Deep Learning for Fine-Grained Visual Recognition

Teng Li

School of Computer Science

The University of Adelaide

A thesis submitted for the degree of

Master of Philosophy

28/03/2016

Contents

Contents **ii**

List of Figures **viii**

1 Introduction **1**

1.1 Overview of fine-grained Visual Recognition 1

1.2 Overview of contributions 5

1.3 Outline 6

2 Deep learning techniques for fine-grained visual recognition **8**

2.1 Convolutional Neural Networks 8

2.1.1 Overview of CNNs 8

2.1.2 Batch Normalization 14

2.1.3 AlexNet, VGGNet and ResNet 15

2.2 Cross-convolutional-layer pooling 17

2.3 Bilinear CNN model 19

3 Spatially Weighted Pooling in deep CNNs for fine-grained visual recognition **21**

3.1 Cross-convolutional-layer pooling experiments on CompCars . . . 22

3.1.1 CompCars dataset 22

3.1.2 Experimental settings 24

3.1.3 Experimental details 26

3.1.4 Experimental results and discussions 27

3.2 The proposed method 32

3.2.1 Spatially weighted pooling 32
3.2.2 Forward and backward propagation of SWP 37
3.3 Experimental results of the proposed method and discussions . . . 38
3.3.1 Datasets 38
3.3.2 Experimental settings 40
3.3.3 Improved architectures as baselines 42
3.3.4 Implementation details 43
3.3.5 Discussion 44
3.3.6 Feature maps and learned filters visualization 58

4 Conclusions **79**

References **81**

Abstract

Fine-grained object recognition is an important task in computer vision. The cross-convolutional-layer pooling method is one of the significant milestones in the development of this field in recent years. Based on the method, we conducted a number of experiments on a new fine-grained car dataset - CompCars. The corresponding experiments illustrate its applicability and effectiveness on this newly-designed dataset. Meanwhile, based on the experiments, we found out that pooling the most distinguishable regions like car logos and headlights areas in the indicator maps, which usually have higher activations, with the local features in the same regions can achieve better results than those by pooling the whole indicator maps with the corresponding local features. Therefore, we conjecture that better performance may be achieved if we have more powerful indicator maps or pooling channels that can better highlight these distinguishable regions.

Based on the above hypothesis and inspired by the cross-convolutional-layer pooling, next we propose the Spatially Weighted Pooling (SWP) method, which is a simple yet effective pooling strategy to improve fine-grained classification performance. SWP learns a dozen of pooling channels or spatial encoding masks that aggregate local convolutional feature maps with learned spatial importance information and produce more discriminative features. It can be seamlessly integrated into existing convolutional neural network (CNN) architectures such as the deep residual network. It also allows end-to-end training. SWP has few parameters to learn, usually in several hundreds, therefore does not introduce much computational overhead.

SWP has shown significant capability to improve fine-grained visual recognition performance by simply adding it before fully-connected layers in off-the-shelf deep convolutional networks. We have conducted comprehensive experiments on a number of widely-used fine-grained datasets with a variety of deep CNN architectures such as Alex networks (AlexNet), VGG networks (VGGNet) and the deep residual networks (ResNet). By integrating SWP into ResNet (ResNet-SWP), we achieve state-of-the-art results on three fine-grained datasets and the MIT67 indoor scene recognition dataset. With ResNet152-SWP models, we obtain 85.2% on the bird dataset CUB-200-2011 without bounding-box annotations and 87.4% with bounding-box, 91.2% on FGVC-aircraft, 94.1% on Stanford-cars with bounding-box information and 82.5% on the MIT67 dataset.

Declaration

Acknowledgements

First I would like to thank Prof. Chunhua Shen, my principal supervisor during my study of the master degree. His dedication to research inspired me a lot, and will definitely influence me throughout my research career in the coming years. Under his supervision, I have learnt not only specific knowledge in the field, but also how to think as a mature researcher, including the ability to find unsolved challenges and address them in novel ways.

I would also like to thank staff and visitors in the Australian Center for Visual Technologies (ACVT). As a non-native English speaker, I made grammar errors in academic manuscripts now and then. I would like to thank my co-supervisor, Dr. Guosheng Lin, for his time and effort in helping my research. In addition, I would like to thank Dr. Lingqiao Liu, a research fellow in our research center. He gave me endless support and advice to help me develop my proposed pooling method which is also a cornerstone of this thesis. Other staffs in the ACVT have given me thoughtful suggestions on research. In a word, I am very lucky to be a member of the ACVT.

As a master by research student, I spent most time with PhD students who all have helped me a lot on both research and life. Having friends with different culture backgrounds really enriches my life experience.

Finally, thanks go to the excellent support from staffs in the School of Computer Science at the University of Adelaide. Julie Mayo and Sharyn Liersch are administrative staffs of our school. Thanks also go to our Head of School, Prof. Katrina Falkner, and our Postgraduate Coordinator Prof. Ian Reid I am really grateful for what they have done.

List of Figures

1.1 Examples from (first row) the birds dataset [1], (second row) the Stanford-Cars dataset [2], (third row) the Aircraft dataset [3], and (last row) the CompCars dataset [4] used in our experiments. . . . 2

1.2 Two SUV models are similar in side view, but are very different in the front views. 3

2.1 **The overview of cross-convolutional-layer pooling.** Pooling local features extracted from Conv. Layer t with pooling channels from Conv. Layer t+1. 18

2.2 **The overview of Bilinear CNN model.** Bilinear CNN architecture consists of two feature extractors whose outputs are multiplied using outer product at each location of the image. 19

3.1 Examples from the CompCars dataset [4]. The images in the five rows illustrate front viewpoint (F), rear viewpoint (R), side viewpoint (S), front-side viewpoint (FS) and rear-side viewpoint (RS). 23

3.2 The tree structure of car model hierarchy. Several car models of Audi A4L in different years are also displayed. 24

3.3 Visualizing of ROI regions by accumulating 512 feature maps of conv53 in the fine-tuned CompCars VGG model. Right: the original image. Left: the indicator map. 31

3.4 The first 81 of total 256 feature maps extracted from the last convolutional layer in AlexNet (left), and the corresponding input image (right). 33

3.5 Visualizing of some feature maps extracted from the last convolutional layer in AlexNet. Three feature maps in the first row indicate some semantically meaningful regions and three feature maps in the second row illustrate the characteristic of similarity. . 34
3.6 **The overview of the proposed method.** An SWP layer is added after the last convolutional layer or the last max pooling layer and before the first FC layer. 35
3.7 **The illustration of SWP in more detail.** An SWP layer is added after the last convolutional layer or the last max pooling layer and before the first FC layer. 36
3.8 Examples from (first row) the birds dataset [1], (second row) aircraft dataset [3], (third row) cars dataset [2], and (last row) the MIT67 dataset [5] used in our experiments. 40
3.9 Nine SWP channels learned on the birds dataset [1] with AlexNet-SWP architecture. 49
3.10 Visualization of 9 weighted masks learnt from Stanford-Cars VGG16-SWP. 50
3.11 Visualizing of six feature maps with the highest activations by pooling with two learned SWP channels for birds ResNet101-SWP. 51
3.12 Visualizing of six feature maps with the highest activations by pooling with two learned SWP channels for CompCars ResNet101-SWP. 52
3.13 Training on CompCars All-View and F-View with ResNet-SWP. . 54
3.14 Plotting training and validation errors training on CompCars and Stanford-Cars. 55
3.15 Other views training samples. 57
3.16 Sample test images that are mistakenly predicted as another model. 58
3.17 Sample test images that are mistakenly predicted as another model in their makes. Each row displays two samples and each sample is a test image followed by another image showing its mistakenly predicted model. 59
3.18 Visualizing Conv1 filters in birds ResNet101-SWP. 60
3.19 Visualizing Conv1 filters in Stanford-Cars ResNet101-SWP. 61

3.20 Visualizing learned 16 SWP filters in birds ResNet101-SWP. The learned weights are between -0.0348003358 and 0.0481069833. . . 62
3.21 Visualizing learned 16 SWP filters in CompCars ResNet101-SWP. The learned weights are between -0.0566353425 and 0.124176443. 63
3.22 Visualizing feature maps in the last convolutional layer with highest activations after pooling with the first 8 filters in the SWP layer in CompCars ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all $14,939$ test images. We obtain six plot images by overlaying the six feature maps on their original images. 64
3.23 Visualizing feature maps in the last convolutional layer with highest activations after pooling with from the 9-th to 16-th filters in the SWP layer in CompCars ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all $14,939$ test images. We obtain six plot images by overlaying the six feature maps on their original images. 65
3.24 Visualizing feature maps in the last convolutional layer with highest activations after pooling with the first 8 filters in the SWP layer in birds ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all $5,794$ test images. We obtain six plot images by overlaying the six feature maps on their original images. 66
3.25 Visualizing feature maps in the last convolutional layer with highest activations after pooling with from the 9-th to 16-th filters in the SWP layer in birds ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all $5,794$ test images. We obtain six plot images by overlaying the six feature maps on their original images. 67

3.26 Patches with highest activations for several filters in the first convolutional layer (in total 64 filters) in the fine-tuned CompCars ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $14,939$ test images. From the top row to the bottom are for the 5-th, 7-th, 17-th, 30-th, 34-th, 51-th, 54-th and 64-th filters. 68

3.27 Patches with highest activations for several filters in the first convolutional layer (in total 64 filters) in the fine-tuned birds ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $5,794$ test images. From the top row to the bottom are for the 6-th, 7-th, 19-th, 25-th, 28-th, 45-th, 46-th and 57-th filters. . . . 69

3.28 Patches with highest activations for several filters in the 6-th layer (in total 64 filters) in the fine-tuned CompCars ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $14,939$ test images. From the top row to the bottom are for the 1-th, 9-th, 13-th, 14-th, 15-th, 18-th, 49-th and 51-th filters. 70

3.29 Patches with highest activations for several filters in the 6-th layer (in total 64 filters) in the fine-tuned birds ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $5,794$ test images. From the top row to the bottom are for the 3-th, 5-th, 16-th, 29-th, 48-th, 57-th, 59-th and 60-th filters. 71

3.30 Patches with highest activations for several filters in the 69-th layer (in total 512 filters) in the fine-tuned CompCars ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $14,939$ test images. From the top row to the bottom are for the 42-th, 55-th, 79-th, 86-th, 107-th, 204-th, 283-th and 446-th filters. . . . 72

3.31 Patches with highest activations for several filters in the 69-th layer (in total 512 filters) in the fine-tuned birds ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $5,794$ test images. From the top row to the bottom are for the 24-th, 40-th, 43-th, 50-th, 92-th, 229-th, 311-th and 465-th filters. 73
3.32 Visualizing 64 Conv1 feature maps in CompCars ResNet101-SWP. 74
3.33 Visualizing 64 feature maps of the 6-th layer (convolution) in CompCars ResNet101-SWP. 75
3.34 Visualizing 128 feature maps of the 18-th layer (convolution) in CompCars ResNet101-SWP. 76
3.35 Visualizing 256 feature maps of the 69-th layer (convolution) in CompCars ResNet101-SWP. 77
3.36 Visualizing 512 feature maps of the 99-th layer (convolution) in CompCars ResNet101-SWP. 78

Chapter 1

Introduction

1.1 Overview of fine-grained Visual Recognition

Fine-grained recognition tasks such as recognizing the models of cars or aircrafts, or the species of birds are very challenging due to the subtle differences among categories. For example, apart from length, other aspects of Boeing 737-400 and 737-500 are almost identical. In addition, the pose and viewpoint of objects, or the cluttered background makes the task even more challenging. For example, even human being cannot easily distinguish the birds standing on the branch from the background in some images from the CUB-200-2011 birds dataset [1] (see Fig. 1.1). We can also observe from Fig. 1.2 that although two SUV models are similar in the side view, but are very different in the front view. Meanwhile, the number of training and testing samples in fine-grained image classification datasets are relatively small compared with generic visual recognition datasets. This makes the task more difficult. However, fine-grained visual recognition is a very significant area in computer vision and has a wide range of applications. For instance, the authors of [6] proposed a new and interesting problem named fine-grained image search. They introduced a baseline system based on fine-grained classification scores to represent and co-index images so that the semantic attributes are better incorporated in the online querying stage, which means better fine-grained classification performance can produce more promising search results that contain more semantically matched or similar images. Fig. 1.1 also shows some examples from other three fine-grained datasets on which we carry out a range of experiments with our proposed pooling method.

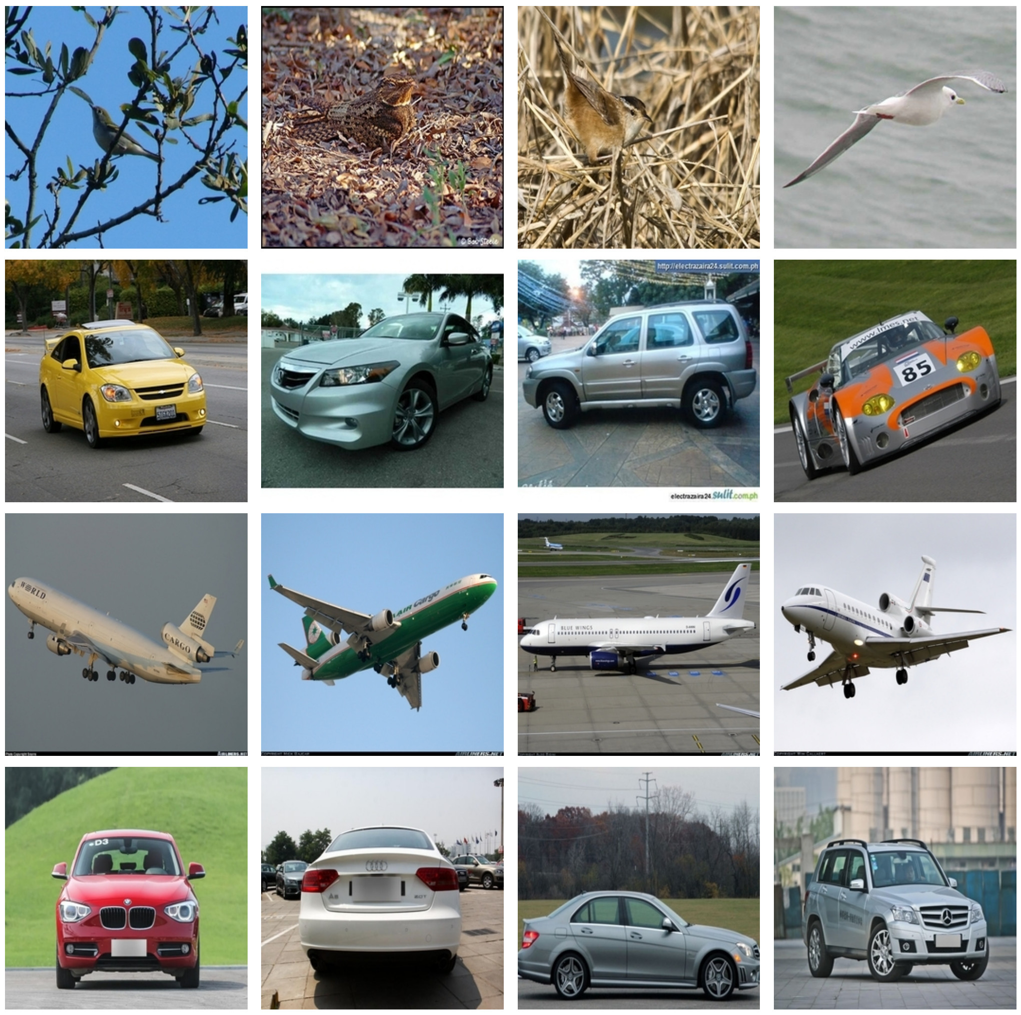

Figure 1.1: Examples from (first row) the birds dataset [1], (second row) the Stanford-Cars dataset [2], (third row) the Aircraft dataset [3], and (last row) the CompCars dataset [4] used in our experiments.

Compared to generic object recognition, the critical parts or regions of objects, such as the heads of birds or the logos of cars that can help discriminate among different categories [7, 8, 9, 10, 11], play a more significant role in the tasks of fine-grained visual recognition. Some works [12, 13] require part annotations to train models in a supervised manner. However, annotating parts is dramatically more expensive than collecting image labels. Thus the work of [14] proposed a method that is based on generating parts using co-segmentation and alignment. They achieved an accuracy of 82.0% without bounding box annotations on the CUB-200-2011 birds dataset [1].

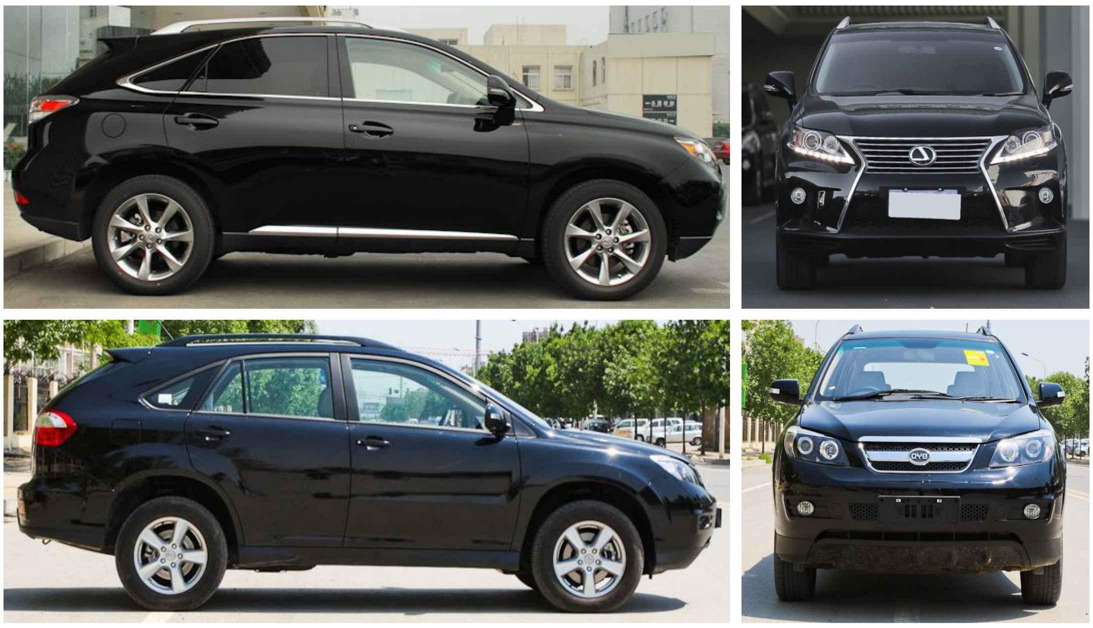
Figure 1.2: Two SUV models are similar in side view, but are very different in the front views.

Early works also made efforts to localize various parts of the objects and train the part detectors with hand-crafted features in a supervised manner. Other methods attempted to use more robust image representations that pool local features in a sophisticated way. For example, VLAD [15] or Fisher vector [16] with SIFT local features. Recently, convolutional neural networks (CNNs) [17, 18, 19] have shown breakthrough performance on various tasks in computer vision such as object detection, localization and recognition. Recent efforts [20, 21, 22, 23] with adapting or utilizing CNNs have been devoted to solving fine-grained object recognitions. For instance, the authors of [22] proposed a DeepBag CNN model to recognize handbag models. The proposed model is also called feature selective joint classification-regression CNN model. It performed favourably for recognizing handbags with 94.48% in accuracy on their newly built branded handbag dataset. Based on the recent work of Spatial Pyramid Pooling [24] in deep convolutional networks (SPP-net), the authors of [23] proposed a task-driven progressive part localization (TPPL) approach for fine-grained object recognition. They took the predicted boxes of a part-based SPP-net as an initial guess and progressively search for more discriminative image regions. The proposed approach is an iterative manner to progressively improve the joint part detection and object classification performance. They achieved an accuracy of 81.69% without bounding box annotations on the CUB-200-2011 birds dataset.

Initially, the studies in generic visual recognition like [25] use the feature vectors of the fully-connected layers in a CNN (FC-CNN) to replace hand-crafted local features such as SIFT. FC-CNN can be considered an object descriptor of a specific region or an entire image as it captures the overall shape of the object contained in the region or the image. Later, cross-convolutional-layer pooling [20] advocates that compared to fully-connected (FC) layer activations, convolutional layer activations can be turned into a more powerful image representation by pooling the extracted features of one convolutional layer with the guidance of feature maps of successive convolutional layer. By pooling the features of two consecutive convolutional layers, they achieved better performance than FC-CNN image representations. Next, the authors of [26] proposed FV-CNN, which is obtained by Fisher Vector pooling of a CNN filter bank. Fisher Vector [16] pools local features densely within the described regions and removes the global spatial information of the regions. FV-CNN is simply computed on the output of a single convolutional layer rather than two convolutional layer as [20]. Here the output feature of a convolutional layer takes the role of local features.

Based on these studies, the work of [21] proposed Bilinear CNN models for fine-grained visual recognition. Bilinear CNN generates bilinear vectors by multiplying two feature vectors of convolutional layers extracted from two different CNNs. It can model local pairwise feature interactions in a translationally invariant manner. They achieved the accuracy of 84.1% without bounding box annotations on the CUB-200-2011 birds dataset. The work of [27] employs the recent Class Activation Mapping (CAM) method [28] to shift the center of attention to increasingly discriminative regions. By alternating stages of classification and re-examination of informative image regions, they obtained an accuracy of 84.48% without bounding box annotations on CUB-200-2011. Recently, the authors of [29] obtained state-of-the-art top-1 accuracy of 92.8% without using any annotated training data, merely training on publicly-available noisy web image search results.

Cross-convolutional-layer pooling [20] and Bilinear CNN [21] take the fine-tuned model trained on a specific dataset as the feature extractor and then pool the extracted features into image-level feature descriptors. Next, the descriptors pass through a linear or non-linear classifier (e.g., SVM) to perform final image classification. The final image descriptors are generally in a very high dimension (e.g., 512×512). This leads to the requirement of dimensionality reduction such as PCA to alleivate the burden of computation and memory cost. In addition, they achieved the best performance on fairly large size of the input image (e.g., 448×448). Bilinear CNN [21] combines the feature extraction stage and bilinear pooling stage together and allows end-to-end training of two CNNs.

Inspired by [20] and [21], we propose a simple yet effective pooling strategy that works on the output features of CNN layers with global spatial information such as the last max pooling layer or the last convolutional layer. Both the methods in [20] and [21] extract local features from one convolutional layer and obtain pooling feature channels from another convolutional layer (for example, 512 feature channels for VGG). Local features are pooled with one pooling channel to form one vector. By concatenating all vectors for all feature maps, we obtain final image-level representations. In the proposed pooling method, we learn a pre-defined number of spatial encoding masks or pooling channels. We show that this spatially weighted pooling significantly improves fine-grained object recognition. Our proposed pooling strategy and all experiments based on it are described in great detail in chapter three.

1.2 Overview of contributions

Our work involves one important topic in the vision community - fine-grained visual recognition. Here, we propose a simple yet effective pooling strategy inspired by cross-convolutional-layer pooling and Bilinear CNN model. Our main contributions to fine-grained visual recognition are as follows.

1. We propose Spatially Weighted Pooling (SWP) that can be seamlessly inte-

grated into existing convolutional neural network (CNN) architectures such as the deep residual network (ResNet). It also allows end-to-end training with image labels only. SWP has few parameters to learn, usually in several hundreds, therefore does not introduce much computational overhead. Testing one image can be done simply by going through the trained or fine-tuned models into which an SWP layer is integrated. It has shown significant capability to improve fine-grained visual recognition performance by simply adding it before fully-connected layers in any off-the-shelf deep convolutional networks.

2. We have conducted comprehensive experiments on three widely-used fine-grained datasets with a variety of deep CNN architectures such as AlexNet, VGGNet and ResNet. We have achieved state-of-the-art results on three fine-grained datasets and the MIT67 indoor scene recognition dataset. With ResNet152-SWP models, we obtain 85.2% on the CUB-200-2011 birds without bounding-box annotations and 87.4% with bounding-box.

1.3 Outline

This thesis will process as follows:

Chapter 2: Deep learning techniques for fine-grained visual recognition. This chapter will cover some background knowledge on both deep learning and fine-grained visual recognition. Specifically, we will first introduce convolutional neural networks in Section 2.1. The basic structure, composition and the training procedure of CNNs will be reviewed. Meanwhile, We will give outlines of three widely used CNNs architectures - AlexNet, VGGNet and ResNet. Finally, two significant studies based on CNNs for fine-grained visual recognition will be introduced.

Chapter 3: Spatially Weighted Pooling in deep CNNs for fine-grained visual recognition. In this chapter, we will present Spatially Weighted Pooling strategy for fine-grained visual recognition. Specifically, we will first report our cross-convolutional-layer pooling experiments on the CompCars dataset in Section 3.1. The experiments indicate that pooling the most distinguishable regions like car logos and headlights areas in the indicator maps, which have higher activations, with the local features in the same regions can achieve better results than those by pooling the whole indicator maps with the corresponding local features. Therefore, we conjecture that if having more powerful indicator maps or pooling channels that can better highlight these distinguishable regions rather than those off-the-shelf pooling channels from one convolutional layer, better performance may be achieved.

Based on above hypothesis, next we propose our pooling strategy in Section 3.2. SWP learns a predefined number of weighted pooling channels or spatial encoding masks that can aggregate local convolutional feature maps with learned spatial importance information and produce more discriminative features. Then we will illustrate the process of forward and backward propagation of an SWP layer, which is similar to that of a convolutional layer.

We will report all experimental results of the proposed method and discussion in Section 3.3. Firstly, three fine-grained image recognition datasets - birds [1], Stanford-Cars [2], aircrafts [3] and one indoor scene recognition dataset - MIT67 [5] are introduced as all experiments are conducted on above four datasets. Secondly, the experiments will illustrate that compared to baseline CNNs structures, CNN architectures with SWP consistently achieve better performance. Furthermore, the state-of-the-art results on all datasets are achieved with ResNet152-SWP fine-tuned on the image size of 293×293 and the crop size of 256×256. Thirdly, we will plot training and validation errors curves of some experiments for all training epochs. Finally, we will show some sample test images that are mistakenly predicted as another model for the birds and CompCars datasets and visualize learned Conv1 and SWP filters. Meanwhile, image patches with highest activations for several convolutional filters and a number of convolutional feature maps will be showed.

Chapter 2

Deep learning techniques for fine-grained visual recognition

2.1 Convolutional Neural Networks

2.1.1 Overview of CNNs

For decades, conventional machine-learning techniques require careful engineering and considerable domain expertise to design a feature extractor that can transform the raw data into an internal representation or feature vector, and then classifiers could be applied on these learned representations or vectors. However, the manual choice of the feature extraction algorithm and the features to classify is often empirical and therefore sub-optimal. A possible solution would be to apply multi-layer feed-forward Neural Networks (NN) on the input data and let the training algorithm find the best feature extractors by adjusting the weights accordingly. Before the prevalence of CNNs, NN has been proved to be a very powerful machine learning technique as they can be trained to approximate complex non-linear functions from high-dimensional inputs.

The problem with NN is that when the input dimension is high, the number of connections is also high because each hidden unit would be fully connected to the input layer. This means the number of free parameters is also high. Convolutional Neural Networks are an approach that tries to alleviate the above mentioned problems. The approach utilizes the principle of weight sharing which can drastically reduces the number of model parameters. Thus this will increase their generalization capacity and reduce the risk of over-fitting.

The authors of [30] indicate that there are two underlying reasons behind the principle of weight sharing. Firstly, local groups of values are usually highly correlated in image patches. Secondly, the local statistics of images are invariant to location. In other words, if a local image pattern appear in one part of the image, it could appear anywhere. Therefore, units at different locations can share the same weights and detect the same pattern in different parts of the image. The role of the convolutional layer in a CNN is to detect local feature conjunctions from the previous layer.

The first implementation of a CNNs was *Neocognitron*, which is proposed in [32, 33, 34, 35]. It has been originally applied to the problem of handwritten digit recognition. The Neocognitron makes use of receptive fields in which each neuron is only connected to a sub-region corresponding to a certain number of neighbouring neurons in the preceding layer. A significant breakthrough of CNNs in the early 1990s came with the widespread use of Back-propagation learning algorithm. In 1990, LeCun *et al.* [36] trained the first CNNs with Back-propagation to solve the problem of handwritten digit recognition.

As long as one module's functions with respect to its inputs and its internal weights are relatively smooth, one can compute gradients using the backpropagation procedure. Then multilayer architectures like CNNs can be trained by simple *Stochastic Gradient Descent*. The goal of the backpropagation algorithm is to compute the gradient of an objective function with respect to the parameters in CNNs. As the decision function $h(x)$ of the neural network is a function of functions, we need to use the chain rule to compute its gradient. The backpropagation algorithm is indeed an implementation of the chain rule specifically designed for neural networks. It can be applied repeatedly to propagate gradients through all modules, starting from the output to the input. Once these gradients have been computed, it is straightforward to compute the gradients with respect to the weights in CNNs.

Stochastic gradient descent (SGD) works according to the same principles as standard gradient descent, but proceeds more quickly by estimating the gradient from just a few examples at a time instead of the entire training set. The use of SGD in the neural network setting is motivated by the high cost of running back propagation over the full training set. SGD can overcome this cost and still lead to fast convergence. The standard gradient descent algorithm updates the parameters θ of the objective $J(\theta)$ as,

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)] \tag{2.1}$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. Stochastic Gradient Descent simply does away with the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by,

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{2.2}$$

with a pair $(x^{(i)}, y^{(i)})$ from the training set. Generally each parameter update in SGD is computed with respect to a few training examples or a mini-batch rather than a single example. The reason for this is twofold: first this reduces the variance in the parameter update and can lead to more stable convergence, second this allows the computation to take advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient. A typical mini-batch size is 256 in training ImageNet, although the optimal size of the minibatch can vary for different applications and architectures.

