 241^{st} epoch – the whole training takes 320 epochs. The training uses data augmentation, as described by [12].

For each experiment, we measure the training and testing classification error, and the proposed measures C_K (6) and L_K (7) – the reported results are actually the mean result obtained from five independently trained models (each model is randomly initialised). All experiments are conducted on an NVidia Titan-X and K40 gpus without the multi-gpu computation. In order to obtain $\widetilde{\mathbf{F}}_k$, the explicit calculation of $\mathbf{J}_k$ is obtained with a modification of the torch [8] NN and CUDNN libraries (convolution, batch normalisation and fully-connected modules) to acquire the Jacobian $\mathbf{J}_k = \nabla \ell(y_i, f(\mathbf{x}_i, \theta_k))$ during back-propagation. Note that the memory complexity to store $\mathbf{J}_k$ scales linearly with respect to the number of model parameters, which is acceptable for the 1.7 million parameters in the 110-layer ResNet. However, the formation of $\mathbf{J}_k$ is performed for each training sample, which reduces the training speed. To alleviate this issue, we compute $\widetilde{\mathbf{F}}_k$ at intervals of 50 mini-batches, resulting in a sampling rate of $\approx 2\%$ of training set. Finally, our stochastic depth is slightly different from [13], where we implement an independent stochastic depth for each sample in the mini-batch (instead of for the whole mini-batch [13]), which shows empirically a slightly improvement in terms of the classification accuracy.

4.1. Mini-batch Size and Dynamic Sampling

Mini-batch size: in Fig. 2, we show our first experiment comparing different mini-batch sizes with respect to the training and testing errors, and the proposed measures C_K (6) and L_K (7) – please focus on results **size**$\{\mathbf{8}, \mathbf{16}, \mathbf{32}, \mathbf{64}, \mathbf{128}, \mathbf{256}, \mathbf{512}\}$. Notice that for CIFAR-10, small mini-batch sizes decrease C_K and increase L_K, and the large mini-batch sizes shows the opposite effect, while for CIFAR-100 mini-bath sizes appears to vary mainly L_K. Also notice that small C_K and large L_K indicate poor training convergence, and large C_K and small L_K show poor generalisation, so the best convergence and generalisation requires a medium value for both measures, which happens for mini-batch sizes between 16 and 64. For CIFAR-10, the minimum error is obtained with mini-batch size 32, with a test classification error of $4.78\% \pm 0.05\%$. For CIFAR-100, the minimum error is also obtained with mini-batch size 32 with test error $23.83\% \pm 0.52\%$.

Initial learning rate: in Fig. 2, we observe that classification errors, C_K and L_K are affected by different values for the initial learning rate (in our experiments, the default is an initial learning rate of 0.1, as mentioned above) – please focus on results **size**$\{\mathbf{16}, \mathbf{256}\}$–**lr**$\{\mathbf{0.05}, \mathbf{0.2}\}$. More specifically, we tested the different initial learning rates of 0.05 and 0.2 on for the mini-batch sizes 16 and 256, and we observe that for both cases, the smaller initial learning rate of 0.05 increases C_K and decreases L_K and the larger larger learning rate 0.2 decreases C_K and increases L_K. As a result, for the mini-batch size 16, the initial learning rate of 0.05 pulls it towards the optimum region (of mini-batch size 32), while the initial learning rate of 0.2 pushes it away from that region. For the mini-batch size 256, the effect is the opposite: the larger initial learning rate of 0.2 pulls the measures towards the optimum region, and the initial learning rate of 0.05 pushes it away from that region. The main conclusion with this experiment is that the mini-batch sizes and initial learning rate are tightly related and can be used to compensate one another, and move the training procedure to different performance regions. For CIFAR-10, the lowest test error (for mini-batch size 32) of $4.78\% \pm 0.05\%$ is matched by the mini-batch size 16 with initial learning rate 0.05 ($4.78\% \pm 0.12\%$), while mini-batch size 256 and initial learning rate 0.2 achieves $5.24\% \pm 0.21\%$. For CIFAR-100, the lowest test error (for mini-batch size 32) of $23.83\% \pm 0.52\%$ is improved by the mini-batch size 16 with initial learning rate 0.05 ($23.54\% \pm 0.21\%$), while mini-batch size 256 and initial learning rate 0.2 achieves $25.38\% \pm 0.19\%$.

