Towards Reliable CNN Architecture Design for Visual Recognition

Lida Li

PhD

The Hong Kong Polytechnic University

2021

The Hong Kong Polytechnic University

Department of Computing

Towards Reliable CNN Architecture Design for Visual Recognition

Lida Li

A thesis submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy

February 2021

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signed)

LIDA LI (Name of student)

Abstract

While several popular network architectures have been developed and widely used, it remains an important topic to design effective and efficient convolutional neural network (CNN) architectures for visual recognition. The design of reliable CNN architectures faces three main challenges, including how to reduce the computational cost, how to improve the accuracy, and how to enhance the robustness against adversarial attacks. In this thesis, we study the design of reliable CNN architectures for visual recognition.

In Chapter 1, we review some common CNN architectures and their design methods for visual recognition, and discuss contribution and organization of this thesis. In Chapter 2, we present a detachable second-order pooling network to improve the performance of first-order CNNs in image classification while keeping the same computational cost at testing stage. In Chapter 3, we propose to train deep CNNs with a learnable sparse transform (LST), which learns to convert the input features into a more compact and sparser domain together with the CNN training process. The proposed LST is more effective in reducing the spatial and channel-wise feature redundancies than the conventional Conv2d, and it can be efficiently implemented with existing CNN modules for seamless training and inference. We also present a hybrid LST-ReLU activation to enhance the robustness of the learned CNN models. In Chapter 4, we further improve LST to faithfully build CNNs for visual recognition. The proposed LST v2 employs hierarchical depth-wise separable convolution

to allow incomplete yet flexible expansion. LST v2 can achieve comparable or even higher accuracy than LST-Net in a wide range of visual recognition tasks. Finally, in Chapter 5, we study the application of LST to adversarial attacks. A robust convolutional layer with multiple kernels, namely RConv-MK, is proposed to improve the robustness of LST against various types of image corruptions and manually designed adversarial attacks.

In summary, in this thesis we present four reliable CNN architecture design methods, including a detachable second-order pooling network, a learnable sparse transform and its improved version, and a robust convolutional layer. Extensive experiments demonstrate their effectiveness and efficiency for accurate, lightweight and robust visual recognition.

Keywords: Convolutional neural network, architecture design, learnable sparse transform, visual recognition, knowledge distillation, adversarial attack

Biography

Journal Papers

1. **Lida Li**, Jiangtao Xie, Peihau Li, and Lei Zhang. Detachable second-order pooling: Towards high performance first-order networks. Accepted to *IEEE Trans. Neural Netw. Learn. Syst.*, 2021, doi: 10.1109/TNNLS.2021.3052829.

2. **Lida Li**, Kun Wang, Shuai Li, Xiangchu Feng, and Lei Zhang. Learning a fast and lightweight transform with hierarchical depth-wise separable convolution. Submitted to *IEEE Trans. Pattern Anal. Mach. Intell.*, 2021.

Conference Papers

1. **Lida Li**, Kun Wang, Shuai Li, Xiangchu Feng, and Lei Zhang. LST-Net: Learning a convolutional neural network with a learnable sparse transform. In *Eur. Conf. Comput. Vis*, 2020.

2. **Lida LI**, Kun Wang, Shuai Li, Xiangchu Feng, and Lei Zhang. Towards Robust 2D Convolution for Reliable Visual Recognition. Submitted to *Int. Conf. Comput. Vis.*, 2021.

Acknowledgements

First of all, I want to express my deepest gratitude to my supervisor, Prof. Lei Zhang, for his incredible guidance, support, and generosity. It has been almost a decade since I first knew Prof. Zhang in his seminar when I was a sophomore. I am deeply impressed by the tremendous work from his Visual Computing Lab as well as the lasting commitment to serving the society with their research output. It is my great pleasure to join this group after five years and spend four wonderful years with so many intelligent, innovative and lovely souls at The Hong Kong Polytechnic University.

I also want to express my gratitude to Dr. Zisheng Cao and other staff with the imaging group of DJI Co., Ltd. During my internship there, they gave me many helpful suggestions from the perspective of industry. Besides, I want to express my gratitude to Prof. Lin Zhang, Dr. Ying Shen and other Visual Computing Group members from Tongji University, Prof. Peihua Li, Mr. Jiangtao Xie and Ms. Zilin Gao from Dalian University of Technology, Prof. Xiangchu Feng and Mr. Kun Wang from Xidian University, Ms. Anqi Yang from Carnegie Mellon University, as well as Ms. Fangrui Zhu from Fudan University for the meaningful and fruitful discussion. I am fortunate to have an opportunity to learn from them.

Finally, I want to thank my family for supporting me to pursue my PhD degree. I want to thank them for their endless love and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional neural network (CNN) has been discovered effective in numerous visual recognition tasks. Thanks to the continuous development of parallel processing devices, the need for reliable CNNs has never stopped in the trend of deep learning. Bolstered by the requirements of numerous applications, such as healthcare, entertainment, security monitoring, environmental protection, and autonmous driving, it has become one of the most heated topics today how to design effective and efficient CNN architectures.

Design of reliable CNN architectures is confronted with three main challenges. First, compared with conventional methods depending on handcrafted features, CNNs always require more parameters and overhead. Though more effective, CNN based methods require more hardware resources for data storage and computation in practice. Besides, large bandwidth technologies are expected to reduce latency of data communication. Both clearly increase the hardware cost. Second, performance of CNNs can still be improved in many applications to reduce waste and protect the environment (*e.g.*, to reduce release of carbon dioxide), to gain profit, to boost production, *etc*. One should note that the performance gap between CNNs and human beings still remains large in many cases. Third, CNNs are vulnerable under adversarial attacks. A well-trained CNN model can be easily fooled to produce wrong

outputs by deliberately constructed inputs that are imperceptible to human beings. Unfortunately, there is little discussion on this issue from the perspective of CNN architecture.

In this thesis, we study reliable CNN architecture design for visual recognition. Here, we are convinced that the *reliablity* of CNN architecture design has at least two meanings. First, it refers to higher accuracy or lower error rates in the fields of computer vision and pattern recognition. Second, we also focus on the robustness of CNN architectures when we are confronted with various types of noises in the real world as well as some deliberately designed attacks. To tackle the above three challenges, we are motivated to identify the drawbacks of existing methods to reduce the cost while boosting the performance. In addition, we discuss reliable CNN architecture under adversarial attacks. The remainder of this section is organized as follows. In Section 1.1, we brief common components of CNNs and some modern CNN architectures. In Section 1.2, we discuss some existing CNN architecture design methods for visual recognition. Finally, we summarize our contributions and the structure of this thesis in Section 1.3.

1.1 Overview of Convolutional Neural Network Architectures

1.1.1 2D convolutional layer

The 2D convolutional layer (Conv2d) is the core building block of a CNN. Given a set of hyper-parameters, including kernel size, input channels, output channels, stride, padding, dilation, *etc.*, the convolutional layer extracts features from a 3D tensor. Conv2d is found redundant according to its definition. To save parameters and overhead of Conv2d layers, group convolution [90] (GConv) and PWConv [105] are popularly employed in the design of bottlenecks. PWConv employs a 1×1 win-

dow, performing a linear combination of the input from all channels. It is often used to align a set of feature maps with different number of channels [154]. GConv assumes that the input features can be decomposed into several groups along the channel dimension, where features from different groups are independent. A successful application of GConv is ResNeXt [186]. DWConv [68] is a special case of GConv when there is only one input channel per group. It is widely used to build lightweight models for mobile devices, such as MobileNet [68, 149]. In addition, some proposed to improve Conv2d layers for better feature extraction. Bello *et al*. [7] introduced a novel two-dimensional relative self-attention mechanism to augment conventional Conv2d. Hu *et al*. [69] adaptively determined aggregation weights based on the compositional relationship of local pixel pairs for more effective extraction of spatial information. Wang *et al*. [165] factorized 2D self-attention into two 1D self-attentions and developed a position-sensitive self-attention to largely reduce both overhead and number of parameters. Chen *et al*. [28] designed dynamic convolution to adaptively generate convolution kernels based on image contents.

1.1.2 Pooling layer

A pooling layer performs dimensionality reduction of spatial cues. In this way, it helps to reduce spatial size of features for faster computation, lower down the risk of overfitting, and increase a model's invariance to spatial transform, such as translation, rotation and scaling. Max pooling and average pooling are the most popular pooling layers to build up CNNs.

A global pooling layer performs a more extreme type of dimensionality reduction, removing both height and width dimensions. A global pooling method can be roughly divided into two categories, including first- and high-order global pooling ones. First-order global pooling methods apply a unary operator to each feature map and concatenate all outputs as the final output. Lin *et al*. [105] first

performed Global Average Pooling (GAP) in a network by averaging final convolutional features to obtain a vector descriptor. Thanks to this design, the cost of high-dimensional dense layers in networks such as AlexNet [90] and VGGNet [151] can be largely reduced. GAP is widely adopted in mainstream CNN architectures, including ResNet [59], DenseNet [72], ResNeXt [186], MobileNet [68], and Inception networks [155, 81, 156]. Statistically, GAP summarizes the first-order statistics (*i.e.*, mean) of high-level convolutional features, neglecting the higher-order statistics. High-order global pooling algorithms aim at more discriminative image representation. Most works in this category exploit pairwise correlations between channels while some others, *e.g.*, [11] , further consider higher-order interactions of features. Bilinear CNN (B-CNN) [109, 110] and DeepO_2P [82] are pioneering works. Both of them compute covariance matrix (or second-order moments) as the global image representations. MPN-COV [100] and its fast version [99] (*i.e.*, iSQRT-COV) have reported compelling performance on large-scale visual recognition and fine-grained classification, significantly outperforming the first-order networks.

1.1.3 Non-linear activation layer

In CNNs, an activation function decides whether a neuron should be activated or not. In this way, non-linearity is introduced to the network, which is believed as one of the most critical factors to the success of a CNN model on different computer vision tasks. ReLU [131] is a pioneer and the most popular non-linear activation function in deep CNN. It is a simple yet highly effective segmented function, forcing the input negative valued features to zeros and keeping only the non-negative features. Parametric ReLU [58], leaky ReLU [169], ELU [31] and SELU [87] consider non-negative features by allowing adaptive negative activation with learnable parameters. Swish [140] is defined as the multiplication of input features and their sigmoid activation, working better than ReLU on deeper models across several challenging datasets. ReLU6 [68]

is a modification of ReLU where an upper limitation of the activation is set to 6 to increase robustness of a lightweight model in low-precision computation.

1.1.4 Modern CNN architectures

We briefly review the development of modern CNN architectures in recent years. Alex *et al*. [90] were the first to train a deep convolutional neural network, namely AlexNet, and obtained considerably better results than the previous state-of-the-art on ImageNet [35]. Simonyan and Zisserman [151] discovered that increase of depth improves performance of CNNs using an architecture with 3×3 convolution filters in the large-scale image recognition. Inception architectures [155, 81, 156, 154] are carefully designed to increase the depth and width of the network while keeping the computational cost constant by using distinct filters and 1×1 convolutions. To make training easier and improve generalization, He *et al*. [59] proposed a milestone residual unit and used it to build hundreds or even one thousand layers of neural networks, called ResNet, where both forward and backward signals can be directly propagated from one block to others. Zagoruyko *et al*. [198] developed wide residual networks by simply decreasing depth and increasing width of residual networks, which demonstrate far superiority over the thin and very deep counterparts on image classification. Xie *et al*. [186] constructed ResNeXt by repeating a building block that aggregates a set of homogeneous and multi-branch formed transformations with the same topology, achieving clear performance boost on image classification under the restricted condition of maintaining complexity of ResNet. Gao *et al*. [48] constructed hiearchical residual-like connections within one single residual block to build CNNs called Res2Net, which demonstrated consistent performance gains on a wide range of visual recognition tasks. Recently, there is a growing interest in building light weight deep neural networks for resource limited devices, such as smart phones, cameras, drones, *etc*. These architectures feature fewer parameters and lower overhead but

they are designed to achieve competitive results on many visual recognition tasks, *e.g.*, image classification, object detection and segmentation. Some representative works include MobileNet [68, 149, 67], ShuffleNet [204, 119], FBNet [179], HBONet [91], ESPNet [127, 128], DiCENet [126], EfficientNet [159], TinyNet [54], *etc.*

1.2 Existing CNN Architecture Design Methods for Visual Recognition

1.2.1 Knowledge distillation

Knowledge distillation is an effective approach to train powerful CNNs. It transfers the knowledge from more powerful entities into weaker ones for inference [64, 52]. Hinton *et al.* [64] proposed to transfer knowledge from an ensemble of acoustic models into a smaller and distilled one for easier deployment. Gupta *et al.* [52] transferred representations from a well labeled domain to an unseen domain. Yim *et al.* [193] introduced a sequential flow between layers to distill knowledge. Furlanello *et al.* [46] trained student models parameterized identically to their teachers so that students can even outperform their teachers in some small scale vision tasks. Guo *et al.* [51] treated all CNNs as student models and generated soft target as supervisions during training by ensembling predictions of all student models and distorting the input images. Zhang *et al.* [205] leveraged a small trusted set to estimate exemplar weights and pseudo label for noisy data so that they can be reused for supervised training of neural networks. Li *et al.* [93] exploited gradient cues for knowledge distiallation and jointly trained both plain CNNs without shortcuts and their ResNet counterpart to deploy shortcut-free models. In this way, entire memory usage can be greatly reduced during inference.

While existing knowledge distiallation methods usually rely on a pre-trained, high performance teacher network, and skillful design of metrics, in this thesis, we

introduce a simple yet effective regularization term by applying the same criterion to both student and auxiliary teacher networks, so that knowledge distiallation can be induced with no extra metric.

1.2.2 Self-attention modules

Self-attention mechanism enables effective capture and fusion of global cues in deep CNNs. In recent years, we have witnessed a growing interest in the development of self-attention modules to improve conventional Conv2d on various visual recognition tasks. Wang *et al*. [170] proposed non-local neural network as a flexible building block that fuses both non-local and local information for intermediate layers of CNNs. SENet [71] constructs reliable features by fusing both spatial and channel-wise information within local receptive fields. CBAM [178] improves spatial attention of SENet by considering two types of local statistics. GE framework [70] aggregates feature responses from a large spatial extent and redistributes the pooled information to local features for better performance. GALA [111] extends SENet by combining visual saliency. Li *et al*. [102] exploited multiple branches with different kernel sizes for information fusion. Cao *et al*. [16] designed a simplified yet effective way to model global contexts, which generally outperforms [170, 71]. LCT block [148] enhances the performance of SENet on image recognition and object detection by capturing dependencies between channels and linearly transforming the global context of each channel. Wang *et al*. [168] developed a local cross-channel interaction method free of dimensionality reduction and validated its efficiency and efffectiveness. Recently, DCA-Net [120] interconnects adjacent attention blocks to boost attention modules.

While self-attention modules have obtained impressive results on visual recognition tasks, they work as a patch of conventional Conv2d, with the need for additional parameters and computational overhead during both training and inference stages. In this thesis, we introduce learnable sparse transform as an alternative of conven-

tional Conv2d to internally perform fusion of channel and spatial cues, while we can achieve comparable or even better results on a wide range of visual recognition tasks.

1.2.3 CNN architecture design in frequency domain

Compared with spatial domain, frequency domain features compactness and sparsity for reliable visual recognition. We review CNN architecture design in frequency domain in three directions. First, Fourier transforms and the inverse are employed to compute convolutions at every convolutional layer, similar to many classical harmonic analysis works, *e.g.* discrete cosine transform [174] and discrete wavelet transform [62, 145, 20]. Some representative works can be found in [146, 135]. Even though methods of this category are more effective in feature extraction, they suffer from intensive compuatation due to back-and-forth transformations. Second, lightweight modules are developed based on frequency transforms and inserted into existing CNN architectures to improve conventional Conv2d. Qin *et al.* [136] designed a multi-spectral channel attention module to perform pre-processing in the frequency domain. Alizadeh vahid *et al.* [1] extended the butterfly operations from FFT algorithm to a general Butterfly Transform and designed a set of criterion for channel fusion to reduce the computational complexity of point-wise convolutions. However, methods in this category usually require additional parameters and overhead. Third, a specific layer is introduced before the main CNN to perform necessary computation of given images in frequency domain. Shipitsin *et al.* [150] learned to select necessary frequencies of input images from the frequency domain for its subsequent CNN model, leading to clear performance boost on classification, segmentation, and other low-level computer vision tasks. Yang *et al.* [191] performed domain adaption on semantic segmentation just using a Fourier Transform and its inverse and achieved state-of-the-art performance on several benchmarks. Works in this line must be aware of the input image size and they use kernels of the same size, leading to two

main drawbacks in real applications: (a) the extra parameters account for a large proportion of the total parameters of a CNN, since the input image size is usually of high resolution; (b) it is not flexible to deal with input images of different size.

In this thesis, we introduce reliable CNN architecture design in frequency domain. Our methods basically follow conclusions of classical harmonic analysis works and we adopt the form of the methods in the first category to save cost. Instead of using fixed Fourier transforms and the inverse, we initialize spatial and feature transforms as the DCT and make them fully learnable in an end-to-end training fashion.

1.2.4 Neural Architecture Search

Neural Architecture Search (NAS) [211, 212, 134, 40] is a heated direction of CNN architecture design to automate the process of model design. By applying automated machine learning techniques, it usually outperforms human-designed models in terms of number of parameters, overhead and accuracy. For completeness, we briefly review some representative works in this line. Early efforts sampled each architecture from a heuristic search space via reinforcement learning [6, 211, 212] or evolutionary algorithms [142, 141, 185] and individually trained it from scratch for performance evaulation. Though effective, these methods were slow as they needed to evaluate a large number of candidate architectures, which usually cost hundreds to thousands of GPU days in total. To reduce computational cost, Cai *et al.* [9] explored the architecture space based on the current network and reusing its weights for similar architectures sampled from the search space. Pham *et al.* [134] constructed a large computational graph to force all architectures to share their parameters and trained a controller to search for an optimal neural network architecture. Tan *et al.* [158] considered NAS for inference on mobile devices by designing a factorized hiearchical search space to encourage layer diversity of the network. Liu *et al.* [112] formulated NAS in a differentiable manner based on continuous relaxation of archi-

tecture representation to discover high-performance CNN architectures for various visual recognition tasks.

In this thesis, we focus on manual CNN architecture design methods for visual recognition. We strive to avoid performance bias caused by any explicit or implicit architecture changes as much as possible and make our methods truly complementary to most existing NAS approaches.

1.3 Contribution and Thesis Organization

This thesis is mainly consisted of four works we have done during the PhD study.

In the first work: We propose a novel method, namely DSoP-Net, to improve the performance of first-order CNNs in image classification. Auxiliary branches are carefully designed to transfer knowledge to the backbone first-order networks during training, which are removable at the testing stage. As a result, the proposed method leverages the advantages of second-order pooling networks while keeping similar complexity to first-order networks during inference. To the best of our knowledge, this is the first attempt to make use of higher-order statistics in knowledge distillation. This work will be introduced in Chapter 2.

In the second work: We learn to convert the input features into a more compact and sparser domain together with the CNN training process by training deep CNNs with a learnable sparse transform (LST). LST can more effectively reduce the spatial and channel-wise feature redundancies than the conventional Conv2d. It can be efficiently implemented with existing CNN modules, and is portable to existing CNN architectures for seamless training and inference. We further present a hybrid ST-ReLU activation to enhance the robustness of the learned CNN models to common types of corruptions in the input. This work will be introduced in Chapter 3.