The learning rate α in SGD is typically much smaller than a corresponding learning rate in standard gradient descent because there is much more variance in the update. Choosing the proper learning rate and schedule (i.e., decreasing the value of the learning rate as learning progresses) can be very difficult. One standard method in practice is to use a small enough constant learning rate that gives stable convergence in the initial epochs (one epoch means one full pass through the training set) and then halve the value of the learning rate as convergence slows down. During SGD training, if the training data is given in some meaningful order (e.g., all images in one mini-batch belong to the same category), this can bias the gradient and lead to poor convergence. Therefore, the training data are required to be randomly shuffled prior to training.

If the objective has the form of a long shallow ravine, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains. Momentum is one method for pushing the objective more quickly away the shallow ravine. The momentum update is given by,

$$v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{2.3}$$

$$\theta = \theta - v \tag{2.4}$$

where v is the current velocity vector which is of the same dimension as the parameter vector θ. Generally the momentum γ is set to 0.9.

CNN was widely applied into various fields in the early 1900s (e.g., time-delay neural networks for speech recognition [37] and document reading [38]). Since the early 2000s, it has been successfully applied to image segmentation, object detection and recognition, particularly in the tasks of the segmentation of biological images [39], face detection and recognition, pedestrian and human body detection [40, 41, 42, 43, 44, 45]. In these research areas, labelled data were becoming relatively abundant. Despite these successes, CNNs were largely forsaken by the mainstream computer-vision and machine-learning communities until the ImageNet competition in 2012 [46]. Krizhevsky *et al.* applied a deep convolutional neural network (AlexNet) to the ImageNet dataset, which consists of about a million images from $1,000$ different categories and achieved spectacular results in the competition. This success came from the efficient use of GPUs, a non-linearity activation function called Rectified Linear Unit (ReLU), a new regularization technique called dropout [47], and effective data augmentation techniques. The success has precipitated the rapid adoption of deep learning by the computer-vision community. Now CNNs are the dominant approach for almost all recognition and detection tasks [48, 45, 49, 50, 51, 18].

CNNs are designed to process data in the form of multiple arrays, for instance a colour image that contains three colour channels, and each channel is a 2D arrays containing pixel intensities. The architecture of a typical CNN is structured as a series of stages. The first few stages are usually composed of two types of layers: convolutional layers and pooling layers. After the convolution operation through a set of weights called a filter bank, we obtain a number of feature maps within which each unit is connected to local patches in the feature maps of the previous layer. Different filter banks produce different feature maps in one convolutional layer. The units in the feature maps are then passed through a non-linearity such as a ReLU. At present, ReLU is the most popular non-linear activation function. It is simply the half-wave rectifier $f(z) = max(z, 0)$. In past decades, smoother non-linearities like $tanh(z)$ or $1/(1 + exp(-z))$ were widely used. But the ReLU has been proved to learn much faster in deep CNNs with many layers.

Different from the role of the convolutional layer that is to detect local conjunctions of features from the previous layer, the role of the pooling layer in CNNs is to merge semantically similar features into one. Average pooling $f_a(v) = \frac{1}{N}\sum_{i=1}^{N} v_i$ and max pooling $f_m(v) = max_i v_i$ are the two most popular pooling approaches. They can reduce the dimension of the feature maps and create an invariance to small distortions and shifts. The authors of [52] argue that max pooling is particularly well suited to the separation of features that are very sparse (i.e., have a very low probability of being active). A typical convolutional neural network is comprised of a number of stages of convolution, ReLU non-linearity (often with a max pooling step after each stage) and then followed by one or more fully-connected layers as in a standard multilayer neural network.

Deep CNNs with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. In order to remedy this issue, Srivastava *et al.* [47] propose a technique called *Dropout*, which can prevent units from co-adapting too much. The key idea is to randomly drop units from the CNNs training. This significantly reduces over-fitting and gives major improvements over other regularization approaches. At the present, Dropout is commonly used in fully-connected layers since the parameters in these layers generally account for the vast majority of the total parameters in a CNN like AlexNet. Finally, a softmax layer is added on top of the fully-connected layer to produce final prediction output for image recognition.

Recent deep CNNs have an enhanced modelling capacity as the model depth becomes increasingly deep. The authors of [53] indicate that they are particularly dependent on the availability of large quantities of training data in order to learn a non-linear function from input to output that generalizes well and yields high classification accuracy on unseen data. A possible explanation for the limited exploration of CNNs and the difficulty to improve on simpler models is the relative scarcity of labelled data for fine-grained visual recognition. An elegant solution to this problem is data augmentation. It is the application of one or more deformations to a collection of annotated training samples which result in new, additional training data.

There are many ways to do data augmentation, such as the popular horizontally flipping, random crops and color jittering. In 2012, Krizhevsky *et al.* [17] proposed fancy PCA when training the famous AlexNet. Fancy PCA alters the intensities of the RGB channels in training images. They perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, they add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1.

The second form of data augmentation in their work consists of generating image translations and horizontal reflections. They do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training the network on these extracted patches. This increases the size of the training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, the network suffers from substantial over-fitting. At test time, the network makes a prediction by extracting five 224×224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the networks softmax layer on the ten patches. We call this approach of image crop "random-crop". In our all AlexNet and VGGNet-related experiments, we employ the random-crop method to do data augmentation.

GoogLeNet [48] proposed one image sampling method which is inspired by [54]. They sample various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area with aspect ratio constrained to the interval [3/4, 4/3]. We call this data augmentation technique "random-sized-crop". In our all ResNet-related experiments, we use the random-sized-crop method to do training sample augmentation.

Nowadays, many state-of-the-arts deep networks models like AlexNet, VGGNet and ResNet are released publicly by famous research groups to facilitate further research. These models are all trained on ImageNet, which consists of more than one million training samples with about $1,000$ categories. Thanks to the wonderful generalization abilities of these pre-trained deep models, we can fine-tune them on our relatively small-scale fine-grained image classification datasets. Different strategies of fine-tuning can be utilized in different situations. For instance, a good case is that your new data set is very similar to the data used for training these pre-trained models. In this case, one can only fine-tune a few top layers of pre-trained models with a small learning rate. We consider our fine-grained datasets like birds, cars and aircraft are quite different from the data used in above pre-trained models for generic image classification, therefore, we organize all our experiments by fine-turning the entire pre-trained models, not just a few layers, with a small learning rate for improving performance.

2.1.2 Batch Normalization

With the convolutional neural network depth increasing, training the network on large-scale image datasets like ImageNet is becoming more and more complicated and time-consuming. For instance, it takes approximately two weeks to train a VGG model with 16-layer on ImageNet with more than one million training images when running at a Tesla K40c GPU. The authors of [55] indicate that the training process is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training process by requiring smaller learning rates and careful parameter initialization. They refer to this phenomenon as *internal covariate shift*. Batch Normalization addresses the problem by normalizing layer inputs, which allows us to use much higher learning rates and be less careful about parameter initialization.

Batch Normalization [55] draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. It also acts as a regularizer, in some cases eliminating the need for Dropout. By applying Batch Normalization to some state-of-the-art image classification models, it achieves the same accuracy with 14 times fewer training steps, and beats the original models by a significant margin. In our all ResNet-related experiments, we also adopt batch normalization in all weight layers including all convolutional layers, fully-connected layers and the layer of our proposed pooling method, which is introduced in Section 3.2.

2.1.3 AlexNet, VGGNet and ResNet

In next two sections, we will briefly review a few well-known CNN architectures as well as two recent studies based on them for fine-grained image recognition. AlexNet [17] consists of five convolutional and three FC layers with a final 1000-way softmax. It has more than 60 million parameters and $650,000$ neurons. Convolutional layer one, two and five are followed by a max-pooling layer. To make training faster, they used non-saturating neurons (ReLU Nonlinearity) and a very efficient GPU implementation of the convolution operation. Dropout [47] is used in the first two FC layers in order to alleviate overfitting. Apart from Dropout, they also employed a couple of data augmentation skills to reduce overfitting. The first form of data augmentation consists of generating image translations and horizontal reflections, and the second one is to alter the intensities of the RGB channels in training images. It has been proved that these two forms of data augmentation are considerably effective, and they are widely used in training deep CNN models. It also employed local response normalization that aids generalization in the first and second convolutional layers. It achieved the best classification accuracy in the 2012 ImageNet competition. OverFeat [50] is an well-known integrated CNN framework for classification, localization and detection. Like AlexNet, OverFeat also has a total of eight layers. But its architecture has two versions - *fast* and *accurate*. For the architecture for *fast* version, there are five convolutional and three FC layers. Its *accurate* model consists of six convolutional layers and two FC layers.

In the 2014 ImageNet competition, VGGNet [18] obtained the best performance with a conventional CNN architecture with substantially increased depth (e.g., 16-19 layers). It employs a relatively small 3×3 receptive field with a fixed stride 1. As very small (3×3) convolution filters are used in all convolutional layer, this allows them to address another important aspect of CNN architecture design - its depth. To this end, they fixed other parameters of the architecture, steadily increased the depth of the network by adding more convolutional layers. Five different convolutional neural network configurations are set up with an increasing depth from 11 to 19. In addition to small receptive fields, VGGNet also removes the layer of local response normalization, which plays an important role in improving performance for AlexNet. It has the same structures as AlexNet in the last three FC layers. Dropout still plays a significant role in reducing ovefitting for the first two fully-connected layers with 4096 dimensions.

They have demonstrated that convolutional networks with substantially increased depth (e.g., 16 weights layers as CNN config. D and 19 weights layers as config. E) can achieve astounding performance for large-scale image classification, and the representation depth is beneficial for the classification accuracy. However, in the paper [18] one can observe that the trained model E with 19 weight layers performs not better than the trained model D with 16 weight layers, and even worse in some cases. There is no discussion about this phenomenon in their work, and this has revealed some indications about the facts that simply adding more convolutional layers does not necessarily mean better performance. This has also been proved by the Deep Residual Network (ResNet) [19] in their experiments with more deeper convolutional networks (e.g., 20-layer, 34-layer and 56-layer).

Recently, the Deep Residual Network (ResNet) [19] made a forceful impression in the computer-vision and machine-learning communities, for it increases the depth of CNNs to more than 100 and achieved state-of-the-art performance on recognition, detection, localization and segmentation tasks. The depth of representations is of central importance for many visual recognition tasks, but deeper neural networks become more difficult to train. The authors carried on experiments on CIFAR-10 [56] with 20-layer and 56-layer networks. They discovered that the deeper network has higher training error and test error. Similar phenomena also occurred while training 18-layer and 34-layer networks on ImageNet. They [19] argue that when deeper networks are able to start converging, a *degradation* problem has been exposed. It is that with the network depth increasing, accuracy gets saturated and then degrades rapidly. The problem is not caused by overfitting, and adding more layers leads to higher training error. To overcome the problem of degradation in training very deep CNNs, ResNet adopts residual learning to every few stacked layers by adding shortcut connections. Coupled with a good weights initialization method [57] and batch normalization [55], ResNet attains better performance with fewer parameters and lower complexity compared to VGGNet as it removes the first two FC layers and dropout as well.

2.2 Cross-convolutional-layer pooling

Inspired by the parts-based pooling strategy [10] used in fine-grained image classification, cross-convolutional-layer pooling [20] employs feature maps of the $(t+1)$-th convolutional layers as D_{t+1} indicator maps. The local features extracted from the t-th convolutional layer are pooled with D_{t+1} pooling channels to obtain final image representations. The method is based on the observation that a feature map of a deep convolutional layer is usually sparse and contains some regions which are semantically meaningful. Cross-convolutional-layer pooling [20] argues that although the filter of a convolutional layer is usually not directly task-relevant compared to the parts detector learned from human-specified parts annotations, cross-convolutional-layer pooling can benefit from combining a much larger number of indicator maps, e.g., 512 for VGG as opposed to 20-30 (the number of parts usually defined by human). It is analogous to applying bagging to improve the performance of multiple weak classifiers. Most previous studies adopts activations of the fully-connected layer of a deep CNN as the image representation and it is believed that convolutional layer activations are less discriminative. But cross-convolutional-layer pooling has illustrated that if used appropriately, convolutional layer activations can be turned into a powerful image representation which enjoys many benefits over fully-connected layer activations. They achieved comparable or in some cases significantly better performance than existing fully-connected layer based image representation on a number of different types of datasets, including a fine-grained image classification dataset -The CUB-200-2011

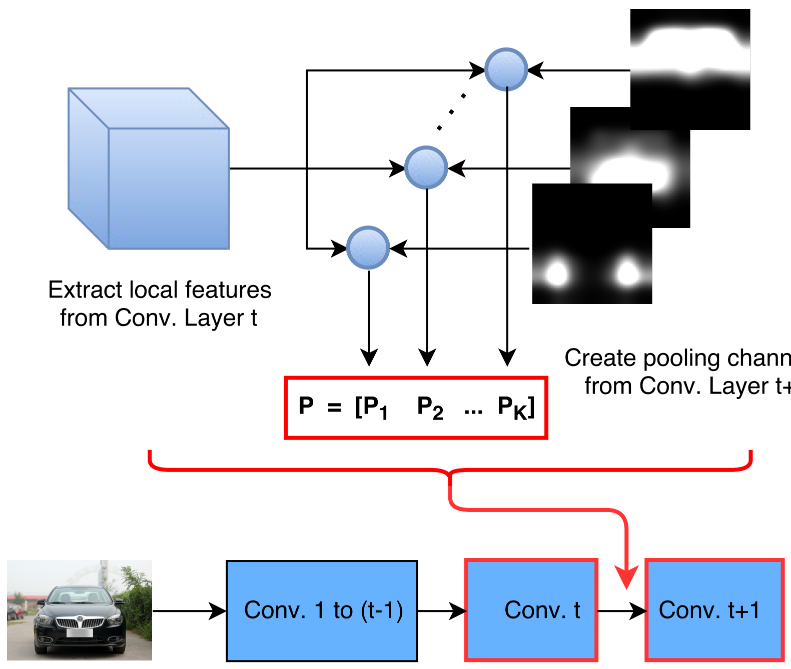
Extract local features from Conv. Layer t
Create pooling channels from Conv. Layer t+1
P = [P1 P2 ... PK]
Conv. 1 to (t-1)
Conv. t
Conv. t+1

Figure 2.1: **The overview of cross-convolutional-layer pooling.** Pooling local features extracted from Conv. Layer t with pooling channels from Conv. Layer t+1.

dataset [1].

Formally, the image representation extracted from cross-convolutional-layer pooling can be expressed as follows:

$$\begin{aligned} P^t &= [{P_1^t}^T, {P_2^t}^T, ..., {P_k^t}^T, ..., {P_{D_{t+1}}^t}^T]^T \\ where, \quad P_k^t &= \sum_{i=1}^{N_t} X_i^t a_{i,k}^{t+1}, \end{aligned} \tag{2.5}$$

where P^t denotes the pooled feature for the $t-$th convolutional layer, which is calculated by concatenating the pooled features of each pooling channel $P_k^t, k = 1, ..., D_{t+1}$. X_i^t denotes the i-th local feature in the t-th convolutional layer. Note that feature maps of the $(t+1)$-th convolutional layer are obtained by convolving the feature maps of the t-th convolutional layer with a $m \times n$-sized kernel. So if we extract local features X_i^t from each $m \times n$ spatial unit in the t-th convolutional layer then each X_i^t naturally corresponds to a spatial unit in the $(t+1)$-th convolutional layer. Let us denote the feature vector in this spatial unit as a_i^{t+1} and the value at its k-th dimension as $a_{i,k}^{t+1}$. Then we use $a_{i,k}^{t+1}$ to weight local feature X_i^t in the k-th pooling channel. See Fig. 2.1 for an overview of cross-convolutional-layer pooling.

Cross-convolutional-layer pooling is inspired by the parts-based pooling strategy. Recent work of [31] is also based on object parts. In their work, the basic idea is to represent an object by a collection of parts arranged in a deformable configuration. The appearance of each part is modeled separately, and the deformable configuration is represented by spring-like connections between pairs of parts. These models allow for qualitative descriptions of visual appearance, and are also suitable for generic recognition problems. Although the above two methods are both based on object parts, but there is a big difference between them. Cross-convolutional-layer pooling is a novel pooling strategy based on convolutional features, whereas [31] is a method based on deformable parts configuration.

2.3 Bilinear CNN model

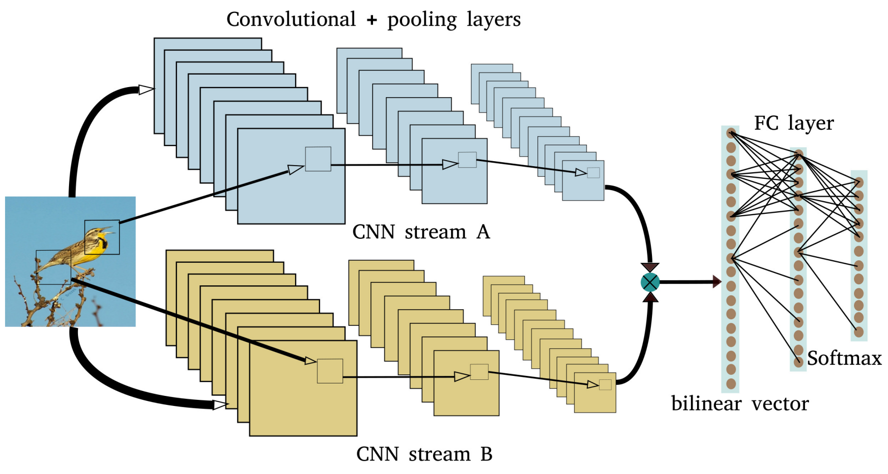

Figure 2.2: **The overview of Bilinear CNN model.** Bilinear CNN architecture consists of two feature extractors whose outputs are multiplied using outer product at each location of the image.

Bilinear CNN models [21] can be seen as a generalization of the cross-convolutional-layer pooling using separate CNNs. Cross-convolutional-layer pooling extracts local features and indicator maps from two consecutive convolutional layers in one CNN model, whereas bilinear CNN architecture consists of two feature extractors whose outputs are multiplied using outer product at each location of the image. See Fig. 2.2 for an overview of Bilinear CNN model. At the stage of training, the generated bilinear vector passes through FC layers and a softmax layer to obtain predictions. This allows a bilinear CNN architecture to do end-to-end training or fine-tuning, which plays a critical role in boosting the final classification accuracy. Bilinear CNN models [21] can model local pairwise feature interactions in a translationally invariant manner which is particularly useful for fine-grained categorization.

One significant difference between bilinear CNN models and cross-convolutional-layer pooling is that the former is a complete CNN architecture which consist of two CNN streams (only contain convolutional and pooling layers), a bilinear vector generation layer and other fully-connected and softmax layers, whereas the latter is only a pooling method that utilizes existing CNN models to generate image representations by extracting local features and indicator maps from two consecutive convolutional layers. The whole architecture of bilinear CNN models can be fine-tuned on a specific domain using two CNN streams initialized from the ImageNet dataset. Both the resulting bilinear vector and the image representation of cross-convolutional-layer pooling are passed through a signed square root step ($y \leftarrow sign(x)\sqrt{|x|}$), followed by ℓ_2 normalization ($z \leftarrow y/||y||_2$). After that, they are trained on one-vs.-all linear SVMs for the final image category predictions.

Chapter 3

Spatially Weighted Pooling in deep CNNs for fine-grained visual recognition

In this chapter, we will elaborate on our proposed Spatially Weighted Pooling strategy for fine-grained visual recognition. We will first report our cross-convolutional-layer pooling experiments on the CompCars dataset in Section 3.1. The authors of [20] have evaluated the proposed cross-convolutional-layer pooling method on four datasets: MIT indoor scene-67 [5], Caltech-UCSD Birds-200-2011 [1], PASCAL VOC 2007 [58] and H3D Human Attributes dataset [59]. Here we are interested in evaluating its applicability and effectiveness on a entirely different fine-grained dataset - CompCars [4]. Based on these experiments, we introduce the proposed Spatially Weighted Pooling method in Section 3.2. To verify its effectiveness, we carried out a range of SWP-related experiments on three fine-grained image classification datasets and the MIT67 dataset.

We consider integrating the proposed pooling strategy into three widely used convolutional neural network architectures - AlexNet, VGGNet and ResNet. Firstly, We take these CNN architectures in which it is a max pooling layer before the first fully-connected layer as the baselines. Next, the aforementioned max pooling is replaced with an SWP layer. Experimental results will show that the modified architectures with SWP not only always perform better than the baselines, but also achieve the state-of-the-art results on all datasets using ResNet152-SWP. Finally, we will visualize learned Conv1 and SWP filters, image patches with highest activations for several convolutional filters and a number of convolutional feature maps to gain a deep insight into the learned models.

3.1 Cross-convolutional-layer pooling experiments on CompCars

3.1.1 CompCars dataset

The CompCars dataset [4] contains images from *web-nature* and *surveillance-nature*. The images of the web-nature are collected from web-based forums, public websites and search engines. The web-nature data contains 163 car makers with $1,716$ car models. There are $136,727$ images in total capturing the entire cars. Fig. 3.1 illustrates some examples of the web-nature images. The car models can be organized into a large tree structure that consists of three levels. They are car make, car model and the year of manufacture from top to bottom. We can observe from Fig. 3.2 that there is subtle difference in appearance for the cars belonging to the same model but produced in different years. For instance, three versions of "Audi A4L" were manufactured between 2009 to 2011 respectively.

Viewpoint	F	R	S	FS	RS
No. in total	4974	3791	5722	9434	7034
No. per model	11.5	8.8	13.3	21.9	16.3

Table 3.1: Quantity distribution for five viewpoints.

For the fine-grained classification task, the authors of [4] only use $30,955$ of a total of $136,727$ web-nature images and classify them into 431 car models. For each car model, the cars manufactured in different years are classified into the same category. Each car image is also labelled as one of five viewpoints, including front (F), rear (R), side (S), front-side (FS) and rear-side (RS). The quantity distribution of five viewpoints for a total of $30,955$ images is shown in Table 3.1 and the examples from five viewpoints are shown in Fig. 3.1. There are in total $16,016$ training images and $14,939$ testing images for the task of fine-grained classification. Note that as an extension to their CVPR paper, the authors of [4] conducted experiments for fine-grained car classification, attribute prediction, and car verification with the entire dataset and different deep models. For fine-grained car classification, they achieved much higher accuracies with a different train/test splits from the original one. There are in total $36,456$ training images and $15,627$ testing images in the new splits, which means training samples are twice as ours.

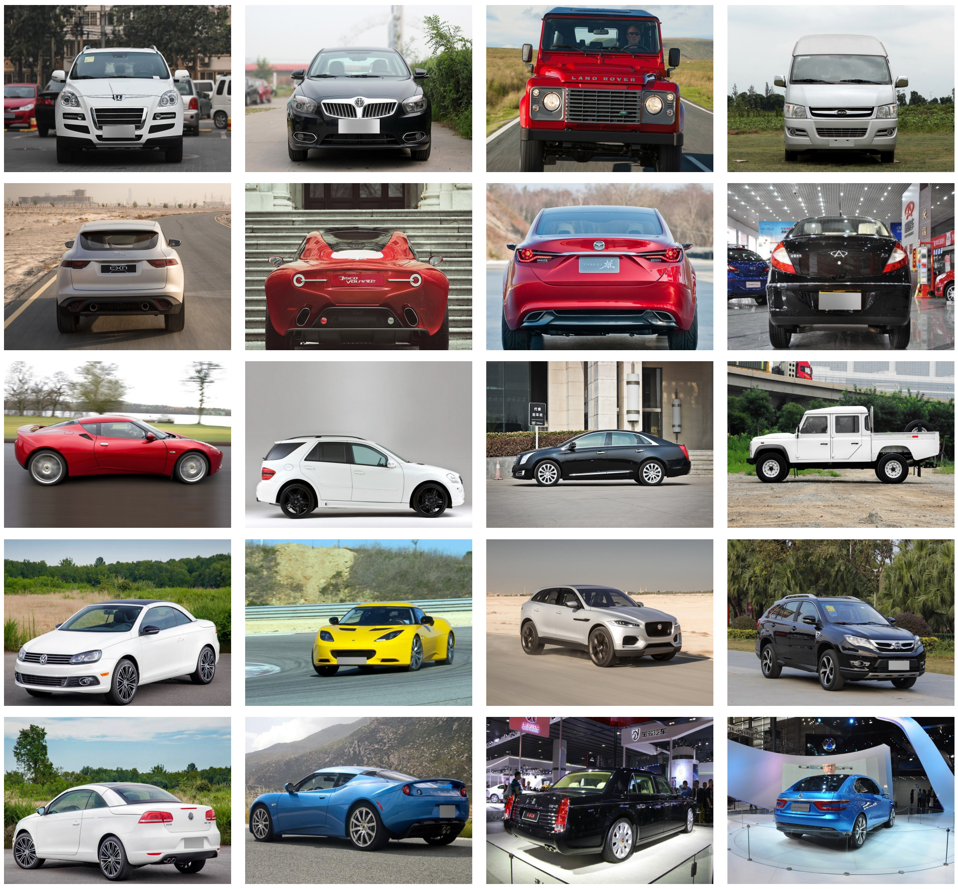

Figure 3.1: Examples from the CompCars dataset [4]. The images in the five rows illustrate front viewpoint (F), rear viewpoint (R), side viewpoint (S), front-side viewpoint (FS) and rear-side viewpoint (RS).

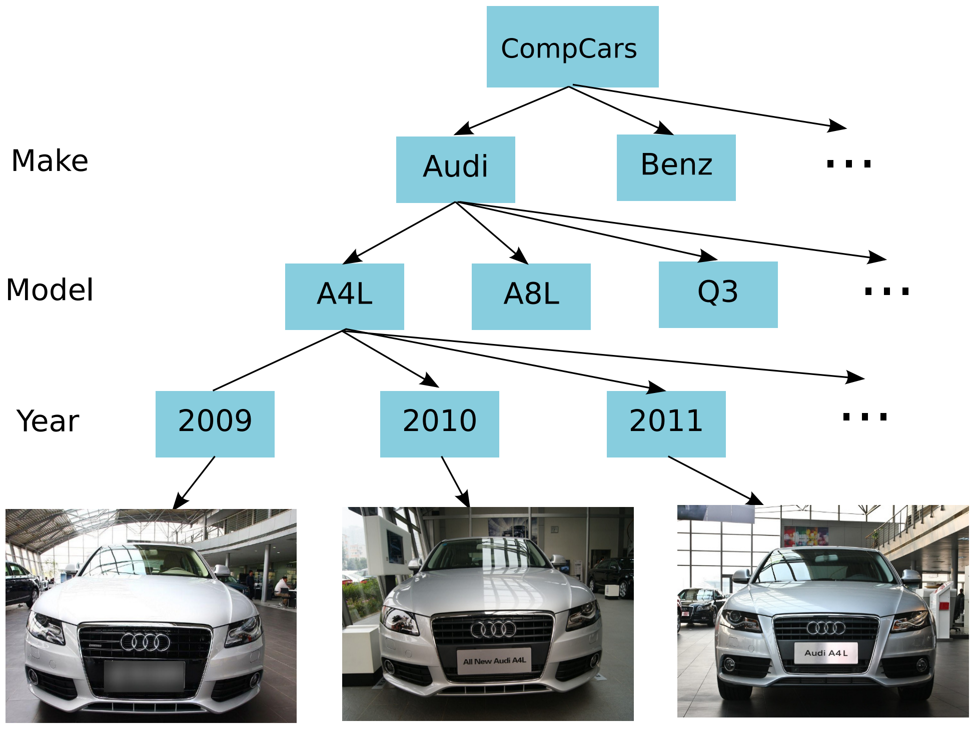

Figure 3.2: The tree structure of car model hierarchy. Several car models of Audi A4L in different years are also displayed.

3.1.2 Experimental settings

The authors of [20] evaluated the proposed cross-convolutional-layer pooling method on four datasets: MIT indoor scene-67 [5], Caltech-UCSD Birds-200-2011 [1], PASCAL VOC 2007 [58] and H3D Human Attributes dataset [59]. Previous studies [25, 60] have shown that activations from the fully-connected layer of a pre-trained deep CNN achieve surprisingly good performance on those datasets. Taking the experimental results of [25, 60] as baselines, the authors of [20] has demonstrated that cross-convolutional-layer pooling can achieve or in some cases significantly better performance by pooling local features extracted from one convolutional layer with the guidance of the feature maps of the successive convolutional layer. They conducted their experiments only on one fine-grained dataset - birds [1]. In order to investigate the effectiveness of their proposed method, we carried out some experiments on another fine-grained dataset - CompCars [4] for the cross-convolutional-layer pooling.

The authors of [4] fine-tuned the OverFeat [50] model, which is pre-trained on ImageNet classification task [46], on the CompCars dataset. The corresponding classification results for five viewpoints and All-View are shown in Table 3.2. In terms of model depth, VGG is much deeper than OverFeat, and it has shown superb performance on ImageNet and many other datasets. Therefore, we consider fine-tuning this more widely-used pre-trained VGG model to obtain initial results that can be used to make a comparison with the results in Table 3.2. Next, we can take these initial experimental results as our baselines to investigate the performance of cross-convolutional-layer pooling when it works on the VGG models. In the initial works of cross-convolutional-layer pooling, all experiments are based on Alex models rather than VGG models. Therefore, we can further investigate the effectiveness of cross-convolutional-layer pooling when it works on deeper CNN models.