Analysis: The main observations of the above experiments are: 1) the mini-batch size experiments can serve as a survey and mapping operation to set up baseline in the measurement space and discover the optimum region within; 2) the initial learning rate experiment shows that we are able to move the measurement readings of one type of hyper-parameter towards or away from the optimum region by tweaking the value of another hyper-parameter. These observations show that C_K and L_K allow the coefficient of multiple hyper-parameters to be examined in one space and the selection of hyper-parameters to be guided with one unified goal–moving the measures towards the optimum region in this space. However, it can be observed from Fig. 2, the "landscapes" of C_K and L_K are different with the use of CIFAR-10 and CIFAR-100 datasets (especially in the CIFAR-100 experiment, the mini-batch sizes has a smaller influence on C_K compared to the counterparts in CIFAR-

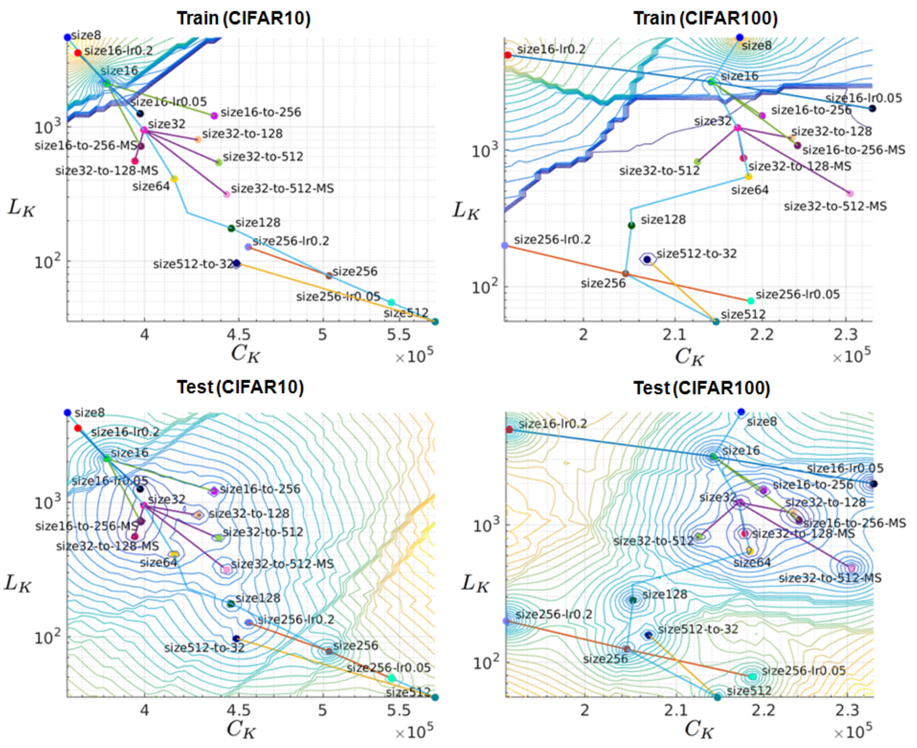


Figure 2. This graph shows how varying mini-batch sizes and initial learning rate affect training performance, which is also related to the proposed measures C_K and L_K. In addition, we also show the performance of dynamic sampling – please see text for more details.

10 experiments), which suggests that C_K and L_K values are relative and dataset dependent, leading to the necessity of set up the baseline experiment before the measures can be useful to guide the selection of the hyper-parameters.

Given that the use of each mini-batch has a unique signature of C_K and L_K and these measures are accumulated over the entire training procedure, we apply the dynamic sampling method to allow the use of non-uniformed mini-batch sizes during the training procedure with the aim to tweak the accumulation of the measures.

Dynamic sampling: in Fig. 2, we show different dynamic sampling alternatives, and how they affect the classification errors, C_K and L_K – please focus on results **size**$\{\mathbf{16}, \mathbf{32}, \mathbf{512}\}-\mathbf{to}-\{\mathbf{32}, \mathbf{128}, \mathbf{256}, \mathbf{512}\}\{\emptyset, -\mathbf{MS}\}$. The first experiment involves a dynamic sampling from mini-batch size 32 to 512, and as a control experiment, we also run an experiment with mini-batch size from 512 to 32 – in both cases, the classification errors, C_K and L_K are pushed away from the initial mini-batch size region towards the final mini-batch size region. Additional experiments with different initial and final mini-batch sizes show similar results as above. Therefore, the dynamically sampling of the training set has the effect of moving the training procedure to different performance regions. For CIFAR-10, the best result achieved with these experiments is with **size32-to-128**, with a test error of $4.9\% \pm 0.05\%$, which is comparable to the error of **size32**. However it is important to mention that even though both training methods achieve similar accuracy, the dynamic sampling allows a faster training process: with our computer setup described above, the full training for **size32** takes 10.9 hours, while **size32-to-128** requires 9.5 hours. For CIFAR-100, the best result achieved with these experiments is with **size32-to-128**, with a test error of $23.90\% \pm 0.31\%$. The training time for respective models are identical to the CIFAR-10 experiments.