In the third work: We develop a fast and lightweight transform with hiearchical

DWConv (HDWConv) based on LST-Net to reduce the redundancy of the conventional Conv2d. To produce a compact feature bank, we allow incomplete yet flexible expansion so that the structure of HDWConv can be completely determined before training. High frequency input channels are expanded more times for near-complete expansion, while low frequency ones are expanded fewer times to save cost. In addition, we partition input channels into groups and assign one or more unique kernels with identical initialization for better representation. This work will be introduced in Chapter 4.

In the fourth work: We discuss the application of LST to adversarial attacks. A novel structure, namely RConv-MK, is proposed to improve the architectural defects of LST. It employs a set of kernels of different size and flexibly applies them to the input features of different frequencies. It enlarges the receptive fields for low frequency features and saves the overhead for sparse high frequency features. Besides, we introduce a normalized soft thresholding operator to adaptively address input samples with different corruption scales for removal of noise and trivial features in the relevant feature domain. This work will be introduced in Chapter 5.

The remainder of this thesis is organized as follows: from Chapter 2 to Chapter 5, we introduce the above four works in order; in Chapter 6, we conclude this thesis and present some future directions.

Chapter 2

Detachable Second-order Pooling: Towards High Performance First-order Networks

In this chapter, we will study how to improve the performance of first-order CNNs in image classification. Second-order pooling has proved to be more effective than its first-order counterpart in visual classification tasks. However, second-order pooling suffers from the high demand of computational resource, limiting its use in practical applications. In this work, we present a novel architecture, namely detachable second-order pooling network, to leverage the advantage of second-order pooling by first-order networks while keeping the model complexity unchanged during inference. Specifically, we introduce second-order pooling at the end of a few auxiliary branches and plug them into different stages of a convolutional neural network. During the training stage, the auxiliary second-order pooling networks assist the backbone first-order network to learn more discriminative feature representations. When training is completed, all auxiliary branches can be removed and only the backbone first-order network is used for inference. Experiments conducted on CIFAR-10, CIFAR-100 and ImageNet datasets clearly demonstrate the improved performance of our network, which achieves even higher accuracy than second-order networks but keeps the low

inference complexity of first-order networks.

2.1 Introduction

Deep convolutional neural networks (CNNs) have been widely used in tackling various computer vision problems, including visual object recognition [90, 41, 207], face recognition [74, 83, 15], person re-identification [29, 206, 190] and scene understanding [184, 96, 209], among others. Tremendous efforts have been devoted to the design of CNN architectures for boosting performance. It is widely acknowledged that deeper and/or wider networks, such as ResNet [59], Inception [154] and ResNeXt [186], could have higher representation learning capability. However, the increase of network depth/width will also bring more overhead and difficulties for network deployment.

Another factor affecting the learning capability of neural networks is the pooling strategy. In recent years, global second-order pooling (GSoP) networks [109, 108, 167, 100, 110, 99] have attracted a lot of attentions. By replacing the classical global average pooling (GAP) with covariance pooling at the end of CNNs, significant improvement has been reported on large-scale visual recognition tasks. For example, ResNet-50 with GSoP surpasses ResNet-152 [99]. The GAP [105] calculates the first-order statistics (*i.e.*, mean) of individual channels without considering the interactions between channels, while the global covariance pooling computes the second-order statistics of high-level convolutional features by exploiting the pair-wise channel correlations, leading to stronger statistical modeling capability. Though the use of covariance matrix to represent image statistics enhances the nonlinear learning capability of networks, the required computational complexity increases quadratically, significantly higher than its first-order counterpart.

Either increasing the width/depth of networks or employing covariance pool-

ing will consume much more computational resources. One interesting question is whether we can leverage the advantage of second-order pooling in the first-order networks while keeping the model complexity unchanged. This work attempts to solve this challenging problem. Inspired by the knowledge distillation method [64], we propose a novel architecture, called detachable second-order pooling network (DSoP-Net), where the covariance pooling networks assist the first-order network to learn more discriminative representations during training; however, during the inference stage, the covariance pooling networks can be removed and only the trained first-order network is deployed. The proposed DSoP-Net achieves significant performance gains without introducing any additional cost. In particular, on the large-scale ImageNet dataset [90], DSoP-Net achieves a top-1 error rate of 21.15% with a single ResNet-50 network.

The key idea of DSoP-Net lies in that a weak pooling method (student) can learn from stronger ones (teachers). Existing methods of this kind, such as knowledge distillation [64, 52, 193], often explicitly minimize the discrepancy between features produced by one or more teacher networks and the student network. The success of such methods largely relies on a pre-trained, high performance teacher network as well as the skillful design of metrics to measure the discrepancy so that the knowledge can be well transferred. In contrast, in this chapter we employ a simple yet effective regularization term by applying the same criterion used in the original first-order pooling to the second-order one. As a result, no extra metric is needed and the knowledge induced by covariance pooling can easily flow into the first-order network.

Figure 2.1 presents an overview of DSoP-Net. During training, auxiliary branches are employed, each with a covariance matrix based second-order pooling and an output header. This allows us to learn spatial information at intermediate layers by adjusting the channel correlations with deep supervision. Once these auxiliary branches are plugged into the backbone architecture at different stages, they actively

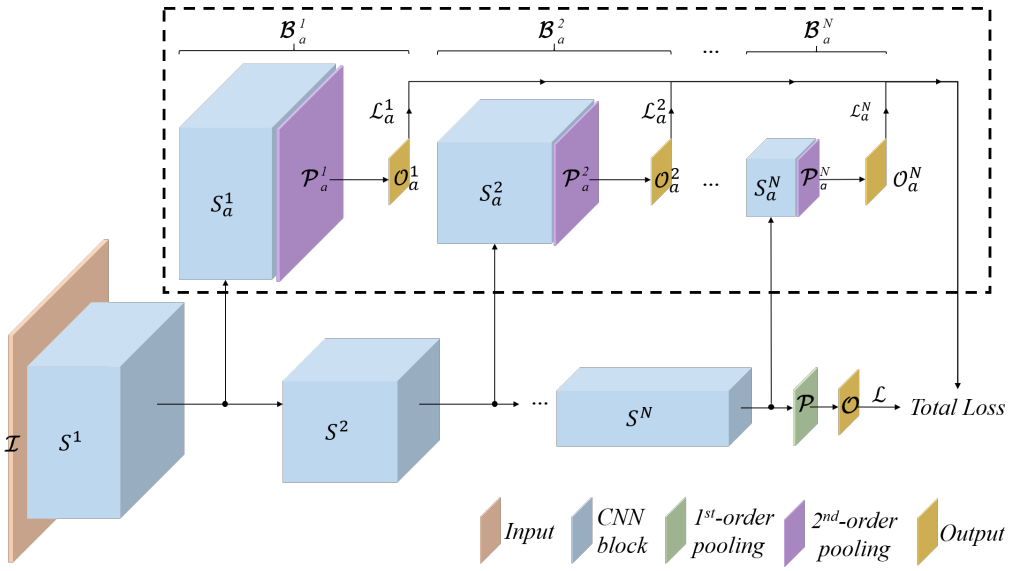


Figure 2.1: The proposed auxiliary second-order pooling networks (subnetworks within the dashed rectangular box) assist the discriminative representation learning of the backbone first-order network. For deployment, we obtain a high-performance first-order network by detaching the auxiliary branches.

cooperate with the first-order pooling and its corresponding output header. The network is optimized with the loss function that is composed of the first-order output header and all extra second-order ones. When training is completed, all auxiliary branches are removed and only the backbone first-order network is preserved for inference. The resulting DSoP-Net is very powerful, even outperforming its second-order counterparts but with much lower model complexity.

The rest of the chapter is organized as follows. Section 2.2 reviews some related works. Section 2.3 and Section 2.4 introduce our methods and implementation details, respectively. Section 2.5 shows experimental results of DSoP-Net and Section 2.6 concludes this chapter.

2.2 Related Work

Section 2.2.1 briefly reviews global pooling methods in the literature. Section 2.2.2 presents the use of auxiliary networks in the training of backbone network. In Section 2.2.3, we discuss squeeze-and-excitation networks and its differences from our DSoP-Net.

2.2.1 Global pooling

The CNN models learn discriminative features in an end-to-end manner. At the end of the network, a global pooling of convolutional features is often performed to represent the whole image for classification. We briefly review the first-order pooling methods [90, 105] and the high-order ones [109, 110, 100, 108, 167, 99, 11], respectively.

First-order global pooling methods apply a unary operator to each feature map and concatenate all outputs as the final output. Lin *et al*. [105] first performed Global Average Pooling (GAP) in a network by averaging final convolutional features to obtain a vector descriptor. Thanks to this design, the cost of high-dimensional dense layers in networks such as AlexNet [90] and VGGNet [151] can be largely reduced. GAP is widely adopted in mainstream CNN architectures, including ResNet [59], DenseNet [72], ResNeXt [186], MobileNet [68], and Inception networks [155, 81, 156]. Statistically, GAP summarizes the first-order statistics (*i.e*., mean) of high-level convolutional features, neglecting the higher-order statistics.

High-order global pooling algorithms aim at more discriminative image representation. Most works in this category exploit pairwise correlations between channels while some others, *e.g*., [11] , further consider higher-order interactions of features. Bilinear CNN (B-CNN) [109, 110] and DeepO_2P [82] are pioneering works. Both of them compute covariance matrix (or second-order moments) as the global image

representations. MPN-COV [100] and its fast version [99] (*i.e.*, iSQRT-COV) have reported compelling performance on large-scale visual recognition and fine-grained classification, significantly outperforming the first-order networks. Unfortunately, the covariance representations are of hundreds of thousands of dimensions. In addition, computing high-order statistics is time-consuming at both training and test stages compared to their first-order counterparts. This limits the practical applications of second-order networks, especially on resource limited devices.

Several methods have been proposed to improve the efficiency of high-order global pooling methods. Compact Bilinear Pooling [49] compresses the full bilinear pooling and achieves comparable performance with significantly reduced parameters. In [100, 99], dimensionality reduction is performed prior to second-order pooling. Moreover, [99] only carries out basic matrix operations suitable for GPU to speed up. Nevertheless, these second-order based methods are still much slower than their first-order counterparts.

2.2.2 Auxiliary networks

Recently, a few methods have been proposed to employ an auxiliary network for backbone network training. To the best of our knowledge, the Inception network architecture [155] is among the first works that utilize auxiliary branches with carefully crafted design for classification and detection tasks. Similarly, knowledge distillation in deep CNNs is an effective approach to transferring the knowledge from more powerful models into weaker ones for inference [64, 52]. Hinton *et al.* [64] proposed to transfer knowledge from an ensemble of acoustic models into a smaller, distilled one for easier deployment. Gupta *et al.* [52] transferred the learned representations from a well labeled domain, obtaining large performance boost by learning rich representations in the unseen domain. Recently, Yim *et al.* [193] introduced a sequential flow between layers to distill knowledge. Furlanello *et al.* [46] trained student models pa-

rameterized identically to their teachers so that students can even outperform their teachers in some small scale vision tasks.

2.2.3 Squeeze-and-excitation networks

Based on the prior that it is important to fuse both spatial and channel-wise information at each layer, a lightweight block, called squeeze-and-excitation (SE) Networks (short for "SE-Net" in the rest of this chapter), has been recently developed in [71]. It first introduces extra GAP layers followed by convolutional and non-linear activation layers. Then, the spatial features are adaptively adjusted along the channel dimension according to the results computed from the last step. Though SE-Net can improve much the performance, computation of the correlations is required so that the extra layers cannot be removed in the inference stage.

We argue that it is possible to design a detachable version of SE-Net which can achieve equivalent or even better performance. Actually, the re-calibration operation at the last step of the original SE block can be approximated and replaced by incorporating additional gradients computed based on channel correlations during training, which are not required and can be omitted in inference. In this chapter, DSoP-Net is proposed as the first attempt to reach this goal.

2.3 Proposed Method

Section 2.3.1 introduces our DSoP-Net in detail, and Section 2.3.2 presents how we solve the issue of dimensionality reduction in second-order pooling methods.

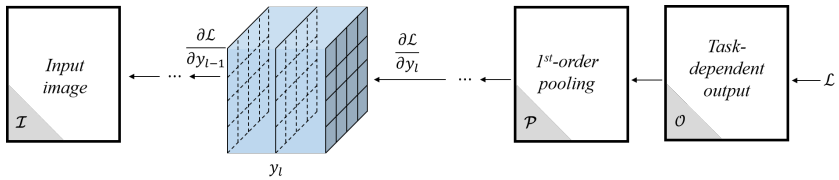
2.3.1 DSoP-Net

Inspired by the works of knowledge distillation [64, 193], we propose to improve the first-order pooling network without introducing extra parameters and computational cost during inference. Our idea is to transfer the knowledge of second-order pool-

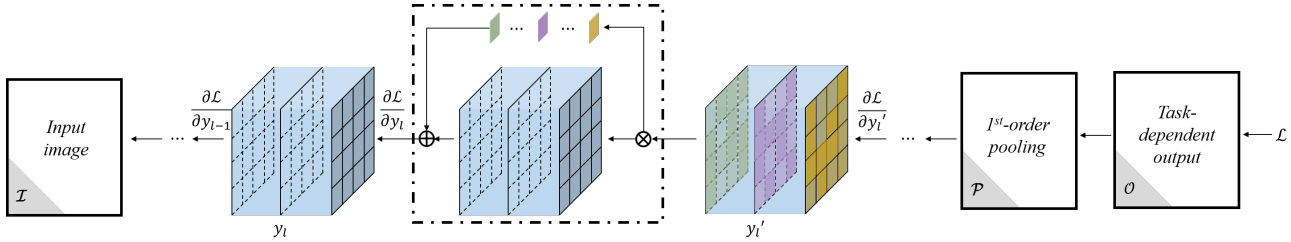
ing networks (teachers) to the first-order pooling network (student) in the training stage, while the teacher networks can be detached from the student in the inference stage. To achieve this goal, existing knowledge distillation frameworks often employ one or more metrics to properly measure the discrepancy between the output of student and teacher networks in the total loss; however, existing metrics, such as p-norm, Kullback-Leibler (KL) divergence, Jensen-Shannon (JS) divergence and Wasserstein divergence, are not suitable to directly measure the distance between first- and second-order pooling outputs.

We propose a simple yet effective solution without the need of extra metrics required in knowledge distillation methods. The auxiliary branch networks are introduced and attached to the first-order pooling network to form a multiple output network, where all outputs are identical to the original output of the first-order network in one task. This allows us to reuse the same criterion, *e.g.*, cross-entropy loss in the classification task, to measure the distance between output of any auxiliary branch and the label. By summing the losses of auxiliary branches into the total loss, knowledge and expertise of the teacher networks can be taught to the student network by computing the gradients of auxiliary branches w.r.t. the first-order network.

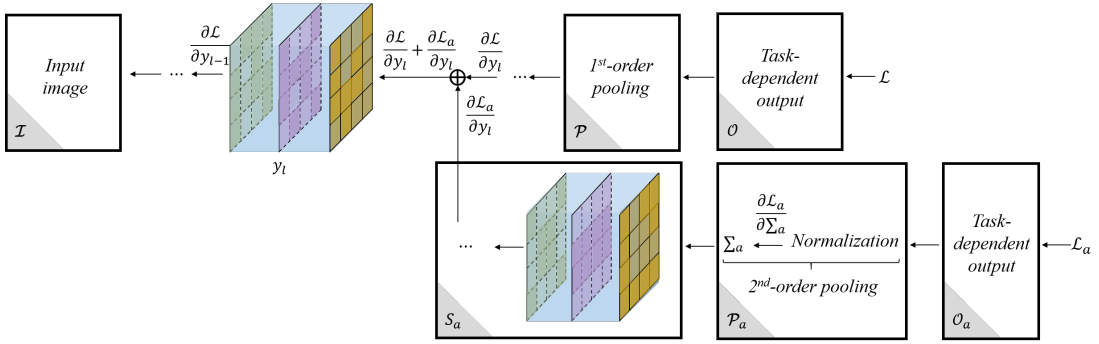
To clearly illustrate the structure of DSoP-Net, we begin with the structure of an auxiliary branch designed in DSoP-Net, followed by the total loss and some discussions.



(a) A normal first-order pooling network



(b) A first-order pooling network with the l-th layer modified as a squeeze-and-excitation block [71]



(c) DSoP-Net with an auxiliary branch plugged after the l-th layer

Figure 2.2: Comparison of back propagation at the l-th layer of a first-order pooling network with different architectures. "$\oplus$" stands for element-wise addition and "$\otimes$" stands for channel-wise multiplication between a scalar and its corresponding feature map in back propagation. y_l' denotes the output of the squeeze-and-excitation block related to the l-th layer.

Structure of an auxiliary branch

Denote by y_l the output of the l-th layer of the backbone model. The i-th auxiliary branch in DSoP-Net is made up of three parts, including a number of convolution and non-linear activation layers S_a^i, a covariance-based second-order pooling layer $\mathcal{P}_a^i$, and task-dependent output O_a^i (please refer to Figure 2.1). Before introducing the general case of auxiliary branches, we first discuss the case when there is only one auxiliary branch inserted after the l-th layer during training, as shown in Figure 2.2(c).

The first part of our auxiliary branch consists of convolutional layers and non-linear activation layers S_a [1], which are used to extract features from y_l for second-order pooling. We reuse the building block of the first-order pooling network, such as bottlenecks of ResNet [59] or its variants [186, 198]. No down-sampling operation is performed so that the output of this part has the same height and width as those of y_l but the channel number can be determined as a hyper-parameter. If we train DSoP-Net without this part in the auxiliary branch, there is a clear performance drop under the same experimental settings. This is because the intermediate features, especially those of layers closer to the input, are not discriminative enough to fulfill a task.

The second part of our auxiliary branch is a covariance matrix based second-order pooling layer $\mathcal{P}_a$. The covariance matrix describes the channel correlations of the output of S_a. The element (i,j) of the covariance matrix is obtained by computing the inner production of the i-th and the j-th channels after they are vectorized. Once the covariance matrix is ready, we proceed to normalization, such as matrix logarithm [3] and matrix power [100, 99], *etc*. The upper- or lower-triangular matrix of the results is re-arranged to form a vector, regarded as the output of the second part.

[1] Superscript is omitted as there is only one branch in this case.

The third part is to produce task-dependent output $\mathcal{O}_a$. It can be easily adapted from the corresponding structure in the backbone model. Since in this work we focus on the classification task, we leverage a fully-connected layer as a linear classifier. Meanwhile, it allows us to reuse the cross-entropy loss in classification tasks to effectively update the weights of an auxiliary branch in DSoP-Net during training. The weights of the l-th layer can be updated by minimizing the loss of the branch and the loss of the backbone first-order pooling network together. From the perspective of back propagation, the gradients related to channel correlations in the branch are merged into the gradients in the first-order pooling network.

The total loss

In the general case, we suppose that there are N auxiliary branches plugged into a first-order pooling network during training. Let $\mathcal{B}_a^1, \ldots, \mathcal{B}_a^N$ denote these branches in order as shown in Figure 2.1. It always holds that $\mathcal{B}_a^i$ is inserted closer to the input than $\mathcal{B}_a^j$ in DSoP-Net, $\forall i < j, i, j = 1, \ldots, N$. Let $\mathcal{L}$ denote the loss computed for the original output of the backbone model $\mathcal{O}$. $\mathcal{O}_a^1, \ldots, \mathcal{O}_a^N$ are the outputs of the auxiliary branches, sharing the same shape and meaning as $\mathcal{O}$. We employ the same criterion to compute the losses of auxiliary branches, $\mathcal{L}_a^1, \ldots, \mathcal{L}_a^N$.