Viewpoint	F	R	S	FS	RS	All-View
Top-1 [4]	52.4	43.1	42.8	56.3	59.8	76.7
Top-5 [4]	74.8	64.7	60.2	76.9	77.7	91.7

Table 3.2: Fine-grained classification results reported by the authors of CompCars [4].

We organized our experiments on the CompCars dataset into four parts. Firstly, we conducted experiments on extracting local features from different layers after fine-tuning the VGG model (e.g., conv53, pool5 and fc6). These local features serve as final image representations and are trained with a linear or non-linear classifier (e.g., SVM). To fine-tune the VGG model on the CompCars dataset, we simply replace the ImageNet specific $1,000$-way final classification layer with a randomly initialized 431-way classification layer, and the rest of the layers keep unchanged. We report the results of first category in Table 3.4. Secondly, we experimented with different combinations of convolutional layers for cross-convolutional-layer pooling (e.g., conv42&conv43, conv52&conv53). The results are illustrated in Table 3.6. Thirdly, we compared the classification accuracy with fine-tuning and without fine-tuning for cross-convolutional-layer pooling. For the experiments without domain fine-tune on the CompCars dataset, we directly utilize the off-the-shelf VGG models pre-trained on ImageNet to extract local features and pooling channels to perform cross-convolutional-layer pooling for the car images. The corresponding experimental results are shown in Table 3.7. Finally, we performed experiments on the CompCars dataset with our proposed method of Spatially Weighted Pooling in Section 3.2, and the results are shown in Table 3.14.

3.1.3 Experimental details

For the experiments without fine-tuning, we adopt the pre-trained VGG model provided in the caffe [61] package to extract CNN activations. To fine-tune the pre-trained VGG model on the CompCars dataset, we replace the original $1,000$-way classification layer with a new 431-way layer specific for CompCars. It is initialized by random weights drawn from Gaussian distributions with fixed standard deviations of 0.01 and trained from scratch with a large learning rate. Other layers are trained from the learned distributions on ImageNet with a relatively small learning rate. We fix the input size of training images at 256×256 when fine-tuning the models on CompCars. We do not use scale and color augmentation during the training process. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted.

After the completion of fine-tuning the VGG model on the car dataset, we carried out all experiments in relation to cross-convolutional-layer pooling by following [20]. But we take the image resolution of 342×342 at testing, and a 300×300 center crop is sampled from a testing image. Next, the center image crops will pass through the fine-tuned VGG model. After that, we can readily extract local features from the different layers in the fine-tuned VGG net (e.g., conv53 and FC6). Once these local features are available, one can perform the cross-convolutional-layer pooling on them to obtain image representations or directly carry out final image classification based on them.

The image representations of cross-convolutional-layer pooling pass through a signed square root step ($y \leftarrow sign(x)\sqrt{|x|}$), followed by ℓ_2 normalization ($z \leftarrow y/||y||_2$). After that, they are trained on one-vs.-all linear SVMs for the final image category predictions. If we take the local features extracted from one specific layer directly as the image representations, it also goes through the same process as the image representations of cross-convolutional-layer pooling does. For the experiments based on the local features extracted from the fully-connected layers, the resolution must be the same for training and testing images. But if utilizing the local features extracted from convolutional layers, we may use different image resolutions. For instance, we fine-tune the VGG model with the training images of 256×256, but perform experiments for cross-convolutional-layer pooling based on the local features extracted from conv53 and conv54 on the testing images of 342×342.

3.1.4 Experimental results and discussions

Viewpoint	F	R	S	FS	RS	All-View
OverFeat fine-tuning [4]	52.4	43.1	42.8	56.3	59.8	76.7
AlexNet fine-tuning	50.2	41.3	40.9	55.1	58.3	75.6
VGGNet fine-tuning	**77.2**	**79.1**	**69.1**	**79.8**	**81.1**	**90.8**

Table 3.3: Comparison of classification accuracy on CompCars between OverFeat fine-tuning [4] and VGG fine-tuning.

Table 3.3 compares the fine-tuned results for the OverFeat model reported in [4] with the results by fine-tuning the Alex and VGG models. All results in the table come from counting the number of correctly classified images, and the final classification result for one test image is to identify which dimension has the largest value in the output of the softmax layer in the model. It is easy to observe that fine-tuning the VGG model achieves much better performance than the OverFeat model does. AlexNet models obtained similar results in comparison with the OverFeat models. Compared to AlexNet and OverFeat with 8 layers in total, VGGNet has a total of 16 layers in depth and is much deeper. Next, we will take these fine-tuned results as our baselines to compare them with a number of experiments based on the cross-convolutional-layer pooling.

After completing the fine-tuning process, we can easily extract local features from one specific layer and take them as image representations to be trained on a classifier. We apply SVM on these image representations. Since, the FC8 layer of the fine-tuned VGG model can directly predict the car model class, there is no need to perform SVM on FC8. Firstly, we carried out the experiment on the first fully-connected layer (FC6). The extracted local features from FC6 has 4096 dimensions, and the input size of testing images is 256×256. To extract local features from one fully-connected layer using the fine-tuned VGG model, we have to maintain the same resolution for training and testing images. Table 3.4 shows that classification accuracy of FC6 (the fourth row) is lower than that of VGGNet fine-tuning, which is our baseline (the last row). However, it is not necessary to maintain the same resolutions for training and testing images if local features are extracted from the layers before the fully-connected. This is clearly evident in the case of Convolutional and Pooling layers because their forward function is independent of the input volume spatial size. For example, the size of the feature maps in conv53 for a 224×224 image crop is 14×14. If we obtain an image crop with the size of 304×304 by resizing testing images, the size of the feature maps in conv53 will be 19×19 (see the first row in Table 3.4).

Viewpoint	F	R	S	FS	RS	All-View
Conv53 ($19 \times 19 \times 512$)	68.1	69.7	59.7	70.5	71.7	79.8
Pool5 ($9 \times 9 \times 512$)	75.5	77.9	67.8	78.3	79.4	87.9
FC6 (4096)	76.3	78.2	68.4	78.9	80.2	88.7
FC7 (4096)	76.6	78.7	68.8	79.2	80.4	89.2
VGGNet fine-tuning (FC8)	**77.2**	**79.1**	**69.1**	**79.8**	**81.1**	**90.8**

Table 3.4: Comparison of classification accuracy for different layers of the fine-tuned VGG model on CompCars.

Some works like [24, 62] also achieved similar experimental results to ours. The work of SPP-net [24] also extracts the representations from the images in the target datasets - Pascal VOC 2007 [58] and re-train SVM classifiers. The extracted image representations are l_2-normalized for SVM training. From the results in Table 3.5, one can easily observe that fc7 performs better than all other layers before it. The layer of fc6 achieves better results than conv4 and conv5. The authors of [62] fine-tuned the pre-trained model of AlexNet on two-class weather dataset [63]. They also performed similar experiments to test classification accuracies for different layers (e.g., conv4, conv5 and fc6). The results they obtained are consistent with those in Table 3.5. As far as image classification concerned, the later layers like fc6 and fc7 in a deep CNN perform much better than the earlier layers like conv3 and conv4. In order to gain a better understanding of CNNs, some studies attempted to dig out underlying explanations of the previous observation by visualizing them. The work of [64] indicates that the earlier features of a deep CNN contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the deep CNN becomes progressively more specific to the details of the classes contained in specific datasets.

Layer	conv4	conv5	pool5	fc6	fc7
Classification accuracy [24]	59.96	66.34	69.14	74.86	75.90

Table 3.5: Comparison of classification accuracy on VOC2007 for different layers in [24].

Next, we carried on experiments based on cross-convolutional-layer pooling using above fine-tuned VGG models on CompCars. After taking local features extracted from one layer like conv53 as indicator maps, we pool these indicator maps with local features extracted from its previous layer (conv52) and obtain cross-convolutional-layer pooling features. Then the cross-convolutional-layer pooling features are passed through a signed square root step ($y \leftarrow sign(x)\sqrt{|x|}$), followed by ℓ_2 normalization ($z \leftarrow y/||y||_2$). After that, they are trained on one-vs.-all linear SVMs for the final image category predictions. Table 3.6 illustrates corresponding results for different layer combinations

Viewpoint	F	R	S	FS	RS	All-View
OverFeat fine-tuning [4]	52.4	43.1	42.8	56.3	59.8	76.7
VGGNet fine-tuning	77.2	79.1	69.1	79.8	81.1	90.8
conv42&conv43	80.9	83.6	73.7	84.4	84.9	92.9
conv52&conv53	**84.3**	**87.4**	**79.4**	**88.8**	**89.5**	**94.6**

Table 3.6: Comparison of classification accuracy on CompCars for different layer combinations of cross-convolutional-layer pooling.

Table 3.6 shows that cross-convolutional-layer pooling improves classification accuracy dramatically compared to our VGGNet fine-tuning baseline. For instance, we obtain 84.3% accuracy for F-View by pooling conv52 with conv53 in comparison with the baseline's 77.2%. Furthermore, the pooled features of conv52&conv53 achieve much better performance than those of conv42&conv43. All above experiments are based on the fine-tuned CompCars VGG model. We also conducted experiments without domain fine-tune for cross-convolutional-layer pooling and report the results in Table 3.7. Without domain fine-tune on CompCars, we obtained slightly worse results for cross-convolutional-layer pooling. Some studies like [25, 21, 20, 62, 26, 24, 65] also reported consistent results on other image classification datasets.

Viewpoint	F	R	S	FS	RS	All-View
conv52&conv53 without fine-tune	78.9	82.2	74.7	82.8	84.1	90.2
conv52&conv53 with fine-tune	**84.3**	**87.4**	**79.4**	**88.8**	**89.5**	**94.6**

Table 3.7: Comparison of classification accuracy on CompCars with or without domain fine-tune for cross-convolutional-layer pooling.

We have mentioned that the method of cross-convolution-layer pooling is inspired by the parts-based pooling strategy used in fine-grained image classification in Section 2.2. In the works of parts-based strategy like [10], multiple regions-of-interest (ROI) are detected and each of them corresponds to specific parts (e.g., the tail of birds and the logo of cars). Next, local features falling into each ROI are pooled together to get image level representations. By accumulating 512 feature maps of conv53 in the fine-tuned CompCars VGG model, we get the left image in Fig. 3.3. One can easily observe that the significant car body parts like the logos and the headlights have highest activations in feature maps of conv53. In other words, the regions near the logos and the headlights are ROI regions. Indeed in terms of distinguishing one car model from the other, they are more discriminative than other regions.

In original cross-convolutional-layer pooling method, suppose that one local feature map extracted from one convolutional layer have the size of $H \times W$, the pooling channel extracted from next convolutional layer has the same size of $H \times W$, where H, W denote the height and width of one local feature map or one pooling channel. The pooling process between the local features and one pooling channel is performed at all $H \times W$ locations. This means that all regions are equally treated and the concept of ROI is not taken into consideration in the pooling process. Moreover, a feature map or a pooling channel is usually sparse (see Fig. 3.4). If the process of pooling is performed at all $H \times W$ locations, we conjecture that this will not only lead to extra computational overhead, but also impair the discriminative ability of pooled features.

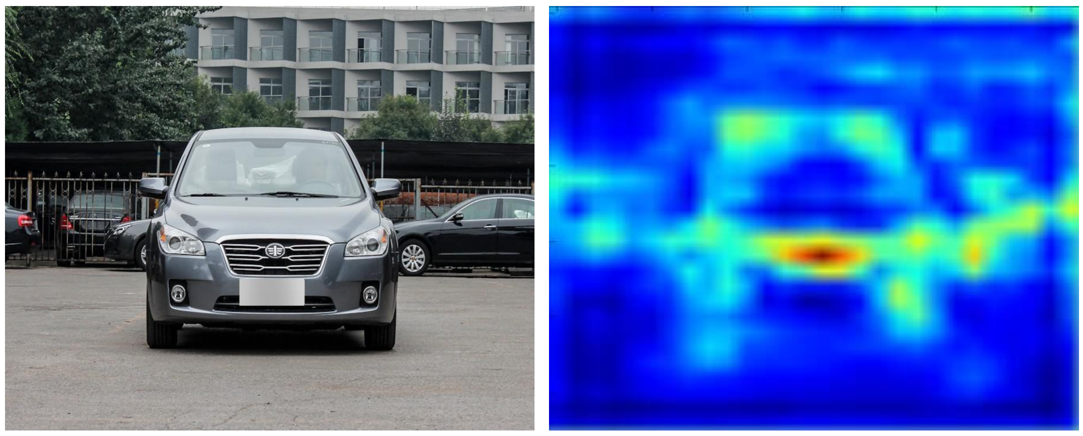

Figure 3.3: Visualizing of ROI regions by accumulating 512 feature maps of conv53 in the fine-tuned CompCars VGG model. Right: the original image. Left: the indicator map.

Next, we conducted some experiments based on the concept of ROI regions. Instead of taking whole Conv52 local features ($19 \times 19 \times 512$) and whole Conv53 indicator feature maps (19×19 for one map), we use fewer points with high activations in indicator maps, as most discriminative car parts like car Logos and headlights have higher activation response. For instance, we take $12 \times 12 = 144$ locations with highest activations in all indicator maps and pooled with corresponding local features ($12 \times 12 \times 512$). The experimental results are reported in Table 3.8. From the table, one can see that the best result of 85.98% is achieved when the top $12 \times 12 = 144$ locations with higher activations are chosen. Even though the cross-convolutional-layer pooling is performed on only 36 locations with highest activations, we can still achieve the same performance as that of the whole indicator map in which there are $19 \times 19 = 381$ points. The results in Table 3.8 indicate that pooling the most distinguishable regions like car logos and headlights areas, which have higher activations, in the indicator maps with the local features in the same regions can achieve better results than those by pooling the whole indicator maps with the corresponding local features. Therefore, we conjecture that if having more powerful indicator maps or pooling channels that can better highlight these distinguishable regions rather than straightforward using those off-the-shelf pooling channels from one convolutional layer, better performance may be achieved.

Based on above hypothesis, we propose a new pooling strategy in which a dozen of pooling channels or masks are learned to replace the roles of extracted local convolutional pooling channels. After carrying out above experiments on CompCars with the cross-convolutional-layer pooling, we have proved that the cross-convolutional-layer pooling strategy also works well on this fine-grained car dataset. In the meantime, we also have gained a deeper understanding of the method. Based on above works and inspired by cross-convolutional-layer pooling, we proposed a simple yet effective pooling strategy called Spatially Weighted Pooling that can improve fine-grained classification performance significantly. We elaborate on the proposed pooling strategy and corresponding experiments in Section 3.2.

Number of Points in indicator maps	F-View
$19 \times 19 = 381$ (whole feature map)	84.28
$17 \times 17 = 289$	85.09
$15 \times 15 = 225$	85.22
$13 \times 13 = 169$	85.55
$12 \times 12 = 144$	**85.98**
$11 \times 11 = 121$	85.93
$9 \times 9 = 81$	85.09
$7 \times 7 = 49$	85.01
$6 \times 6 = 36$	84.29

Table 3.8: Comparison of classification accuracy on CompCars for different number of interesting points in indicator maps. The cross-convolutional-layer pooling is performed between conv53 and conv54 layers for the fine-tuned CompCars VGG model.

3.2 The proposed method

3.2.1 Spatially weighted pooling

Cross-convolutional-layer pooling [20] and bilinear CNN [21] possess a number of similarities in the process of training and testing. For instance, they both extract local features and pooling channels from different convolutional layers, which pass through similar pooling processes. Besides, they both use linear SVMs as classifiers to train the pooled image representation or the bilinear vector, and both obtain the best performance on a large size of input images. Among them, the most significant one is that they use a similar pooling strategy to generate image representations or bilinear vectors. They both encode the local features extracted from one convolutional layer with indicator maps or pooling channels extracted from another convolutional layer. Cross-convolutional-layer pooling extracts the pooling channels and the local features from the adjacent convolutional layers in a single CNN, whereas bilinear CNN employs two CNNs to extract them separately.

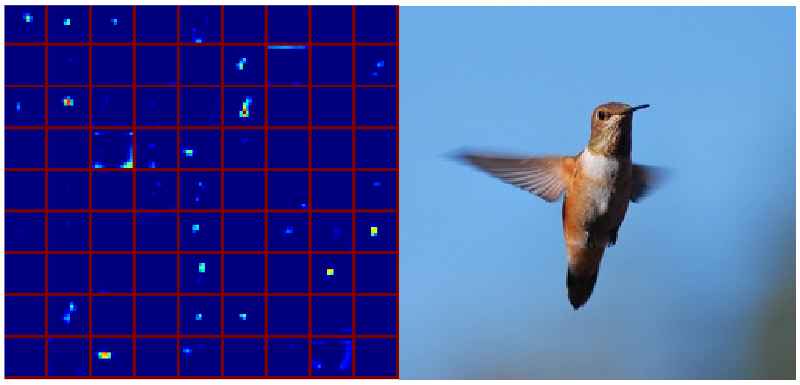

Figure 3.4: The first 81 of total 256 feature maps extracted from the last convolutional layer in AlexNet (left), and the corresponding input image (right).

Suppose that the local features extracted from one convolutional layer have the size of $H \times W \times D$, where H, W denote the height and width of each local feature map and D denotes the number of local feature maps. The pooling channel extracted from another convolutional layer has the same size of $H \times W$, the pooling process between the local features and one pooling channel can be seen as a linear combination of all local features at all $H \times W$ locations to form one D-dimensional pooled feature. If there are 512 pooling channels (e.g., the last convolutional layer in VGG), we can obtain 512 linear combinations of local features. Concatenating all 512 pooled D-dimensional feature produces a $512 \times D$ dimensional image representation or a bilinear vector.

A feature map or a pooling channel is usually sparse (see Fig. 3.4) and indicates some semantically meaningful regions (see the first row in Fig. 3.5). Ad-

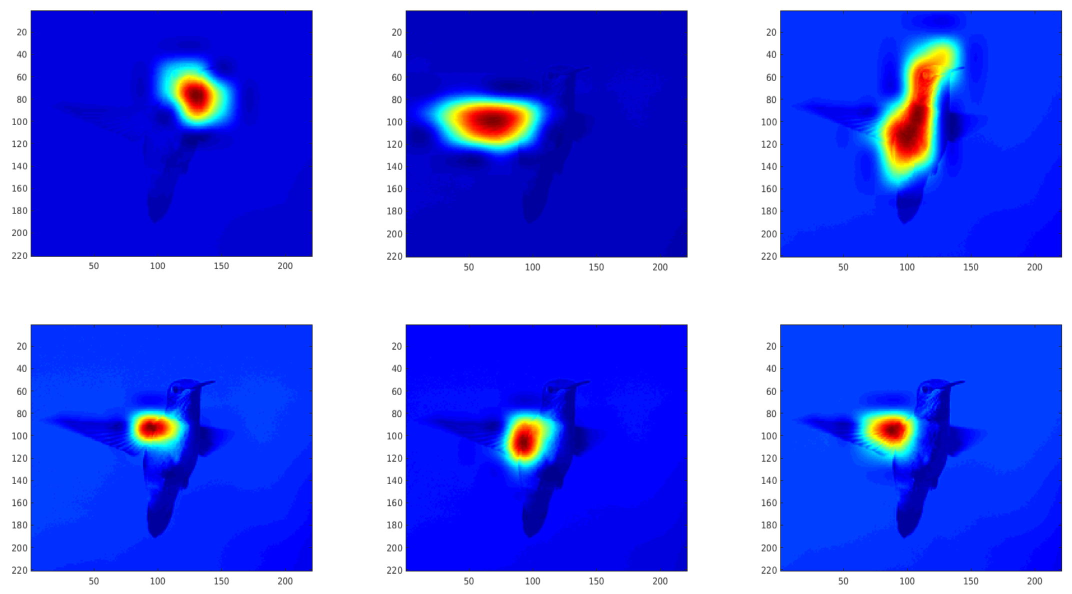
Figure 3.5: Visualizing of some feature maps extracted from the last convolutional layer in AlexNet. Three feature maps in the first row indicate some semantically meaningful regions and three feature maps in the second row illustrate the characteristic of similarity.

ditionally, a large number of feature maps in one convolutional layer are similar to each other (see the second row in Fig. 3.5). The characteristic of sparsity in one feature map and similarity among many feature maps may result in the dimension redundancy of the pooled image representation or the bilinear vector. Compact bilinear pooling [66] derived through a novel kernelized analysis has proved that with only a few thousand dimensions, it can maintain the same discrimination power as the full bilinear representations with several hundreds thousand dimensions.

Instead of using off-the-shelf feature maps extracted from one convolutional layer as pooling channels to do linear combinations or encodings for local features extracted from another convolutional layer, we argue that a number of spatial encoding maps or masks can be learned from the end-to-end training process to replace these off-the-shelf feature maps or pooling channels. Therefore, we propose the spatially weighted pooling (SWP) method that can improve classification performance by adding it after either the last convolutional layer for

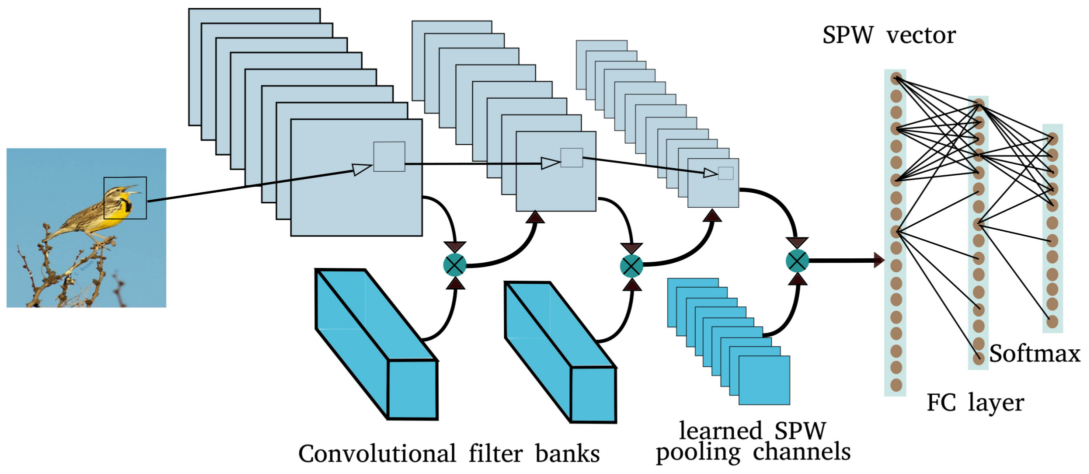

Figure 3.6: **The overview of the proposed method.** An SWP layer is added after the last convolutional layer or the last max pooling layer and before the first FC layer.

ResNet or the last max pooling layer for AlexNet and VGGNet. Then the pooled SWP features pass through one or several FC layers and a softmax layer to get the final image category predictions. In an SWP layer, we learn a predefined number of weighted pooling channels or spatial encoding masks that can aggregate local convolutional feature maps with learned spatial importance information and produce more discriminative features (see Fig. 3.6 for an overview of the proposed method and see Fig. 3.7 for an more detailed illustration). Formally, SWP can be expressed as follows:

$$\begin{aligned} P &= [{P_1}^T, {P_2}^T, ..., {P_k}^T, ..., {P_d}^T]^T \\ where, \quad P_k &= \sum_{i=1}^{H \times W} f_i w_i^k, \end{aligned} \tag{3.1}$$

where P denotes the pooled SWP feature for the last convolutional layer or the last max pooling layer, which is calculated by concatenating the pooled feature of each learned pooling channel $P_k, k = 1, ..., d$ and d refers to the number of learned pooling channels or spatial masks. f_i denotes the i-th D dimensional feature in the size of $H \times W$ (D is the number of feature maps). w_i^k denotes the i-th learned

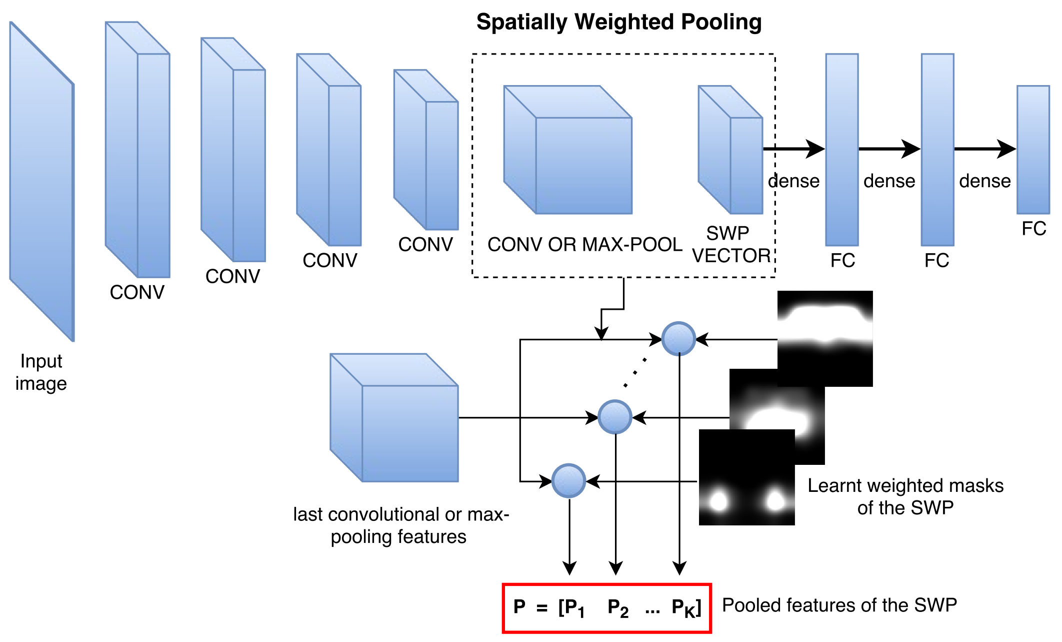
Spatially Weighted Pooling
Input image
CONV
CONV
CONV
CONV
CONV OR MAX-POOL
SWP VECTOR
dense
dense
dense
FC
FC
FC
last convolutional or max-pooling features
Learnt weighted masks of the SWP
P = [P1 P2 ... PK]
Pooled features of the SWP

Figure 3.7: **The illustration of SWP in more detail.** An SWP layer is added after the last convolutional layer or the last max pooling layer and before the first FC layer.

weight in the k-th pooling channel or spatial mask of the same size of $H \times W$.

By comparing Fig. 2.1 with Fig. 3.7, one can observe that the proposed SWP is very similar to cross-convolutional-layer pooling, for they both employ nearly identical pooling strategy. However, the latter differs from the former in one significant fact that SWP learns a predefined number of pooling channels, but cross-convolutional-layer pooling takes extracted local features from one convolutional layer as pooling channels. This allows SWP to be seamlessly integrated into any existing CNN architectures (e.g., VGGNet and ResNet). We can effortlessly apply an SWP layer to those deep CNNs with some slight modifications of their structures. The new CNN architectures with SWP can be end-to-end trained from scratch or fine-tuned from some pre-trained models. Furthermore, the number of the learned SWP pooling channels (16 for VGGNet-SWP and ResNet-SWP) is much smaller than that of the off-the-shelf pooling channels (the numbers are 512 and 2048 in the last convolutional layer for VGGNet and ResNet50-152 respectively). This makes the output features of an SWP layer have

16×512 dimensions, whereas the bilinear vector or the cross-convolutional-layer pooling features have 512×512 dimensions for VGGNet, which means the vector dimension of SWP is only 3% of that of Bilinear CNN models. In the meantime, one can imagine that it is quite hard to apply cross-convolutional-layer and Bilinear CNN models to ResNet because ResNet has not only very high dimensions in the last convolutional layer (2048 dimensions), but also different number of feature channels for any two successive convolutional layers (512 dimensions for the second-to-last convolutional layer and 2048 for the last convolutional layer)

3.2.2 Forward and backward propagation of SWP

The process of forward and backward propagation of SWP is similar to that of a convolutional layer. For example, the input of a convolutional layer or an SWP layer can be formulated as a tensor of size $H \times W \times D$, where H, W denote the height and width of each feature map and D denotes the number of feature maps. The learned pooling channels or spatial masks of an SWP layer can be formulated as a tensor of the size $H \times W \times d$, where d denotes the number of pooling channels. The learned pooling channels and the input feature maps of an SWP layer have the same height and width (e.g., 7×7 for VGG), whereas one learned filter of convolutional layers usually has a relatively small size (e.g., 3×3 receptive field for VGG). Forward propagation of SWP can be formulated as follows:

$$f = w^T x \tag{3.2}$$

where w^T denotes the transpose of learned SWP channels with the size of $d \times H \times W$. x denotes the input of the SWP layer with the size of $H \times W \times D$. After multiplication of two tensors, the pooled SWP features have $d \times D$ dimensions.

Backward propagation of SWP can be calculated by following the chain rule of back-propagation. To compute gradients with respect to input data x and learned weights w, we have:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial f} * \frac{\partial f}{\partial x} = \frac{\partial u}{\partial f} w \tag{3.3}$$

$$\frac{\partial u}{\partial w} = \frac{\partial u}{\partial f} * \frac{\partial f}{\partial w} = \frac{\partial u}{\partial f} x \tag{3.4}$$

where $\partial u/\partial f$ denotes the computed gradient of previous layer with respect to the output of the SWP layer f. The SWP layer output f is also the input of that layer. We do not use bias term b in the SWP layer.