4.2. (Dynamic) Stochastic Depth

Dynamic sampling for multi-step learning rate decay: following the intuition that learning rate decay causes the training process to focus on specific regions of the energy landscape, we test if the dynamic sampling should be performed within each particular value of learning rate, instead of the approach above, where the sampling is done over all training epochs and decreasing learning rates. This new approach is marked with **-MS** in Fig. 2, where for each learning rate value, we re-iterate through the sequence of mini-batch sizes of the corresponding dynamic sampling policy. In general, these **-MS** polices are more effective at pushing the measurements within the optimum region of the graph. For CIFAR-10, the best results achieved with this new policy are obtained with **size16-to-256-MS** and **size32-to-128-**

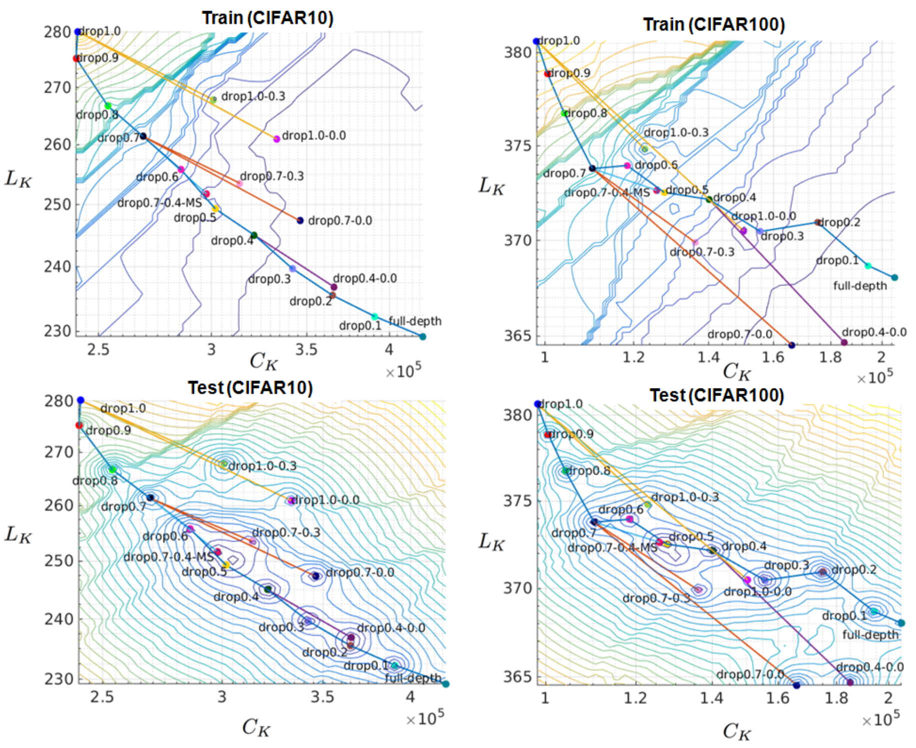


Figure 3. This graph shows how varying the dynamic stochastic depth affects training performance, which is also related to the proposed measures C_K and L_K – please see text for more details.

MS with test errors of $4.76\% \pm 0.22\%$ and $4.76\% \pm 0.13\%$, respectively, which are slightly better than the above original dynamic sampling polices. For CIFAR-100, the best result is obtained by **size32-to-128-MS** with a test error of $23.69\% \pm 0.34\%$, showing a noticeable margin to **size32-to-128**.

Baseline stochastic depth: using drop rates $p_L \in [0.1, 1.0]$ in (8) (labelled in the results as **drop**$\{\mathbf{0.1}, ..., \mathbf{1.0}\}$) on a ResNet model trained with mini-batch size=100, it is clear from Fig. 3 that for CIFAR-10 and CIFAR-100 an increase of p_L is associated with a reduction in C_K and an increase of L_K, which in turn leads to worse training convergence. Interestingly, the generalisation seems stable over a fairly large rage of drop rates (between 0.2 and 0.6). For CIFAR-10, the best results are: **drop**$\{\mathbf{0.2}, \mathbf{0.3}\}$ with a test error of $4.66\% \pm \{0.07\%, 0.13\%\}$, **drop**$\mathbf{0.4}$ with $4.57\% \pm 0.16\%$, and **drop**$\{\mathbf{0.5}, \mathbf{0.6}\}$ with $4.79\% \pm \{0.07\%, 0.20\%\}$. For CIFAR-100, the best results are: **drop**$\mathbf{0.3}$ with $22.21\% \pm 0.22\%$, **drop**$\mathbf{0.4}$ with $22.44\% \pm 0.22\%$, and **drop**$\mathbf{0.5}$ with $22.36\% \pm 0.11\%$.

By analysing the baseline experiment, we can see p_L has a major influence on C_K while its influence on L_K is minimal (see Fig. 1(a) for the comparison between the lines of mini-batch sizes and stochastic depth baselines and see Fig. 3 for the scale difference of C_K over L_K axes). Therefore, the guide to select the drop rate is to focus on tweaking C_K closer to the optimum region defined by **drop**$\{\mathbf{0.2}, \mathbf{0.3}\}$.