We sum up $\mathcal{L}_a^1, \ldots, \mathcal{L}_a^N$ together with $\mathcal{L}$ as the total loss of the DSoP-Nets to minimize:

$$\mathcal{L}_{DSoP-Net} = \alpha\mathcal{L} + \sum_{i=1}^{N} \beta_i \mathcal{L}_a^i, \tag{2.1}$$

where α and $\{\beta_i\}_{i=1}^N$ are weights used to balance the contribution of each term. We set $\alpha = \beta_1 = \ldots = \beta_N = 1$ by default except stated otherwise.

For any layer of the backbone model between insertion point of $\mathcal{B}^k$ (included) and the insertion point of $\mathcal{B}^{k-1}$ (excluded), its gradients are determined by outputs

of the original first-order pooling and all the auxiliary branches inserted after that layer. The gradients of the l-th layer of the backbone model can be written as

$$\frac{\partial \mathcal{L}_{DSoP-Net}}{\partial y_l} = \frac{\partial(\alpha\mathcal{L} + \sum_{i=k}^{N} \beta_i \mathcal{L}_a^i)}{\partial y_l}, \tag{2.2}$$

where y_l is the output of the l-th layer. For those outputs of branches plugged before the specific layer, it can be clearly observed that they are irrelevant to the update of the l-th layer's weights.

It should also be noted that a layer in the backbone model will not be updated with additional second-order statistics if there is no auxiliary branch inserted after that layer. In practice, we always insert the last auxiliary branch $\mathcal{B}^N$ right before the first-order pooling layer to make sure that each layer of the backbone model (except the original pooling layer and its classifier) can acquire extra cues during training.

After training is completed, we remove all auxiliary branches so that the first-order network has the same parameters and computational overhead as it originally has during test.

Discussions

Once optimized, DSoP-Net can work without extra parameters and computation of channel-information in auxiliary branches. As we will see in the section of experimental results, DSoP-Net exhibits highly competitive performance with second-order pooling networks. This leads to an interesting question: how does channel-based information contribute to a CNN model during training and test?

It is important to fuse spatial and channel-based cues of a CNN model to boost model performance. As the channel-based cues can be directly computed from the spatial responses, optimizing spatial responses *w.r.t.* channel-based information can improve the latter in return. Therefore, there is no need to explicitly compute channel-based cues during test as the channel-based cues have been determined once

the spatial responses are given. In other words, when we jointly optimize the spatial and channel-based cues during training, the spatial cues can be further enhanced.

It is critical to ensure that the backbone network used for training is equivalent to the one used for testing after we detach all the auxiliary branches from it. When two CNNs have the same weights and the same computational graph, they will obtain the same output for the same input. As detaching the branches does not affect the value of any weight, all weights of DSoP-Net remain unchanged in the testing stage. As for the computational graph, though all auxiliary branches are detached, no operations are removed or added in the computational graph of the remained DSoP-Net. Therefore, the entire route from the input to the desired output $\mathcal{O}$ is the same before and after the detachment. The DSoP-Net used for training is equivalent to the one used for testing.

To further explain the detachable property of DSoP-Net, in Figure 2.2 we compare DSoP-Net with the original first-order pooling network and SE-Net [71] from the perspective of back propagation. One can see that gradients at the l-th layer of the original first-order pooling network are computed only w.r.t. the original loss $\mathcal{L}$. In contrast, gradients obtained at the l-th layer of both DSoP-Net and SE-Net can be written as the weighted summation of two terms corresponding to the spatial and channel-based information. In the dash-dotted rectangle, it can be observed that the weight of one term in SE-Net is partially determined by the output of the other one according to the definition of channel-wise multiplication. However, weights of both terms in DSoP-Net are independent to each other. They are pre-defined by the relative contributions of $\mathcal{L}$ and $\mathcal{L}_a$ in the total loss, which is concise and easy to compute. It remains unsolved in [71] whether the channel-wise information plays a more important role than the spatial information as an SE block can be hardly partitioned for in-depth study. With DSoP-Net, however, we are able to unveil that it is merely important to explicitly compute channel-wise information during test

when the spatial information is well learned. It can be seen that at the test stage DSoP-Net can be regarded as a special SE-Net whose channel-wise information is always trivial, *e.g.*, **1s**. Besides, it can be found from the experimental results in Section 2.5 that DSoP-Net can achieve equivalent or even better performance than SE-Net.

2.3.2 Progressive supervised dimensionality reduction

The output feature dimension of covariance matrix based second-order pooling method is proportional to the square of input channels. As a result, dimensionality reduction (DR) is required prior to a second-order pooling layer to save computational overhead and prevent over-fitting. However, a large channel reduction ratio often leads to significant performance drop. The DR operator in [100, 99], for example, consists of a 1×1 convolution, followed by BN and ReLU layers. When it reduces the channel number from 2048 to 64 so that the dimension of second-order representation is comparable to that of the vanilla ResNet-50 model (with GAP) at $2K$-d, the performance gain almost fades out while it costs 24% more inference time to perform GSoP. Clearly, it is of vital importance to develop a compact yet effective DR operator.

In this chapter, we propose a compound DR operator which consists of a sequence of lightweight DR operators, called progressive supervised dimensionality reduction (PSDR). PSDR shares the same design principle of DSoP. Their main difference lies in where they locate in a CNN model. A CNN model can be regarded as a multi-step transform that maps its input domain to the desired output domain. Both PSDR and DSoP aim to introduce the second-order statistics from the detachable branches to actively guide and improve the learning of each step, *i.e.*, they both exploit the second-order statistics. A straight of M intermediate layers are created for PSDR during training. Each intermediate layer gently reduces the channel to some extent

without decreasing the performance, and all these layers work together to reduce the channel to the desired dimension. In addition, we incorporate auxiliary branches to better transfer the expertise and knowledge in high-dimensional domain to low-dimensional domain at all intermediate layers. Specifically, an auxiliary branch is inserted after each intermediate layer during training, and there are M auxiliary branches $\mathbb{B}^1, \ldots, \mathbb{B}^M$. Similar to $\mathcal{B}^1, \ldots, \mathcal{B}^N$, each auxiliary branch used in PSDR is composed of three parts. The first part is formed by a 1×1 convolutional layer followed by batch normalization and non-linear activation layer, *e.g.* ReLU, reducing the given number of channels to the desired number. We reuse the covariance matrix based second-order pooling layer to construct the second part of an auxiliary branch in PSDR. The third part is built to produce task-dependent output, which can borrow from the corresponding layers of the backbone model.

Let $\mathbb{P}_a^1, \ldots, \mathbb{P}_a^M$ denote the second part of an auxiliary branch in PSDR. We denote the output of auxiliary branches by $\mathbb{O}_a^1, \ldots, \mathbb{O}_a^M$, sharing the same form and meaning as the original output $\mathbb{O}$. As the same criterion is adopted to measure these output, we can obtain their corresponding losses $\mathcal{J}_a^1, \ldots, \mathcal{J}_a^M$. All these extra losses are summed up with the original loss $\mathcal{J}$ to minimize. Thus, the total loss function of PSDR can be written as

$$\mathcal{L}_{PSDR} = \lambda \mathcal{J} + \sum_{i=1}^{M} \gamma_i \mathcal{J}_a^i, \tag{2.3}$$

where λ and $\{\gamma_i\}_{i=1}^M$ are scalars used to balance the contribution of each term. By default, $\lambda = \gamma_1 = \ldots = \gamma_M = 1$.

Figure 2.3 compares a direct DR operation with the proposed PSDR. Before the second-order pooling layer, the direct DR operation takes only one step to reduce 2048 channels to 64, running the risk of severe information loss in the output $\mathbb{O}$. In contrast, it takes four more steps with PSDR. One can see that on the right-

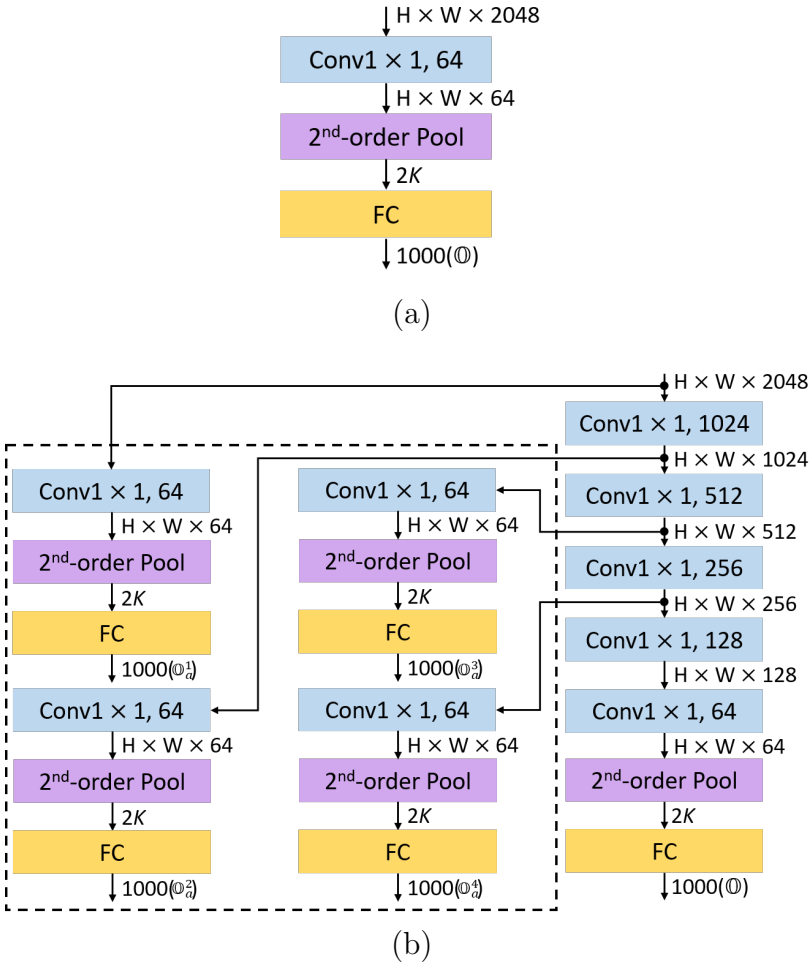


Figure 2.3: Comparison of (a) a direct DR operation and (b) our PSDR used to reduce channels from 2048 to 64 for second-order pooling. The blue, purple and yellow rectangles denote a 1×1 convolutional layer followed by batch normalization and ReLU (number of channels is presented after comma), a second-order pooling layer, and a fully-connected layer used as a linear classifier, respectively. Data shape is formatted as *height* × *width* × *channel*, located next to the related arrow. Four identical auxiliary branches (dashed rectangle) are created for PSDR during training. Names of the output are presented in the bracket.

hand side of Figure 2.3(b), there are 5 1×1 convolutional layers with 1024, 512, 256, 128 and 64 filters, where each layer is followed by batch normalization and ReLU. In the dashed rectangle, four auxiliary branches are attached at the first four steps, resulting in $\mathbb{O}_a^1, \mathbb{O}_a^2, \mathbb{O}_a^3$ and $\mathbb{O}_a^4$, respectively. With the help of these branches, channel correlations of features obtained at intermediate steps are strengthened in the process of channel reduction.

2.4 Implementation Details

2.4.1 Architecture of DSoP-Net

It is flexible to implement our DSoP-Net with many standard architectures. In this chapter, we implement DSoP-Net with the popular and powerful ResNet [59]. By default, we create 4 auxiliary branches with second-order pooling and output header, which are inserted after conv2_x, conv3_x, conv4_x and conv5_x, respectively. We repeat the ResNet bottleneck defined at each stage for several times to set up the first part of the auxiliary branch described in Section 2.3, where strides of convolutional layers are fixed to 1 so that the input and output of the auxiliary layers have identical size. We name the first part of auxiliary branches as a_conv2_x, a_conv3_x, a_conv4_x and a_conv5_x, respectively. For example, a_conv2_x is inserted after conv2_x.

For the second-order pooling method used in each auxiliary branch, we use iSQRT-COV [99] for fast speed and reliable performance. Before the iSQRT-COV meta-layer, we perform a DR following [100, 99] so that the number of channels is at most 256 if necessary. This reduces the computational cost during training and prevents potential over-fitting. In our implementation, we follow the default setting of hyper-parameters described in [99]. We use trace based pre-normalization and 5 Newton-Schultz iterations to solve matrix power.

The ResNet-18, ResNet-34 and ResNet-50 based DSoP-Net models are presented

in Table 2.1. The input size is 224×224×3. We use the same names for those layers as in the original ResNet structure [59]. We use prefix "a_" to indicate the corresponding layers in the auxiliary branches. For instance, a_conv2_x is inserted after conv2_x, and a_pds2 is on top of a_conv2_x. For comparison, we present the structure of ResNet-101 in the rightmost column. We only change the stride of convolutional layers in conv5_x from 2 to 1 so that the output size is 14×14. Let us consider the modified ResNet-18 model as an example. We replicate the blocks at stage2∼stage5 and accordingly put them in the auxiliary branches. We plug a direct DR layer to reduce the channel number to 256 right before the second-order pooling operation. However, values obtained after a_conv2_x∼a_conv4_x are directly fed into a_pds2∼a_pds4 without DR, respectively. The majority of extra parameters come from the dense layers in a_pds2∼a_pds5 as the dimension of image representation of the auxiliary branches increases from $2K$ to $32K$.

The training of DSoP-Net costs more than twice the time than training its backbone model with the same experimental configuration. The extra time comes from two parts: auxiliary branches and second-order pooling layers. As the total number of convolutional layers in our auxiliary branches are by default the same as that of the backbone model, the auxiliary convolutional layers roughly double the training time. The extra training time costed by the second-order layers depends on the number of auxiliary branches. For example, with DSoP-Net for ImageNet, it takes approximately 74, 97, and 158 hours to train a ResNet-18, ResNet-34 and ResNet-50, respectively.

Table 2.1: DSoP-Net with modified ResNet architecture. The input image size is 224×224×3. 1^{st}- and 2^{nd}-PDS are short terms for 1^{st}- and 2^{nd}-order pooling, and dense layer and softmax, respectively. Figures in bracket indicate kernel size and number of channels used for pooling.

Layer Name	Output Size	18-layer	34-layer	50-layer	101-layer
conv1	112×112	7×7, 64, stride 2			
conv2_x	56×56	3×3, max pool, stride 2			
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} times3$
a_conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	N.A.
a_pds2	1×1	2^{nd}-PDS(64)	2^{nd}-PDS(64)	2^{nd}-PDS(64)	N.A.
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
a_conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	N.A.
a_pds3	1×1	2^{nd}-PDS(128)	2^{nd}-PDS(128)	2^{nd}-PDS(256)	N.A.

conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256\ 1\times1, 1024 \end{bmatrix} \times 23$
a_conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 13$	N.A.
a_pds4	1×1	2^{nd}-PDS(256)	2^{nd}-PDS(256)	2^{nd}-PDS(256)	N.A.
conv5_x	14×14	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
a_conv5_x	14×14	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	N.A.
a_pds5	1×1	2^{nd}-PDS(256)	2^{nd}-PDS(256)	2^{nd}-PDS(256)	N.A.
		1^{st}-PDS(512)		1^{st}-PDS($2K$)	
GFLOPs (Train.)		7.26	9.81	13.45	10.01
GFLOPs (Test)		3.06	5.61	6.27	10.01
#Param (Train)		100.6M	110.7M	181.9M	44.5M
#Param (Test)		11.7M	21.8M	25.6M	44.5M

2.4.2 Organization of PSDR

To balance precision and speed of PSDR, we reduce half of the channels at the first intermediate layer in our implementation. For the rest of intermediate layers, we introduce a decay factor $\delta \in (0, 1)$ for gentle DR. The number of output channels at an intermediate layer can be computed by

$$C_a^i = \begin{cases} 0.5C_a^0, & i = 1, \\ \lfloor \delta C_a^{i-1} \rfloor, & otherwise, \end{cases} \tag{2.4}$$

where C_a^0 is the number of given feature maps for PDSR. We set $\delta = 0.25$ in the remaining of this chapter unless otherwise specified. For example, given C_a^0=2048 feature maps, we set up 2 intermediate layers for PSDR and they are associated with 1024 and 256 channels, respectively. Besides, we directly reduce the number of current channels to the desired number in each auxiliary branch before we perform second-order pooling in case of limited GPU memory.

2.5 Experiments

We evaluate the proposed method on large-scale image classification dataset ImageNet, as well as CIFAR-10 and CIFAR-100 datasets. All experiments are conducted on a machine equipped with dual Intel Xeon Gold 6136@3.0GHz CPUs, 128G DDR4 2666MHz RAM, 1T nvme m.2 SSD and 8 NVIDIA Tesla P100 GPU cards. We implement our method by using PyTorch [133] compatible with CUDA and cuDNN.

2.5.1 Datasets

We adopt ImageNet LSVRC2012 dataset [35] with 1,000 classes for large-scale image classification task. The dataset contains over 1.2 million images for training, 50 thousand images for validation, and 100 thousand images for testing. As labels of

Table 2.2: Comparison of error rates (%) achieved by different methods with ResNet-50 on ImageNet.

Method	Pooling Type (Train.)	Pooling Type (Test)	GFLOPs	Extra GFLOPs	#Param.	Extra #param.	Top-1/5
ResNet-50-s2 [59]	1^{st}	1^{st}	3.86	N.A.	25.6M	N.A.	23.85/7.13
ResNet-50-s1 [59]	1^{st}	1^{st}	6.27	N.A.	25.6M	N.A.	23.57/6.85
KD [64] w/ ResNet-50-s1	1^{st}	1^{st}	3.86	0	25.6M	0	23.73/7.03
KD [64] w/ ResNet-101	1^{st}	1^{st}	3.86	0	25.6M	0	23.54/6.70
KD [64] w/ SE-ResNeXt-101	1^{st}	1^{st}	3.86	0	25.6M	0	23.46/6.66
FBN [103]	1^{st}	1^{st}	N.A.	N.A.	N.A.	N.A.	24.00/7.10
SORT [171]	1^{st}	1^{st}	N.A.	N.A.	N.A.	N.A.	23.82/6.72
SE-Net [71]	1^{st}	1^{st}	3.87	0.01	28.1M	2.5M	23.29/6.62
CBAM [178]	1^{st}	1^{st}	3.86	0.004	28.1M	2.5M	22.66/6.31
MPN-COV [100]	2^{nd}	2^{nd}	6.43	0.16	56.9M	31.3M	22.74/6.54
iSQRT-COV [99]	2^{nd}	2^{nd}	6.43	0.16	56.9M	31.3M	22.14/6.22
ResNet-50-s2 w/ DSoP-Net	1^{st} &2^{nd}	1^{st}	3.86	0	25.6M	0	23.14/6.56
ResNet-50-s1 w/ DSoP-Net	1^{st} &2^{nd}	1^{st}	6.27	0	25.6M	0	**21.15/5.70**

the test images are not released, we follow [81, 59] and compare methods on the validation set.

We also evaluate the generalization capability of the proposed DSoP-Net on the CIFAR-10 dataset and the CIFAR-100 dataset [89]. Both datasets are well balanced, consisting of 60,000 32×32 colour images from 10 and 100 classes, respectively. For each dataset, 50,000 images are used for training and the remaining 10,000 are adopted for testing.

2.5.2 Experimental Settings

We closely follow standard experimental settings on ImageNet as well as CIFAR-10 and CIFAR-100 datasets for fair comparison. Details are presented below.