The output features of an SWP layer have $d \times D$ dimensions, whereas the bilinear vector or the cross-convolutional-layer pooling features have $D \times D$ dimension. The number of local feature maps D is 512 in VGGNet and 2048 in ResNet50-152 respectively. Here d is much smaller than D. In the following experiments, we achieved the best performance on all datasets by setting the number of SWP channels d to 16 for ResNet and VGGNet and 9 for AlexNet. In AlexNet and VGGNet architectures, an SWP layer is added between the last max pooling layer and the first FC layer. The feature map size after the max pooling layer is 6×6 for AlexNet and 7×7 for VGGNet. Then it is further reduced to 3×3 by the SWP layer if d is equal to 9 ($3 \times 3 = 9$ for AlexNet). It means that by applying the SWP layer before the FC6 layer, we can reduce the parameters of the FC6 layer significantly. For the SWP layer, it only has several hundreds of weights to learn(e.g., $6 \times 6 \times 9$ for AlexNet architecture). In AlexNet and VGGNet, the parameters of three FC layers account for the majority of the model parameters and FC6 also has much more parameters compared with the other two FC layers. Therefore, the modified MAX-SWP-FC architecture has much fewer parameters to learn compared to the original MAX-FC architecture.

3.3 Experimental results of the proposed method and discussions

3.3.1 Datasets

We evaluate the proposed method on three fine-grained image recognition datasets - birds [1], Stanford-Cars [2], aircrafts [3] and one indoor scene recognition dataset - MIT67 [5]. Birds are relatively smaller than aircrafts and cars in the images. Cars and birds appear in a more cluttered way compared with aircrafts. Aircrafts has few viewpoints in comparison with cars and birds. Fig. 3.8 shows some examples for the four datasets.

The CUB-200-2011 dataset [1]: It contains 11,788 images of 200 bird species. There are 5,994 training images and 5,794 testing images in total. The dataset also provides bounding-box annotations for both training and testing. We evaluate our methods in two protocols - "birds" where the object bounding-box is not provided both at training and testing phase, and "birds + box" where the bounding-box is provided for both stages. A large number of images in the dataset are not aligned in terms of the proportion and location of bird objects in the images.

The Stanford Cars Dataset [2]: The Cars dataset contains 16,185 images of 196 classes of cars. It is split into 8,144 training images and 8,041 testing images. The dataset also provides bounding-box annotations for both training and testing. We evaluate our methods in two protocols as well - "cars" where the object bounding-box is not provided both at training and testing phase, and "cars + box" where the bounding-box is provided for both stages. Different from the CUB-200-2011, the car objects in [2] usually occupy one entire image.

The FGVC-aircraft Dataset [3]: The Aircraft dataset consists of 10,000 images of 100 aircraft variants. It is split into 6,667 training images and 3,333 testing images. Aircraft models are organized in a four-levels hierarchy. The four levels, from finer to coarser, are Model, Variant, Family and Manufacturer. Since certain models are nearly visually indistinguishable, the "model" level is not used in the evaluation. A variant collapses all the models that are visually indistinguishable into one class. The Aircraft dataset has fewer viewpoints compared to other datasets. The majority of planes are horizontal in the images.

The MIT67 Indoor Scene Recognition [5]: The MIT67 dataset contains 67 indoor scene categories, for example, the scene in libraries and kitchens. There are 5,353 training images and 1,338 testing images. Unlike other three fine-grained objects datasets in which one image usually contains one object in a well-aligned position, the images in MIT67 usually contain a number of objects in a more cluttered way. No bounding-box annotations are provided in this dataset.

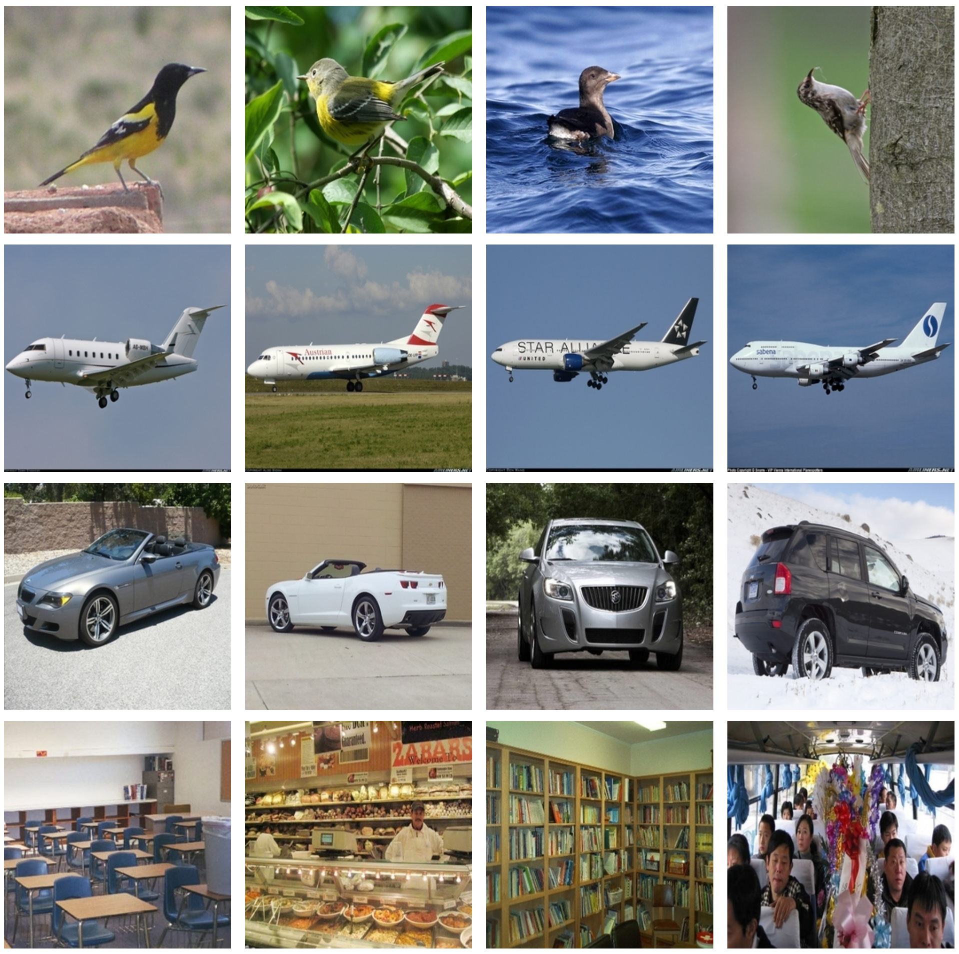

Figure 3.8: Examples from (first row) the birds dataset [1], (second row) aircraft dataset [3], (third row) cars dataset [2], and (last row) the MIT67 dataset [5] used in our experiments.

3.3.2 Experimental settings

We consider three widely-used and well-known CNN architectures - AlexNet, VGGNet and ResNet to illustrate the performance improvement of the proposed method. For the residual network, we also conducted comprehensive experiments on different depth of models such as ResNet-34, ResNet-50, ResNet-101 and ResNet 152.

At first, we replace the 1000-way classification layer trained on the ImageNet dataset with a randomly initialized N-way classification layer, where N is the category number of a specific dataset and train this layer from scratch with a high learning rate. Other layers of the pre-trained models are fine-tuned with a relatively small learning rate. AlexNet and VGGNet have three FC layers after the combinations of a number of convolutional layers, max-pooling layers and other layers (e.g., Local Response Normalization). The first two FC layers of AlexNet and VGGNet models with a dimension of 4096 are followed by ReLU non-linear activations and a drop-out with a rate of 0.5. The above setting is devised to train a CNN model on the ImageNet, which comprises more than one million training images. However, the aforementioned four datasets only contain training images in the order of thousands. To some extent, we have observed the phenomenon of over-fitting with this setting during the training process. Therefore, we reduce the dimension of first two FC layers from 4096 to 512 and keep the drop-out rate of 0.5. This modification can help alleviate over-fitting and achieve better performance for the aircrafts and cars datasets.

ResNet has a different network architecture from AlexNet and VGGNet. Firstly, ResNet only has one FC classification layer after several convolutional blocks and a global average pooling. The global average pooling down-samples all 2048 feature maps from 7×7 to 1×1, and all spatial information is discarded in this process. The output features of the global average pooling with 2048 dimensions are passed through a softmax layer to obtain final image classification. For achieving better classification results, we replace the global average pooling of ResNet with a max-pooling layer and then followed with a 1024 FC layer. Secondly, ResNet employs batch normalization [55] to accelerate the training process and get better performance. With batch normalization, it allows to train a CNN with a high global learning rate and to be less sensitive about initialization. It also acts as a regularizer and eliminates the need for dropout in some cases. In ResNet, all convolutional layers are followed with batch normalization in order to address the phenomenon of internal covariate shift. We similarly add batch normalization after the newly added 1024 FC layer.

In our all AlexNet and VGGNet-related experiments, we do not adopt batch normalization into our modified models because these experiments are all based on fine-tuning the pre-trained AlexNet and VGGNet models, but these pre-trained models were trained prior to the appearance of batch normalization. Nevertheless, all pre-trained ResNet models are trained with batch normalization. Therefore, when fine-tuning the modified ResNet models, such as adding a fully-connected layer or our proposed pooling layer, we have to integrate batch normalization into all newly added weight layers. Otherwise, the modified models cannot be properly trained.

3.3.3 Improved architectures as baselines

In our initial experimental setting, we simply replace pre-trained models' ImageNet-specific 1000-way classification layer and fine-tune them to obtain initial experimental results. However, these revised CNN architectures are not optimal for these relatively small fine-grained datasets. Hence, we reduce the dimension of the first two FC layers from 4096 to 512 in order to alleviate over-fitting. For AlexNet and VGGNet, we keep all convolutional layers, max-pooling layers and other auxiliary layers (e.g., local response normalization) before the FC layers unchanged. For ResNet, all layers before the original global average pooling remain unchanged, and the global average pooling is replaced by a down-sampling max pooling (e.g., down-sampling from 7×7 to 4×4) followed by a 1024 FC layer and the batch normalization.

We take these revised CNN architectures as baselines to illustrate the performance enhancement by integrating an SWP layer into them. For ResNet, we replace the last max pooling layer with an SWP layer and add batch normalization and ReLU activations after it. The SWP layer also reduces the dimension of output of the previous layer from 7×7 to 4×4 for 2048 feature channels as the last max-pooling does exactly. Thus the SWP layer and the last max-pooling layer have the exactly same output features dimension. However, the SWP layer has several hundreds additional parameters to learn compared to the last max-pooling layer without learned parameters. This is also why after the SWP layer, we have to employ batch normalization and ReLU activations. For AlexNet-MAX and VGGNet-MAX, we add an SWP layer after the last max-pooling layer, and the SWP layer further down-samples feature map from 7×7 to 3×3 and 4×4 before the FC 512 layer respectively.

The feature map size of the outputs of the last convolutional layers is different for ResNet, VGGNet and AlexNet (e.g., 14×14 for VGGNet, 13×13 for AlexNet and 7×7 for ResNet). VGGNet is twice as large as that of ResNet. Thus we integrate SWP into these three CNN architectures with two different strategies. After down-sampling feature maps with max pooling from 14×14 to 7×7 for VGGNet or from 13×13 to 6×6 for AlexNet, we add an SWP layer after the max pooling layer. For ResNet, the size of feature maps is already 7×7 and we simply replace the max-pooling layer with an SWP layer and insert a FC layer after the SWP layer. We report experimental results in Table 3.10, which compares classification results with or without SWP. We also compare our best results with the state-of-the-art results in Table 3.11.

3.3.4 Implementation details

All training and testing procedures of ResNet architectures generally follow [19]. We fix training and test image size with 256×256. Then a 224×224 crop is randomly sampled from an image or its horizontal flip. But our process of cropping images is slightly different from that in [19]. We follow the process of training GoogleNet [48], which includes sampling of various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area with aspect ratio constrained to the interval [3/4, 4/3]. Next, we apply color jitter that randomly varies brightness, contrast and saturation of the image with a constant of 0.4, following [19]. Finally, we subtract a mean value and divide by standard deviation for three RGB channels. During the testing, we only subtracted mean values and divided by standard deviation for the image crops. We also adopt batch normalization (BN) [55] after the SWP layer and the first FC layer before ReLU activations. The SWP layer and the following FC layers are initialized by random weights drawn from Gaussian distributions with fixed standard deviations of 0.005 and trained from scratch. We use stochastic gradient descent (SGD) solver with a mini-batch size of 20-60 (varies according to the model size) for all four datasets. The learning rate starts from 0.001 and is divided by 10 after 40 training epochs. The whole training process completes after 90 epoch. We use a weight decay of 0.0005 and a momentum of 0.9.

For fine-tuning AlexNet and VGGNet architectures, we follow the exactly

# weighted masks	AlexNet-SWP	VGG16-SWP	ResNet50-SWP	ResNet101-SWP
$K = 4$	82.9%	89.3%	91.6%	92.9%
$K = 9$	**84.3**%	90.7%	91.8%	93.0%
$K = 16$	83.9%	**91.0**%	**92.1**%	**93.4**%
$K = 25$	83.4%	90.5%	92.0%	93.2%
$K = 36$	82.5%	89.8%	91.9%	92.9%

Table 3.9: Comparison of classification results with different mask numbers on the Stanford Cars-196 dataset.

same implementation as ResNet architectures but with a small learning rate of 0.0005. Additionally, we do not use scale and color augmentation during training and testing. A 227×227 crop for AlexNet or a 224×224 crop for VGGNet is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted. A dropout layer with a ratio of 0.5 is added after the FC layers, and no batch normalization is used in AlexNet and VGG architectures. In testing, we adopt the standard 10-crop testing [17] and report all experiments with a single model at a single scale (images are resized to 256×256). We do not adopt multiple GPU implementation and all experiments run at a Tesla K40c GPU.

3.3.5 Discussion

Before we set up all SWP-related experiments, one important prerequisite is to figure out what the best predefined numbers of SWP pooling channels are for different models. Then we firstly carried out experiments in relation to this predefined number on the Stanford Cars-196 dataset. Table 3.9 shows the comparison of classification results for different mask numbers. For the AlexNet-SWP and VGG16-SWP, one can observe that the accuracies generally improve as we increase the number of weighted masks up to 9. However, setting the number of masks to be too large (*e.g.*, $K = 36$) may hurt the performance. By observing the 36 learnt masks of AlexNet-SWP, we found out that some masks have similar weighted distributions. It means that these masks focus on the same parts of cars and generate redundant pooled features. For ResNets-SWP, the performance is not very sensitive to the number of weighted masks. We conjecture that these deep architectures can learn more semantically meaningful information for each

CNN architectures	birds	birds + box	aircrafts	aircrafts + box	cars	cars + box	MIT67
AlexNet-4096	58.7	70.3	74.1	78.0	68.5	79.4	**63.7**
AlexNet-MAX	58.6	71.2	77.0	82.2	71.3	81.9	58.7
AlexNet-SWP	**62.9**	**72.6**	**77.2**	**82.3**	**76.6**	**84.3**	62.3
VGG16-4096	74.1	80.4	84.2	87.5	84.8	89.2	**74.5**
VGG16-MAX	73.6	80.3	86.1	88.9	86.1	90.2	73.0
VGG16-SWP	**76.4**	**81.2**	**86.3**	**89.0**	**88.5**	**91.0**	73.1
ResNet101-GLOBAL-AVE	82.0	84.8	87.1	89.2	90.4	91.0	79.8
ResNet101-MAX	83.4	86.4	88.3	89.9	91.7	92.9	80.4
ResNet101-SWP	**83.8**	**86.7**	**89.4**	**91.0**	**92.1**	**93.4**	**81.6**

Table 3.10: **Comparison of classification results**. The table compares the classification results of three variants of AlexNet, VGG-16 and ResNet101. VGG16-4096 refers to the VGG-16 model that only changes the last ImageNet specific 1000-way classification layer, and VGG16-MAX is the CNN architecture that further replaces the first two FC 4096 dimension layers with two 512 dimension layers. VGG16-SWP is the model that adds an SWP layer between the last max-pooling layer and the first FC 512 layer of VGG16-MAX. ResNet-GLOBAL-AVE is the ResNet variant that keeps the global average pooling and only changes the last ImageNet specific 1000-way to corresponding N-way cclassification layer, where N is the categories of specific dataset. ResNet-MAX is ResNet architectures that further replace global average with max-pooling (down-sampling feature map from 7×7 to 4×4). Then a 1024 FC layer is added after max-pooling. ResNet-SWP replaces max-pooling with SWP for ResNet-MAX.

mask. We can also observe that the accuracies decline with the N increasing in the AlexNet-SWP and VGGNet-SWP. The large N value may lead to a under-fitting issue since the SWP layer will have more parameters to learn from scratch, but learning samples are relatively insufficient on a small-scale fine-grained dataset. The highest accuracy of 93.4% is achieved by the ResNet101-SWP with setting $K = 16$. The highest accuracy for VGGNet-SWP and AlexNet-SWP are achieved by setting $K = 16$ and $K = 9$ respectively. Therefore, we set the number of SWP masks d to 16 for ResNet and VGGNet and 9 for AlexNet for the rest of our experiments.

We compare the classification results of three variants of AlexNet, VGG-16 and ResNet-101 in Table 3.10. AlexNet-MAX and VGG16-MAX refer to baseline CNN architectures that reduce the number of neurons the first two FC layers from 4096 to 512. We get better results by fine-tuning them on aircrafts and

CNN architectures	birds	birds + box	aircrafts	aircrafts + box	cars	cars + box
FC-CNN[D] [21]	70.4	76.4	74.1	–	79.8	–
FV-CNN[D] [21]	81.3	83.6	77.6	–	85.7	–
B-CNN[D,M] [21]	84.1	85.1	83.9	–	91.3	–
B-CNN[D,D] [21]	84.0	84.8	84.1	–	90.6	–
STANFORD [14]	82.0	82.8	–	–	92.6	92.8
VCRLII [27]	84.5	–	80.89	–	–	–
ResNet152-SWP	**85.2**	**87.4**	**91.2**	**92.5**	**93.4**	**94.1**

Table 3.11: **Comparison of classification results**. We report classification accuracy on three fine-grained datasets without (e.g., birds) and with bounding-boxes (e.g., birds + box). FC-CNN uses features from the last FC layer of a CNN, and FV-CNN uses FV pooling of CNN filter banks [26]. B-CNN [21] is the bilinear model consisting of two CNNs shown in brackets. M-Net refers to [67] and D-Net is [18]. VCRLII refers to [27], which combines Class Activation Mapping and CNN to shift its center of attention to increasingly discriminative regions. STANFORD [14] is based on generating parts using co-segmentation and alignment.

cars datasets and obtain slightly poor performance on birds dataset. Table 3.10 also shows that there is a large decline in performance if reducing FC layers dimensions for AlexNet and VGGNet on the MIT67 dataset. Next, we add an SWP layer after the last max pooling and before the first FC layer to make a comparison with baselines. Likewise, ResNet101-MAX refers to ResNet baseline CNN architecture that replaces global average pooling with max pooling followed with a 1024 FC layer. Next, we replace the max pooling layer with an SWP layer and get ResNet101-SWP. Table 3.10 illustrates that the proposed SWP works on all three CNN architectures. Compared to three baselines, CNN architectures with SWP consistently achieve better performance.

Table 3.11 compares our best results with the state-of-the-art approaches. We achieve the best results on all six datasets with ResNet152-SWP trained on the image size of 293 $\times$ 293 and the crop size of 256 $\times$ 256. We only use this image resolution on the experiments in Table 3.11 and other experiments are conducted with the image size of 256 $\times$ 256 and the crop size of 224 $\times$ 224. We merely report the works without using extra data (e.g., Web data). All methods are only trained on one specific dataset from scratch or fine-tuned with the models

CNN architectures	birds	birds + box	aircrafts	aircrafts + box	cars	cars + box	MIT67
ResNet34-MAX	79.1	81.1	84.6	87.0	87.4	89.0	77.0
ResNet34-SWP	**80.1**	**82.2**	**85.4**	**87.9**	**88.1**	**90.2**	**78.7**
ResNet50-MAX	82.1	84.4	86.8	88.4	90.8	91.6	79.0
ResNet50-SWP	**82.5**	**84.7**	**87.1**	**88.9**	**91.1**	**92.1**	**79.5**
ResNet101-MAX	83.4	86.4	88.3	89.9	91.7	92.9	81.3
ResNet101-SWP	**83.8**	**86.7**	**89.4**	**91.0**	**92.1**	**93.4**	**82.5**
ResNet152-MAX	83.9	86.6	90.0	91.2	92.1	93.4	80.8
ResNet152-SWP	**84.2**	**87.0**	**91.0**	**92.1**	**92.6**	**93.8**	**81.9**

Table 3.12: **Comparison of classification results for ResNet 34-152**. The table compares the classification results between ResNet-MAX and ResNet-SWP for four variants of Residual Networks architectures.

pre-trained on the ImageNet, and no extra data are involved.

We can see that ResNet152-SWP with the crop size of 256×256 achieves best performance on all six datasets. In particular, we significantly improve the best result from previous 84.1% to 91.2% on the aircrafts dataset. The vast majority of aircraft objects occupy the entire image. Furthermore, aircrafts in images are generally horizontal. It means that most aircrafts have a simple side view, while cars and birds are photographed from multiple viewpoints. In other words, aircrafts are more well-aligned than birds and cars dataset. This may indicate that SWP could be able to enhance the performance significantly on well-aligned datasets.

Table 3.12 reports the classification accuracies of all ResNet architectures and Table 3.13 illustrates how much accuracy rate is improved by incorporating SWP into baseline CNN architectures. We can observe according to Table 3.12 that regardless of the depths, ResNet-SWP consistently performs better than ResNet-MAX, and ResNet34-SWP can improve the performance more than other ResNet-SWP architectures. This indicates that the effectiveness of SWP decreases as the depth of CNNs increases. The columns of birds and cars in Table 3.13 can clearly demonstrate this trend. By integrating SWP into AlexNet, we can get additional 4.3% performance gain, and 2.8% for VGGNet, 1.0% for ResNet34, 0.4% for ResNet50 and ResNet101.

Table 3.13 also shows that SWP works on all six different CNN architectures. However, it is more effective on AlexNet and VGGNet than ResNet. In AlexNet-

CNN architectures	birds	birds + box	aircrafts	aircrafts + box	cars	cars + box	MIT67
AlexNet-SWP	4.3	1.4	0.2	0.1	5.3	2.4	3.6
VGGNet-SWP	2.8	0.9	0.2	0.1	2.4	0.8	0.1
ResNet34-SWP	1.0	1.1	0.8	0.9	0.7	1.2	1.7
ResNet50-SWP	0.4	0.3	0.3	0.5	0.3	0.5	0.5
ResNet101-SWP	0.4	0.3	1.1	1.1	0.4	0.5	1.2
ResNet152-SWP	0.3	0.4	1.0	0.9	0.5	0.4	1.1

Table 3.13: **Performance improvement results**. The table illustrates the improvement of results for six different CNN architectures that add an SWP layer to AlexNet and VGGNet baselines or replace max-pooling with an SWP layer in all ResNets baselines.

MAX and VGGNet-MAX baselines, last max pooling layers down-sample all feature maps from 13×13 to 6×6 or 14×14 to 7×7 respectively. Its counterpart in ResNet-MAX baseline reduces the size of feature maps from 7×7 to 3×3. In other words, AlexNet and VGGNet keep more spatial information in the feature of the last max pooling layer than ResNet. We know that after the last max pooling operation, all features are fed into the FC layers to learn non-linear functions in that space. All spatial and local information is discarded after that.

We can observe from Table 3.13 that SWP can enhance the performance in the birds and cars without bounding-box much more significantly than on these two datasets with bounding-box. As the object size and location of the birds and the cars without bounding-box vary enormously, their features in last max pooling layer in AlexNet or VGGNet contains more complicated spatial information than those of birds and cars with bounding-box. After adding the SWP layer after the max pooling layer, the features after the SWP layer become more general and contain less spatial information. Therefore, SWP can better enhance performance for AlexNet and VGGNet on these two datasets without bounding-box. We argue that this is also the reason behind the performance drop by replacing 4096 FC layers with 512 FC layers for the MIT67 and birds dataset in Table 3.10, for the images in these two datasets have more complicated viewpoints, background and object size variation.

Fig. 3.9 shows learned SWP channels for AlexNet-SWP on the birds dataset. We can observe that each learned pooling channel encodes the features of last

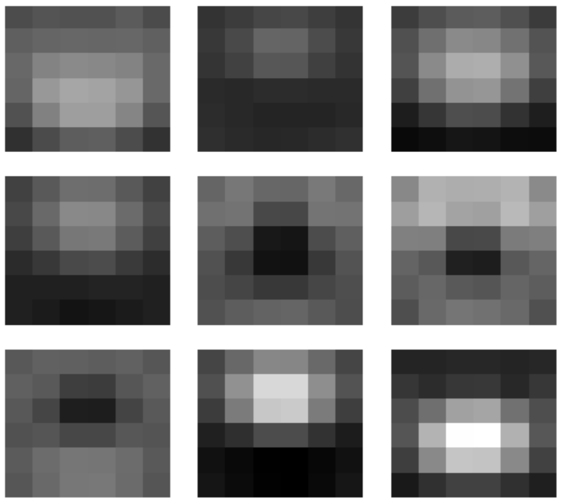
Figure 3.9: Nine SWP channels learned on the birds dataset [1] with AlexNet-SWP architecture.

max pooling layer from different spatial locations and viewpoints. Global average pooling can be seen as a spatial pooling with equal weights at all locations, whereas SWP pooling encodes the features of the previous layer with different weights distribution. SWP benefits from aggregating local convolutional feature maps with learned spatial importance information and producing more discriminative features that are fed into the fully-connected layers.

Weighted masks in an SWP layer can be treated as a set of various pooling kernels. These weighted masks can indicate some discriminative image regions used by deep CNNs to identify the categories of the objects. The features extracted from these regions contribute significantly to the pooled features. This observation is showed in Fig. 3.10. We show 9 weighted masks learned from the SWP layer in Stanford-Cars VGG-SWP and overlay them on the original images for better visualization. As can be observed from the figure, the activated regions of the weighted masks are actually semantically meaningful. For example, the activated region of the upper left graph indicates that both the front lights and fog lights are discriminative parts of the car. The bottom right graph in the figure corresponds to the front face of the car. The bottom left graph shows that the activated region corresponds to the log area of the car. Hence, these learnt weighted masks serve as a similar role as the part-region indicator maps. Compared to the human-specified part annotations, these masks can be optimized in training without any domain-specific knowledge.

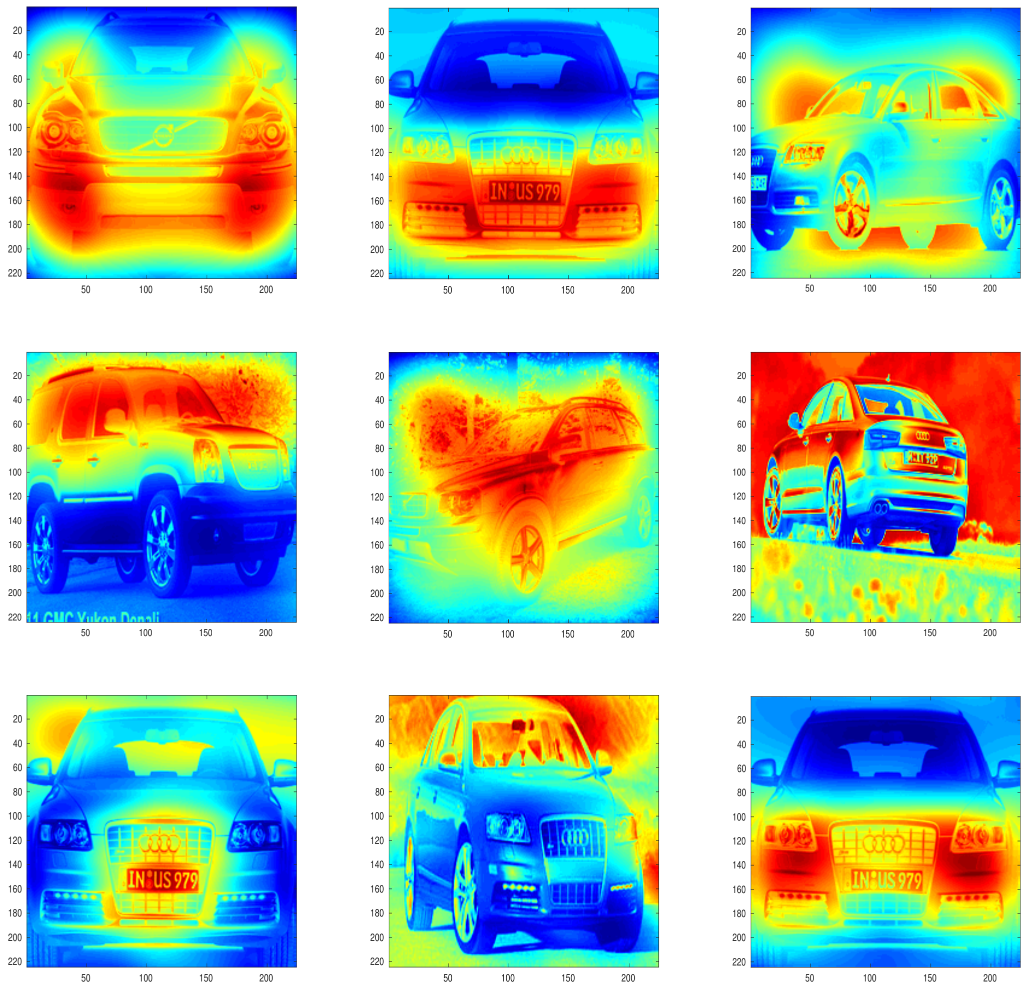

Figure 3.10: Visualization of 9 weighted masks learnt from Stanford-Cars VGG16-SWP.