Dynamic stochastic depth: following a similar intuition used for the dynamic sampling, we design a dynamic stochastic depth, which starts with high drop rate that has poor convergence, but good generalisation (similarly to a mini-batch of small size), and as training progresses, this drop rate is gradually reduced. These approaches are labelled as **drop**$\{\mathbf{0.4}, ..., \mathbf{1.0}\}$-$\{\mathbf{0.0}, ..., \mathbf{0.4}\}\{\emptyset, -\mathbf{MS}\}$, where the first number indicates the drop rate at the first iteration, the second means the drop rate at the last iteration, and **-MS** represents the multi-step learning rate decay. The empirical results in Fig. 3 show that a large value of drop rate is less favorable in terms of training convergence. The two polices that are close to **drop0.4** are: **drop0.7-0.3**, which achieves $4.76\% \pm 0.13\%$ on CIFAR-10, and $23.01\% \pm 0.19\%$ on CIFAR-100; and **drop0.4-0.0**, which achieves $4.68\% \pm 0.16\%$ on CIFAR-10, and $23.54\% \pm 0.22\%$ on CIFAR-100. In addition, similarly to multi-step dynamic sampling polices, the **-MS** polices are proposed to gradually decay the drop rate during each period of learning rate. The **-MS** policy improves the test error **drop0.7-0.4-MS** reaching $4.62\% \pm 0.15\%$ on CIFAR-10 and $22.68\% \pm 0.27\%$ on CIFAR-100.

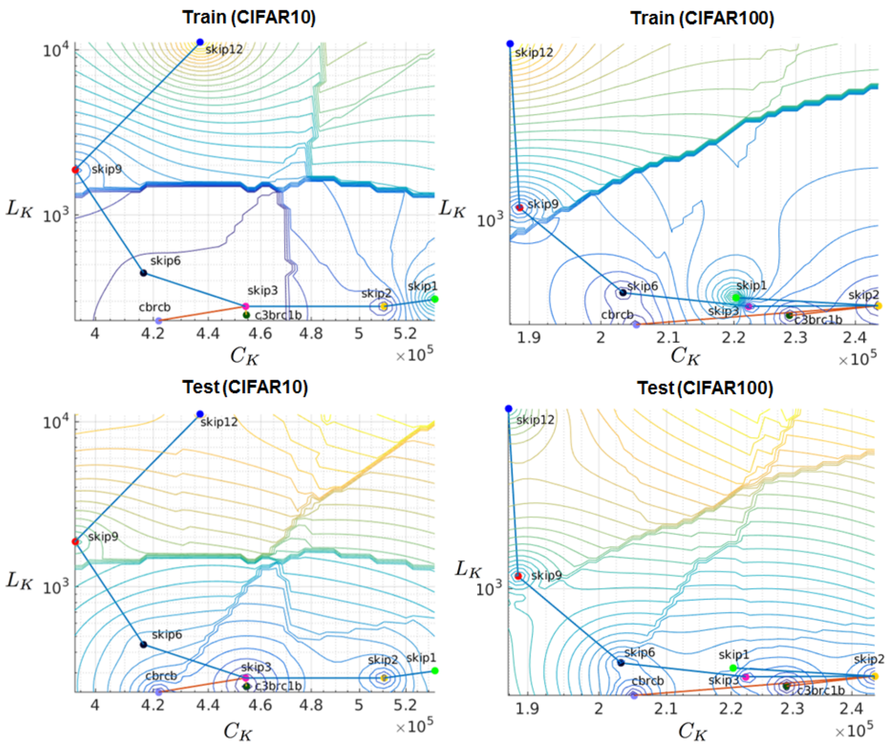


Figure 4. This graph shows how different ResNet architectures affect training performance, which is also related to the proposed measures C_K and L_K – please see text for more details.

4.3. Model Selection

In Fig. 4, we compare different ResNet architectures with respect to their classification performance and our proposed measurements C_K and L_K. The models explored in this section are trained with mini-batches of size 100, and are summarised as follows: 1) the baseline model that used in the above mini-batch size and stochastic depth experiments, which has 110 layers, containing residual units with the following operators: 3×3 convolution, batch normalisation, ReLU, 3×3 convolution, and batch normalisation – this model is denoted as **cbrcb**; 2) baseline **skip**$\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{6}, \mathbf{9}, \mathbf{12}\}$: each skip connection shortcuts between 1 and 12 layers, each containing batch normalisation, ReLU, and 3×3 convolution [12]; and 3) a simple variation of **cbrcb** containing the following operators: 3×3 convolution, batch normalisation, ReLU, 1×1 convolution, and batch normalisation - this model is denoted by **c3brc1b** (due to the smaller number of parameters per residual unit, the **c3brc1b** has 194 layers to match 1.7M parameter).

Fig. 4 shows that models **skip**$\{\mathbf{2}, \mathbf{3}\}$, **cbrcb** and **c3brc1b** present good training convergence and good generalisation, while **skip**$\{\mathbf{6}, \mathbf{9}\}$ show good convergence, but poor generalisation, and **skip**$\{\mathbf{1}, \mathbf{12}\}$ have poor convergence. In general, looking at Figures 1 and 4 for model selection, we notice that, similarly to stochastic depth approaches, a mid-range values (compared to all other approaches with different mini-batch sizes and stochastic drop rates) for L_K and C_K appear to bring good training convergence and generalisation. Large values of C_K indicate poor convergence, and small values of C_K suggest poor generalisation. For CIFAR-10, the top performing three models are: **c3brc1b** with $4.80\% \pm 0.08\%$, **cbrcb** with a test error of $5.11\% \pm 0.07\%$, and **skip3** with $5.25\% \pm 0.19\%$. For CIFAR-100, the top performing models are: **c3brc1b** with $23.70\% \pm 0.24\%$, **cbrcb** with a test error of $24.70\% \pm 0.18\%$, and **skip2** with $25.63\% \pm 0.34\%$.