Experimental setting for ImageNet. In the training phase, we first resize each image so that its shorter side is randomly sampled on [256, 512] [154]. Then, a fixed-size 224×224 patch is randomly cropped from the down-scaled image or its horizontally flipped version. Finally, we normalize each patch by subtracting the dataset mean and dividing it by the dataset standard deviation. In the testing phase, we resize each test image so that its shorter side is 256 and a single 224×224 center crop is applied for inference. We use SGD [153] with a mini-batch of 256 for optimization and set weight decay to 1×10^{-4} and momentum to 0.9. We train DSoP-Net from scratch for 90 epochs. Learning rate starts at 0.1, and is reduced to 0.01 and 0.001 at Epoch 30 and 60, respectively.

Experimental setting for CIFAR-10 and CIFAR-100. Standard data augmentation strategy [105, 73, 71] is adopted in training, where images are horizontally flipped at random and zero-padded on each side with 4 pixels before conducting a random 32×32 crop. For evaluation, we report the error computed on the test images of original size.

2.5.3 Evaluation on ImageNet

We compare our DSoP-Net with 4 categories of competing networks, including: (1) networks with quadratic transformation instead of just linear convolutions, such as FBN [103] and SORT [171]; (2) vanilla ResNet-50 trained with deeper or wider models in terms of knowledge distillation [64], which plays the role of student network jointly optimized with a teacher network, *e.g.* a modified ResNet-50 that uses stride=1 for all convolutional layers at stage 5, ResNet-101 and SE-ResNeXt-101; (3) architectures developed with fixed design such as SE-Net [71], and CBAM [178]; and (4) networks that use GSoP at the network end, such as MPN-COV [100] and iSQRT-OV [99]. We compare the results reported in the original papers for methods in categories (1), (3) and (4). We run the methods in category (2) with PyTorch official implementation of ResNet family models [2] and third-party [3] implementation of SE-ResNeXt-101, with which we achieved very close results to ResNet family models reported in [129]. Due to limited computational resources, we adapted the final size of a random image crop to 256×256 pixels so that experiments can be conducted with a single set of 8-way GPU server; meanwhile, we reduced the mini-batch size from 1024 to 256 for SE-ResNeXt-101 as the original setting reported in [71] is highly in favor of the distributed training system. Accordingly, we reduced the initial learning rate from 0.6 to 0.15. Following the settings in [100, 99], to obtain higher resolution feature maps, we further changed the value of stride at stage 5 from 2 to 1. We use the original ResNet-50 architecture (stride=2) and the modified one (stride=1) as two baselines for fair comparison, and denote by ResNet-50-s2 and ResNet-50-s1 the two baselines, respectively.

Table 2.2 compares the performance of DSoP-Net with the state-of-the-arts on ImageNet. We have three observations articulated below. First, the proposed

[2] `https://github.com/pytorch/vision`

[3] `https://github.com/Cadene/pretrained-models.pytorch`

ResNet-50-s1 w/ DSoP-Net achieves top-1/5 error rates at 21.15%/5.70% on ImageNet, significantly outperforming all the competing methods. It is even better than those methods that explicitly use GSoP at the network end, including MPN-COV [100] and iSQRT-COV [99]. Second, with equivalent number of parameters used in training, ResNet-50-s2 w/ DSoP-Net outperforms joint training with a single deeper and/or wider teacher model. It shows that a student model is not able to well fuse the channel correlations and the spatial information without explicit modelling during training. However, it is hard to design metrics to directly measure the discrepancy of channel correlations between the student and the teacher models. In contrast, DSoP-Net is free of this issue and it allows us to easily transfer knowledge and expertise of second-order statistics to the student model. Third, DSoP-Net uses the same architecture as the two baseline models during inference, making it the most lightweight one among all competing methods. However, it can still obtain equivalent or even better performance than the heavier models, such as SE-Net [71].

We also study the performance of the latest state-of-the-art architectures with our DSoP-Net. EfficientNet-b0 [159] is selected as the backbone model. Without a TPU cluster in hand, we made a few changes of the original experimental settings to fit our own GPU server. Specifically, the mini-batch size is reduced from 2048 to 768, and the initial learning rate is accordingly set to 0.048 for RMSProp. First, we used a third-party implementation [4] and managed to achieve a top-1/5 accuracy of EfficientNet-b0 at 76.81%/93.32% on ImageNet with baseline ResNet preprocessing. The results we obtained are almost the same as the official TPU implementation [5] under similar settings (top-1 accuracy at 76.8%). Then, we replaced the original first-order pooling of EfficientNet-b0 with the second-order pooling structure by reducing the number of channels from 1280 to 64 with a sequence of conv1 $\times$ 1, BN and swish

[4] `https://github.com/rwightman/pytorch-image-models`

[5] `https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet`

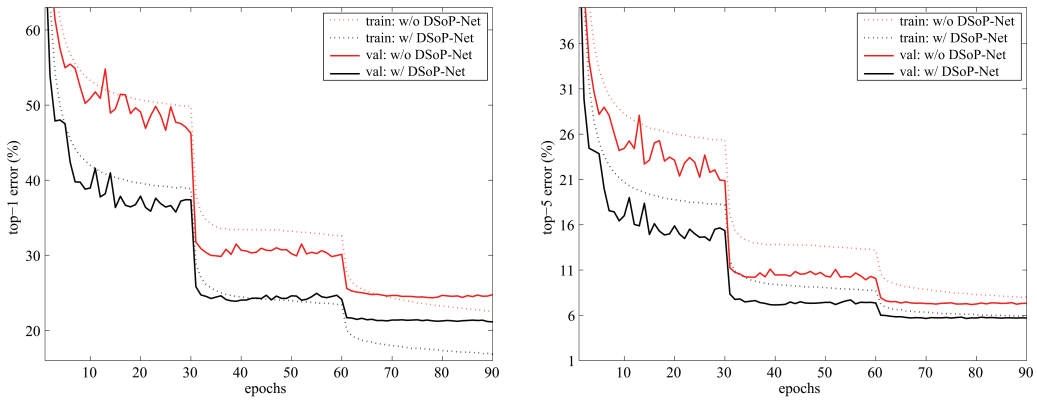


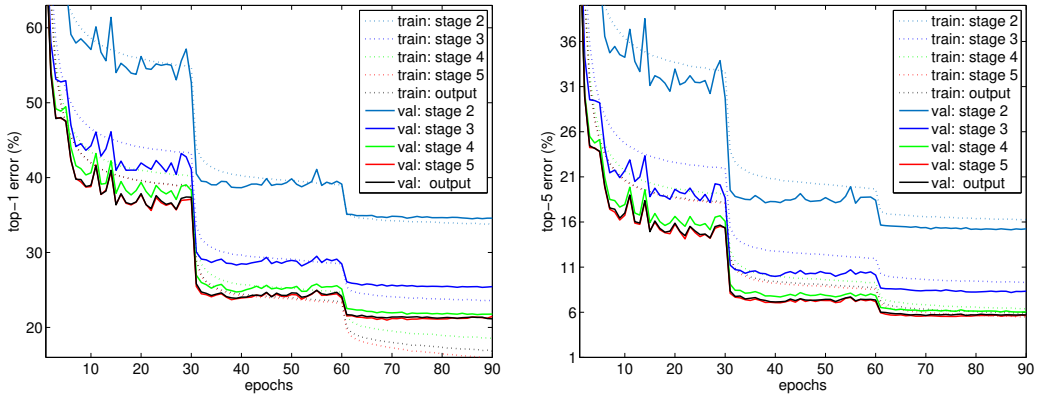
Figure 2.4: Convergence curves regarding (a) top-1 and (b) top-5 error rates achieved by ResNet-50-s1 with and without DSoP-Net on ImageNet.

activation [140]. To prevent over-fitting, we inserted a dropout layer before the last linear classifier and set its dropout rate to 0.2. It turns out that the EfficientNet-b0 with second-order pooling obtains a top-1/5 accuracy at 77.02%/93.38%. Finally, we constructed the DSoP-Net for EfficientNet-b0 by inserting two auxiliary branches with second-order pooling. One was inserted at the middle of the backbone model and the other was inserted at the end. With DSoP-Net, the top-1/5 accuracy of the original first-order pooling head is further improved to 77.12%/93.56%. This validates the complementary nature of DSoP-Net to state-of-the-art CNN architectures such as EfficientNet.

Figure 2.4 compares the convergence curves of our DSoP-Net (output with GAP) and those of ResNet-50-s1. We can see that our DSoP-Net consistently outperforms ResNet-50-s1 by a large margin. This shows that knowledge acquired from the auxiliary classifiers with second-order pooling improves the original classifier with first-order pooling throughout the whole training process. Figure 2.5 presents the convergence curves of different output headers. We have three observations. First, the first-order output achieves very close performance to its teacher output in the

Table 2.3: Comparison of error rates (%) achieved by different training policies with DSoP-Net on ImageNet.

Training policy	#Epoch	a_pds2 Top-1/5	a_pds3 Top-1/5	a_pds4 Top-1/5	a_pds5 Top-1/5	GAP Top-1/5
One-phase (default)	90	34.59 /15.24	25.44 /8.33	21.77 /6.04	21.11 /5.57	21.15 /5.70
Two-phase	90+15	43.52 /22.36	30.00 /11.30	24.68 /7.54	23.34 /6.50	23.57 /6.85



(a) (b)

Figure 2.5: Convergence curves regarding (a) top-1 and (b) top-5 error rates achieved by all output headers of DSoP-Net under ResNet-50 architecture on ImageNet, including the original one and the four obtained in the auxiliary branches.

last detachable branch. Second, output of a branch inserted at a later layer always performs better than that of a branch inserted at an earlier layer. The first-order output has even better results than those second-order outputs of the branches inserted at earlier layers. Third, compared to the original backbone model, the benefit gained by DSoP-Net during the very first epochs is significant while later the gap between the curves gradually reduces. This raises an interesting question whether early removal of auxiliary branches in DSoP-Net can help to boost the performance.

We employed ResNet-50-s1 as the backbone and inserted auxiliary branches at the beginning of training on ImageNet. All experimental settings remain unchanged except that auxiliary branches are removed at epoch 30, 45, 60, respectively, to train the DSoP-Net. Experimental results show that the top-1/5 accuracies are decreased by 0.87%/0.31%, 0.41%/0.14% and 0.33%/0.11%, respectively, compared to adopting auxiliary branches during the whole training. That is, early removal of auxiliary branches in DSoP-Net harms the final performance of the backbone model.

We continue to investigate the importance of engaging second-order statistics at earlier layers of first-order pooling networks. The default training policy of our DSoP-Net can be regarded as one-phase where first- and second-order statistics jointly help to update the weights of the backbone model. For comparison reasons, we define a two-phase training policy where first- and second-order statistics are independently computed in order. Specifically, we first attach the same 4 auxiliary branches, including a_pds2,3,4,5, to the baseline ResNet-50 model (stride=1) after it converges under default training settings so that it has the same structure as DSoP-Net. Then, we proceed to fine-tune the parameters in auxiliary branches and fix those of the backbone model for 15 epochs on ImageNet. Learning rate starts at 0.1, and is reduced to 0.01 and 0.001 after 5 and 10 epochs, respectively. From Table 2.3, one can see that the performance of the outputs at the 4 auxiliary branches achieved by the two-phase training policy can be largely improved by the default training policy. This suggests that optimizing networks simultaneously with first- and second-order statistics lead to better performance at earlier layers than computing the two kinds of statistics separately during training. Besides, with the default training policy of DSoP-Net, the strengthened intermediate layers help to improve the classification performance of the backbone model.

Table 2.4: Summary of DSoP-Net with different backbone models on CIFAR-10 and CIFAR-100 datasets.

Backbone	#Block per stage	#Branch per stage	#Total branches	#Conv. per stage	Stage spec.
ResNet-110	18	6	18	72	[0,0,3]×6
ResNet-164	18	6	18	108	[0,0,3]×6
WRN-22-10	3	2	6	12	[1,0,2]
WRN-28-10	4	2	6	16	[0,2,0,2]
ResNeXt-64-8	3	2	6	18	[1,0,2]
ResNeXt-64-16	3	2	6	18	[1,0,2]

2.5.4 Evaluation on CIFAR-10 and CIFAR-100 Datasets

We further implement DSoP-Net with other modern backbone models to evaluate its generalization ability on CIFAR-10 and CIFAR-100 datasets, including two Pre-activation ResNet models [59], *i.e.* ResNet-110 and ResNet-164, two Wide Residual Networks (WRN) [198] models, *i.e.* WRN-22-10 and WRN-28-10, as well as two ResNeXt [186] models, *i.e.* ResNeXt-64-8 and ResNeXt-64-16. During training, for all models, we used SGD with momentum as optimizer and set the batch size as 128 and momentum as 0.9. For Pre-activation ResNet models, we set the weight decay as 1×10^{-4} and the number of training epochs as 110. The learning rate starts from 0.3, and it is divided by 10 at 80 and 95 epochs. For WRN models, we set the weight decay as 5×10^{-4} and the number of training epochs as 200. The learning rate begins from 0.1, and is divided by 5 at epochs 60, 120 and 160. For ResNeXt models, we set the weight decay as 5×10^{-4} and the number of training epochs as 300. The initial learning rate is 0.1, and is divided by 10 at 150 and 225 epochs. To construct auxiliary branches for each model, definition of the same building block is reused at the stage where one auxiliary branch is attached. Meanwhile, we keep the total number of building blocks at one stage the same as that of the blocks used in

the backbone model. Therefore, there are the same number of convolutional layers in the backbone model and its auxiliary branches.

Table 2.4 summarizes some key statistics of DSoP-Net with different backbone models on CIFAR-10 and CIFAR-100 datasets. Take ResNet-110 as an example. We uniformly insert 18 branches with second-order pooling into the backbone network. The pattern of each stage is specified as $[0, 0, 3] \times 6$, where the non-zero values suggest the number of auxiliary convolutional layers inserted at the corresponding position while the zero values mean there are no auxiliary branches inserted at that position. To this end, there is a detachable branch at the end of every 3 bottlenecks; 6 auxiliary branches are attached to each stage; and there are $6 \times 3 = 18$ auxiliary branches in total. For each branch, it adopts the same network structure as its associated 3 bottlenecks before the second-order pooling layer. As each original/auxiliary bottleneck depicts 2 convolutional layers, $36 + 36 = 72$ convolutional layers are used at each stage during training. The target number of channels are set to 128 for all second-order pooling layers, including DSoP-Nets and iSQRT-COV [99] for comparison. For DR, we follow [100, 99] and use the direct DR operator. To avoid over-fitting, we incorporate a dropout layer with dropout rate of 0.5 before a dense layer in an auxiliary branch if 128 channels are used for second-order pooling. In this way, the image representation in auxiliary branches is $2K$-d, $4K$-d, and $4K$-d for branches attached to Stage 1, Stage 2, and Stage 3, respectively.

To adapt to the CNN models on CIFAR-10 and CIFAR-100 datasets, we fixed the weight of original loss as $\alpha = 1$ in Equation 2.1. Weights of losses connected with auxiliary outputs at Stage 1, Stage 2 and Stage 3 of all models are empirically set as 0.1, 0.2 and 0.3, respectively. This significantly helps to prevent gradient exposure accumulated at earlier layers in back-propagation. Meanwhile, this also strengthens the role of the original prediction of backbone model. We observed that all losses are basically on the same order of magnitude; thus, the desired loss of the original

Table 2.5: Comparison of error rates (%) achieved by different backbone models on CIFAR-10.

Backbone	w/o DSoP-Net (Original)	w/ DSoP-Net (Ours)	Gain
ResNet-110	6.37	5.73	0.64
ResNet-164	5.46	4.55	0.91
WRN-22-10	4.44	3.91	0.53
WRN-28-10	4.17	3.81	0.36
ResNeXt-64-8	3.65	3.44	0.21
ResNeXt-64-16	3.58	3.20	0.38

prediction given by the 1^{st}-order pooling layer only takes up a very small portion ($\sim 5\%$) if all components are equally weighted in the total loss, challenging the performance of the backbone model in inference. In contrast, with our implementation, loss of the original prediction actually takes up $30\% \sim 50\%$ in the total loss to train more effective CNN model.

To prevent over-fitting caused by auxiliary branches, a direct DR operator is inserted before the 2^{nd}-order pooling layer to keep at most 128 channels if there are more channels for GSoP. Meanwhile, a dropout layer ($p = 0.5$) is plugged between the second-order pooling layer and the dense layer in each branch. Consequently, the dimension of image representation produced by any auxiliary branch on CIFAR-10 and CIFAR-100 datasets is equal to or less than $128\times(128+1)\times0.5\times0.5=4K$.

Table 2.5 and Table 2.6 demonstrate the results achieved by different backbone models with and without DSoP-Net on CIFAR-10 and CIFAR-100 datasets. One can clearly see that models with DSoP-Net effectively outperforms the same architecture without DSoP-Net on both CIFAR-10 and CIFAR-100. Besides, it can be observed that DSoP-Net is more useful when training a narrow network. For example, the performance gain on CIFAR-10 for the narrowest model, ResNet, reaches $0.64\% \sim$

Table 2.6: Comparison of error rates (%) achieved by different backbone models on CIFAR-100.

Backbone	w/o DSoP-Net (Original)	w/ DSoP-Net (Ours)	Gain
ResNet-110	26.88	25.43	1.45
ResNet-164	24.33	21.06	3.27
WRN-22-10	20.75	18.91	1.84
WRN-28-10	20.50	18.50	2.00
ResNeXt-64-8	17.77	16.34	1.43
ResNeXt-64-16	17.31	16.23	1.08

Table 2.7: Comparison of error rates (%) achieved by different pooling methods in the auxiliary branches on ImageNet. ResNet-50-s1 is employed as the backbone model.

With DSoP-Net	Aux. Pool. Method	Order	Top-1/5
✗	N.A.	N.A.	23.57/6.85
✓	GAP [105]	1^{st}	22.74/6.61
	iSQRT-COV [99]	2^{nd}	**21.15/5.70**

0.91% while those for the WRN models, and the widest ResNeXt models, are reduced to 0.36% $\sim$ 0.53%, and 0.21% $\sim$ 0.38%, respectively.

2.5.5 Ablation Study

DSoP-Net

We conduct ablation study of DSoP-Net from three aspects, as presented in detail as follows.

Pooling methods in auxiliary branches. We study the impact of different pooling methods in auxiliary branches. ResNet-50-s1 is employed as the backbone model and we modify the network architecture of auxiliary branches during training.

Table 2.8: Comparison of error rates (%) achieved by different number of channels used for second-order pooling at a_pds2,3,4,5 on ImageNet. ResNet-18-s1 is employed as the backbone model. An asterisk symbol in superscript indicates a DR operation before 2^{nd}-order pooling.

With DSoP-Net	Target #Chnl	#Chnl	GFLOPs / #Param.(M)	Top-1/5
✗	N.A.	N.A.	3.06/11.7	30.11/10.78
✓	64	64,64*,64*,64*	5.69/32.7	29.19/10.12
	128	64,128,128*,128*	5.95/51.3	28.88/9.92
	256	64,128,256,256*	7.26/100.6	**28.69/9.85**

Table 2.9: Comparison of top-1 error rates of ResNet-20 on CIFAR-10 by different optimizers (%).

Optimizer	w/o DSoP-Net	w/ DSoP-Net
SGD [153]	7.92	**7.59**
ADAHESSIAN [192]	**7.87**	7.70
RAdam [114]	8.62	8.07

Table 2.10: Comparison of top-1 error rates of ResNet-18-s1 on ImageNet by different optimizers (%).