Fig. 3.11 visualizes six feature maps of the last convolutional layer with the highest activations for birds ResNet101-SWP. The images in the left column are original 224 $\times$ 224 crops and the feature maps in the middle column and the right column are those with the highest activations by pooling with the SWP channels of the 14-th and the 16-th respectively. We can see the learned the 14-th SWP channel mainly encodes the body part of heads and the 16-th channel more focuses on the main body or the wing of birds. This shows that the pooling channels of SWP indicate some semantic meaningful regions of input images.

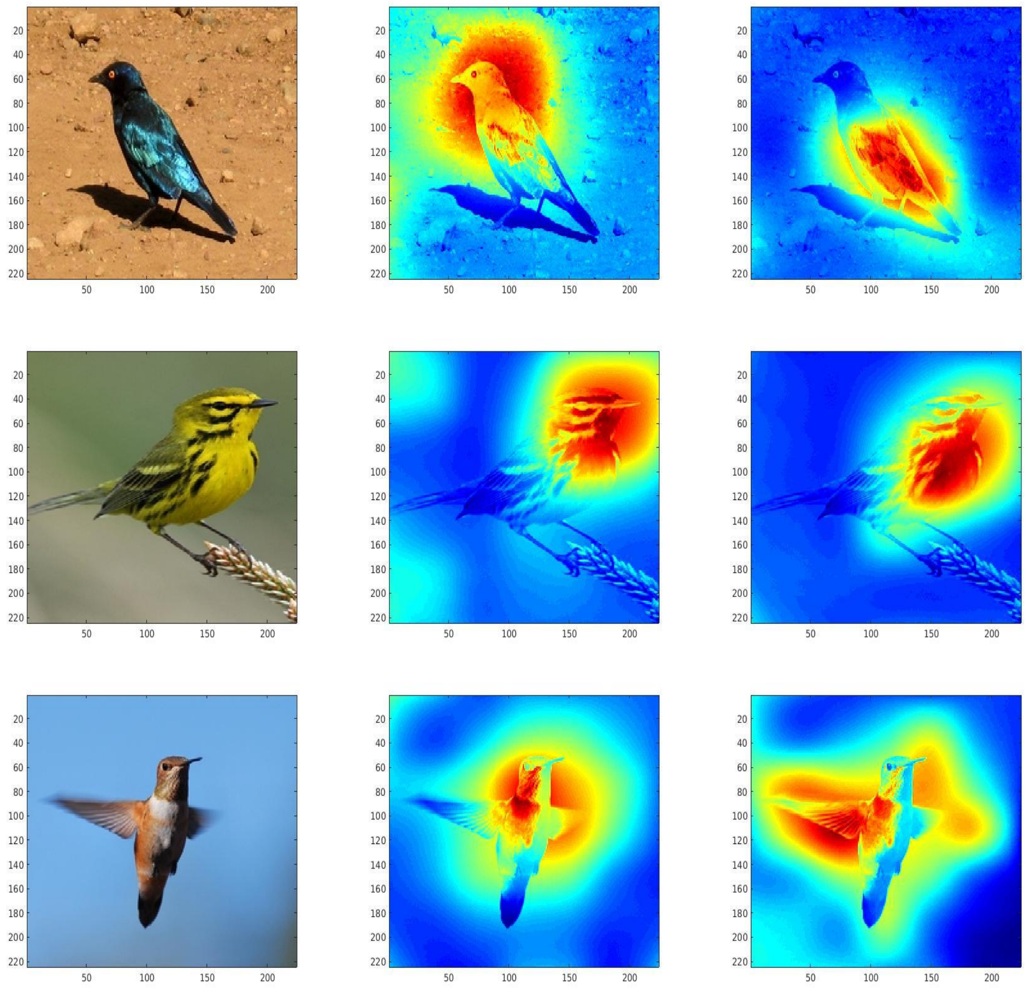

Figure 3.11: Visualizing of six feature maps with the highest activations by pooling with two learned SWP channels for birds ResNet101-SWP.

Fig. 3.12 visualizes six feature maps of the last convolutional layer with the highest activations for Stanford-Cars ResNet101-SWP. The images in the left column are original 224×224 crops and the feature maps in the middle column and the right column are those with the highest activations by pooling with the SWP channels of the 8-th and the 12-th respectively. We can see the learned the 8-th SWP channel mainly encodes the body part of headlights and tail-lights, and the 12-th channel more focuses on the front or rear face of the car.

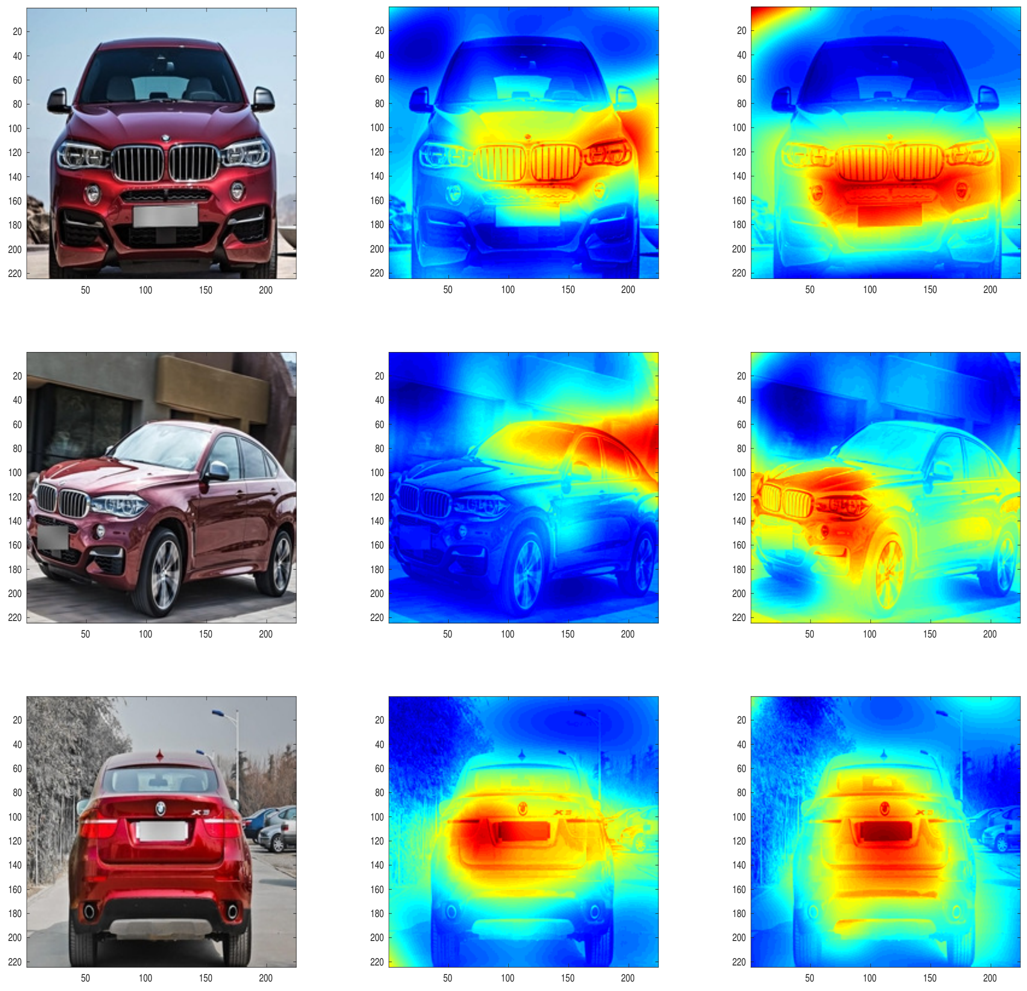

Figure 3.12: Visualizing of six feature maps with the highest activations by pooling with two learned SWP channels for CompCars ResNet101-SWP.

We also carried out experiments with the proposed ResNet-SWP method on the CompCars dataset. The corresponding results are illustrated in Table 3.14. One can see from the table that by fine-tuning the pre-trained VGGNet models on the CompCars dataset, we obtain much better results than those by fine-

Viewpoint	F	R	S	FS	RS	All-View
OverFeat fine-tuning [4]	52.4	43.1	42.8	56.3	59.8	76.7
AlexNet fine-tuning	50.2	41.3	40.9	55.1	58.3	75.6
VGGNet fine-tuning	77.2	79.1	69.1	79.8	81.1	90.8
CCL pooling conv52&conv53	84.3	87.4	79.4	88.8	89.5	94.6
ResNet101 fine-tuning	84.7	87.7	79.8	89.1	89.8	94.8
ResNet101-SWP fine-tuning	**86.7**	**89.8**	**81.6**	**91.3**	**91.9**	**96.8**

Table 3.14: Comparison of classification accuracy on CompCars between Over-Feat fine-tuning [4], VGG fine-tuning, cross-convolutional-layer pooling and the proposed ResNet-SWP (CCL pooling refers to cross-convolutional-layer pooling).

tuning the pre-trained OverFeat and AlexNet models. The better results have been obtained by fine-tuning the pre-trained ResNet-101 model in comparison with the pre-trained VGGNet model. By integrating the proposed SWP pooling into ResNet architecture, ResNet101-SWP achieves the best performance using a single fine-tuned model trained on 224×224 image crops. The results of the fourth row in the table are obtained by performing cross-convolutional-layer pooling between the local features of conv53 and the indicator maps of conv54 both extracted from the fine-tune VGGNet model. The input test image resolution of above extracting process is 342×342, and the corresponding crop size is 300×300. Although compared to straightforward results from softmax layer in the fine-tuned VGGNet model, cross-convolutional-layer pooling can enhance the performance considerably. Its improvement is built upon the cost of increasing the resolution of test images and consequent computational overhead.

During the training of ResNet-related experiments, we saved training and validation errors of each epoch to have a clear understanding of the whole training process. Note that the training process has 90 epochs, and the initial learning rates of all ResNet-related experiments all start from 0.001 and are divided by 10 after 40 training epochs. This means there are two drops for the learning rates, one is from 0.001 to 0.0001, and another one is from 0.0001 to 0.00001. We can only observe one significant decrease in training and validation errors when the learning rates drop from 0.001 to 0.0001 at the 41-th epoch (see Fig. 3.13). The left graph in Fig. 3.13 compares the top-1 training and validation errors of 90 epochs for All-View and F-View CompCars. There are in total $16,016$ training

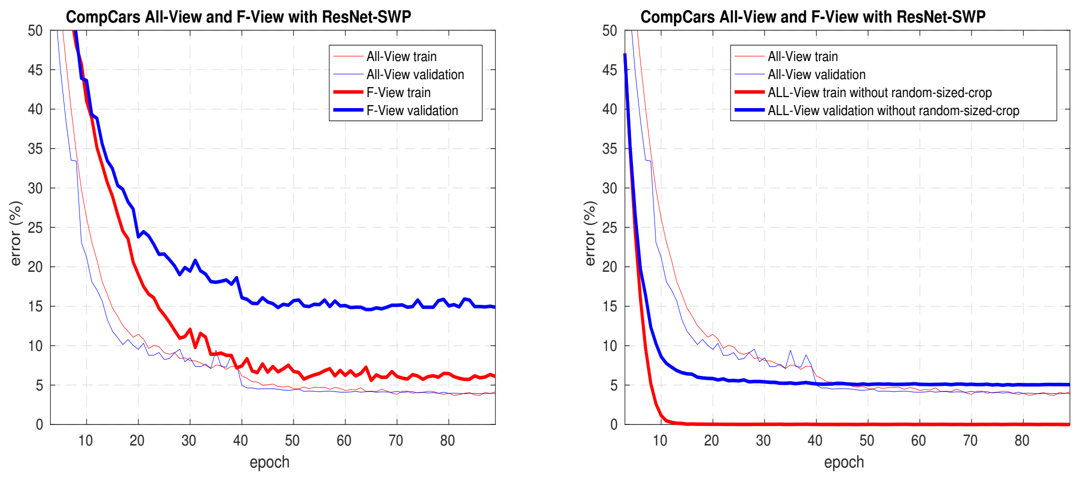

Figure 3.13: Training on CompCars All-View and F-View with ResNet-SWP.

images and 14, 939 testing images for All-View and 2, 593 training images and 2, 381 testing images for F-View.

All-View has much more training and testing images than F-View as the latter is only one of the five views of the former. The thin and bold curves in the left graph in Fig. 3.13 denote All-View and F-View training and validation errors respectively. Red curves represent training errors, and blue curves refer to validation errors. From the left graph in the figure, one can obtain two significant observations. The first is that the training and validation errors of All-View decrease much faster than those of F-View. After 10-epochs training, both errors of All-View drop to around 20%, whereas the errors of F-View are still more than 40%. The second observation is that the validation errors of F-View are much higher than training errors while two types of errors for All-View are almost equal. This indicates that F-View training suffers from over-fitting issue to some extent.

We argue that it is caused by the fact that such a large model is fine-tuned on such a small-scale dataset with only around 2, 600 training images. Conversely, All-View training never incur the same issue as it is performed on more than 16, 000 images. To explain the first observation, we believe that compared to All-View training, the F-View training process is more complicated and difficult due to less distinguishable features for one single view. In the right graph, All-View training and validation errors with or without random-sized-crop are plotted. One can observe that the training process does benefit from *random-sized-crop*, which is a data augmentation technique that randomly crops various sized patches whose size is distributed evenly between 8% and 100% of the image. With random-sized-crop training, we achieved the accuracy of 96.1% on All-View CompCars, compared to the result of 94.9% without random-sized-crop.

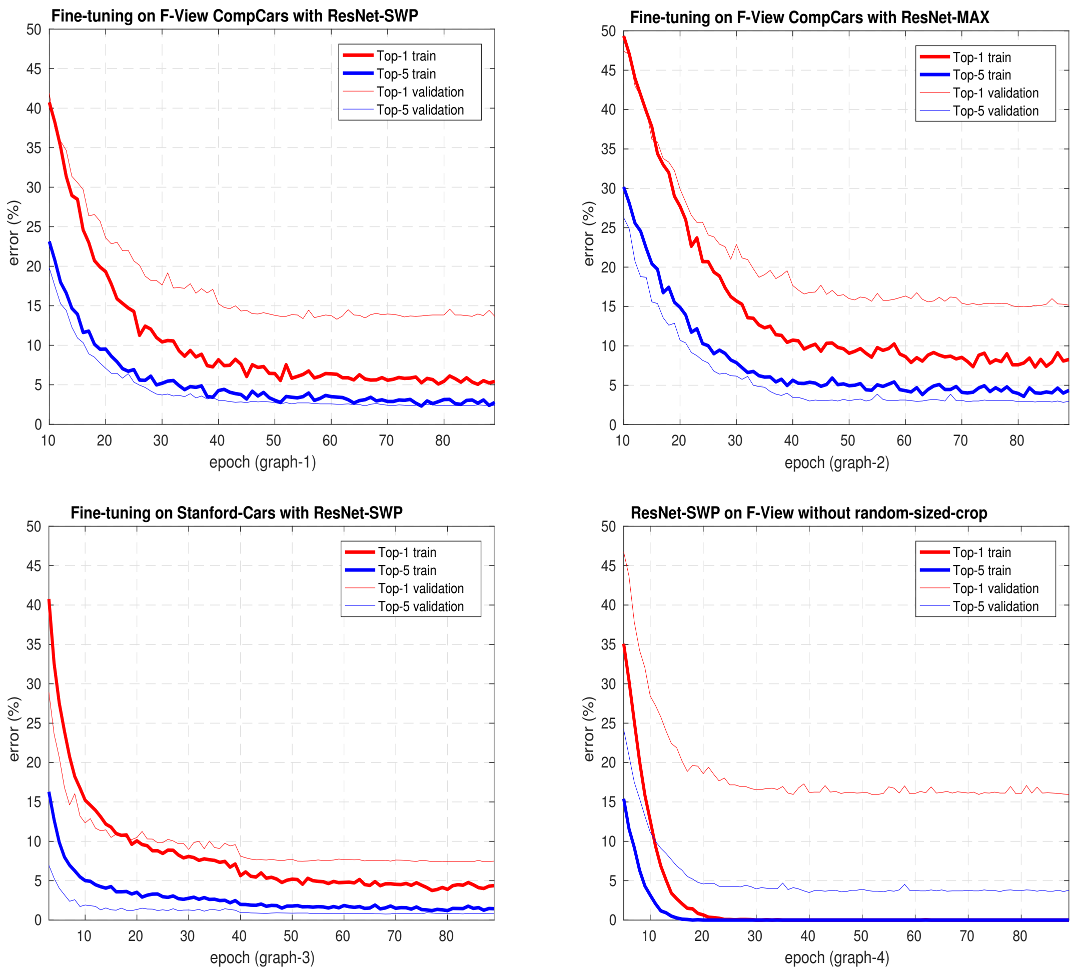

Figure 3.14: Plotting training and validation errors training on CompCars and Stanford-Cars.

Fig. 3.14 compares the top-1 and top-5 training and validation errors of 90 epochs for four different training. The upper left graph (graph-1) in the figure shows the errors of fune-tuning ResNet-SWP on F-View CompCars. The upper right graph (graph-2) refers to the erros in fin-tuning ResNet-MAX without SWP.

The graph in the bottom left corner (graph-3) plots the errors in fune-tuning ResNet-SWP on Stanford-Cars. The bottom right one (graph-4) plots the errors in fine-turning ResNet-SWP on F-View CompCars without random-sized-crop. Note that our process of cropping images is to follow the process of training GoogleNet [48], which crops various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area with aspect ratio constrained to the interval [3/4, 4/3]. This process is called "random-sized-crop". "No random-sized-crop" refers to the process that a fixed sized 224×224 patch is cropped randomly from the 256×256 whole image. This process is called "random-crop".

By comparing graph-1 and graph-4, we can recognize that fine-tuning ResNet-SWP on F-View CompCars without random-sized-crop suffers from more serious over-fitting issue than its counterpart with random-sized-crop. The classification accuracy obtained by the former model is 84.1% while the latter model can achieve 86.7% on F-View CompCars. This shows this data augmentation method is really effective in alleviating over-fitting and enhancing performance when a relatively large model is fine-tuned on a small dataset. Comparing graph-1 with graph-2 illustrates that the proposed ResNet-SWP does perform better than ResNet-MAX. Note that ResNet-SWP is a model that replaces an SWP layer with the max-pooling layer in ResNet-MAX. The SWP layer generate more discriminative pooling vectors than those produced by original max-pooling layer, which can make the proposed model converge faster and better on the fine-grained datasets. After 10-epochs training, both errors of ResNet-SWP decrease to around 40%, whereas the errors of ResNet-MAX are still around 50%.

Graph-1 and graph-3 plot errors on different dataset with the same model - ResNet-SWP. The Stanford-Cars dataset contains $16,185$ images of 196 classes of cars. It is split into $8,144$ training images and $8,041$ testing images. The number of its training samples is much higher than F-View's, but still only half of All-View's. Therefore, we can observe from graph-3 that over-fitting issue on Stanford-Cars is not so serious compare to it on F-View. Only All-View CompCars training does not suffers from over-fitting issue, and we achieved 96.8% top-1 accuracy on it. This implies that if there are more powerful and effective training data augmentation techniques available, better performance can be achieved with the proposed SWP pooling strategy on small-scale fined-grained datasets.

Viewpoint	F	R	S	FS	RS	All-View
ResNet101 fine-tuning	84.7	87.7	79.8	89.1	89.8	94.8
ResNet101-SWP fine-tuning	86.7	89.8	81.6	91.3	91.9	96.8
Testing results	**95.9**	**96.3**	**93.9**	**96.4**	**97.6**	**96.8**

Table 3.15: Comparison of classification accuracy on CompCars.

Table 3.15 shows the test results for each single view using fine-tuned All-View CompCars ResNet101-SWP. This means the model was trained on the entire CompCars with five views and tested on one of the five views. The results are reported in the bottom line of the table, and we can see that the test results have been improved enormously. For instance, the test result of F view is 86.7%, and if the model is fine-tuned on five views, the test result on F view can reach 95.9%. Although there is no increase in the number of F view training samples, we do introduce more other views samples (see Fig. 3.15). The training can benefits from two aspects. Firstly, this helps to remedy over-fitting issue. Secondly, the model can learn more discriminative features from other views images.

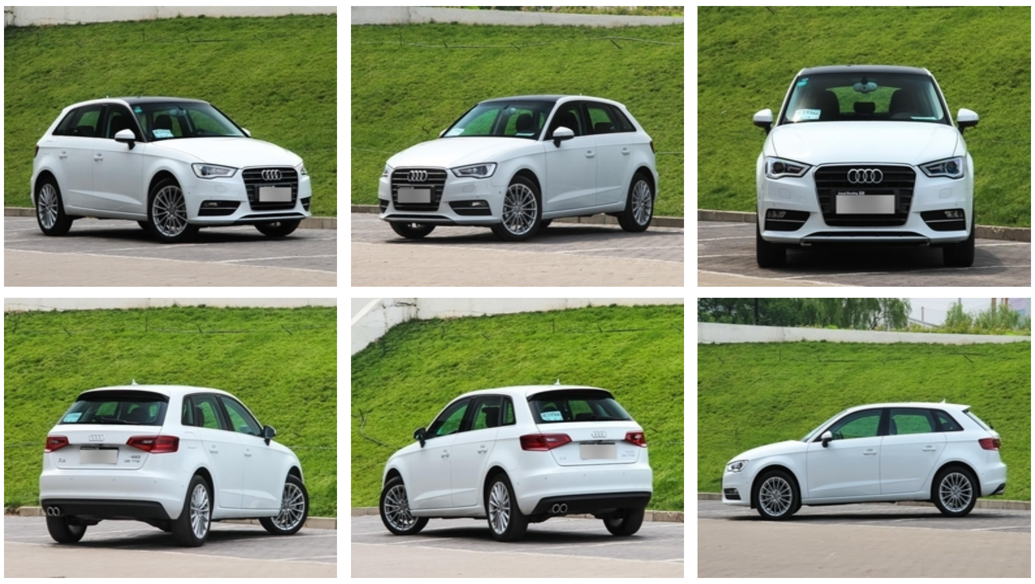

Figure 3.15: Other views training samples.

3.3.6 Feature maps and learned filters visualization

Several challenging cases are given in Fig. 3.16 and Fig. 3.17, whre the images on the left hand side are the testing images and the images on the right hand side are the examples of the wrong predictions. Most of the wrong predictions in Fig. 3.17 belong to the same car makes as the test images, and it is even hard for human beings to distinguish one test image from its wrong prediction. In the same way, most of the wrong predictions in Fig.3.16 look very similar to the test images.

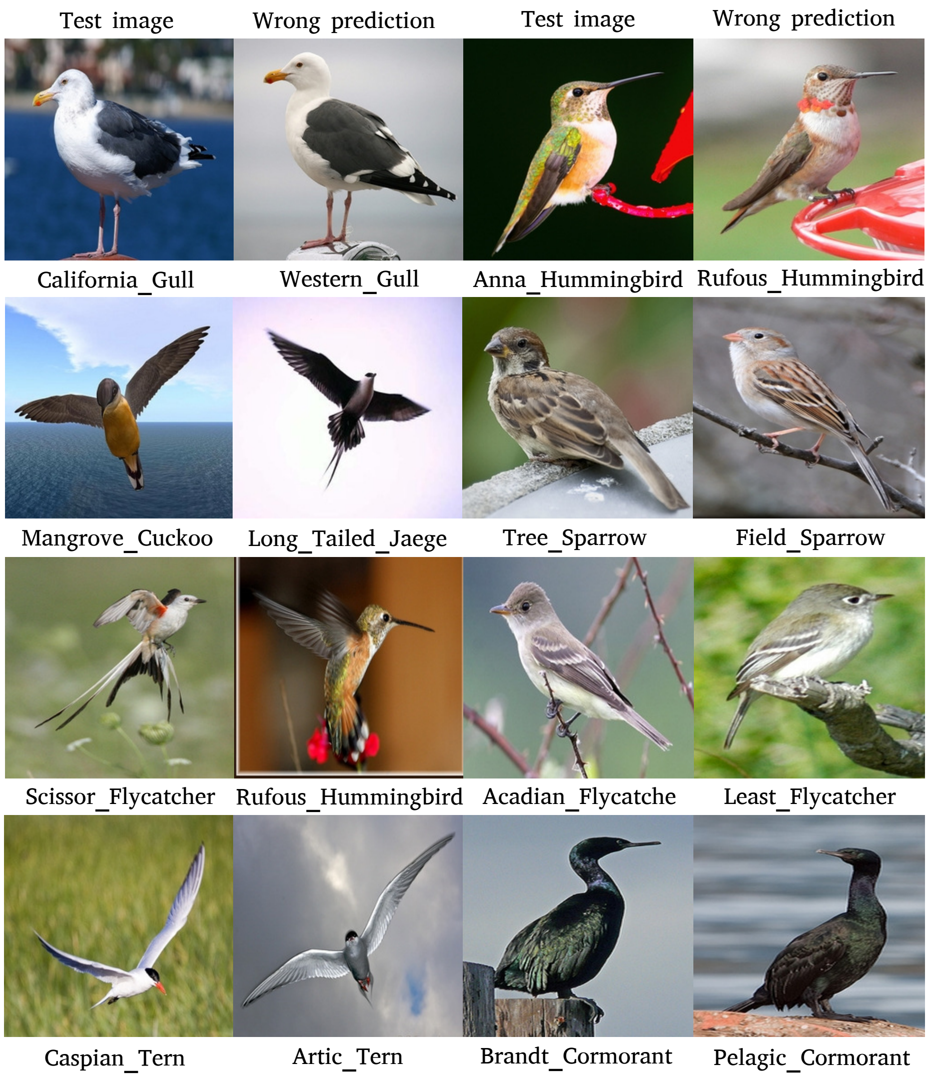

Figure 3.16: Sample test images that are mistakenly predicted as another model.

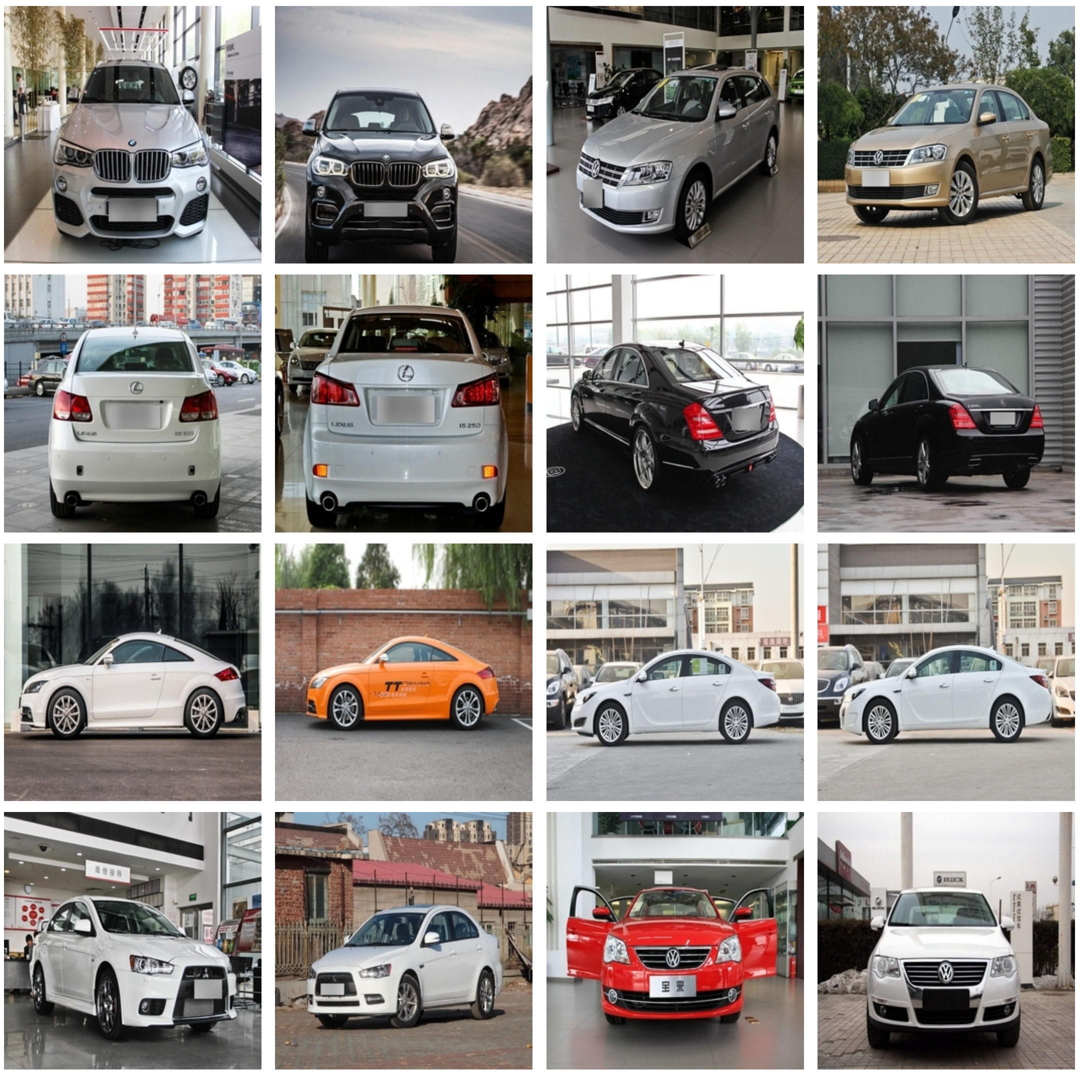
Figure 3.17: Sample test images that are mistakenly predicted as another model in their makes. Each row displays two samples and each sample is a test image followed by another image showing its mistakenly predicted model.