The top performing model of this experiment can be characterised based on the following reasoning. Comparing the performance of **skip2** and **skip3**, we notice that larger C_K seems to marginally improve the results from **skip2** ($5.31\% \pm 0.19\%$) to **skip3** ($5.25\% \pm 0.19\%$) in the CIFAR-10 experiment, and from **skip3** ($25.63\% \pm 0.34\%$) to **skip2** ($25.90\% \pm 0.54\%$) in CIFAR-100 experiment, both with a stable value for L_K. Furthermore, the location of **cbrcb** shows small L_K and much smaller C_K to the optimum location (i.e., **skip2** in CIFAR-10 and **skip3** in CIFAR-100). So by analogy, if we design a model that has a similar value of L_K, but larger C_K, then we can improve the test error result of **cbrcb**. Unlike the other types of hyper-parameter which can be tuned by value, the model selection can only be explored by making small structure adjustment to the model structure of **cbrcb** and see how C_K and L_K change accordingly. During our experiment, we found that **c3brc1b** has exactly these C_K and L_K values, which indeed improved the results from **cbrcb**.

5. Discussion and Conclusion

For all the experiments above, we notice that our test error results are quite competitive with respect to the state of the art using models that are significantly more efficient, particularly with respect to memory complexity. In particular, we are aware that the state of the art on CIFAR-10 and CIFAR-100 have been pushed to 3.89% and 18.85% (respectively) by Wide ResNet model [31]. However, the Wide ResNet model contains 56 million parameters, whereas our models contain only 1.7 million parameters ($\approx$ 3% of the Wide ResNet model) and show a minimum test error of 4.33% (result on five independently trained models: 4.47% $\pm$ 0.12%) and 21.36% (result of five models: 21.64% $\pm$ 0.17) on CIFAR-10 and CIFAR-100 when we combine **c3brc1b** model with **drop-0.4**. To the best of our knowledge, the best performing model containing 1.7 million parameter is the pre-act-ResNet [12], which shows 5.46% and 24.33% on CIFAR-10 and CIFAR-100 respectively. The take-home message of this paper is the following: training deep networks, and in particular ResNets, is still an art, but the use a few easily computed measures from SGD can provide substantial help in the selection of model parameters that lead to good training convergence and generalisation.

In conclusion, we propose a novel methodology to characterise the performance of deep ResNets regarding training convergence and generalisation as a function of mini-batch size, model structure, learning rate and stochastic depth rate. This proposed methodology defines a space that can be used for guiding the training of ResNets, which led us to propose two new training mechanisms: dynamic sampling and dynamic stochastic depth rate. We believe that the newly proposed measures will help researchers make important decisions about the ResNet structure and training procedure. We also expect that this paper has the potential to open new research directions on how to assess and predict top performing ResNet models with the use of the proposed measures (C_K and L_K) and perhaps new other measures that can be proposed in the future.

References

[1] Training and investigating residual nets. Training and investigating Residual Nets [Online]. Accessed: 2017-03-13.

[2] S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[3] D. P. Bertsekas. Incremental least squares methods and the extended kalman filter. *SIAM Journal on Optimization*, 6(3):807–822, 1996.

[4] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[5] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[6] P. Chaudhari, A. Choromanska, S. Soatto, and Y. LeCun. Entropy-sgd: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations (ICLR)*, pages 1–19, 2017.

[7] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, volume 38, pages 192–204, 2015.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, EPFL-CONF-192376, 2011.

[9] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12(Jul):2121–2159, 2011.

[10] R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[11] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations (ICLR)*, pages 1–20, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[13] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 646–661. Springer, 2016.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pages 448–456, 2015.

[15] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, pages 1–16, 2017.

[16] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, pages 1–15, 2015.

[17] R. Kiros. Training neural networks with stochastic hessian-free optimization. In *International Conference on Learning Representations (ICLR)*, pages 1–11, 2013.

[18] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[19] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht. Gradient descent only converges to minimizers. In *Conference on Learning Theory*, pages 1246–1257, 2016.

[20] E. Littwin and L. Wolf. The loss surface of residual networks: Ensembles and the role of batch normalization. *arXiv preprint arXiv:1611.02525*, 2016.

[21] J. Martens. Deep learning via hessian-free optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 735–742, Haifa, Israel, 2010. Omnipress.

[22] J. Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[23] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of International Conference on Machine Learning*, pages 807–814, 2010.

[24] B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[25] L. Sagun, L. Bottou, and Y. LeCun. Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*, 2016.

[26] N. N. Schraudolph. Fast curvature matrix-vector products. In *International Conference on Artificial Neural Networks*, pages 19–26. Springer, 2001.

[27] D. Soudry and Y. Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

[28] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[29] J. H. Wilkinson. *The algebraic eigenvalue problem*, volume 87. Clarendon Press Oxford, 1965.