Optimizer	w/o DSoP-Net	w/ DSoP-Net
SGD [153]	30.11	**28.69**
ADAHESSIAN [192]	**29.86**	29.55
RAdam [114]	32.12	31.29

Table 2.11: Comparison of error rates(%) achieved by different insertion points with ResNet-18 on ImageNet.

#Conv. Layers in a_conv2,3,4,5_x	With DSoP-Net	#Branch	GFLOPs	#Param. (M)	Top-1	Top-5
0,0,0,0	✗	N.A.	3.06	11.7	30.11	10.78
16,0,0,0	✓	1	4.94	94.8	29.09	10.04
0,0,0,16	✓		11.23	127.4	28.88	9.92
8,8,0,0	✓	2	4.34	95.4	29.06	10.03
8,0,0,8	✓		8.47	108.7	29.00	9.98
0,0,8,8	✓		8.50	112.0	28.77	9.90
4,4,4,4	✓	4	7.26	100.6	28.69	9.85

Table 2.12: Comparison of error rates (%) achieved by different DR methods with ResNet-50-s1 on ImageNet in reducing the 2048 channels.

Method	Target #Channel	δ	w/ Aux. Branches	Top-1	Top-5
Direct DR [100, 99]	64	N.A.	N.A.	23.73	6.99
	128	N.A.	N.A.	22.78	6.43
	256	N.A.	N.A.	22.14	6.22
PSDR (Ours)	64	0.25	✗	23.30	6.86
	128	0.25	✗	22.18	6.09
	64	0.5	✗	23.24	6.72
	128	0.5	✗	22.02	6.15
	256	0.5	✗	21.84	5.99
	64	0.5	✓	22.71	6.42
	128	0.5	✓	21.89	6.11
	256	0.5	✓	**21.65**	**6.00**

All layers used for channel reduction are removed and the second part is changed from iSQRT-COV to GAP while the other parts remain unchanged. We train the backbone model with the modified auxiliary branches from scratch with the default training strategy. We compare it to default architecture defined in DSoP-Net under the same backbone model. Table 2.7 presents the results. Compared with the baseline, we can see that the introduction of auxiliary branches with first-order pooling method improves top-1 error rates by 0.8%. If we use the covariance matrix based second-order pooling as the second part of an auxiliary branch, it further boosts the result by nearly 1.4%. It shows that the second-order channel correlations are difficult to be directly modelled by first-order pooling such as GAP. Instead, knowledge and expertise from the second-order pooling layer in an auxiliary branch are effective in adjusting the spatial responses in the backbone model.

Number of channels for GSoP. Covariance matrix as an image representation quadratically increases the computational complexity of the second-order pooling method. It also affects the number of parameters of the dense layer in an auxiliary branch. To study the impact, in each auxiliary branch, we decrease the number of channels for second-order pooling from 256 to 128 and 64, respectively. ResNet-18 (stride=1), denoted by ResNet-18-s1, is employed as the backbone model. Table 2.8 compares the results by using different number of channels. One can see that the total number of parameters used in training decreases sharply from 100.6M to 32.7M with only 64 channels. It reduces the computational complexity by approximately 21.6%. Meanwhile, top-1/5 error rates increase moderately by 0.5% and 0.27%. Even so, DSoP-Nets still reduces the top-1 error rate by almost 1% with the same backbone inference architecture.

Insertion points of the auxiliary branches. To study the impact of different insertion points, we employ ResNet-18 model as backbone and fix the total number of ResNet bottlenecks used in all auxiliary branches as 16. Specifically, we remove one

or more auxiliary branches based on the default architecture of DSoP-Net under the ResNet-18-s1 architecture (please refer to Table 2.1), where each auxiliary branch contains four ResNet bottlenecks. We uniformly distribute the bottlenecks to the remaining branches. Table 2.11 presents the results. One can see that performance of first-order pooling can be improved by inserting more branches at different stages. With the same number of branches, plugging branches at the layers closer to the first-order pooling benefits performance boost at the price of more parameters and computations in training, but not during the inference stage.

Choice of optimizer. It has been shown [84, 118] that SGD is a very stable and effective network optimizer for various CNN architectures, especially on large scale datasets such as ImageNet. It is interesting to investigate whether other optimizers can further improve the performance of DSoP-Net than SGD. We test two recent optimizers: ADAHESSIAN [192] and Rectified Adam (a.k.a. RAdam) [114]. We use the official implementation of ADAHESSIAN[6] and RAdam[7]. In the experiments, we opt to SGD [153] as the baseline and evaluated ResNet-20 on CIFAR-10 and ResNet-18-s1 on ImageNet, which are widely used to compare different optimizers (especially those designed for CNN models). We use the same hyper parameter settings reported in the original paper of each optimizer, including weight decay, initial learning rate, total epochs, learning rate schedule, *etc*.

Table 2.9 and Table 2.10 summarize the results. One can have the following three observations. First, for each optimizer, model trained with DSoP-Net outperforms the same model trained without DSoP-Net on both CIFAR-10 and ImageNet. This shows that our method is effective with different optimizers. Second, compared to the baseline optimizer, DSoP-Net slightly reduces the gap between RAdam and SGD on CIFAR-10 from 0.70% to 0.48%. However, the gap between RAdam and

[6]`https://github.com/amirgholami/adahessian`

[7]`https://github.com/LiyuanLucasLiu/RAdam`

SGD on ImageNet is enlarged from 2.01% to 2.60%. Third, ADAHESSIAN results in better performance than SGD for the original CNN models on both datasets, but it is not as good as SGD for models with DSoP-Net. We believe that the performance loss is caused by the approximation of the Hessian matrix as a diagonal operator. Such an approximation may be suitable for the sequential models (such as the vanilla CNN models), but for multiple-header models like DSoP-Net, it is critical to consider the correlations among parameters in the backbone model and those in the auxiliary model. As a result, some second-order cues in the auxiliary branches may be implicitly lost during the optimization with ADAHESSIAN, and the performance becomes worse than the DSoP-Net optimized with SGD. Besides, SGD is friendly to computational resources in terms of both memory and overhead, and hence it is easier to apply to deeper or wider CNN architectures.

PSDR

We employ ResNet-50-s1 as the backbone model in the ablation study of PSDR. In addition, we focus on the parameter selection of δ in Equation 2.4 and the introduction of auxiliary branches for DR. Table 2.12 demonstrates the results achieved by the direct DR operator adopted in [100, 99] and the proposed PSDR in reducing the 2048 channels of the last convolutional layer. It can be observed that PSDR outperforms direct DR when the same reduction ratio is applied. A larger δ allows more intermediate layers, leading to a clear performance boost. Meanwhile, introduction of auxiliary branches enables transfer of second-order statistics from high-dimension domain to low-dimension domain, which also improves the DR operation. Particularly, PSDR significantly improves the direct DR when the given 2048 channels are reduced to 64 channels. We achieve a top-1/5 error rate of 22.71%/6.42%, even better than when keeping 128 channels with direct DR.

2.6 Conclusion

In this chapter, we propose a novel method which significantly improves the performance of first-order CNNs in image classification. Auxiliary branches are carefully designed to transfer knowledge to the backbone first-order networks during training, which are however removable at the testing stage. As a result, the proposed method leverages the advantages of second-order pooling networks while keeping similar complexity to first-order networks during inference. To the best of our knowledge, this is the first attempt to make use of higher-order statistics in knowledge distillation. Experiments conducted on ImageNet as well as CIFAR-10 and CIFAR-100 datasets demonstrate the effectiveness of our network. In particular, we achieve a top-1 error rate of 21.15% with single center-crop using ResNet-50 network.

In the next chapter, we will move from the perspective of overall CNN architecture in this chapter to the design of fast and lightweight CNN modules.

Chapter 3

LST-Net: Learning a Convolutional Neural Network with a Learnable Sparse Transform

The 2D convolutional (Conv2d) layer is the fundamental element to a deep convolutional neural network (CNN). Despite the great success of CNN, the conventional Conv2d is still limited in effectively reducing the spatial and channel-wise redundancy of features. In this chapter, we propose to mitigate this issue by learning a CNN with a learnable sparse transform (LST), which converts the input features into a more compact and sparser domain so that the spatial and channel-wise redundancy can be more effectively reduced. The proposed LST can be efficiently implemented with existing CNN modules, such as point-wise and depth-wise separable convolutions, and it is portable to existing CNN architectures for seamless training and inference. We further present a hybrid soft thresholding and ReLU (ST-ReLU) activation scheme, making the trained network, namely LST-Net, more robust to image corruptions at the inference stage. Extensive experiments on CIFAR-10/100, ImageNet, ImageNet-C and Places365-Standard datasets validated that the proposed LST-Net can obtain even higher accuracy than its counterpart networks with fewer parameters and less overhead.

3.1 Introduction

The past decade has witnessed a great success of deep convolutional neural netowrk (CNN) in various computer vision problems, such as visual object recognition [90, 41], object detection [144, 143, 101], face recognition [74, 83], scene understanding [184, 209], *etc*. The 2D convolutional (Conv2d) layer [90] is one of the key elements in a CNN to extract powerful features from the input image. Despite the great success of CNN, the conventional Conv2d is limited in effectively reducing the spatial and channel-wise redundancy of features. When image features are propagated through Conv2d, it usually requires a large number of kernels to model the data and hence introduces exaggerated parameters and overhead. Meanwhile, Conv2d simply sums up all convolutional responses along the channel dimension for the same kernel and takes little advantage of inter-channel cues [71, 33], which is less effective.

A lot of efforts have been devoted to improving the performance of Conv2d. Recent works can be roughly categorized into two categories. The first category of works aim to enhance what a Conv2d layer sees in the spatial domain. For representative works in this category, dilated convolution [195] effectively expands its receptive field by applying predefined gaps, while deformable convolutional networks [32, 210] improve the performance of Conv2d by learning internal parameters to model geometric transformation or variations so as to adaptively focus on some more important areas. Though these methods make better use of spatial information, they fail to take advantage of the channel-wise cues. The second category of works strengthen the performance of Conv2d by combining both spatial and channel-wise attentions. Representative works in this category can be found in [71, 178, 45, 16]. For example, squeeze-and-excitation networks (SENet) [71] re-weights the features along the channel dimension using an efficient squeeze-and-excitation block. Usually, these works rely on an extra network path to adjust spatial and channel-wise attentions after

the conventional Conv2d is computed. The redundancy of conventional Conv2d remains but it requires additional network parameters and overhead. It is interesting to investigate whether we can develop a new convolutional module, which can better describe the local features, reduce the spatial and channel-wise feature redundancies, and reduce the parameters and overhead while keeping the accuracy unchanged or even improved.

We propose to mitigate these issues by learning a CNN with a learnable sparse transform (LST). We are motivated by the classical harmonic analysis works such as discrete cosine transform (DCT) [174] and discrete wavelet transform (DWT) [62, 145, 20], which can convert the given image into a more compact and sparse domain to reduce the spatial and channel redundancy of features. In DCT and DWT, the sparse transforms are manually pre-designed, while in our proposed LST, the sparse transform is learned from training data together with the process of CNN training. The proposed LST learning can be efficiently implemented with existing CNN modules, such as point-wise convolutions [105] (PWConvs) and depth-wise separable convolutions [68] (DWConvs). This makes LST compatible with existing CNN architectures for seamless training and inference without additional operations.

The proposed LST promotes sparser features. In light of the sparsity priors [162, 13, 14], we further present a hybrid soft thresholding [39] and ReLU [131] (ST-ReLU) activation scheme. Compared with the standard ReLU, the ST-ReLU activation can suppress the noise and trivial features in the learning process, making the trained network more robust to image corruptions, such as noise, blur, digital compression, *etc*. Overall, the proposed LST module can be applied to existing state-of-the-art network architectures such as ResNet and VGGNet. The obtained new network, namely LST-Net, achieves more robust and accurate performance with fewer parameters and less overhead. Our major contributions are summarized as follows.

- A novel learnable sparse transform based Conv2d module is developed, which can be efficiently implemented and seamlessly integrated into existing CNN learning process, producing sparser features and improving the effectiveness of learned CNN models.

- A new activation function is presented by properly combining soft-thresholding and ReLU operations, which endows the proposed LST-Net with better robustness to image trivial features and corruptions.

3.2 Related Work

3.2.1 Network bottleneck

To save parameters and overhead of Conv2d layers, group convolution [90] (GConv) and PWConv [105] are popularly employed in the design of bottlenecks. PWConv employs a 1×1 window, performing a linear combination of the input from all channels. It is often used to align a set of feature maps with different number of channels [154]. GConv assumes that the input features can be decomposed into several groups along the channel dimension, where features from different groups are independent. A successful application of GConv is ResNeXt [186]. DWConv [68] is a special case of GConv when there is only one input channel per group. It is widely used to build lightweight models for mobile devices, such as MobileNet [68, 149], ShuffleNet [119, 204], *etc*.

Xie *et al*. [186] improved ResNet bottleneck [59] by substituting the conventional 3×3 Conv2d in the middle with a GConv of slightly more channels. One problem of this method is how to set the group number. A larger number of groups can easily cause loss of inter-channel cues while a smaller number of groups can hardly reduce redundancy of Conv2d. Recently, Res2Net [48] was developed by fusing the group with the intermediate results obtained from the latest group in a recursive manner.

Though Res2Net demonstrates higher accuracy, it actually sacrifices parallel execution on devices such as GPUs. In this paper, we naturally incorporate DWConvs and PWConvs to facilitate transforms in spatial and channel-wise fields.

3.2.2 Learning space

The conventional Conv2d layer is less effective in reducing the spatial and channel-wise feature redundancies because each Conv2d kernel interacts with input features locating in a local grid of limited size and cannot take features outside the grid into consideration. To mitigate this issue, dilated convolution [195] applies predefined gaps to enlarge spatial receptive field of Conv2d. Deformable convolutional networks [32, 210] learn to adaptively focus on some more important areas by modelling geometric transformation or variations with internal parameters; however, they fail to further consider the channel-wise cues of features and require sophisticated implementation skills. SENet [71] and its variants [178, 45, 16] focus on designing lightweight network paths to fuse channel-wise and spatial features to improve the attention of the conventional Conv2d. Though these methods is effective to boost accuracy, they remain inefficient as they use more parameters and require extra overhead.

To improve the performance of Conv2d layer, it's more straightforward to perform convolution in a more compact and sparser domain. The classical DCT [174] and DWT [62, 145, 20] transform the input image into a sparse space for manipulation and they have a wide range of successful applications [174, 43, 203, 10, 20, 75]. The sparse coding [132] techniques encode the image patches as a sparse linear combination of learned atoms. However, the transformation filters used in DCT and DWT are manually designed and they are not effective enough to represent image structures, while sparse coding is computationally inefficient and is hard to be extended for deep feature extraction. In this paper, we propose to learn a sparse transformation

together with the deep CNN learning so that the network can be more efficiently and effectively learned in a sparser space.

3.2.3 Activation function

Non-linearity introduced by the activation function is among the most critical factors to the success of a CNN model in various computer vision tasks. ReLU [131] is a pioneer and the most popular non-linear activation function in deep CNN. It is a simple yet highly effective segmented function, forcing the input negative valued features to zeros and keeping only the non-negative features. To make use of the information of negative features, parametric ReLU [58], leaky ReLU [169], ELU [31] and SELU [87] are proposed to allow adaptive negative activation with learnable parameters. However, negative activation functions need to be carefully designed and they only exhibit better performance in specific applications. One problem of ReLU and its variants is that they are not very robust to noise or other corruptions in the input image. It is well-known that by soft-thresholding the image features in some sparse domain, such as DWT domain [62, 145, 20] and sparse coding domain [132], the latent image features can be well recovered. In our proposed LST, we adaptively learn a sparse transform together with the CNN learning, which can make the CNN features sparser. This motivates us to develop a new activation scheme, *i.e.*, hybrid soft-thresholding and ReLU (ST-ReLU), to better exploit the merit of sparser features. The ST-ReLU further enhances the robustness of learned CNN models to various types of corruptions.

3.3 Proposed Method

3.3.1 Learnable sparse transform (LST)

Denote by $\mathcal{I} \in \mathcal{R}^{H_{in} \times W_{in} \times C_{in}}$ the input feature and $\mathcal{O} \in \mathcal{R}^{H_{out} \times W_{out} \times C_{out}}$ the output feature of a Conv2D layer, where H_{in}/H_{out}, W_{in}/W_{out}, C_{in}/C_{out} denote the height, the width, and the channel number of the input/output feature, respectively. The sliding window Ω of the Conv2D can be parameterized by the kernel size $s_H \times s_W$ (for simplicity of expression, we omit the subscripts H and W in the remaining of this paper), number of kernels C_{out}, stride, as well as padding. We denote the k^{th} kernel by $\mathcal{K}^{(k)}$.

The Conv2d output feature is redundant in both spatial and channel dimensions. When the sliding window Ω is centered at spatial location (i,j) of $\mathcal{I}$, the output $\mathcal{O}_{i,j,k}$ by convolving $\mathcal{I}$ with kernel $\mathcal{K}^{(k)}$ is computed as

$$\mathcal{O}_{i,j,k} = \sum_{x=1}^{s} \sum_{y=1}^{s} \sum_{z=1}^{C_{in}} \Omega(\mathcal{I}; i, j)_{x,y,z} \cdot \mathcal{K}^{(k)}_{x,y}, \tag{3.1}$$

where $\Omega(\mathcal{I}; i, j)_{x,y,z}$ is the pixel at (x, y, z) of the tensor extracted from $\mathcal{I}$ by Ω, and $\mathcal{K}^{(k)}_{x,y}$ means the pixel at (x, y) of $\mathcal{K}^{(k)}$.

We have two observations from Equation 3.1. First, all feature pixels in the local neighborhood of the pixel at spatial location (i,j) are involved in the computation. While this is helpful to extract the high frequency features, it is redundant for extracting the low frequency features, which usually occupy most of the pixels in a feature map. Second, the subscript z does not follow $\mathcal{K}$ but only comes up with Ω. That is to say, all pixels in the same channel are equally weighted to produce $\mathcal{O}_{i,j,k}$. It has been found that the input features have strong similarities along the channel dimension [176, 61]. Therefore, there exists much redundant channel-wise computations. All these motivate us to develop a learnable sparse transform (LST),

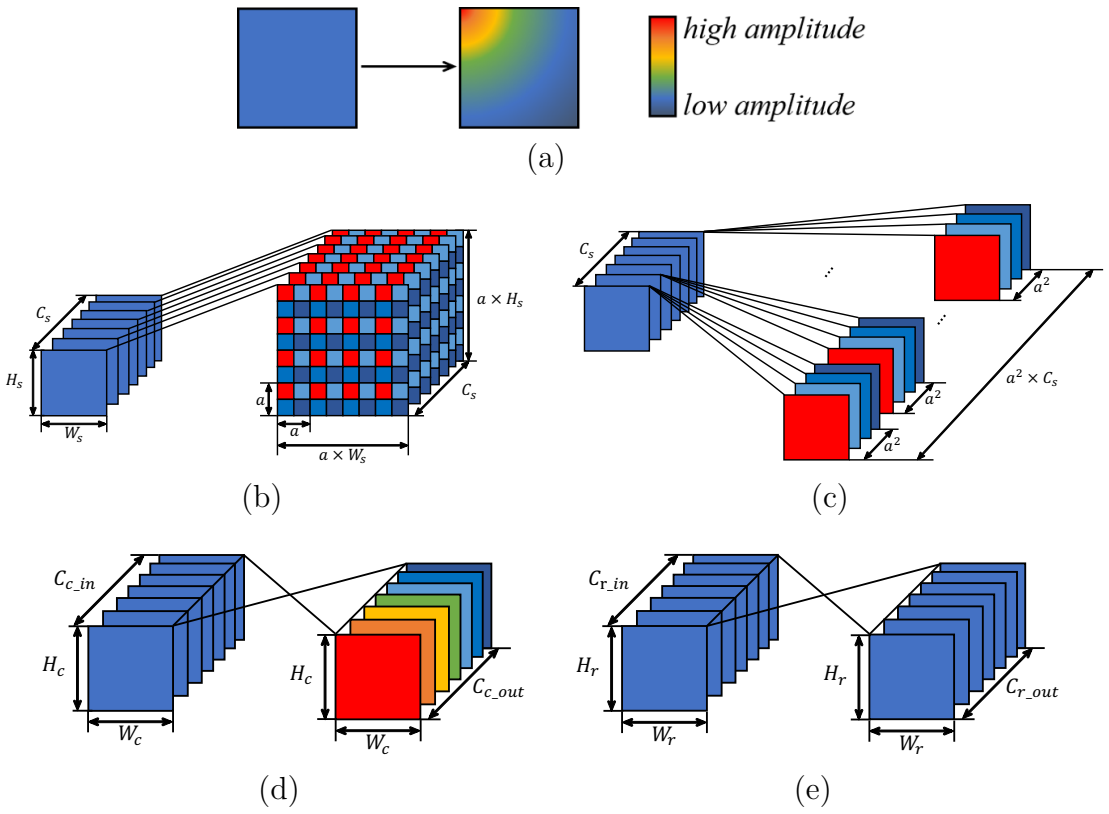


Figure 3.1: Illustration of different transforms. (a) 2D-DCT, (b) a tiled spatial transform, (c) T_s, (d) T_c, and (e) T_r.

with which the redundancy of conventional Conv2D can be reduced and hence a more efficient CNN can be learned.