Fig. 3.18 and Fig. 3.19 visualize the learned first convolutional layer filters for the birds and Stanford-Cars datasets respectively. We can see that two sets of convolutional filters learned from two different fine-grained datasets are almost identical. However, they do have different values for each pair of convolutional filters. In the work of [64], it is illustrated that the first two convolutional layers respond to corners and other edge/color conjunctions, and the third convolutional layer has more complex invariances, capturing similar textures. Although two sets of filter banks are fine-tuned on two different datasets, but they are from the same initializations learned from ImageNet, and they both have the same functionality to respond to edge, color and blob. Therefore, we argue that they are why two sets of filter banks are nearly identical.

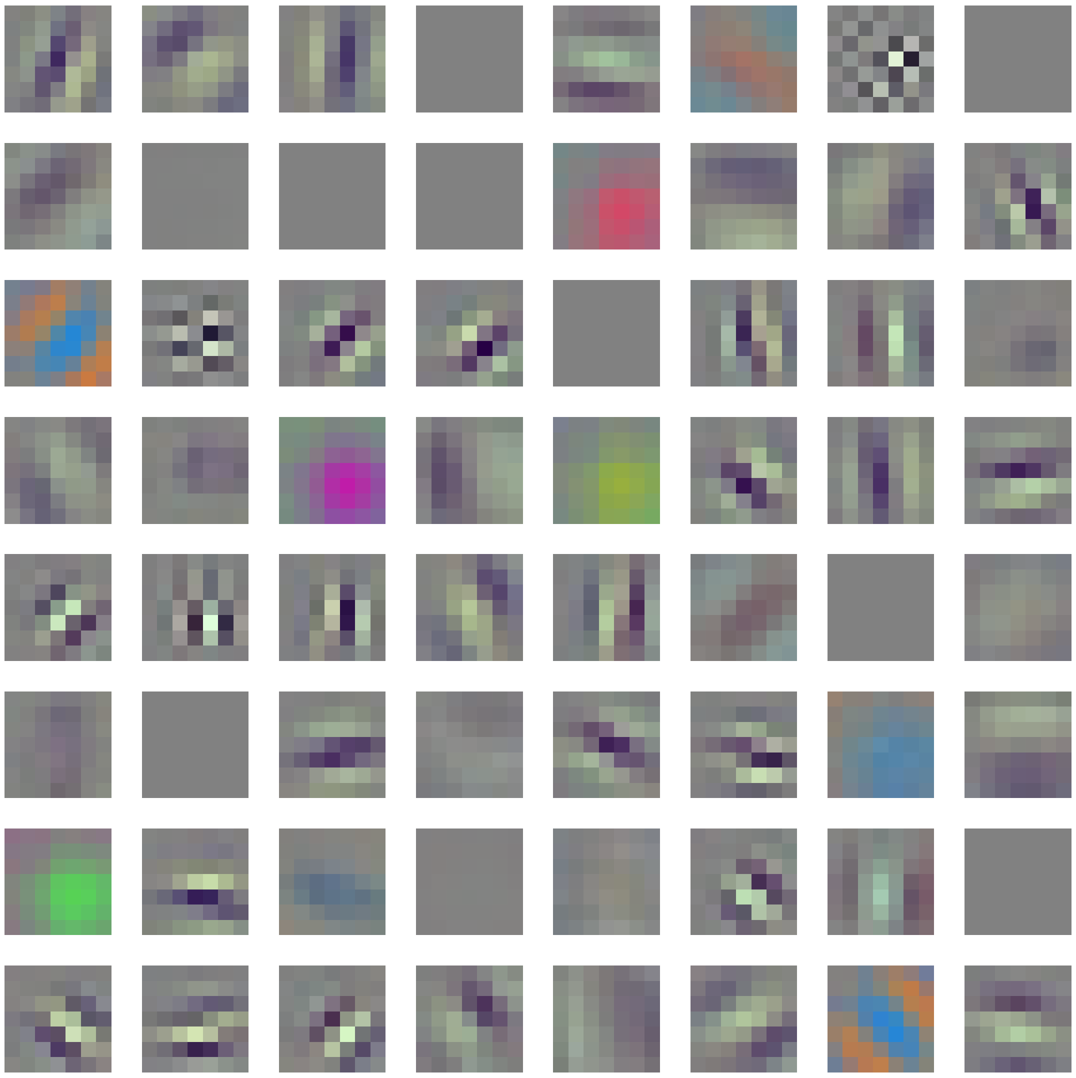

Figure 3.18: Visualizing Conv1 filters in birds ResNet101-SWP.

Fig. 3.20 and Fig. 3.21 visualize the learned 16 SWP filters in birds ResNet101-SWP and CompCars ResNet101-SWP respectively. Fig. 3.26, Fig. 3.27, Fig. 3.28, Fig. 3.29, Fig. 3.30 and Fig. 3.31 show patches with highest activations for filters in the first, 6-th and 69-th layer in the fine-tuned CompCars ResNet101-SWP and birds ResNet101-SWP. Fig. 3.32, Fig. 3.33, Fig. 3.34, Fig. 3.35 and Fig. 3.36 visualize the feature maps of the first, 6-th, 18-th, 69-th and 99-th layer in CompCars ResNet101-SWP. Fig. 3.22 and Fig. 3.24 visualize feature maps in the last convolutional layer with highest activations after pooling with the first 8 filters in the SWP layer in CompCars and birds ResNet101-SWP. Fig. 3.23 and Fig. 3.25 are for the 9-th to 16-th filers.

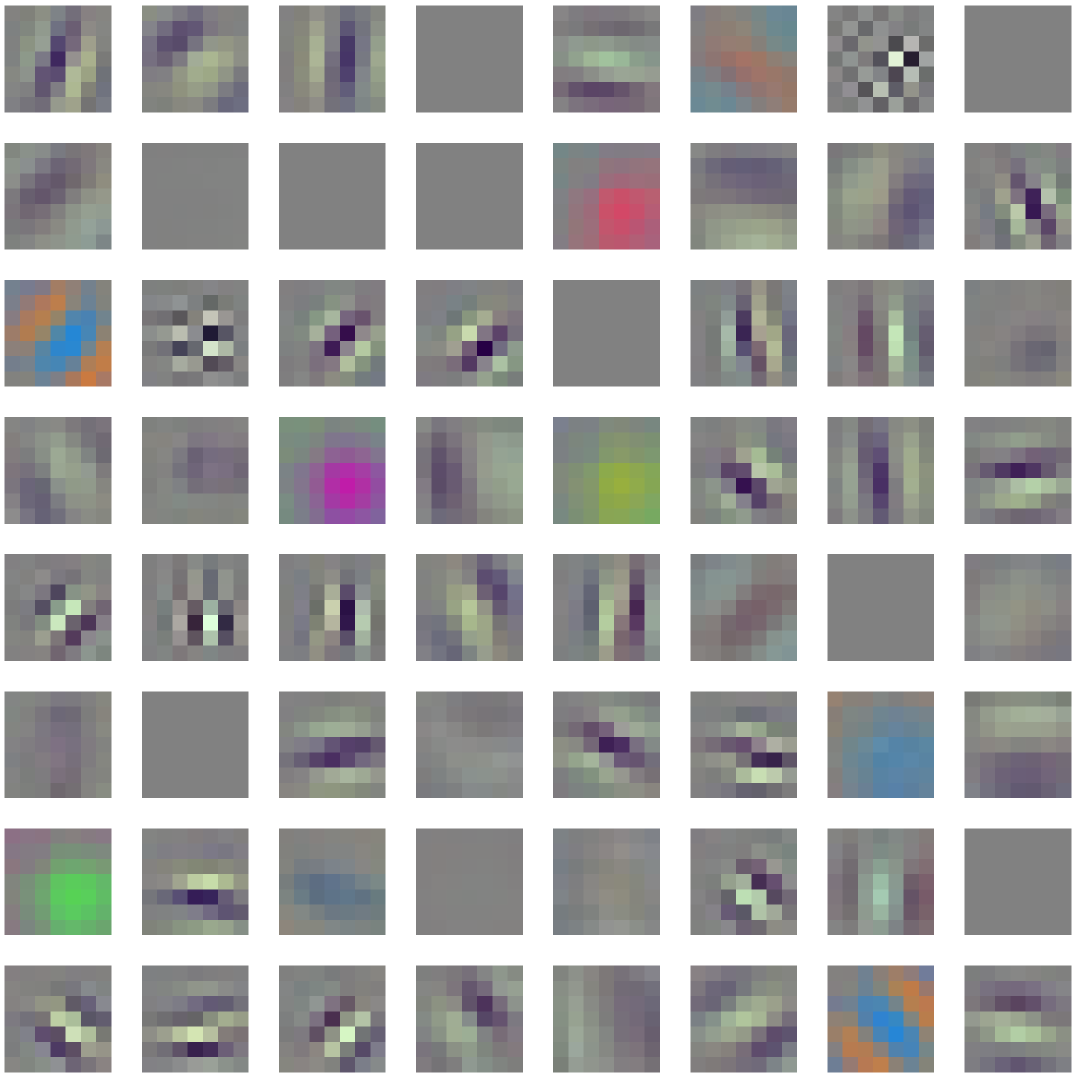

Figure 3.19: Visualizing Conv1 filters in Stanford-Cars ResNet101-SWP.

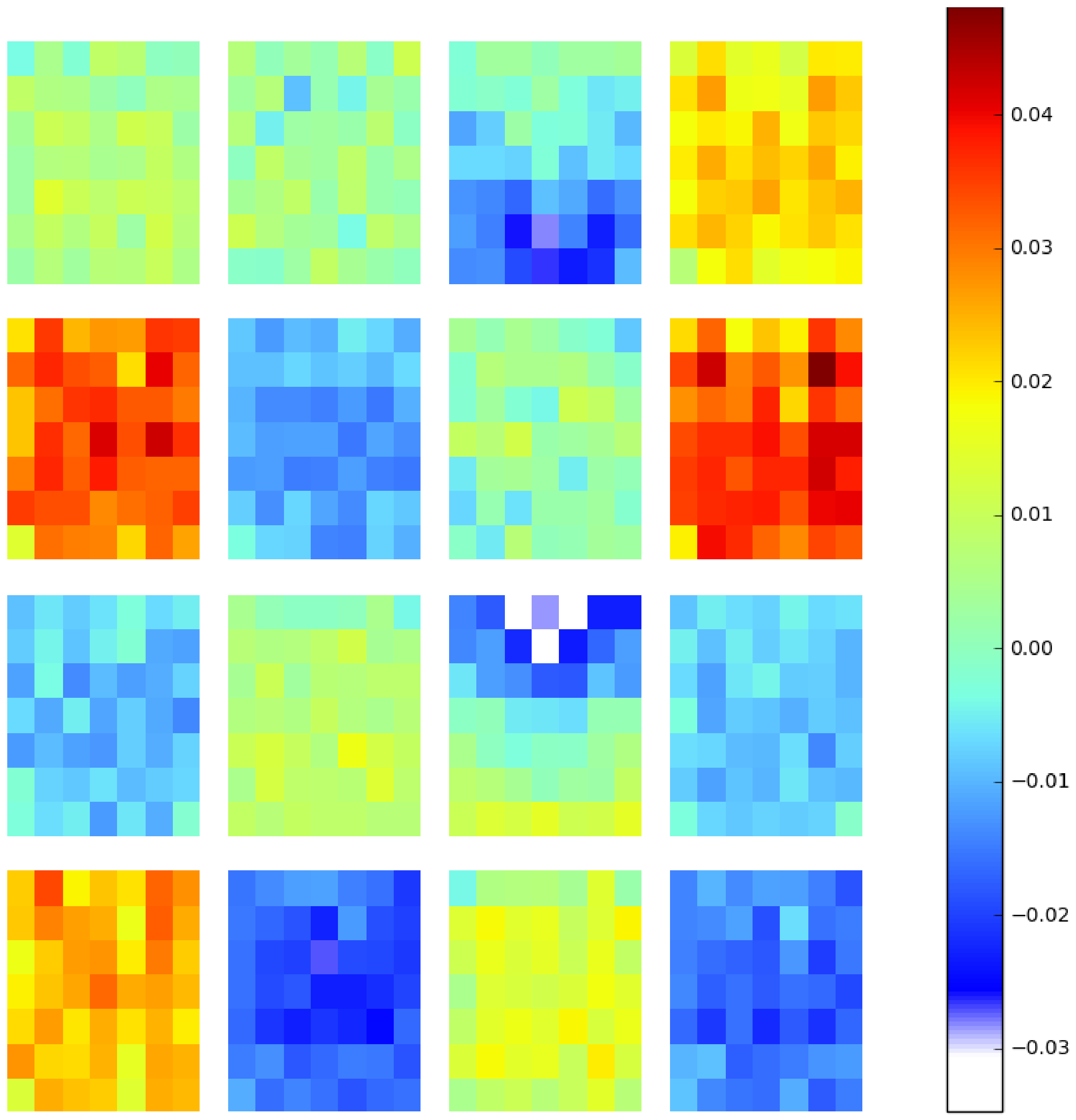

Figure 3.20: Visualizing learned 16 SWP filters in birds ResNet101-SWP. The learned weights are between −0.0348003358 and 0.0481069833.

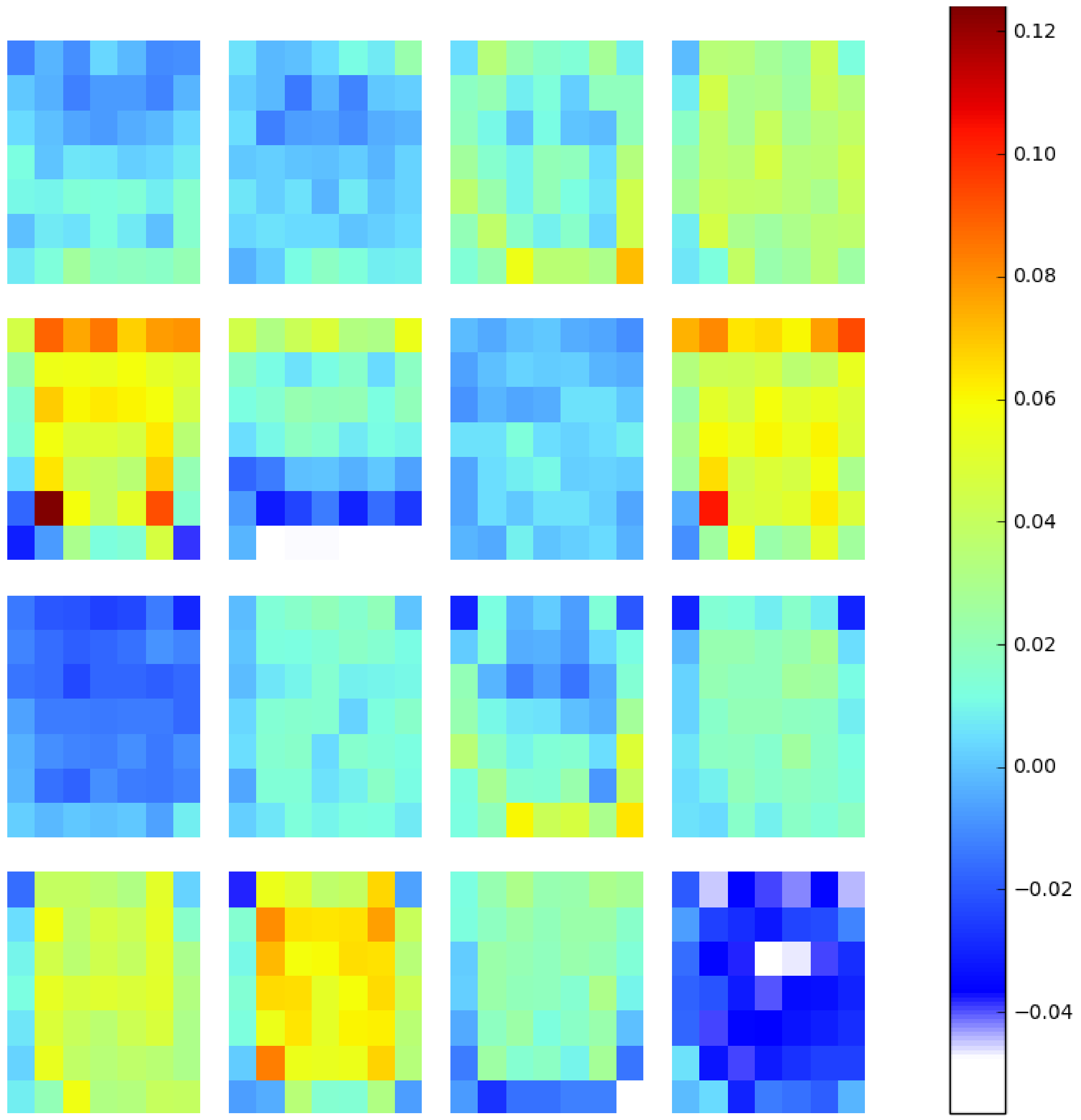

Figure 3.21: Visualizing learned 16 SWP filters in CompCars ResNet101-SWP. The learned weights are between −0.0566353425 and 0.124176443.

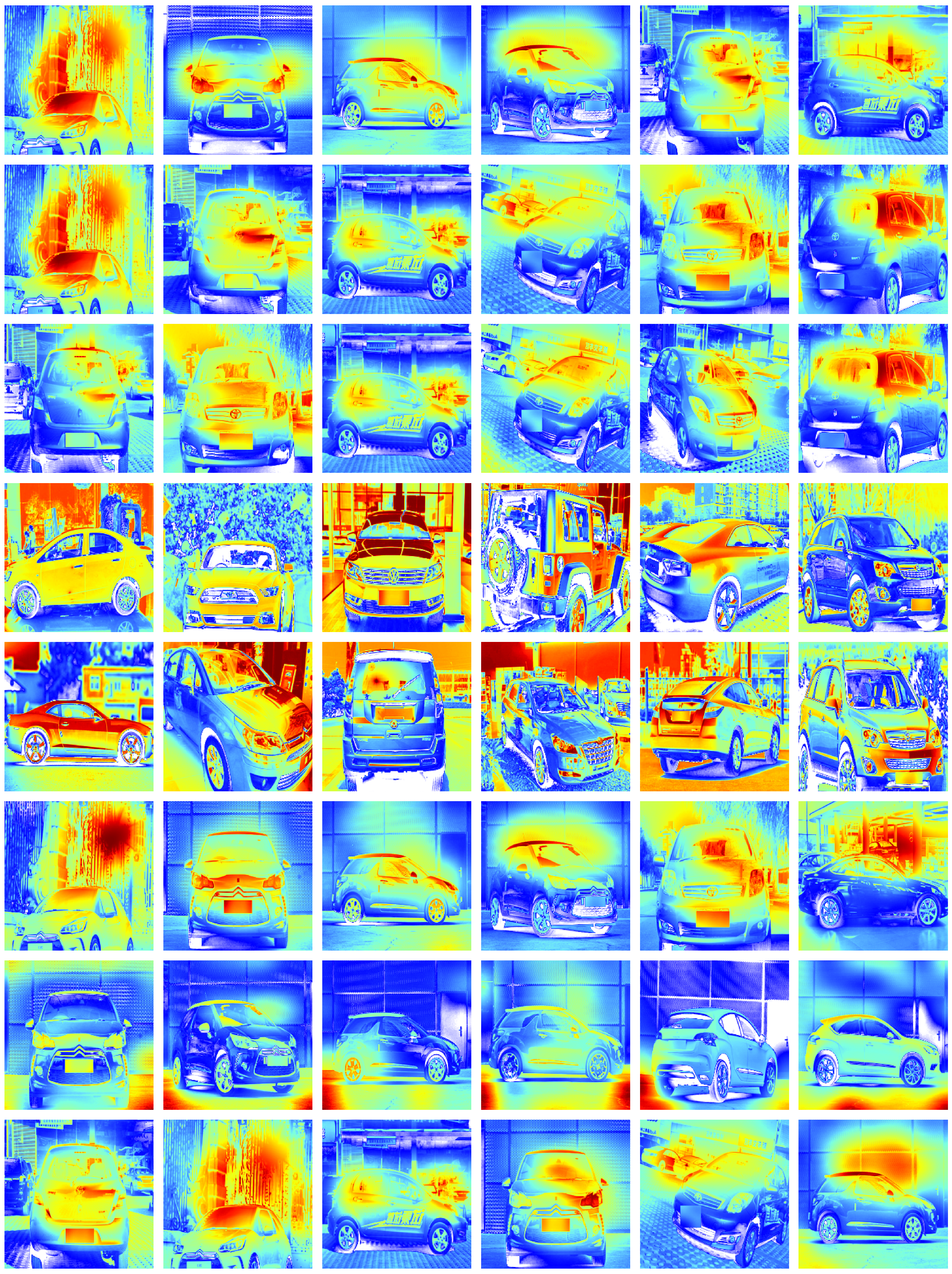

Figure 3.22: Visualizing feature maps in the last convolutional layer with highest activations after pooling with the first 8 filters in the SWP layer in CompCars ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all $14,939$ test images. We obtain six plot images by overlaying the six feature maps on their original images.

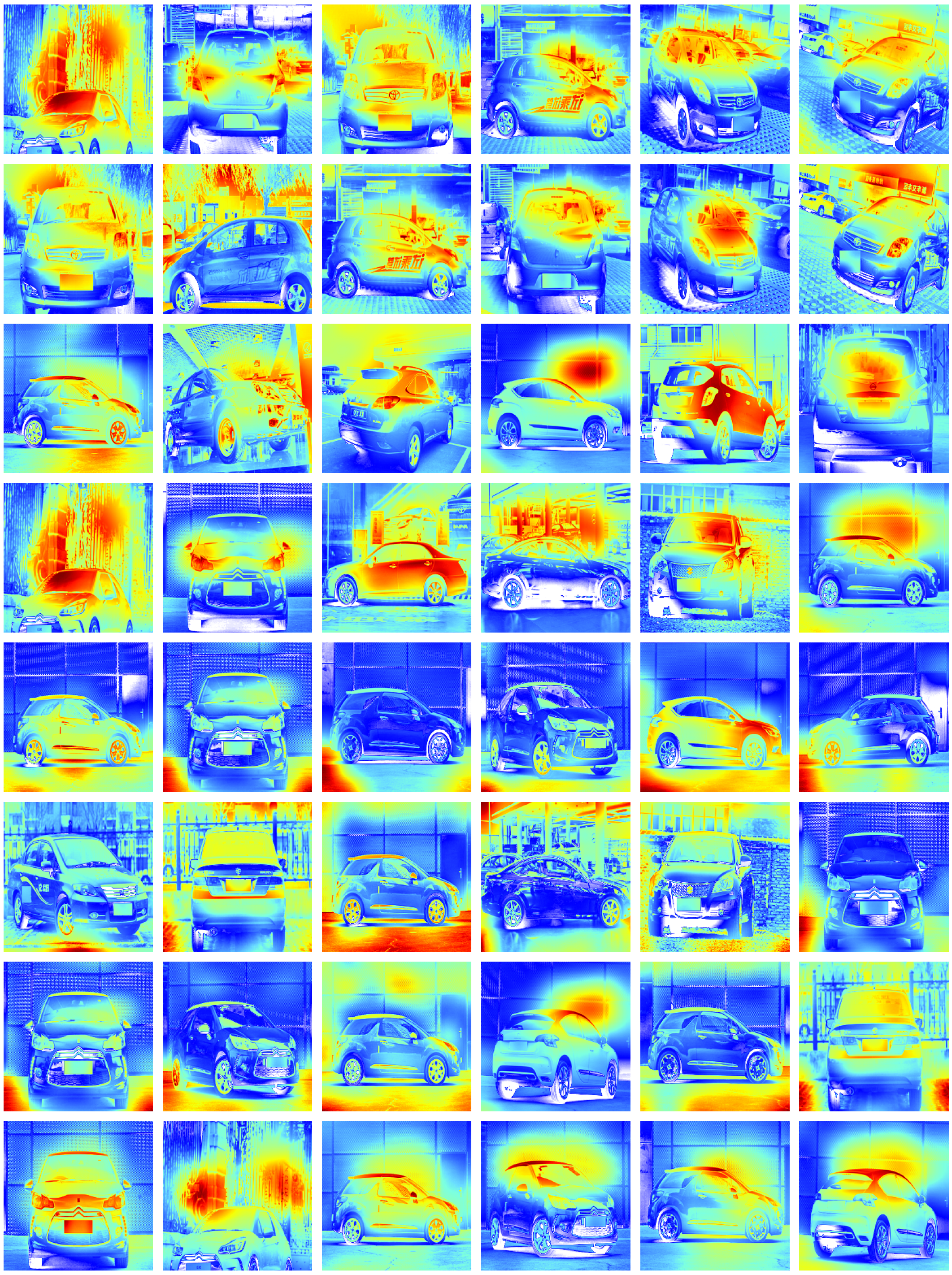

Figure 3.23: Visualizing feature maps in the last convolutional layer with highest activations after pooling with from the 9-th to 16-th filters in the SWP layer in CompCars ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all 14, 939 test images. We obtain six plot images by overlaying the six feature maps on their original images.

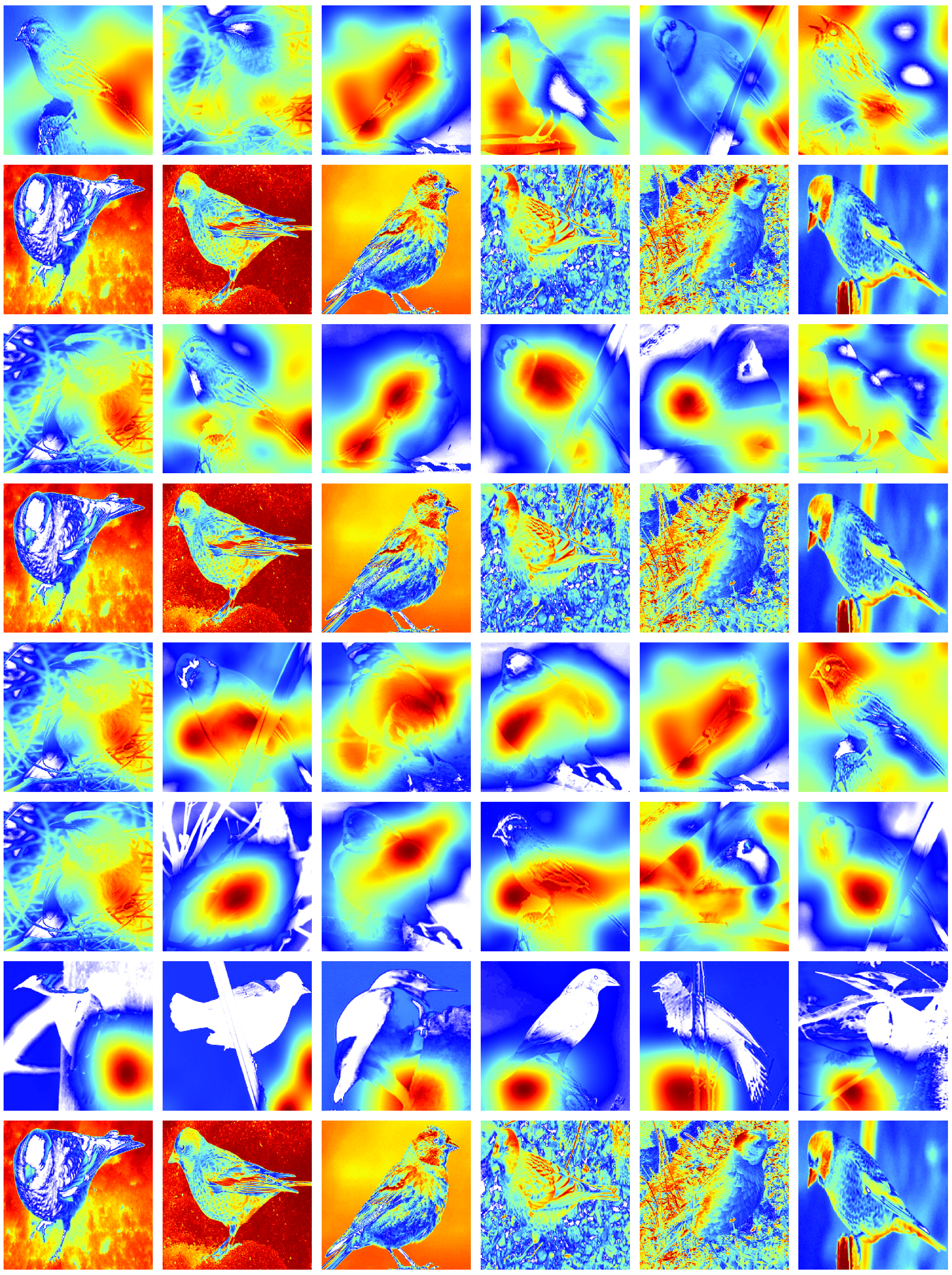

Figure 3.24: Visualizing feature maps in the last convolutional layer with highest activations after pooling with the first 8 filters in the SWP layer in birds ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all 5,794 test images. We obtain six plot images by overlaying the six feature maps on their original images.