[30] S. Wright and J. Nocedal. Numerical optimization. *Springer Science*, 35:67–68, 1999.

[31] S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, pages 1–15, 2016.

[32] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Chapter 8

Conclusion and Future Works

Deep learning has shown promising results on many computer vision problems, where the stunning performance of deep learning methods has attracted a great deal of attention. Nevertheless, our understanding of deep learning methods is still preliminary. This thesis contributes to a better understanding of deep learning methodologies by studying four fundamental deep learning problems: nonlinear classification using CNN features, normalisation of piecewise linear activation units, multiple-size features and CNN Maxout activation units, and training characterisation of residual networks. Our proposed methodologies in this thesis have been evaluated on challenging publicly available visual classification datasets, and shown competitive results compared to state-of-the-art methods on these datasets.

In this chapter, we first summarise the main contributions of our work, followed by discussions on the limitations of our methods and possible future directions.

8.1 Summary of Contributions

The main contributions of this thesis are summarised as the followings:

- In Chapter 4, we show a novel nonlinear hierarchical classifier designed to work with features of large dimensionality, such as the CNN features. This nonlinear classifier is a binary tree classifier that uses logistic regression classifiers as internal tree nodes and linear SVM [10] as the leaf classifiers, which maintains low runtime and memory costs in contrast to other nonlinear classifiers. Furthermore, we introduce a novel loss function to learn this classifier, which minimises the classification error in a non-greedy way and at the same time delays hard classification to nodes further down the tree. Our classifier has been evaluated on the Pascal VOC 2007 [66] dataset, and results show that the classification performance of our classifier is better than the compared nonlinear shallow and hierarchical classifiers, and competitive to linear SVM classifier, with the use of features extracted from two CNN models: OverFeat [30] and VGG-CNN [8], suggesting that our classifier is robust to different types of features (See Table 1 in Chapter 4);

- In Chapter 5, we show how to increase the model capacity of the NIN [31] model with the use of the Maxout [53] activation function to replace ReLU [48]. We found that the model capacity may not increase as expected and can lead to an ill-conditioned training, if the input data to the Maxout activation function is not properly normalised. Our main contribution of this work is to correct the normalisation issue with the use of a Batch Normalisation [45] unit inside the Maxout unit, resulting in a better pre-conditioning of the unit. Experiments show that our extension of the NIN [31] model: Maxout-network in Maxout-network model surpasses the performance of several state-of-the-art methods on MNIST [47], CIFAR-10/100 [72], and SVHN [119] datasets (see Section 3 in Chapter 5);

- In Chapter 6, we propose a competitive multiple-size CNN module composed of convolutional filters of up to four filter sizes that are joined by a Maxout activation unit, which promotes competition and reduces the co-adaptation amongst these filters, and also addresses the channel growth issue of the Inception [29] multiple-size CNN module. We show that the classification results produced by a number of network architectures can be improved with the use of our module. We evaluate our proposed module on MNIST, CIFAR-10/100, SVHN, and ImageNet ILSVRC 2012 [9] datasets (see Section 4 in Chapter 6);

- In Chapter 7, we propose two novel measures derived from the eigenvalues of the approximate empirical Fisher matrix to evaluate the training of residual networks [24]. Our proposed measures can be efficiently calculated within the stochastic gradient descend (SGD) iteration; hence they do not result in significant overhead to the training process. We show that these measures can be used to guide the selection of mini-batch size, model structure, learning rate and stochastic depth rate. We also propose a new way to schedule dynamic sampling and dynamic stochastic depth, leading to competitive classification results on CIFAR-10/100 datasets compared to other state-of-the-art models that have significantly larger capacity (See Section 4 in Chapter 7).

8.2 Future Works

As stated in the above section, our methods show competitive performance to the state-of-the-art methods on various datasets, and they also shed light in the understanding of deep learning models. Nevertheless, we believe that our methodologies could be further improved by a few possible directions:

- In Chapter 4, we show the comparison of our classifier and the state-of-the-art methods with the use of OverFeat [30] and VGG-CNN [8] feature models on PASCAL VOC 2007 [66], where the performance of the nonlinear classifiers are sub-optimal compared to the linear SVM. We plain to extend this evaluation with the use of feature extracted by recently proposed CNN feature models such as GoogLeNet [29] and ResNet [24]. We also plan to verify the results of our approach on other visual classification datasets, such as the ImageNet [9] visual classification task.

- We intend to integrate the nonlinear classification model proposed in Chapter 4 and the CNN feature model to enable end-to-end training. One possible way to realise this idea is to integrate the nonlinear classification model as a CNN module in order to replace the linear layer before the softmax loss layer, and train the classification model from scratch. The results show that this modification only increases the model capacity at the final layer of the model, leading to marginal improvement compared to the baseline model. This module is actually a piecewise linear activation unit, which can be used to replace ReLU and increase the model capacity throughout the entire model, potentially leading to substantial improvements. This module motivated our work in Chapter 5, where we replace this module by a Maxout [53] unit that can be trained faster with a comparable performance. Note that the piecewise linear activation functions follows a greedy loss by nature (i.e., Maxout always pick the highest activation), where we plan to explore the possibility to integrate our non-greedy loss function (3.5) with piecewise linear activation function in the future.