Overview of LST. Our LST consists of three transforms: a spatial transform T_s, a channel-wise transform T_c, and a resize transform T_r. T_s and T_c strive to reduce the spatial and channel-wise redundancies by transforming the corresponding field into a more compact domain, while T_r aims to resize the input to obtain the desired shape of output. T_r can be placed either before or after T_s and T_c. The LST, denoted by T_{LST}, can be implemented as

$$T_{LST} \circ \mathcal{I} = T_r \circ T_s \circ T_c \circ \mathcal{I}, \tag{3.2}$$

or in the form of

$$T_{LST} \circ \mathcal{I} = T_s \circ T_c \circ T_r \circ \mathcal{I}. \tag{3.3}$$

The spatial transform T_s. We propose to reduce the spatial redundancies of local features by using a learnable spatial transform T_s with associated weights $\mathcal{W}_s \in \mathcal{R}^{a^2 \times 1 \times s \times s}$ (dimensions are organized in PyTorch [133] style). Inspired by the success of classical 2D-DCT [124], which decomposes the image local region into different frequency bands by using sequential column and row transforms, we can implement T_s by applying column and row transforms, denoted by T_{column} and T_{row}, respectively. Mathematically, the corresponding weights W_s can be expressed as:

$$\mathcal{W}_s = \mathcal{W}_{column} \otimes \mathcal{W}_{row}, \tag{3.4}$$

where $\otimes$ means the Kronecker product with necessary dimension insertion and removal, $\mathcal{W}_{column} \in \mathcal{R}^{a \times 1 \times s \times 1}$ and $\mathcal{W}_{row} \in \mathcal{R}^{a \times 1 \times 1 \times s}$ are the weights of T_{column} and T_{row}, respectively, and a is a hyper parameter specifying the number of coefficients to keep.

As illustrated in Figure 3.1(a), a 2D-DCT transforms a local region into different frequencies. The low frequency coefficients concentrate at the top left corner and they dominate the energy (high amplitude), while many high frequency coefficients are close to zero (low amplitude) and can be neglected. Based on this fact, to save unnecessary parameters and computation, we set $1 \leqslant a < s$ so that the low amplitude trivial features can be excluded from calculation. Since for almost all existing CNN architectures, it is true that the kernel size $s \geqslant 3$, we set $a = \lceil \frac{s}{2} \rceil$ by default in this chapter.

Figure 3.1(c) depicts our implementation of T_s. One can see that the output of T_s is arranged along the channel dimension (this will ease much the implementation of our resize transform T_r). For each $s \times s$ local region, by convolving it with W_s, we

obtain an a^2-*dim* output vector. Thus, for each input feature map, it is transformed into a number of a^2 feature maps by aggregating the a^2-*dim* output vectors. That is, T_s maps $\mathcal{R}^{H_{s\_in} \times W_{s\_in} \times C_s}$ to $\mathcal{R}^{H_{s\_out} \times W_{s\_out} \times (a^2 \times C_s)}$, where C_s is the channel number of the input argument of T_s, and H_{s_in}/H_{s_out} and W_{s_in}/W_{s_out} are the input/output height and width, respectively. In contrast, the conventional spatial transform organizes the output in the height and width fields, instead of the channel domain. We term the conventional spatial transform as tiled spatial transform, which maps $\mathcal{R}^{H_{s\_in} \times W_{s\_in} \times C_s}$ to $\mathcal{R}^{(a \times H_{s\_in}) \times (a \times W_{s\_in}) \times C_s}$, as illustrated in Figure 3.1(b). Comparing our T_s with tiled spatial transform, we can obtain three findings.

First, T_s is simpler to implement than tiled spatial transform. In practice, we can adopt a DWConv operation to implement it. Second, T_s only affects the channel dimension, which allows us to easily use the existing efficient implementations for the resize transform T_r. (Please see the following section of resize transform for details.) In contrast, a tiled spatial transform increases both the height and width of feature maps so that T_r must be changed to deal with the enlarged spatial dimensions. Third, our T_s always holds memory continuity, making it faster in both training and inference. In contrast, a tiled spatial transform needs channel shuffle, which requires extra memory alignment.

Owe to the physical meaning of T_s (*i.e.*, to reduce feature spatial redundancy), 2D-DCT can be effectively used to initialize $\mathcal{W}_s$ with Equation 3.4. This makes the training of LST converge efficiently to a good local minimum. In Section 3.4.3, we will show that initialization of W_s by 2D-DCT exhibits much better performance than random initialization.

Channel-wise transform T_c. T_c is used to reduce the redundancy along channel dimension. It is a $\mathcal{R}^{H_c \times W_c \times C_{c\_in}} \rightarrow \mathcal{R}^{H_c \times W_c \times C_{c\_out}}$ mapping, where C_{c_in}/C_{c_out} is the channel number of the input/output of T_c, and H_c and W_c denote the height and width of the input, respectively. T_c encourages features to be more separable along

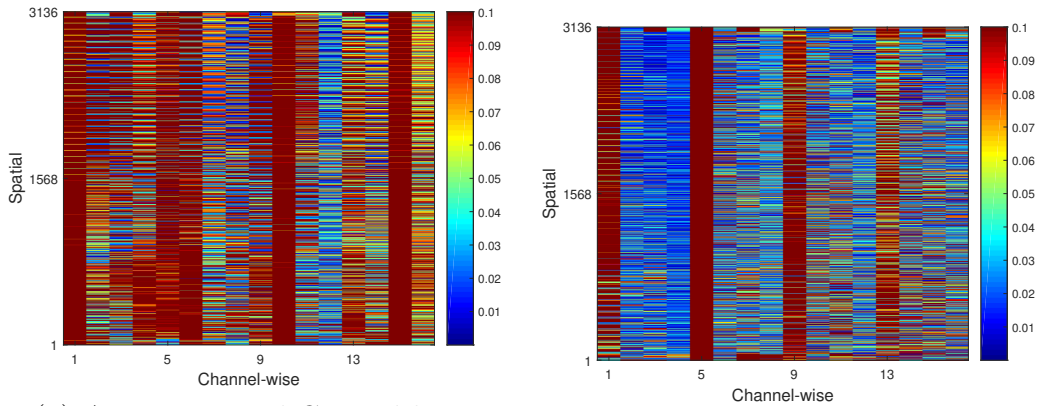
the channel and simplifies the resize transform T_r in reweighting data.

A PWConv operation can be naturally leveraged for T_c with its associated weights $\mathcal{W}_c \in \mathcal{R}^{C_{c\_out} \times C_{c\_in} \times 1 \times 1}$. Similar to T_s, 2D-DCT can be used to initialize T_c for compact features. We fill $\mathcal{W}_c$ with the 2D-DCT basis functions shaped as $C_{c\_in} \times C_{c\_out}$ and expand its dimensions where necessary. Figure 3.1(d) illustrates the implementation of T_c. One can see that the output of T_c is organized in order by the expected feature amplitude like 2D-DCT. It should be noted that T_c is similar to the resize transform T_r since both of them adopt PWConv for implementation. However, they are initialized in different ways.

The resize transform T_r. A conventional Conv2d equally treats all the samples in the window without considering their importance. To fill this gap, the resize transform T_r is designed as a $\mathcal{R}^{H_r \times W_r \times C_{r\_in}} \rightarrow \mathcal{R}^{H_r \times W_r \times C_{r\_out}}$ mapping, where H_r/W_r is the height/width of the input, and C_{r_in} and C_{r_out} are the channel number of the input and output, respectively. T_r is learned to adaptively reweight the input features with its weights $\mathcal{W}_r \in \mathcal{R}^{C_{r\_out} \times C_{r\_in} \times 1 \times 1}$. Figure 3.1(e) illustrates how T_r works. With the help of our design of T_s and T_c, T_r can be implemented by directly leveraging a normal PWConv operation in our paper.

Discussions. To better understand the role of LST, in Figure 3.2 we visualize the learned features by the standard ResNet50 (with conventional Conv2d) and our LST-Net with a ResNet50 architecture on ImageNet [35]. (The details of the model can be found in Section 3.4.2.) Once trained, a validation image is randomly selected and its center crop is fed into the two models. Figure 3.2 visualizes 16 channels of the features (other channels are similar) from the first bottleneck (the features after the T_s transform are visualized for our LST-Net). We clip the amplitude of the feature in the range of [0, 0.1] and stretch the features in each channel as a vector. Each column in Figure 3.2 represents the vectorized features of a channel.

One can see that the output features of conventional Conv2d in ResNet50 are



(a) A conventional Conv2d layer in ResNet50

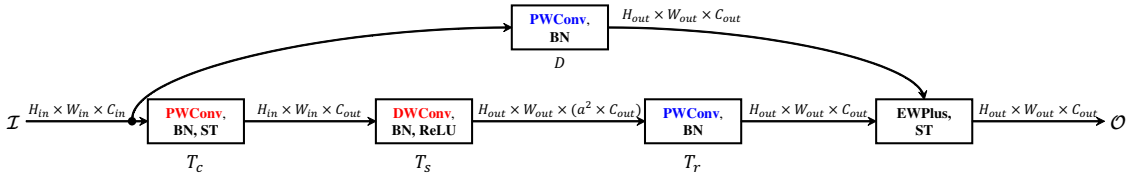
(b) Our LST-Net after T_s

Figure 3.2: Visualization of output features. One can see that the features output by LST-Net are sparser and well-structured along the channel dimension.

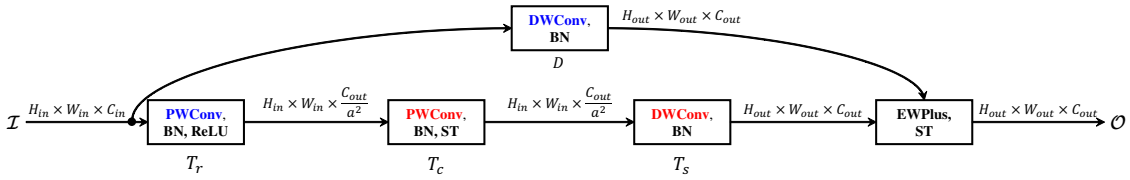
mixed up along the channel dimension. In contrast, the features output by LST-Net are sparser (with lower amplitude) and well-structured along channel dimension. Specifically, every $a^2 = 2^2 = 4$ channels form a unit where the four channel features are de-correlated into different frequency bands (please also refer to Figure 3.1(c)). Such kind of sparser and structured features are highly suited to the successive channel-wise operations such as PWConv (used by resize transform T_r) or a sequential of global average pooling (GAP) [105] plus a dense layer.

3.3.2 Hybrid ReLU-ST activation scheme

By using the proposed LST introduced in Section 3.3.1, we are able to generate more compact and sparser features than the conventional Conv2D layers in a CNN, as illustrated in Figure 3.2. It has been shown in many works of WT [39, 38] and sparse coding [132] that a soft-thresholding (ST) operation in the sparse feature domain can increase the robustness to noise and trivial features. The ST operation for the input feature x can be written as



(a) LST-I



(b) LST-II

Figure 3.3: Illustration of the two LST bottlenecks with downsample operators. EWPlus means element-wise addition. PWConv/DWConv in red font indicates initialization with 2D-DCT while blue font suggests random initialization.

$$y = \begin{cases} sgn(x)(|x| - \tau), & |x| \geqslant \tau, \\ 0, & otherwise. \end{cases} \tag{3.5}$$

where τ is a hyper parameter for the threshold. To exploit the merit of sparser features brought by LST, we propose a new activation scheme for our LST-Net by jointly using ST and ReLU, namely ST-ReLU.

Specifically, ST is adopted at two places in LST-Net; otherwise, ReLU is used. First, ST is inserted in the middle of T_c and T_s. It not only reduces the noises along the channel dimension but also further forces sparsity and suppresses trivial features in the spatial domain. Second, ST is used as the last activation function for an LST to allow adaptive negative activation. Unlike existing methods such as parametric ReLU [58], leaky ReLU [169], ELU [31] and SELU [87], ST is a natural selection of activation in the sparse feature domain, and it accords with the findings on spiking states of neurons in neuroscience [77, 78, 199, 147, 34].

3.3.3 The bottleneck

We construct a novel bottleneck, namely LST bottleneck, to wrap LST and the hybrid ST-ReLU activation scheme. A shortcut path is introduced in our LST bottleneck to avoid gradient exploding or vanishing when a model goes deeper. As a result, an LST bottleneck can be written as follows when the shape of input feature $\mathcal{I}$ is the same as that of output $\mathcal{O}$:

$$\mathcal{O} = T_{LST} \circ \mathcal{I} + \mathcal{I} \tag{3.6}$$

If the input shape is different from the output shape, the bottleneck becomes

$$\mathcal{O} = T_{LST} \circ \mathcal{I} + D \circ \mathcal{I}, \tag{3.7}$$

where D is a downsample operator to adjust the shape of features. D is adopted when the stride of Ω is greater than 1 or $C_{in} \neq C_{out}$.

According to the arrangement of T_s, T_c and T_r defined in Equation 3.2 and Equation 3.3, we design two bottleneck structures, namely LST-I and LST-II, as illustrated in Figure 3.3. One difference between LST-I and LST-II lies in how the bottleneck expands. LST-I is similar to the basic bottleneck in [59]. It first expands the number of channels by a^2 times with T_s; then, it reduces the number of channels back to C_{out} with T_r. The expansion factor of LST-I is 1. In contrast, LST-II adopts a similar ideology to the ResNet bottleneck [59]. It starts to reduce the channel number to C_{out}/a^2 with T_r and then increases it to C_{out} with T_s. Like ResNet bottleneck [59], we refer the planes (core number of channels) of an LST-II bottleneck to C_{ch_out}, *i.e.*, C_{out}/a^2. Meanwhile, the expansion factor of LST-II is determined by T_s, which equals a^2.

Another difference between the two bottlenecks lies in the implementation of D. LST-I adopts the widely used structure, *i.e.*, a PWConv followed by a BN. In contrast, we propose to leverage a 1×1 DWConv followed by BN for the downsample

operator D of LST-II by assuming that C_{out} is divisible by C_{in}. Such an assumption usually holds in many modern architectures, *e.g.*, VGG [151], ResNet [59], ResNeXt [186], *etc*. It shifts the original definition of "identity" in such cases to a group-wise mapping by expanding one channel to C_{out}/C_{in} channels. Each input feature map only interacts with its C_{out}/C_{in} associated output feature maps by DWConv, making it very efficient to handle hundreds or even thousands of feature maps.

With LST-I or LST-II, one can easily build an LST-Net by using existing network architectures with fewer parameters and less overhead. Examples of LST-Net with some common CNNs can be found in Section 3.4.2.

3.4 Experiments

3.4.1 Experiment setup and datasets

All experiments are conducted using an 8-way NVIDIA Tesla P100 GPU server with 2 Intel Xeon Gold 6136 CPUs and 128G RAM. To evaluate our method, we build up LST-Nets by replacing conventional Conv2d operations with our proposed LST bottlenecks w.r.t. some widely used CNN architectures. The datasets used include CIFAR-10/100 [89] and ImageNet [35]. Besides, ImageNet-C [63] dataset is used to demonstrate the robustness of LST-Net to common image corruptions. In addition, MS-COCO dataset is also used to study the performance in object detection and instance segmentation. Ablation studies are performed to discuss the initialization, the selection of parameter τ in ST-ReLU, the difference between LST-I and LST-II and comparison of ReLU-ST to other activations.

CIFAR-10 and CIFAR-100 datasets. Standard data augmentation strategies [105, 73] were adopted in training, including random horizontal flip, padding of four extra pixels on each side, random crop, etc. Each model was trained for 160 epochs. We used SGD with a mini-batch of 128 samples for optimization. Weight decay and

momentum were set to 5×10^{-4} and 0.9, respectively. Learning rate started at 0.1, and was reduced by a factor of 10 after $32K$ and $48K$ iterations. One GPU card was used to train LST-Nets constructed w.r.t. ResNet-20 and ResNet-56 architectures; two GPU cards were employed for 110- and 164-layer LST-Nets; four GPU cards were adopted to train LST-Nets built under other architectures. We did not use any SyncBN layers.

ImageNet LSVRC2012 dataset. By default, we follow the settings in [81, 59] to compare different methods on the validation set (no test labels are released). SGD with a mini-batch of 256 samples was used for optimization. Weight decay was set to 1×10^{-4} and momentum to 0.9. We trained each model from scratch for 90 epochs. Learning rate started at 0.1, and was reduced by a factor of 10 for every 30 epochs. We employed four GPU cards to train LST-Net constructed w.r.t. ResNet-18, ResNet-34, ShiftNet, AlexNet and MobileNet V2. Eight GPU cards were all used to train other models.

ImageNet-C dataset. We used the ImageNet-C dataset to study the robustness of those models trained on ImageNet. No fine-tuning was conducted for test on ImageNet-C.

Places365-Standard dataset. We reused the same training settings on ImageNet. We report the best top-5 test accuracy achieved by ten-crop estimation for each model.

MS-COCO dataset. We used MS-COCO dataset to study the performance on object detection, instance segmentation and human pose estimation. For the first two tasks, all experiments were conducted using default settings of MMDetection [22]. Specifically, we resized images to 1333×800 at both training and test stages. Besides, we applied $1\times$ learning rate schedule policy for fair comparison, *i.e.*, each ImageNet pretrained backbone model was fine-tuned for 12 epochs. Neither synchronized batch normalization nor image scale augmentation was leveraged in our experiments. In

terms of human pose estimation, we leveraged SimpleBaseline [183] as the key-points estimation method. To locate key-points, human images are detected by a detector having human AP of 56.4 on COCO val2017 dataset and resized to 256×192 or 384×288 during training and inference.