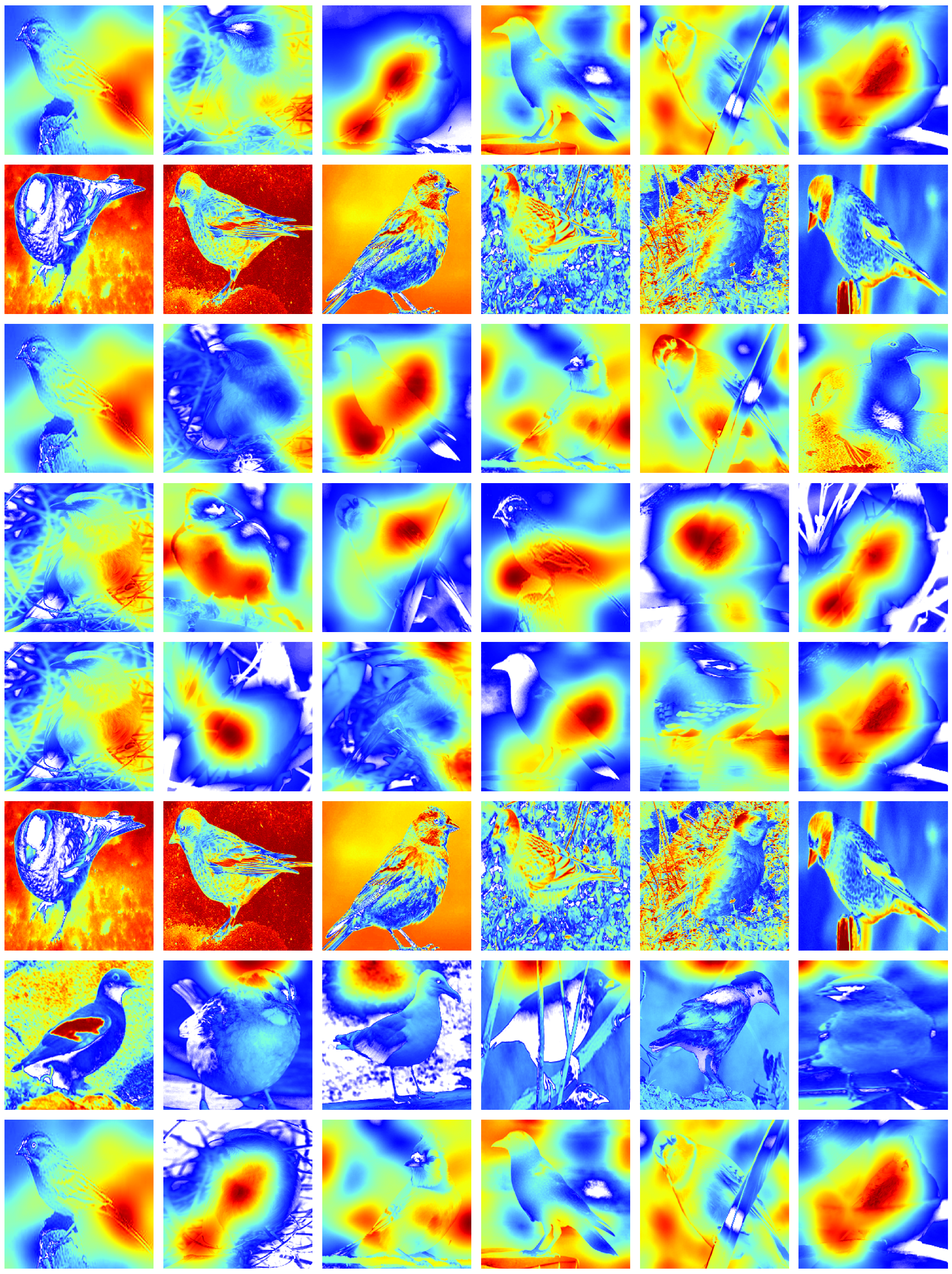

Figure 3.25: Visualizing feature maps in the last convolutional layer with highest activations after pooling with from the 9-th to 16-th filters in the SWP layer in birds ResNet101-SWP. Each row refers to one learned SWP filter. Six images in one row represent the top six feature maps with the highest SWP pooling activations for all 5, 794 test images. We obtain six plot images by overlaying the six feature maps on their original images.

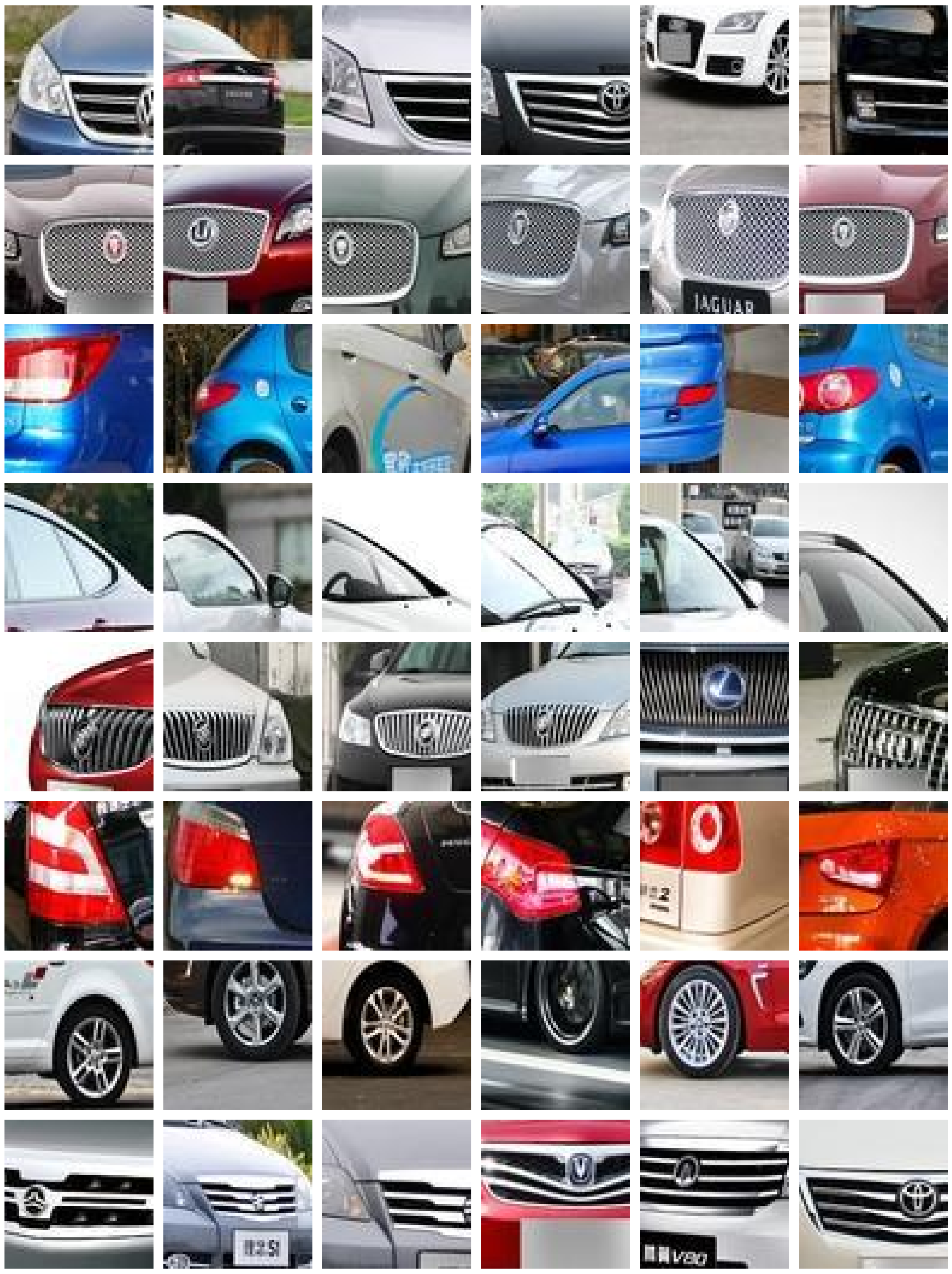

Figure 3.26: Patches with highest activations for several filters in the first convolutional layer (in total 64 filters) in the fine-tuned CompCars ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all 14, 939 test images. From the top row to the bottom are for the 5-th, 7-th, 17-th, 30-th, 34-th, 51-th, 54-th and 64-th filters.

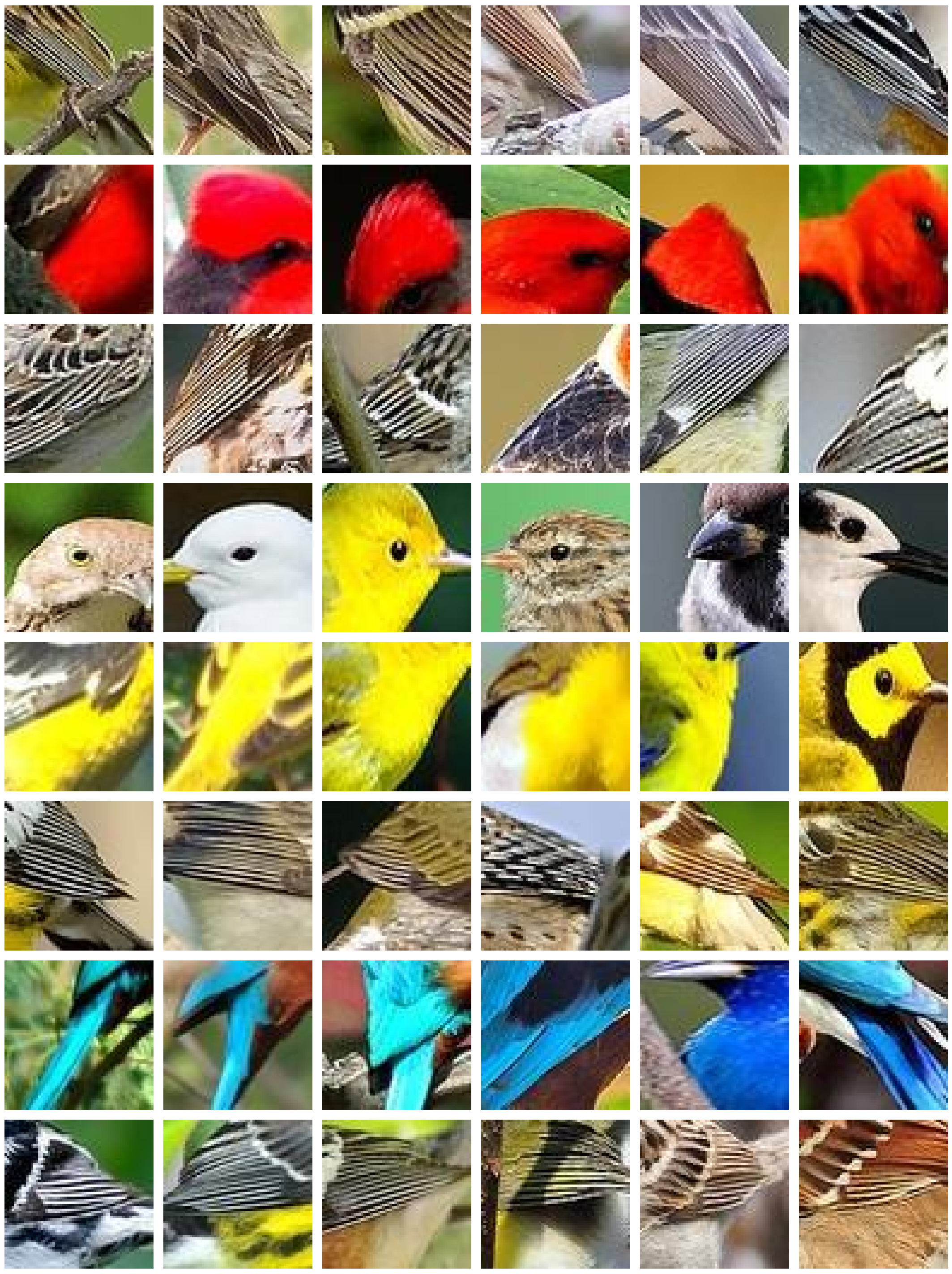

Figure 3.27: Patches with highest activations for several filters in the first convolutional layer (in total 64 filters) in the fine-tuned birds ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all 5,794 test images. From the top row to the bottom are for the 6-th, 7-th, 19-th, 25-th, 28-th, 45-th, 46-th and 57-th filters.

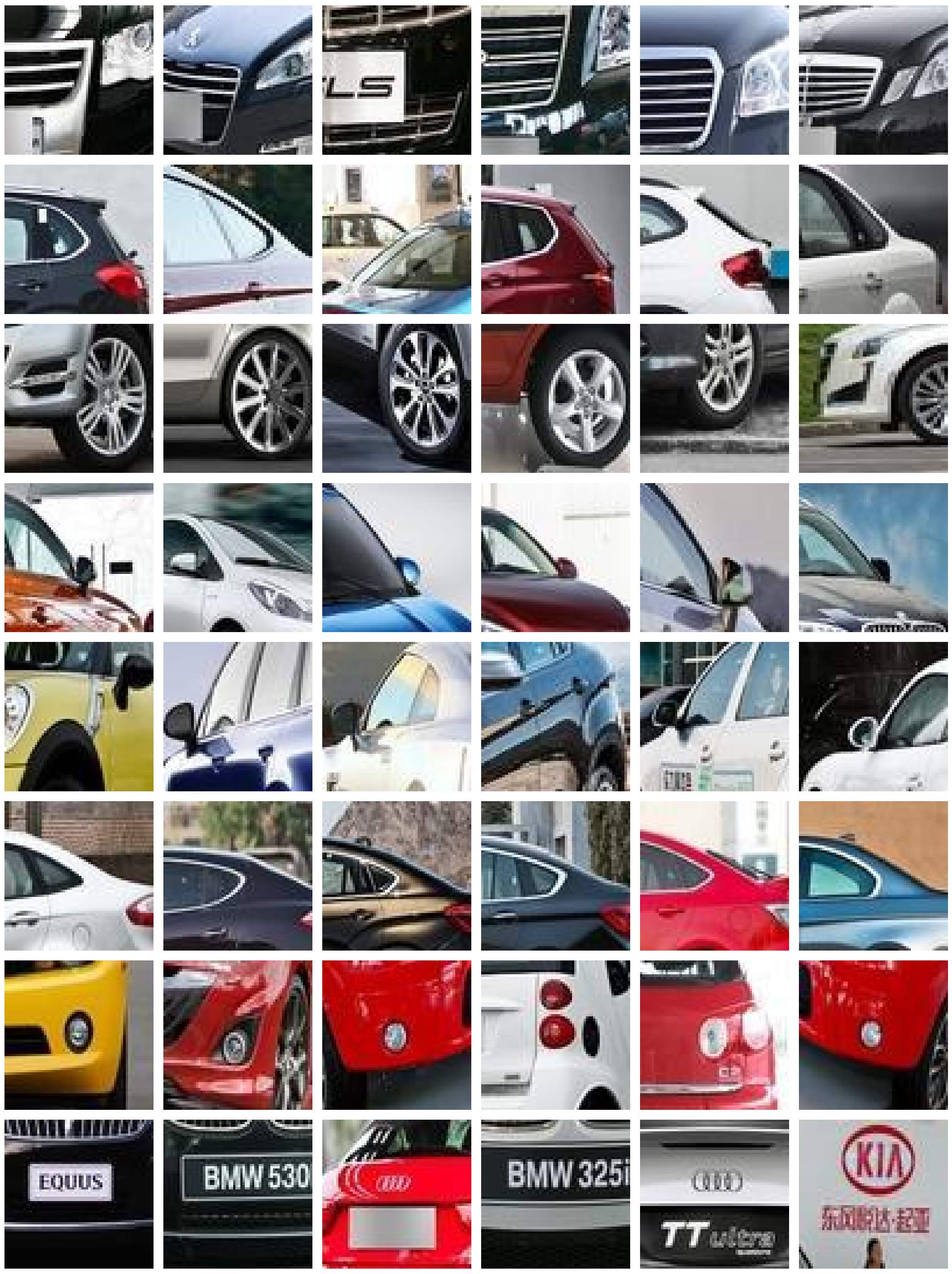

Figure 3.28: Patches with highest activations for several filters in the 6-th layer (in total 64 filters) in the fine-tuned CompCars ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all 14, 939 test images. From the top row to the bottom are for the 1-th, 9-th, 13-th, 14-th, 15-th, 18-th, 49-th and 51-th filters.

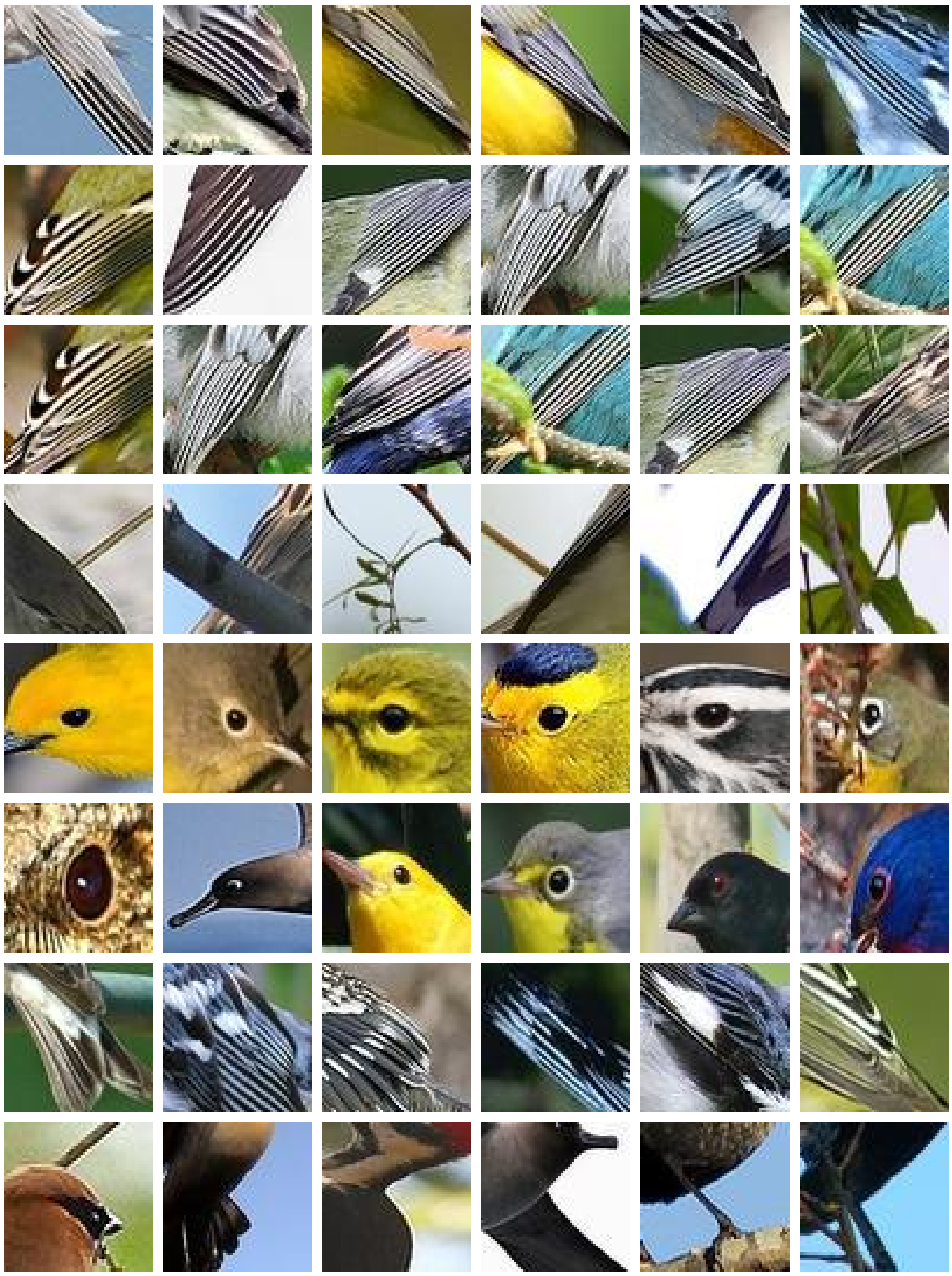

Figure 3.29: Patches with highest activations for several filters in the 6-th layer (in total 64 filters) in the fine-tuned birds ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all $5,794$ test images. From the top row to the bottom are for the 3-th, 5-th, 16-th, 29-th, 48-th, 57-th, 59-th and 60-th filters.

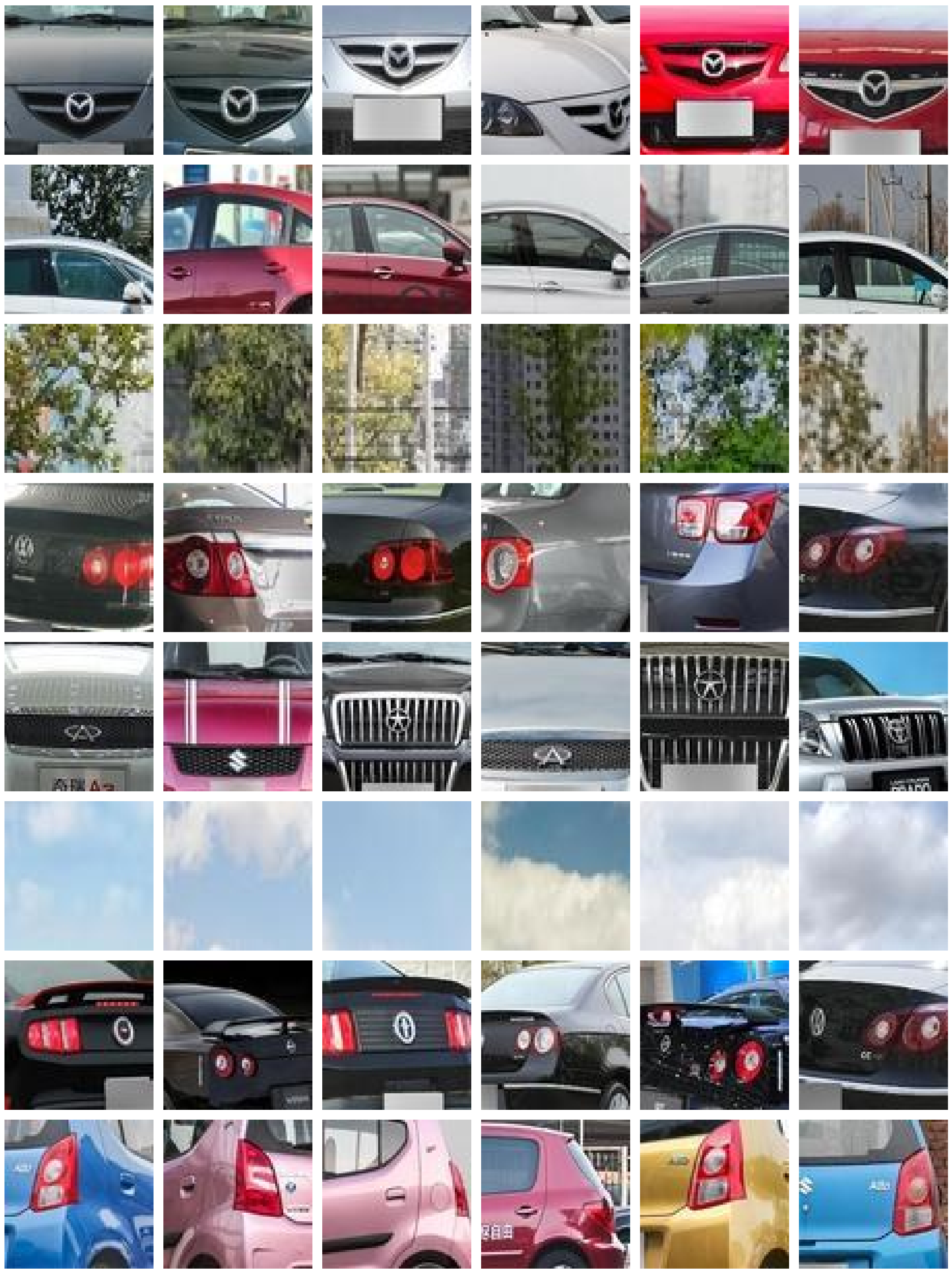
Figure 3.30: Patches with highest activations for several filters in the 69-th layer (in total 512 filters) in the fine-tuned CompCars ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all 14, 939 test images. From the top row to the bottom are for the 42-th, 55-th, 79-th, 86-th, 107-th, 204-th, 283-th and 446-th filters.

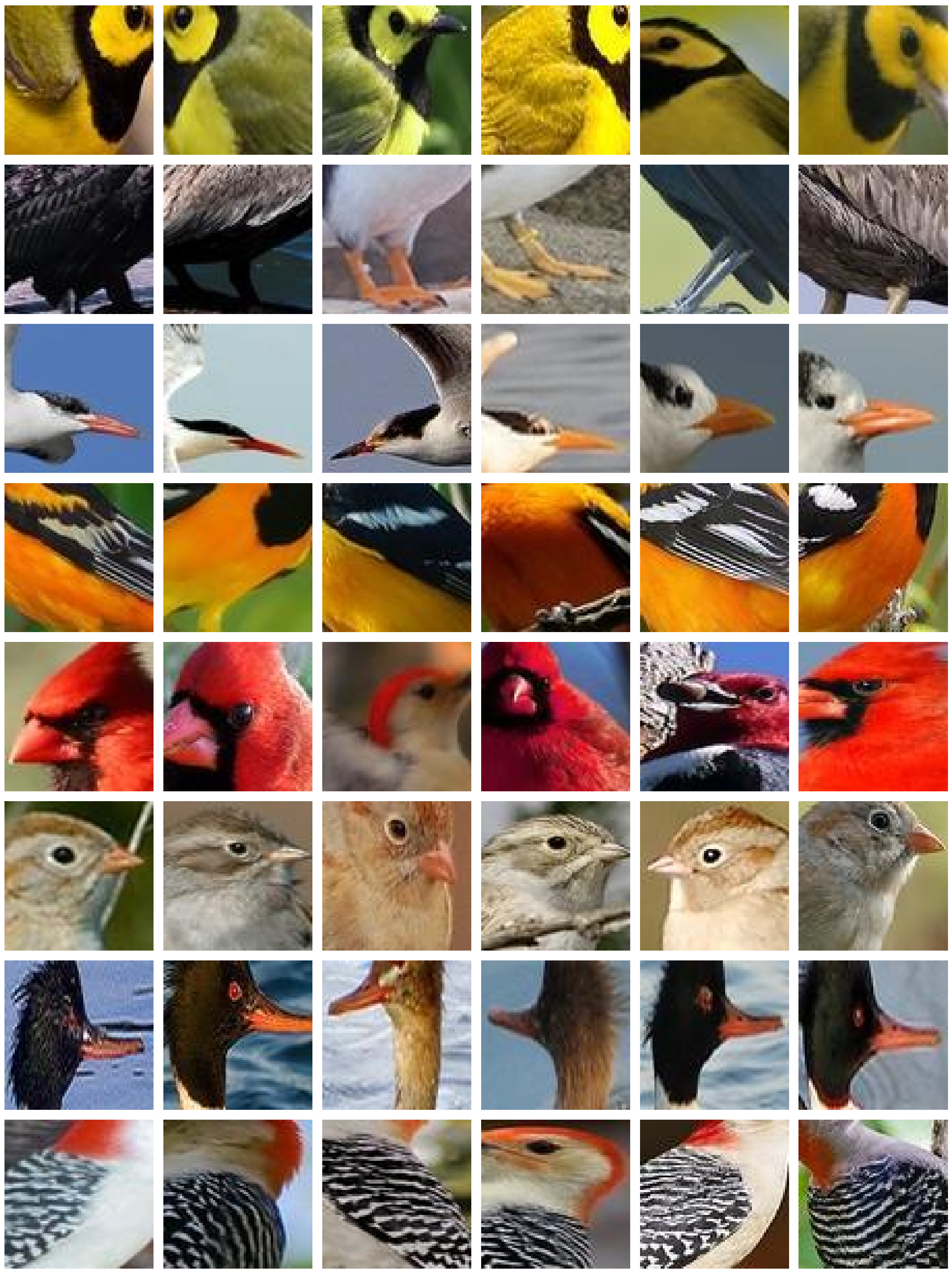

Figure 3.31: Patches with highest activations for several filters in the 69-th layer (in total 512 filters) in the fine-tuned birds ResNet101-SWP. Each row refers to one learned convolutional filter. Six patches in one row represent the top six highest activations among all 5, 794 test images. From the top row to the bottom are for the 24-th, 40-th, 43-th, 50-th, 92-th, 229-th, 311-th and 465-th filters.