- In Chapter 5, we empirically demonstrated that the combination of Maxout activation unit and Batch Normalisation (BN) unit provides a powerful framework, where the BN unit aims to keep a balanced distribution of the input data for these units. We anticipate that other types of normalisation (e.g., layer normalisation [46]) can replace BN, and the performance of these new combinations can be explored in the future. Furthermore, the empirical demonstration in Chapter 5 can be addressed theoretically in the future.

- In Chapter 6, we show that our competitive CNN module includes the use of large size filters, which can induce a large number of parameters on large size networks. Our proposal of the bottleneck-CMSC (see Section 4.3 in Chapter 6) serves as one solution to solve this issue. We believe that the use of multiple-size features is advantageous, so we plan to further investigate other means to solve the parameter issue involved with the use of large filters to make it computationally affordable to be used in large size deep learning models (i.e., one possible solution is to use a stack of two stacks of 3×3 convolution layer with nonlinearities to simulate a convolution with 5×5 receptive field, etc.).

- In Chapter 7, we show two types of measures to characterise the training of residual networks (ResNets) [24]. Our implementation is based on a single-GPU computation environment, which limits our ability to test bigger networks (i.e., in a GTX Titan-X GPU with 12G memory, the network size is limited to 5 millions parameters). We plan to extend our implementation to multi-GPU programming, which can be used to test recent deep learning models (not only the ResNets) on more challenging large-scale datasets, such as the ImageNet [9] dataset.

- We plan to further analyse the impact of other types of hyper-parameters with our proposed measures, such as the depth of the model. One principle we followed during the research in Chapter 7 is that we restrict the comparison of models with similar number of parameters (i.e., within $\pm 5\%$ margin of the number of parameters of the referenced 110-layer ResNet model) and trained these models with the rest of hyper-parameters unchanged. This allowed the approximate Fisher matrix and the derived C_K and L_K measures to be comparable among different models and hyper-parameter settings. However, we found that if we adjust only the depth of the networks (i.e., we do not change the number of filters per layer to compensate the change of number of parameters), deeper models always show larger C_K and L_K values. This brings up a question: maybe these measures are meant to be normalised by number of parameters. Furthermore, these measures are also functions of the number of epochs, maybe the values should be normalised by training epochs in order to compare training procedures with different numbers of epochs. We plan to investigate these questions in a future work.

- We plan to explore whether applying our measures to other types of loss function can be useful to characterise the training procedure. For instance, we shall test if the softmax log-loss function of a model was replaced with the Hinge loss function, then use the same methodology to calculate C_K and L_K values.
- Finally, an issue associated with both proposed measures is that they are relative measures and dataset dependent, where some baseline experiments are necessary to quantify the measurement space and discover the optimum region first. Such practice are time consuming and may not be possible for large-scale datasets such as the ImageNet. We plan to search for new measures which are invariant and less dependent on prior experiments.

Bibliography

[1] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[2] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

[3] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Proceedings of the ECCV Workshop on Statistical Learning in Computer Vision*, volume 1, pages 1–2. Prague, 2004.

[4] Josef Sivic, Andrew Zisserman, et al. Video google: A text retrieval approach to object matching in videos. In *International Conference on Computer Vision (ICCV)*, volume 2, pages 1470–1477, 2003.

[5] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2169–2178. IEEE, 2006.

[6] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 143–156, 2010.

[7] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(9):1627–1645, 2010.

[8] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference (BMVC)*, pages 1–12, 2014.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. In *International Journal of Computer Vision*, pages 1–42. Springer, 2014.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[11] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

[12] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266): 1332–1338, 2015.

[13] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NIPS)*, pages 3320–3328, 2014.

[14] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[15] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2366–2374, 2014.

[16] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW '14, pages 512–519, Washington, DC, USA, 2014. IEEE Computer Society.

[17] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional

architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[18] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, EPFL-CONF-192376, 2011.

[19] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[21] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, pages 1097–1105, 2012.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, pages 1–14, 2015.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[25] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, pages 1–15, 2016.

[26] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, pages 1–16, 2017.

[27] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2924–2932, 2014.

[28] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1475, 2015.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[30] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR)*, pages 1–16, 2014.

[31] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, pages 1–10, 2014.

[32] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, pages 1–13, 2015.

[33] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *Deep Learning Workshop, International Conference on Machine Learning*, pages 1–6, 2015.

[34] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations (ICLR)*, pages 1–11, 2016.

[35] Falong Shen, Rui Gan, and Gang Zeng. Weighted residuals for very deep networks. In *International Conference on Systems and Informatics (ICSAI)*, pages 936–941. IEEE, 2016.

[36] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[37] Ke Zhang, Miao Sun, Xu Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2017.

[38] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 550–558, 2016.

[39] Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *International Conference on Learning Representations (ICLR)*, pages 1–14, 2017.