PASCAL VOC dataset. We used PASCAL VOC dataset [41] to study the performance on semantic segmentation. In the training phase, we randomly cropped a fixed-size 513×513 patch from an image or its horizontally flipped version. In the testing phase, a single 513×513 center crop is applied for inference. We used SGD [153] with a mini-batch of 16 for optimization. Output stride is fixed to 16 in both training and testing phases.

MPII dataset. We used MPII dataset [2] to evaluate LST-Net on human pose estimation. We trained all models on MPII training set and evaluate them on MPII test set. SimpleBaseline [183] is used as the key-points estimation method.

FGVC-Aircraft dataset. FGVC-Aircraft dataset [123] includes 10,000 images across 100 aircraft categories. We used it to evaluate LST-Net on fine-grained visual recognition.

Birds-CUB200-2011 dataset. Birds-CUB200-2011 dataset [175] consists of 11,788 images from 200 bird species. We used the original dataset partition for evaluation. It is used for fine-grained visual recognition.

FGVC-Cars dataset. FGVC-Cars [88] contains 16,185 images from 196 car classes. It is also used for fine-grained visual recognition.

Stanford Dogs dataset. The Stanford Dogs [85] dataset includes 20,580 images of 120 breeds of dogs around the world. We also used it for fine-grained image categorization.

Describing Textures Dataset. Describing Textures Dataset (DTD) [30] collects 5,640 material images from 47 classes. A total of 10 pre-defined splits are used to evaluate models on texture classification. We used the mean and standard deviation

of the best top-1 accuracy over all splits as our criteria.

Indoor67 dataset. Indoor67 dataset [137] consists of 6,700 images from 67 indoor scene classes. For each class, 80 and 20 samples are used for training and test, respectively. We also employed this dataset for texture classification.

DUTS dataset. DUTS dataset [166] contains 10553 training images and 5019 test images for saliency object detection. Following [66], we employed F-measure and Mean Absolute Error (MAE) as the metrics for evaluation. For a fair comparison, the latest PoolNet [113] with ResNet-50 backbone is used as our baseline. We trained all models by taking use of the training set of DUTS dataset, and we studied the performance by using its test set.

Extended Complex Scene Saliency Dataset. Extended Complex Scene Saliency Dataset (ECSSD) [188] is a benchmark dataset for evaluation of saliency detection algorithms. It comprises 1000 images, including a large number of semantically meaningful but structurally complicated samples. The entire ECSSD is used for test.

PASCAL-S dataset. PASCAL-S dataset [104] is a saliency object detection benchmark constructed on the validation set of PASCAL VOC 2010 segmentation challenge [41]. It contains 850 natural images and all of them are used for test.

DUT-OMRON dataset. DUT-OMRON dataset [189] is a challenging saliency object detection dataset. It consists of 5168 complicated images with accurate ground truth. The whole dataset is used for evaluation.

HKU-IS dataset. HKU-IS dataset [92] is built for saliency object detection. It contains 4447 natural images, where 2500, 500 and 1447 images are used for training, validation and test, respectively. We followed [113] and used the test set of HKU-IS dataset for evaluation.

Salient Objects Dataset. Salient Objects Dataset (SOD) [130] is a pioneer work on salient object detection. It collects 300 images used in Berkeley Segmentation

Dataset [125]. In this chapter, all images are employed for evaluation.

3.4.2 Detailed structures of LST-Net

We can construct our LST-Nets w.r.t. existing CNN architectures (e.g., ResNet, VGG and AlexNet, etc.) by replacing their main building blocks, such as conventional Conv2d layers or featured bottlenecks, with our proposed LST-I or LST-II bottleneck. For each existing CNN architecture, we closely followed its instantiation on different datasets to construct our corresponding LST-Net.

LST-Net w.r.t. ResNet on CIFAR-10/100. Table 3.1 and Table 3.2 show the structures of LST-Net constructed w.r.t. ResNet on CIFAR-10/100 using LST-I and LST-II bottlenecks, respectively. We substituted each basic bottleneck of ResNet with a pair of LST-I or LST-II bottlenecks as there are two Conv2d operations in each original bottleneck. To keep the same classifier (the last FC layer), we inserted a PWConv before GAP (please refer to the third last row of Table 3.2) so that C_{in} of FC remains 64.

LST-Net w.r.t. ResNet on ImageNet. Table 3.3 and Table 3.4 present the architectures of LST-Net w.r.t. ResNet on ImageNet. For shallow models, such as ResNet-18 and ResNet-34, we built up LST-Net for ImageNet in the same way as that for CIFAR-10/100. For deep models, such as ResNet-50 and ResNet-101, we did not introduce extra PWConv before the GAP layer. We employed LST-II bottlenecks at each stage of conv2_x∼conv5_x with comparable number of parameters and computational cost.

LST-Net w.r.t. WRN on CIFAR-10/100. Table 3.5 and Table 3.6 demonstrate the details of LST-Net constructed w.r.t. WRN on CIFAR-10/100 when width multiplier is set to 8 and 10, respectively. We adopted LST-II bottlenecks for construction. Following WRN, we enlarged the core channels of each LST-II bottleneck, *i.e.*, C_{r_out}, for a few times according to the pre-defined width multiplier.

LST-Net w.r.t. WRN on ImageNet. LST-II bottlenecks are adopted to construct LST-Net w.r.t. WRN on ImageNet. Following WRN, we enlarged the core channels of each LST-II bottleneck, *i.e.*, C_{r_out}, for a few times according to the pre-defined width multiplier. Thus, it has very similar structure to the one built up w.r.t. ResNet on ImageNet.

LST-Net w.r.t. VGG on ImageNet. Table 3.7 and Table 3.8 present the LST-Nets constructed w.r.t. VGG on ImageNet using FC and GAP, respectively. As VGG has a larger spatial size at various layers than that of the corresponding layers in ResNet, we adopt LST-I bottleneck for VGG to save overhead. LST-Net (FC) adopts the same classifier as the standard VGG, *i.e.* three FC layers. In contrast, classifier of LST-Net (GAP) is similar to that of ResNet.

LST-Net w.r.t. AlexNet on ImageNet. Table 3.9 and Table 3.10 present the LST-Net constructed w.r.t. AlexNet on ImageNet using FC and GAP, respectively. For the same reason as that of VGG, we employed LST-I bottleneck. LST-Net (FC) has the same classifier as the original AlexNet. In contrast, LST-Net (GAP) takes the same classifier structure as ResNet.

LST-Net w.r.t. ShiftNet on ImageNet. Table 3.11, Table 3.12 and Table 3.13 show the LST-Net constructed w.r.t. ShiftNet on ImageNet. We employ LST-I bottleneck for the same reason as VGG and AlexNet. Besides, we set $a = 2$ for all bottlenecks. Following ShiftNet, we set the base width to 32 for LST-Net (A) and half the number for LST-Net (B) and LST-Net (C). We reduced a few bottlenecks at each stage to match its original expansion rate.

3.4.3 Ablation study

Initialization. As discussed in Section 3.3, 2D-DCT is used to initialize our spatial and channel-wise transforms W_s and W_c to reduce the feature redundancy. It is important to study whether random initialization (R.I.) can achieve similar results.

Table 3.1: LST-Net constructed using LST-I bottleneck w.r.t. ResNet on CIFAR-10/100.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat			
				20	56	110	164
Conv3x3/1	3	N.A.	16	1			
LST-I/1	16	64	16	5	17	35	53
LST-I/2	16	128	32	1			
LST-I/1	32			5	17	35	53
LST-I/2	32	256	64	1			
LST-I/1	64			5	17	35	53
GAP	64	N.A.	64	1			
FC	64	N.A.	10/100	1			

Table 3.2: LST-Net constructed using LST-II (default) bottleneck w.r.t. ResNet on CIFAR-10/100.

Type/Stride	C_{in}	C_{r_out}	C_{out}	Repeat			
				20	56	110	164
Conv3x3/1	3	N.A.	16	1			
LST-I/1	16	16	64	1			
	64			5	17	35	53
LST-II/2	64	32	128	1			
LST-II/1	128			5	17	35	53
LST-II/2	128	64	256	1			
LST-II/1	256			5	17	35	53
Conv1x1/1	256	N.A.	64	1			
GAP	64	N.A.	64	1			
FC	64	N.A.	10/100	1			

Table 3.3: LST-Net constructed w.r.t. ResNet on ImageNet and Places365-Standard (18 and 34 layers).

Name	Type/Stride	C_{in}	C_{r_out}	C_{out}	Repeat	
					18	34
conv1	Conv7×7/2	3	N.A.	64	1	1
conv2_x	MaxPool3×3/2	64	N.A.	64	1	1
	LST-II/2	64	64	256	1	1
	LST-II/1	256			1	5
conv3_x	LST-II/2	256	128	512	1	1
	LST-II/1	512			1	7
conv4_x	LST-II/2	512	256	1024	1	1
	LST-II/1	1024			1	11
conv5_x	LST-II/2	1024	512	2048	1	1
	LST-II/1	2048			1	5
	Conv1x1/1	2048	N.A.	512	1	1
	GAP	512	N.A.	512	1	1
	FC	512	N.A.	$365/1K$	1	1

We build LST-Nets of 20∼164 layers in depth using the vanilla ResNet architecture [59] to test this (LST-II bottleneck is used). We use the uniform distribution within $[-\sqrt{u}, \sqrt{u}]$ to randomly initialize W_s and W_c, where $u = 1/(C_{in} \times a^2 \times s^2)$ for W_s and $u = 1/(C_{in} \times a \times s)$ for W_c.

Table 3.14 summarizes the error rates on CIFAR-100 (similar conclusions can be made on CIFAR-10). One can see that 2D-DCT initialization obtains much better performance than R.I., which lags behind the former by 2.7% $\sim$ 7.0%. Besides, an LST-Net with R.I. is even worse than the baseline vanilla ResNet. This is because LST-Net will drop a certain amount of trivial frequencies after 2D-DCT initialization, while R.I. may be difficult to transform the channel and spatial fields of the input

Table 3.4: LST-Net constructed w.r.t. ResNet on ImageNet and Places365-Standard (50 and 101 layers).

Name	Type/Stride	C_{in}	C_{r_out}	C_{out}	Repeat	
					50	101
conv1	Conv7×7/2	3	N.A.	64	1	1
conv2_x	MaxPool3×3/2	64	N.A.	64	1	1
	LST-II/2	64	64	256	1	1
	LST-II/1	256			9	9
conv3_x	LST-II/2	256	128	512	1	1
	LST-II/1	512			13	13
conv4_x	LST-II/2	512	256	1024	1	1
	LST-II/1	1024			20	77
conv5_x	LST-II/2	1024	512	2048	1	1
	LST-II/1	2048			9	9
	GAP	2048	N.A.	2048	1	1
	FC	2048	N.A.	$365/1K$	1	1

feature into different frequencies with PWConv and DWConv operations, resulting in unnecessary loss of some crucial information.

The selection of parameter τ. We study the effect of parameter τ (refer to Equation 3.5) on LST-Net. We built a 20-layer LST-Net in favor of ResNet architecture, and tested on CIFAR100. We search for the optimal value of τ in the range of {0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}}. The error rates are {28.92%, 28.32%, 28.28%, 28.86%, 28.21%, 28.43%, 28.70%}, where the best result is 28.21% when $\tau = 10^{-4}$. Thus, we set $\tau = 10^{-4}$ by default in all experiments of this chapter. Note that when $\tau = 0$, ST-ReLU is reduced to standard ReLU, but its error rate is larger than other values of τ. This validates that our hybrid ReLU-ST activation scheme works better than ReLU for LST-Net.

LST-I vs LST-II. We discuss the pros and cons of our proposed LST-I and

Table 3.5: LST-Net constructed w.r.t. WRN on CIFAR-10/100 (width multiplier = 8).

Type/Stride	C_{in}	C_{r_out}	C_{out}	Repeat			
				16	22	28	40
Conv3x3/1	3	N.A.	16	1			
LST-II/1	16 512	128	512	1 1	2	3	5
LST-II/2 LST-II/1	512 1024	256	1024	1 1	2	3	5
LST-II/2 LST-II/1	1024 2048	512	2048	1 1	2	3	5
Conv1x1/1	2048	N.A.	512	1			
GAP	512	N.A.	512	1			
FC	512	N.A.	10/100	1			

Table 3.6: LST-Net constructed w.r.t. WRN on CIFAR-10/100 (width multiplier = 10).

Type/Stride	C_{in}	C_{r_out}	C_{out}	Repeat			
				16	22	28	40
Conv3x3/1	3	N.A.	16	1			
LST-II/1	16 640	160	640	1 1	2	3	5
LST-II/2 LST-II/1	640 1280	320	1280	1 1	2	3	5
LST-II/2 LST-II/1	1280 2560	640	2560	1 1	2	3	5
Conv1x1/1	2560	N.A.	640	1			
GAP	640	N.A.	640	1			
FC	640	N.A.	10/100	1			

Table 3.7: LST-Net (FC) constructed w.r.t. VGG on ImageNet and Places365-Standard.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv3×3/1	3	N.A.	64	1
MaxPool2×2/2	64	N.A.	64	1
LST-I/1	64	512	128	
MaxPool2×2/2	128	N.A.	128	1
LST-I/1	128	1024	256	
LST-I/1	256			
MaxPool2×2/2	256	N.A.	256	1
LST-I/1	256	2048	512	
LST-I/1	512			
MaxPool2×2/2	512	N.A.	512	1
LST-I/1	512	2048	512	2
FC	25088	N.A.	4096	1
FC	4096	N.A.	4096	
FC	4096	N.A.	$365/1K$	

LST-II bottlenecks in building up a LST-Net. For LST-I, one is free to replace a conventional Conv2d by LST-I in many existing architectures, such as ResNet [59], AlexNet [90], VGG [151], *etc*. For example, a basic ResNet bottleneck can be replaced by a pair of LST-I bottlenecks as it has two Conv2d operations. For LST-II, due to the expansion factor of LST-II, parameters and overhead of the associated PWConv operation in the shortcut path are increased by a^2 times compared to LST-I. Thus, LST-II is not suitable to architectures with larger spatial size at earlier layers, such as AlexNet and VGG. LST-II will also increase the output channel number of the last bottleneck, but this issue can be easily solved with an extra PWConv operation, which is cheap compared to the entire CNN model in terms of number of parameters and computational cost. When building a deeper CNN model, such as ResNet-50 or

Table 3.8: LST-Net (GAP) constructed w.r.t. VGG on ImageNet and Places365-Standard.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv3×3/1	3	N.A.	64	1
MaxPool2×2/2	64	N.A.	64	1
LST-I/1	64	512	128	
MaxPool2×2/2	128	N.A.	128	1
LST-I/1	128	1024	256	
LST-I/1	256			
MaxPool2×2/2	256	N.A.	256	1
LST-I/1	256	2048	512	
LST-I/1	512			
MaxPool2×2/2	512	N.A.	512	1
LST-I/1	512	2048	512	2
GAP	512	N.A.	512	1
FC	512	N.A.	$365/1K$	1

ResNet-101, it is more suitable to use LST-II than LST-I. In Table 3.15, we construct LST-Nets with LST-I and LST-II bottlenecks w.r.t. ResNet architecture and compare them on CIFAR-100. The vanilla ResNet is included as the baseline. Both LST-I and LST-II outperform the baseline by a large margin, while LST-II performs better than its LST-I counterpart. In the remaining experiments of this chapter, if not specified, we adopt LST-II bottleneck to build ResNet models by default.

Comparison of ST-ReLU to other activations. We use a 20-layer and a 164-layer LST-Net to compare our ReLU-ST with ReLU [131], leaky ReLU (LReLU) [121], parametric ReLU (PReLU) [58], ELU [31] and SELU [87] on CIFAR-100. For comparison, we remove ST operations in LST bottleneck and replace ReLU by other activations. Table 3.16 presents the top-1 error rates achieved by different activations. One can see that ReLU-ST outperforms other activations for both 20- and 164-layer

Table 3.9: LST-Net (FC) constructed w.r.t. AlexNet on ImageNet and Places365-Standard.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv11×11/4	3	N.A.	64	1
MaxPool3×3/2	64	N.A.	64	1
LST-I/1	64	768	192	
MaxPool3×3/2	192	N.A.	192	1
LST-I/1	192	1536	384	
MaxPool3×3/2	384	N.A.	384	1
LST-I/1	384	1024	256	
LST-I/1	256			
FC	9216	N.A.	4096	1
FC	4096	N.A.	4096	
FC	4096	N.A.	$365/1K$	

Table 3.10: LST-Net (GAP) constructed w.r.t. AlexNet on ImageNet and Places365-Standard.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv11×11/4	3	N.A.	64	1
MaxPool3×3/2	64	N.A.	64	1
LST-I/1	64	768	192	
MaxPool3×3/2	192	N.A.	192	1
LST-I/1	192	1536	384	
MaxPool3×3/2	384	N.A.	384	1
LST-I/1	384	1024	256	
LST-I/1	256			
GAP	512	N.A.	512	1
FC	512	N.A.	$365/1K$	1

Table 3.11: LST-Net (A) constructed w.r.t. ShiftNet-A on ImageNet.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv7×7/2	3	N.A.	32	1
LST-I 5×5/2	32	128	32	1
LST-I 5×5/1				4
LST-I 5×5/2	32	256	64	1
LST-I 5×5/1	64			2
LST-I 3×3/2	64	512	128	1
LST-I 3×3/1	128			
LST-I 3×3/2	128	1024	256	1
LST-I 3×3/1	256			
GAP	256	N.A.	256	1
FC	256	N.A.	$1K$	1

Table 3.12: LST-Net (B) constructed w.r.t. ShiftNet-B on ImageNet.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv7×7/2	3	N.A.	16	1
LST-I 5×5/2	16	64	16	1
LST-I 5×5/1				4
LST-I 5×5/2	16	128	32	1
LST-I 5×5/1	32			2
LST-I 3×3/2	32	256	64	1
LST-I 3×3/1	64			
LST-I 3×3/2	64	512	128	1
LST-I 3×3/1	128			
GAP	128	N.A.	128	1
FC	128	N.A.	$1K$	1

Table 3.13: LST-Net (C) constructed w.r.t. ShiftNet-C on ImageNet.

Type/Stride	C_{in}	$a^2 \times C_s$	C_{out}	Repeat
Conv7×7/2	3	N.A.	16	1
LST-I 5×5/2 LST-I 5×5/1	16	64	16	1
LST-I 5×5/2 LST-I 5×5/1	16 32	128	32	1
LST-I 3×3/2 LST-I 3×3/1	32 64	256	64	1
LST-I 3×3/2 LST-I 3×3/1	64 128	512	128	1
GAP	128	N.A.	128	1
FC	128	N.A.	$1K$	1

Table 3.14: Comparison (error rates, %) of different initialization methods on CIFAR-100.

Method / Depth	20	56	110	164
ResNet [59] (R.I.)	30.88	27.62	26.23	26.07
LST-Net (R.I.)	31.12	29.92	28.95	28.94
LST-Net (2D-DCT)	**28.21**	**24.09**	**22.66**	**21.94**

Table 3.15: Comparison (error rates, %) of different bottlenecks on CIFAR-100.

Method / Depth	20	56	110	164
ResNet [59]	30.88	27.62	26.23	26.07
LST-I	**27.64**	25.08	23.76	23.15
LST-II	28.21	**24.09**	**22.66**	**21.94**

Table 3.16: Comparison (error rates, %) of different activation methods on CIFAR-100.

Depth	ReLU-ST	LReLU [121]	SELU [87]	ReLU [131]	PReLU [58]	ELU [31]
20	**28.21**	28.37	28.69	28.92	29.35	31.60
164	**21.94**	22.44	22.84	22.86	23.91	29.94

Table 3.17: Results (error rates, %) by different networks under ResNet on CIFAR-10/100.