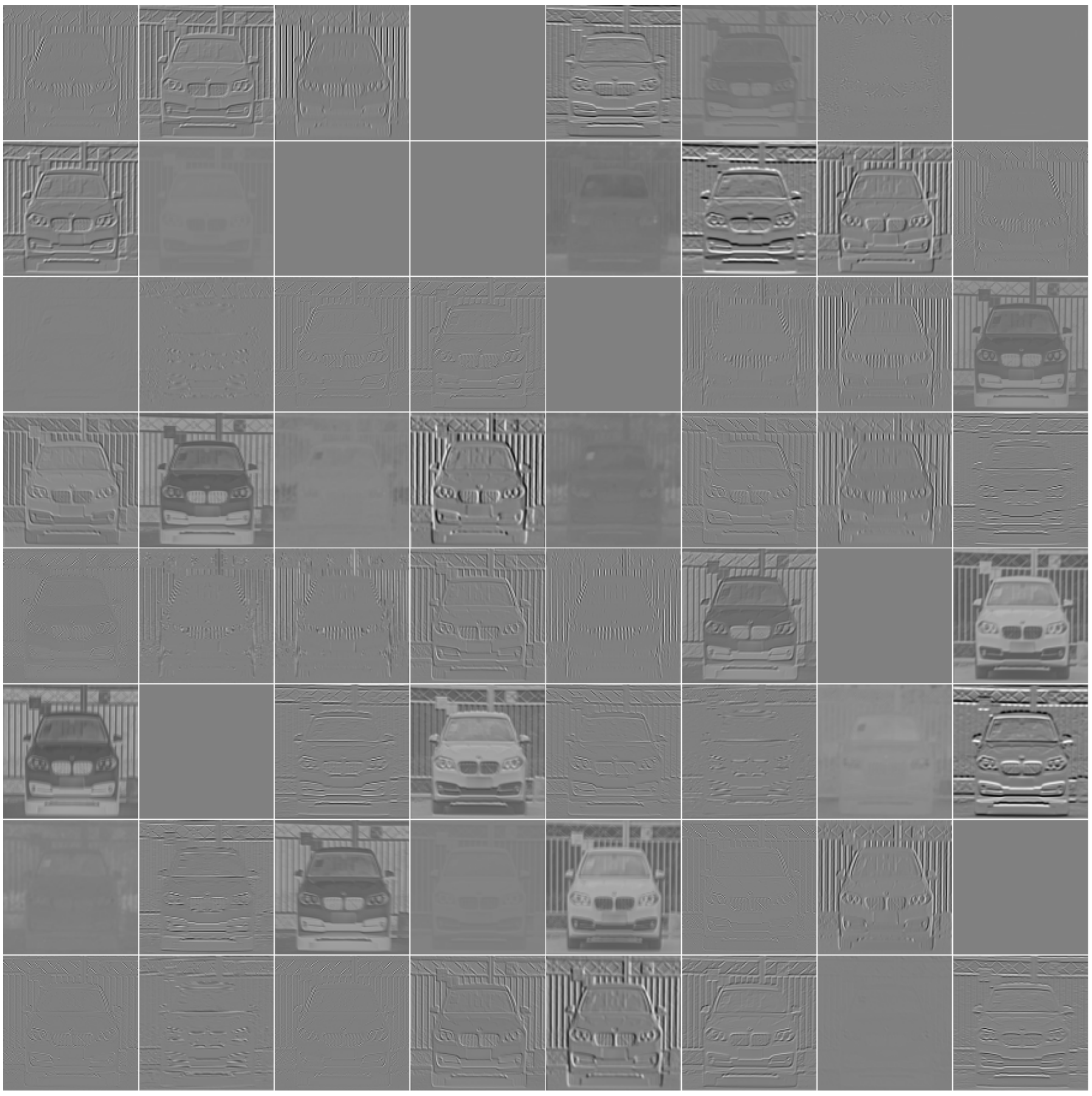

Figure 3.32: Visualizing 64 Conv1 feature maps in CompCars ResNet101-SWP.

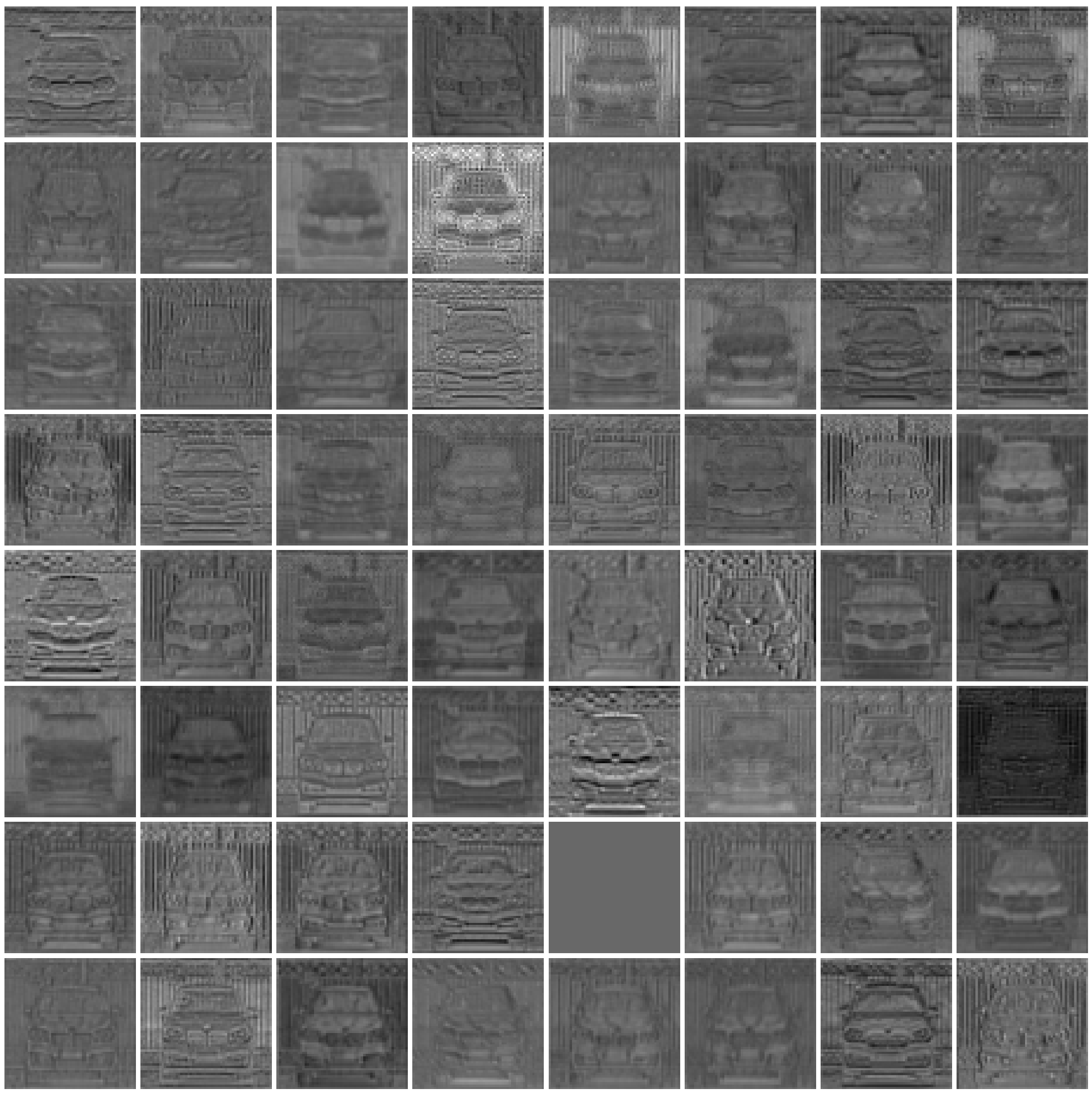

Figure 3.33: Visualizing 64 feature maps of the 6-th layer (convolution) in CompCars ResNet101-SWP.

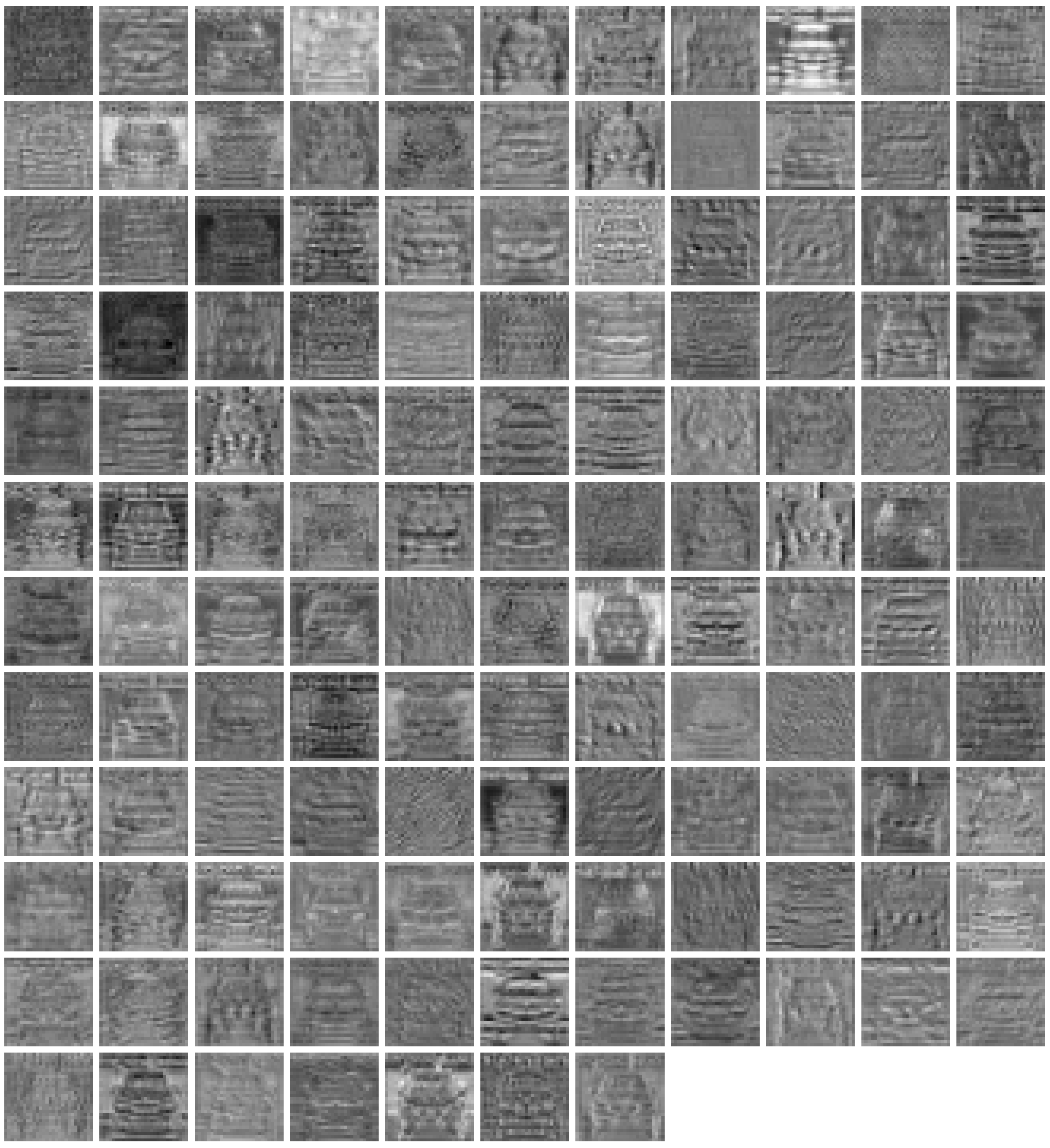

Figure 3.34: Visualizing 128 feature maps of the 18-th layer (convolution) in CompCars ResNet101-SWP.

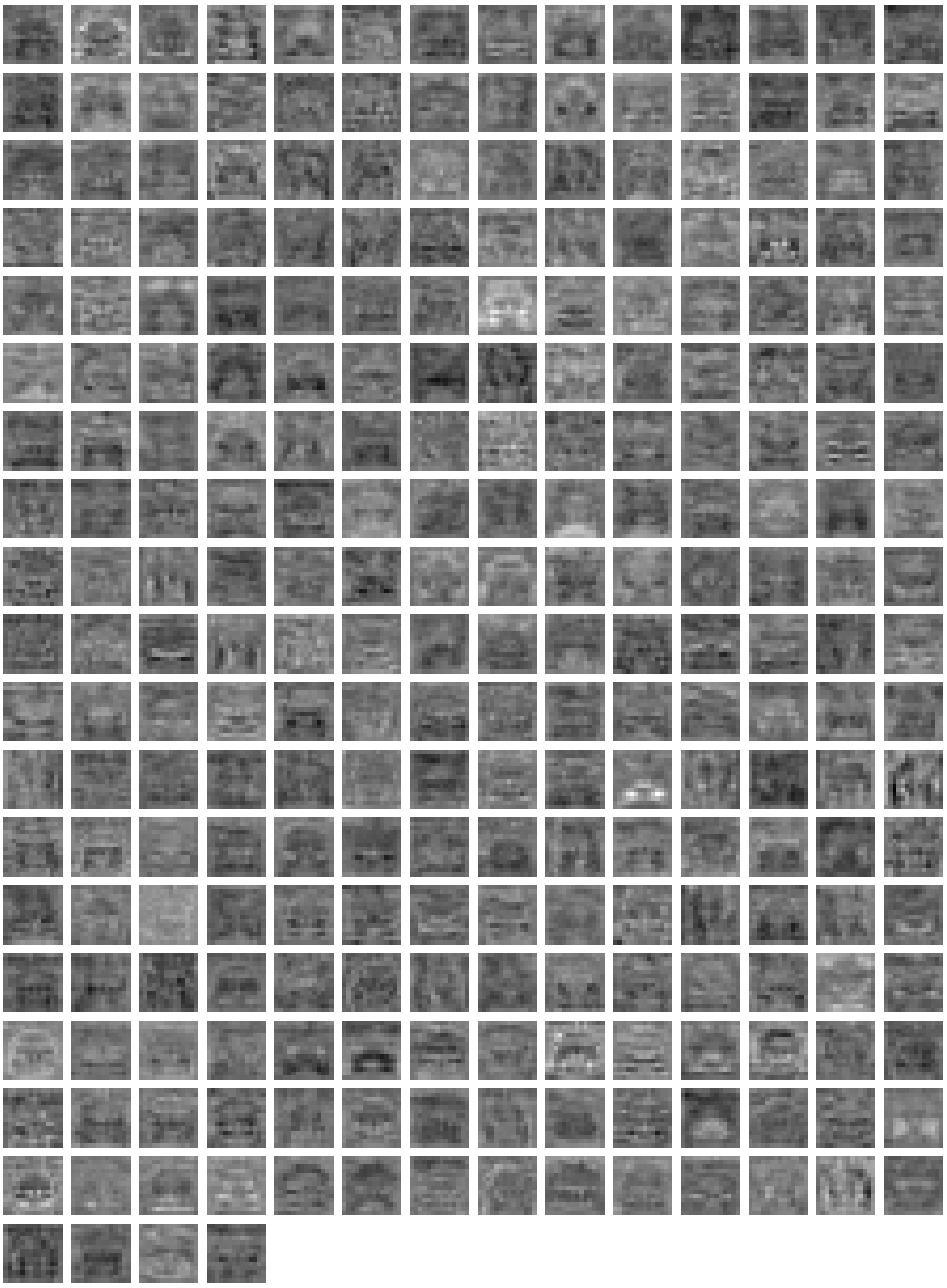

Figure 3.35: Visualizing 256 feature maps of the 69-th layer (convolution) in CompCars ResNet101-SWP.

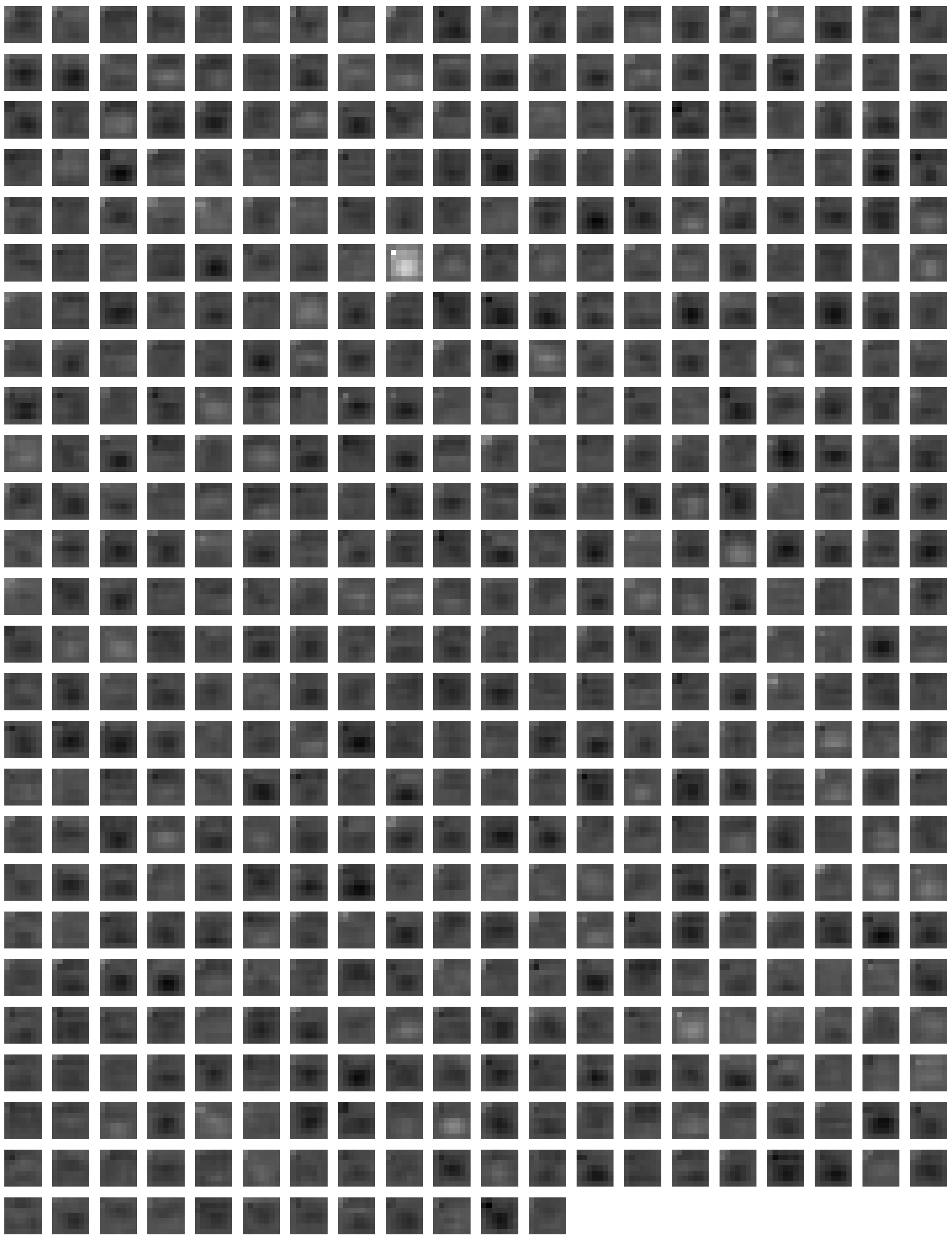

Figure 3.36: Visualizing 512 feature maps of the 99-th layer (convolution) in CompCars ResNet101-SWP.

Chapter 4

Conclusions

This thesis has presented a detailed study of one important topic in the compute-vision and machine-learning communities, *deep learning for fine-grained visual recognition*. In order to gain a deep insight into the domain of fine-grained visual recognition. Firstly, we conducted a number of experiments based on the cross-convolutional-layer pooling on the CompCars dataset. The corresponding experiments illustrate its applicability and effectiveness on this newly-designed fine-grained dataset. Meanwhile, based on above experiments, we found out that that pooling the most distinguishable regions like car logos and headlights areas, which have higher activations, in the indicator maps with the local features in the same regions can achieve better results than those by pooling the whole indicator maps with the corresponding local features. Therefore, we conjecture that if having more powerful indicator maps or pooling channels that can better highlight these distinguishable regions rather than those off-the-shelf pooling channels from one convolutional layer, better performance may be achieved.

Based on the above hypothesis, next we have developed and proposed a simple yet effective pooling strategy - Spatially Weighting Pooling to enhance fine-grained visual recognition performance. The proposed method can considerably improve the robustness and effectiveness of the feature representation of deep CNNs. More specifically, the SWP is a novel pooling strategy which contains a predefined number of spatially weighted masks or pooling channels. The SWP pools the extracted features of deep CNNs with the guidance of its learnt masks, which measures the importance of the spatial units in terms of discriminative power. It can be seamlessly integrated into any existing convolutional neural network architectures by simply adding it before fully-connected layers in them. It also allows end-to-end training with only image labels. SWP has few parameters to learn, usually in several hundreds, therefore does not introduce much computational overhead. Testing one image can be done simply by going through the trained or fine-tuned models into which an SWP layer is integrated.

We have conducted comprehensive experiments on three widely-used fine-grained datasets with a variety of deep CNN architectures such as AlexNet, VGGNet and ResNet. State-of-the-art results have been achieved on three fine-grained datasets and the MIT67 indoor scene recognition dataset. Finally, we visualize learned Conv1 and SWP filters, image patches with highest activations for several convolutional filters and a number of convolutional feature maps to gain a deep insight into the learned models.

References

[1] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," in *Technical Report CNS-TR-2011-001, CalTech*, 2011. viii, ix, 1, 2, 3, 7, 18, 21, 24, 38, 39, 40, 49

[2] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, 2013. viii, ix, 2, 7, 38, 39, 40

[3] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," in *arXiv preprint arXiv:1306.5151*, 2013. viii, ix, 2, 7, 38, 39, 40

[4] L. Yang, P. Luo, C. C. Loy, and X. Tang, "A large-scale car dataset for fine-grained categorization and verification," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015. viii, 2, 21, 22, 23, 24, 25, 27, 29, 53

[5] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2009, pp. 413–420. ix, 7, 21, 24, 38, 39, 40

[6] L. Xie, J. Wang, B. Zhang, and Q. Tian, "Fine-grained image search," in *IEEE Trans. Multimedia.*, vol. 17, no. 5, 2015, pp. 636–647. 1

[7] T. Berg and P. N. Belhumeur, "POOF: part-based one-vs.-one features for fine-grained categorization, face verification, and attribute estimation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2013, pp. 955–962. 2

[8] R. Farrell, O. Oza, N. Zhang, V. I. Morariu, T. Darrell, and L. S. Davis, "Birdlets: Subordinate categorization using volumetric primitives and pose-

normalized appearance," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2011, pp. 161–168. 2

[9] N. Zhang, R. Farrel, and T. Darrell, "Pose pooling kernels for sub-category recognition," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2012, pp. 3665–3672. 2

[10] Y. Chai, V. Lempitsky, and A. Zisserman, "Symbiotic segmentation and part localization for fine-grained categorization," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2013, pp. 321–328. 2, 17, 30

[11] J. Deng, J. Krause, and L. Fei-Fei, "Fine-grained crowdsourcing for fine-grained recognition," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2013, pp. 580–587. 2

[12] N. Zhang, J. Donahue, R. B. Girshick, and T. Darrell, "Part-based r-cnns for fine-grained category detection," *Proc. Eur. Conf. Comp. Vis.*, pp. 834–849, 2014. 2

[13] S. Branson, G. V. Horn, S. J. Belongie, and P. Perona, "Improved bird species recognition using pose normalized deep convolutional nets," in *In British Machine Vision Conference*, 2014. 2

[14] J. Krause, H. Jin, J. Yang, and L. Fei-Fei, "Fine-grained recognition without part annotations," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015, pp. 5546–5555. 2, 46

[15] H. Jegou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2010. 3

[16] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Proc. Eur. Conf. Comp. Vis.*, 2010. 3, 4

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012. 3, 13, 15, 44

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015. 3, 11, 15, 16, 46

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016. 3, 16, 17, 43

[20] L. Liu, C. Shen, and A. van den Hengel, "The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015. 3, 4, 5, 17, 21, 24, 26, 30, 32

[21] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2015, pp. 1449–1457. 3, 4, 5, 19, 20, 30, 32, 46

[22] Y. Wang, S. Li, and A. C. Kot, "Deepbag: Recognizing handbag models," in *IEEE Trans. Multimedia.*, vol. 17, no. 11, 2015, pp. 2072–2083. 3

[23] C. Huang, W. Cao, and Z. He, "Task-driven progressive part localization for fine-grained object recognition," in *IEEE Trans. Multimedia.*, 2016. 3

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, 2015, pp. 1904–1916. 3, 28, 29, 30

[25] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: An astounding baseline for recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 806–813. 4, 24, 30

[26] M. Cimpoi, S. Maji, and A. Vedaldi, "Deep filter banks for texture recognition and description," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015. 4, 30, 46

[27] A. Rosenfeld and S. Ullman, "Visual concept recognition and localization via iterative introspection," in *arXiv preprint arXiv:1603.04186*, 2016. 4, 46

[28] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016. 4

[29] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei, "The unreasonable effectiveness of noisy data for fine-grained recognition," in *Proc. Eur. Conf. Comp. Vis.*, 2016. 4

[30] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," in *Nature*, vol. 521. Nature Publishing Group, 5 2015, pp. 436–444. 9

[31] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *Int. J. Comput. Vision*, vol. 61, no. 1, pp. 55–79, Jan. 2005. 19

[32] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," in *Biological Cybernetics*, vol. 36, 1980, pp. 193–202. 9

[33] K. Fukushima and T. Imagawa, in *Original Contribution: Recognition and Segmentation of Connected Characters with Selective Attention*, vol. 6, no. 1, Jan. 1993, pp. 33–41. 9

[34] K. Fukushima, "A neural network model for selective attention in visual pattern recognition," in *Biological Cybernetics*, vol. 55, 1986, pp. 5–15. 9

[35] ——, "Analysis of the process of visual pattern recognition by the neocognitron," in *Neural Networks*, vol. 2, no. 6, 1989, pp. 413–420. 9

[36] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan Kaufmann Publishers Inc., 1990, pp. 396–404. 9

[37] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1990, pp. 393–404. 11

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” in *Proceedings of the IEEE*, 1998, pp. 2278–2324. 11

[39] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, “Toward automatic phenotyping of developing embryos from videos,” in *IEEE Transactions on Image Processing*, vol. 14, no. 9, 2005, pp. 1360–1371. 11

[40] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. Lecun, “Pedestrian detection with unsupervised multi-stage feature learning,” in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2013. 11

[41] R. Vaillant, C. Monrocq, and Y. LeCun, “Original approach for the localisation of objects in images,” in *IEE Proc on Vision, Image, and Signal Processing*, vol. 141, no. 4, August 1994, pp. 245–250. 11

[42] S. J. Nowlan and J. C. Platt, “A convolutional neural network hand tracker,” in *Proc. Adv. Neural Inf. Process. Syst.*, 1994, pp. 901–908. 11

[43] M. Delakis and C. Garcia, “Convolutional face finder: A neural architecture for fast and robust face detection,” in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, 2004, pp. 1408–1423. 11

[44] M. Osadchy, Y. LeCun, and M. L. Miller, “Synergistic face detection and pose estimation with energy-based models,” in *J. Mach. Learn. Res.*, vol. 8, may 2007, pp. 1197–1215. 11

[45] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, “Efficient object localization using convolutional networks,” *CoRR*, vol. abs/1411.4280, 2014. 11

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2009. 11, 25

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," in *J. Mach. Learn. Res.*, vol. 15, no. 1. JMLR, 2014, pp. 1929–1958. 11, 12, 15

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015. 11, 13, 43, 56

[49] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, June 2014. 11

[50] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *International Conference on Learning Representations (ICLR2014)*. CBLS, April 2014. 11, 15, 25

[51] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. 11

[52] Y.-L. Boureau, J. Ponce, and Y. Lecun, "A theoretical analysis of feature pooling in visual recognition," in *27TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, HAIFA, ISRAEL*, 2010. 12

[53] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *CoRR*, vol. abs/1608.04363, 2016. 13

[54] A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *CoRR*, vol. abs/1312.5402, 2013. 13

[55] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015. 14, 17, 41, 43

[56] A. Krizhevsky, "Learning multiple layers of features from tiny images," in *Tech Report*, 2009. 16

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2015. 17

[58] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," in *Int. J. Comp. Vis.*, vol. 88, no. 2, 2010, pp. 303–338. 21, 24, 28

[59] L. Bourdev, S. Maji, and J. Malik, "Describing people: a poselet-based approach to attribute classification," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2011. 21, 24

[60] H. Azizpour, A. Sharif Razavian, J. Sullivan, A. Maki, and S. Carlsson, "From generic to specific deep representations for visual recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2015. 24

[61] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *arXiv preprint arXiv:1408.5093*, 2014. 26

[62] M. Elhoseiny, S. Huang, and A. Elgammal, "Weather classification with deep convolutional neural networks," in *Proc. IEEE Int. Conf. Image Process.*, 2015. 28, 30

[63] C. Lu, D. Lin, J. Jia, and C.-K. Tang, "Two-class weather classification," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2014. 28

[64] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, 2014, pp. 818–833. 29, 59

[65] P. Lamblin and Y. Bengio, "Important gains from supervised fine-tuning of deep architectures on large labeled sets," in *JMLR: Workshop and Conference Proceedings 27:17-37 Workshop on Unsupervised and Transfer Learning*, 2012. 30

[66] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell, “Compact bilinear pooling,” in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016. 34

[67] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, “Return of the devil in the details: Delving deep into convolutional nets,” in *Proc. Brit. Mach. Vis. Conf.*, 2014. 46