[40] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

[41] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 28, pages 1058–1066, 2013.

[42] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 646–661. Springer, 2016.

[43] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *Advances in Neural Information Processing Systems (NIPS)*, pages 28–36, 2016.

[44] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, pages 249–256, 2010.

[45] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pages 448–456, 2015.

[46] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[47] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

[48] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of International Conference on Machine Learning*, pages 807–814, 2010.

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[50] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 30, 2013.

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[52] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations (ICLR)*, pages 1–14, 2016.

[53] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 28, pages 1319–1327, 2013.

[54] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2310–2318, 2013.

[55] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[56] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[57] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12(Jul):2121–2159, 2011.

[58] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[59] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, pages 1–15, 2015.

[60] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, volume 38, pages 192–204, 2015.

[61] Etai Littwin and Lior Wolf. The loss surface of residual networks: Ensembles and the role of batch normalization. *arXiv preprint arXiv:1611.02525*, 2016.

[62] Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

[63] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent only converges to minimizers. In *Conference on Learning Theory*, pages 1246–1257, 2016.

[64] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations (ICLR)*, pages 1–20, 2015.

[65] Levent Sagun, Léon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*, 2016.

[66] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC 2007) Results. The PASCAL Visual Object Classes Challenge 2007 (VOC 2007) Results [Online].

[67] Zhuowen Tu. Probabilistic boosting-tree: Learning discriminative models for classification, recognition, and clustering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1589–1596. IEEE, 2005.

[68] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 32, pages 647–655, 2014.

[69] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[70] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1717–1724, 2014.

[71] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[72] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[73] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[74] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[75] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

[76] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[77] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[78] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[79] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[80] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[81] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, pages 1–9, 2013.

[82] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pages 1462–1471, 2015.

[83] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *International Conference on Learning Representations (ICLR)*, pages 1–10, 2015.

[84] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2204–2212, 2014.

[85] Frank Rosenblatt. The perceptron: A perceiving and recognizing automaton. *Technical Report 85-460-1, Project PARA*, 1957.

[86] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[87] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*, pages 267–285. Springer, 1982.

[88] Kevin J Lang and Geoffrey E Hinton. *A time-delay neural network architecture for speech recognition*. Carnegie Mellon University, Computer Science Department, 1988.

[89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[90] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of International Conference on Document Analysis and Recognition*, volume 3, pages 958–962. Citeseer, 2003.

[91] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[92] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2): 197–227, 1990.

[93] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 1–1. IEEE, 2001.

[94] Tianshi Gao and Daphne Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2072–2079. IEEE, 2011.

[95] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[96] Rupesh Kumar Srivastava, Jonathan Masci, Faustino Gomez, and Jürgen Schmidhuber. Understanding locally competitive networks. In *International Conference on Learning Representations (ICLR)*, pages 1–11, 2015.

[97] Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 392–407. Springer, 2014.

[98] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 346–361. Springer, 2014.

[99] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4353–4361, 2015.

[100] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[101] Thomas Serre, Minjoon Kouh, Charles Cadieu, Ulf Knoblich, Gabriel Kreiman, and Tomaso Poggio. A theory of object recognition: computations and circuits in the feedforward path of the ventral stream in primate visual cortex. Technical report, DTIC Document, 2005.

[102] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3), 2007.

[103] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016.

[104] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[105] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[106] Dimitri P Bertsekas. Incremental least squares methods and the extended kalman filter. *SIAM Journal on Optimization*, 6(3):807–822, 1996.

[107] Nicol N Schraudolph. Fast curvature matrix-vector products. In *International Conference on Artificial Neural Networks*, pages 19–26. Springer, 2001.

[108] Stephen Wright and Jorge Nocedal. Numerical optimization. *Springer Science*, 35:67–68, 1999.

[109] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[110] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 735–742, Haifa, Israel, 2010. Omnipress.

[111] Ryan Kiros. Training neural networks with stochastic hessian-free optimization. In *International Conference on Learning Representations (ICLR)*, pages 1–11, 2013.

[112] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 265–272. ACM, 2011.

[113] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 32, pages 604–612, 2014.

[114] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[115] Hyeyoung Park, S-I Amari, and Kenji Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764, 2000.

[116] Gaétan Marceau-Caron and Yann Ollivier. Practical riemannian neural networks. *arXiv preprint arXiv:1602.08007*, 2016.

[117] Training and investigating residual nets. Training and investigating Residual Nets [Online]. Accessed: 2017-03-13.

[118] Cifar: Alternate training strategies (rmsprop, adagrad, adadelta). CIFAR: Alternate training strategies (RMSPROP, Adagrad, Adadelta) [Online]. Accessed: 2017-04-21.

[119] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on Deep Learning and Unsupervised Feature Learning*, page 5, 2011.

[120] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15, pages 315–323, 2011.

[121] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, and Yann LeCun. Entropy-sgd: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations (ICLR)*, pages 1–19, 2017.

[122] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[123] James Hardy Wilkinson. *The algebraic eigenvalue problem*, volume 87. Clarendon Press Oxford, 1965.