Depth	Model	Param/FLOPs	C10/C100
20	ResNet [59]	0.27M/40.8M	7.7/30.9
	PreactResNet [59]	0.27M/40.8M	7.7/30.8
	ShiftResNet [180]	**0.16M/27M**	9.0/31.4
	FE-Net [27]	**0.16M/27M**	8.3/30.8
	SENet [71]	0.28M/40.8M	7.6/30.5
	CBAM [178]	0.28M/40.8M	7.3/30.3
	LST-Net	0.20M/34M	**6.7/28.2**
56	ResNet [59]	0.86M/126M	6.6/27.6
	PreactResNet [59]	0.86M/126M	6.5/27.6
	ShiftResNet [180]	**0.55M/84M**	7.3/27.9
	FE-Net [27]	**0.55M/84M**	8.3/30.8
	SENet [71]	0.87M/126M	6.4/27.5
	CBAM [178]	0.87M/126M	6.0/27.1
	LST-Net	0.59M/94M	**5.6/24.1**
110	ResNet [59]	1.73M/253M	6.6/25.2
	PreactResNet [59]	1.73M/253M	6.2/24.1
	ShiftResNet [180]	1.18M/187M	6.8/27.4
	FE-Net [27]	N.A.	N.A.
	SENet [71]	1.74M/253M	5.2/23.9
	CBAM [178]	1.74M/253M	5.1/23.5
	LST-Net	**1.17M/183M**	**5.0/22.7**

Table 3.18: Results (error rates, %) by different networks under WRN on CIFAR-10/100.

Depth	Multiplier	Model	Param/FLOPs	C10/C100
16	8	WRN [198]	10.96M/2.00G	4.80/22.03
		LST-Net	**7.41M/1.30G**	**4.70/20.88**
	10	WRN [198]	17.12M/3.12G	4.49/21.52
		LST-Net	**11.53M/2.01G**	**4.46/20.21**
22	8	WRN [198]	17.16M/2.91G	4.56/21.21
		LST-Net	**10.94M/1.82G**	**4.40/19.33**
	10	WRN [198]	26.80M/4.54G	4.44/20.75
		LST-Net	**17.01M/2.82G**	**4.31/18.57**
28	10	WRN [198]	36.48M/5.95G	4.17/20.50
		LST-Net	**22.50M/3.63G**	**4.03/18.23**
	12	WRN [198]	43.42M/8.56G	4.33/20.41
		LST-Net	**32.29M/5.20G**	**3.94/17.93**
40	4	WRN [198]	8.91M/1.41G	4.97/22.89
		LST-Net	**5.52M/0.87G**	**4.31/19.14**
	8	WRN [198]	35.75M/5.63G	4.66/19.38
		LST-Net	**21.52M/3.38G**	**3.76/18.56**

LST-Nets. The gain is higher for deeper models.

3.4.4 Evaluation on image classification

Evaluation on CIFAR-10 and CIFAR-100. We build our LST-Net models w.r.t. the popular architectures, including ResNet [59] and Wide Residual Networks (WRN) [198], and compare LST-Net with state-of-the-art CNNs in those families, e.g. Pre-activation ResNet [59], SENet [71], CBAM [178], and two other models, i.e., ShiftResNet [180] and FE-Net [27].

Table 3.17 and Table 3.18 present the results on CIFAR-10/100. We can have the following findings. First, LST-Net achieves the lowest error rates under differ-

ent network depths with almost the least number of parameters and FLOPs (very close to ShiftResNet and FE-Net). This validates its effectiveness and efficiency. LST-Net outperforms ResNet and PreactResNet while reducing over 40% parameters and 35% overhead. Compared to SENet and CBAM, LST-Net does not need extra paths while it achieves even better results. For instance, a 110-layer LST-Net improves SENet/CBAM of the same depth by 0.2%/0.1% and 1.2%/0.8% on CIFAR-10 and CIFAR-100, respectively. Besides, LST-Net outperforms both ShiftResNet and FENet by a large margin with comparable parameters and overhead. For example, a 20-layer LST-Net reduces the error rates of ShiftResNet and FE-Net by 2.3/3.2% and 1.6/2.6% on CIFAR-10/100, respectively.

Second, when we switch to wider CNN models, our bottleneck can save more parameters and computational cost because the computation of PWConv dominates an entire LST bottleneck when it is wide enough (the cost of DWConv can be neglected). We can obtain consistent performance boost of our LST-Net with the increase of width and/or depth. In contrast, the corresponding WRN architecture is less effective in improving its results with more channels and/or layers. For example, for a 28-layer WRN, the error rates will rise by 4.17% $\sim$ 4.33% on CIFAR-10 when the width multiplier is increased from 10 to 12.

Evaluation on ImageNet. We then evaluate LST-Net on ImageNet [35] for large-scale image classification. We construct LST-Nets under the widely used network architectures, including ResNet [59], WRN [198], AlexNet [90] and VGG (with 11 layers) [151]. We also build LST-Nets w.r.t. ShiftNet [180] and MobileNet V2 [149].

Specifically, for ResNet or WRN architecture, we construct LST-Net using LST-II bottleneck, and for AlexNet/VGG, we build LST-Net (FC) by replacing Conv2d layers with LST-I bottlenecks. We also change the original classifier layer in AlexNet/VGG into GAP [105] followed by a dense layer in the same way as [208], resulting in LST-

Net (GAP). Similarly, the standard AlexNet/VGG can be modified in the same way, resulting in AlexNet (GAP)/VGG (GAP). Since BN [81] is used in our bottleneck, we further insert a BN layer after each Conv2d of AlexNet/VGG, termed as AlexNet/VGG (BN). For ShiftNet architecture, we build the LST-Nets by adjusting the stride, kernel size, number of stages, *etc.*, according to its variants A, B, and C with different depth and width. For MobileNet V2, we build LST-Net (M-V2) by replacing the Inverted Residual bottlenecks in MobileNet V2 with the LST-I bottlenecks and reusing the original settings, including kernel size, stride, expansion rate $\mathcal{E}$, number of bottlenecks, *etc*. To adapt LST-I bottleneck to the Inverted Residual bottleneck, we made three changes: (1) we replaced each ReLU in the original LST-I bottleneck by ReLU6 and the ReLU-ST activation scheme was adapted to ReLU6-ST, where we set $\tau = 1 \times 10^{-8}$ in ST to take care of the need for a linear transform; (2) we removed PWConv and BN in channel-wise transform T_c when the expansion rate $\mathcal{E} = 1$; (3) we removed the downsample operator D and element-wise plus when $\mathcal{E} > 1$ while stride>1 or $C_{in} \neq C_{out}$. Table 3.19 shows the structure of LST-Net built up w.r.t. MobileNet V2, where we modified LST-I bottleneck in this experiment. Figure 3.4 illustrates the LST-I bottlenecks corresponding to MobileNet V2 bottlenecks. Batch size, initial learning rate and weight decay are set to 256, 0.05 and 5×10^{-4}, respectively. We adopted a cosine learning rate decay strategy and trained our model for 150 epochs.

Table 3.20, Table 3.21, and Table 3.22 summarize the results. One can see that LST-Net consistently surpasses ResNet, SENet and CBAM of the same depth with fewer parameters and less overhead. An 18-layer LST-Net even achieves lower top-1 error rates than the standard ResNet-34 on ImageNet. Despite different depth, increasing width of LST-Net with WRN architecture steadily increases its accuracy. Meanwhile, LST-Net saves larger proportion of parameters and overhead compared to WRN. LST-Net with AlexNet or VGG architecture is much more robust to differ-

Table 3.19: LST-Net constructed w.r.t. MobileNet V2 on ImageNet.

Type/Stride	C_{in}	$a^2 \times C_s$ ($\mathcal{E}$)	C_{out}	Repeat
Conv3x3/1	3	N.A.	16	1
Modified LST-I/1	16	16 (1)	16	1
Modified LST-I/2	16	96 (6)	24	1
Modified LST-I/1	24	144 (6)		1
Modified LST-I/2	24	144 (6)	32	1
Modified LST-I/1	32	192 (6)		2
Modified LST-I/1	32	192 (6)	64	1
	64	384 (6)		3
Modified LST-I/2	64	384 (6)	96	1
Modified LST-I/1	96	576 (6)		2
Modified LST-I/2	96	576 (6)	160	1
Modified LST-I/1	160	960 (6)		2
Modified LST-I/1	160	960 (6)	320	1
PWConv	320	N.A.	1280	1
GAP	1280	N.A.	1280	1
FC	1280	N.A.	$1K$	1

ent classifier structures than the standard AlexNet or VGG because LST-Net learns structured features, which are well suited for channel-wise operations (see our discussion in Section 3.3.1). Meanwhile, LST-Net (FC) can reduce top-1/top-5 error rates of AlexNet (BN) and VGG (BN) by 2.61%/2.62% and 1.06%/0.40%, respectively. LST-Net also shows better performance under the ShiftNet architecture. Compared with all the three variants, our LST-Net reduces the top-1 error rate of its corresponding counterpart by 0.6% $\sim$2.3% with similar number of parameters. LST-Net (M-V2) achieves a 72.3% top-1 accuracy, outperforming MobileNet V2 by 0.4% using the same number of parameters and computational cost. This again validates the

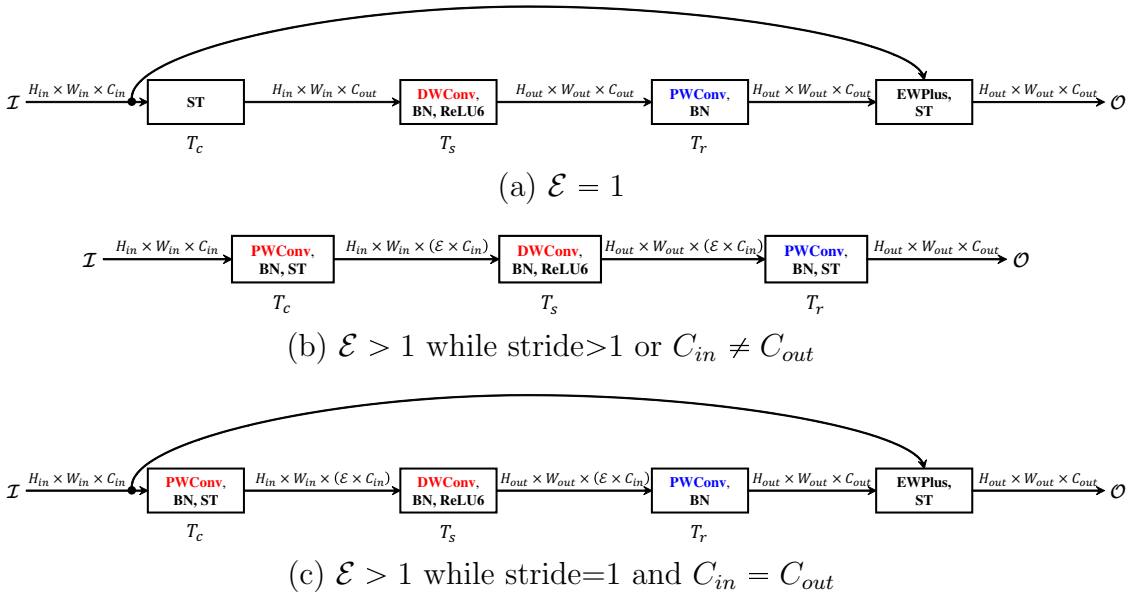


Figure 3.4: Illustration of LST-I bottleneck w.r.t. the Inverted Residual bottleneck in MobileNet V2. EWPlus means element-wise plus. PWConv/DWConv in red font indicates initialization with 2D-DCT while blue font suggests random initialization.

generality and superiority of our LST method.

We present the convergence curves of LST-Net on ImageNet in terms of top-1 and top-5 error rates. Figs. 3.5 and 3.6 compare the convergence curves of ResNet-18, ResNet-50 and their corresponding LST-Nets. One can see that our LST-Nets achieve lower error rates during the entire training process.

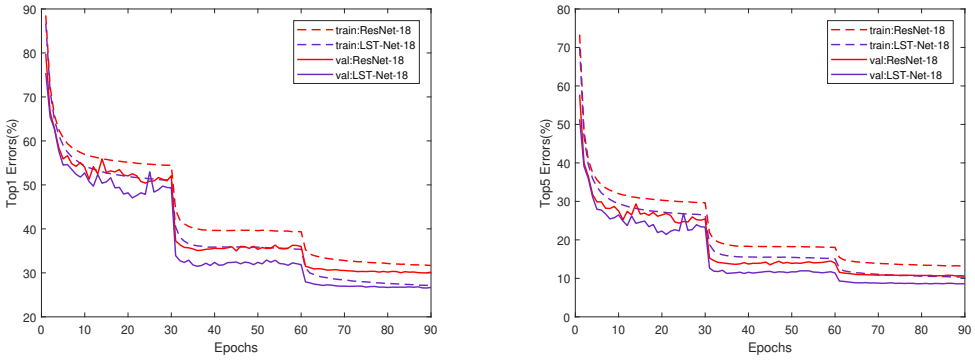
3.4.5 Evaluation on robustness to common corruptions

We study the robustness of LST-Net to common corruptions in input by using the ImageNet-C dataset [63]. The mean corruption error (mCE) defined in [63] is used as our criteria. We construct LST-Net according to the ResNet architecture and compare it with the vanilla ResNet [59], SENet [71] and CBAM [178]. To examine the role of ST (please refer to Section 3.3.2) in improving the robustness of LST-Net, we also test LST-Net without ST in activation.

Table 3.23 lists the mCE and corruption errors for each type of corruption. One

Table 3.20: Results (error rates, %) by different networks under ResNet on ImageNet.

Depth	Model	Param/FLOPs	Top-1/Top-5
18	ResNet [59]	11.69M/1.81G	30.24/10.92
	SENet [71]	11.78M/1.81G	29.41/10.22
	CBAM [178]	11.78M/1.82G	29.31/10.17
	LST-Net	**8.03M/1.48G**	**26.55/8.59**
34	ResNet [59]	21.79M/3.66G	26.70/8.58
	SENet [71]	21.96M/3.66G	26.13/8.35
	CBAM [178]	21.96M/3.67G	26.01/8.40
	LST-Net	**13.82M/2.56G**	**23.92/7.24**
50	ResNet [59]	25.56M/4.09G	23.85/7.13
	SENet [71]	28.09M/4.09G	23.14/6.70
	CBAM [178]	28.09M/4.10G	22.98/6.68
	LST-Net	**23.33M/4.05G**	**22.78/6.66**
101	ResNet [59]	44.55M/7.80G	22.63/6.44
	SENet [71]	49.29M/7.81G	22.35/6.19
	CBAM [178]	49.29M/7.81G	21.65/5.95
	LST-Net	**42.36M/7.75G**	**21.63/5.94**

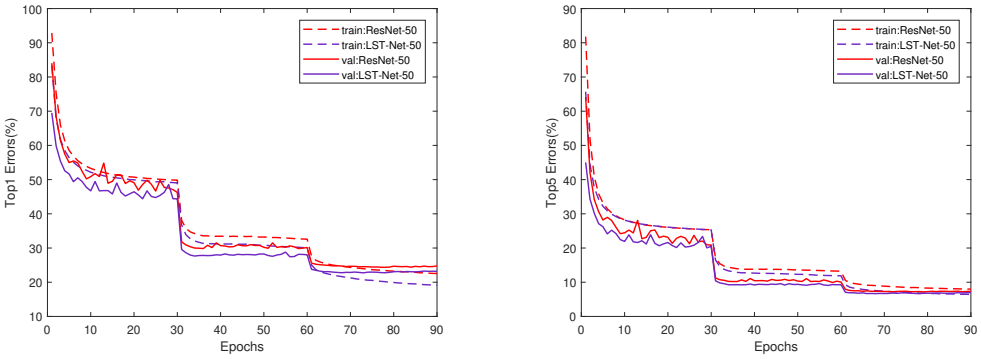


(a) Top-1 error rates (b) Top-5 error rates

Figure 3.5: Convergence curves of ResNet-18 and our LST-Net on ImageNet.

Table 3.21: Results (error rates, %) by different networks under WRN on ImageNet.

Depth	Mulp.	Model	Param/FLOPs	Top-1/Top-5
18	1	WRN [198]	11.69M/1.81G	30.24/10.92
		LST-Net	**8.03M/1.48G**	**26.55/8.59**
	1.5	WRN [198]	25.88M/3.87G	27.06/9.00
		LST-Net	**17.53M/3.21G**	**24.44/7.51**
	2	WRN [198]	45.62M/6.70G	25.58/8.06
		LST-Net	**30.68M/5.59G**	**23.49/6.93**
	3	WRN [198]	101.78M/14.72G	24.06/7.33
		LST-Net	**67.96M/12.31G**	**22.33/6.52**
34	1	WRN [198]	21.79M/3.66G	26.70/8.58
		LST-Net	**13.82M/2.56G**	**23.92/7.24**
	1.5	WRN [198]	48.61M/8.03G	24.50/7.58
		LST-Net	**30.42M/5.59G**	**22.29/6.30**
	2	WRN [198]	86.04M/14.09G	23.39/7.00
		LST-Net	**53.49M/9.79G**	**21.44/6.11**



(a) Top-1 error rates (b) Top-5 error rates

Figure 3.6: Convergence curves of ResNet-50 and our LST-Net on ImageNet.

Table 3.22: Results (error rates, %) by different networks under other architectures on ImageNet.

Model	Param/FLOPs	Top-1/Top-5
AlexNet [90]	61.10M/0.71G	43.45/20.91
AlexNet (BN)	61.10M/0.71G	41.93/20.02
AlexNet (GAP)	2.73M/0.66G	51.13/26.33
LST-Net (FC)	**60.30M/0.62G**	**39.32/17.40**
LST-Net (GAP)	**2.25M/0.60G**	**39.91/17.86**
VGG [151]	132.86M/7.61G	30.98/11.37
VGG (BN)	132.86M/7.61G	29.62/10.19
VGG (GAP)	9.73M/7.49G	33.40/12.20
LST-Net (FC)	**128.63M/5.89G**	**28.56/9.79**
LST-Net (GAP)	**6.63M/5.04G**	**29.23/10.26**
ShiftNet-A [180]	**4.1M**/1.4G	29.9/10.3
ShiftNet-B [180]	**1.1M**/N.A.	38.8/16.4
ShiftNet-C [180]	**0.78M**/N.A.	41.2/18.0
LST-Net (A)	4.3M/**1.2G**	**29.3/10.0**
LST-Net (B)	1.2M/389.5M	**36.9/14.8**
LST-Net (C)	0.84M/342.5M	**38.9/16.3**
MobileNet V2 [149]	**3.4M/300M**	28.1%/N.A.
LST-Net (M-V2)	**3.4M/300M**	**27.7%/9.4%**

can see that LST-Net achieves lower mCE than its competitors of the same depth. It significantly reduces the mCE of the vanilla ResNet by 3.69% (18-layer) / 6.47% (50-layer), and also improves SENet and CBAM by at least 2.76% (18-layer) / 2.02% (50-layer). Though SENet and CBAM use extra paths which work well on clean images, the pooling operations in these paths may produce biased results in the existence of corruptions when the model is shallow. In contrast, LST does not need such extra paths and its robustness comes from the compact and sparser features. In addition, the ST operation in our ST-ReLU activation function can strengthen the robustness of LST-Net to most types of corruptions. With ST, the mCE of

LST-Net-18/50 is reduced by 0.45%/0.31%.

3.4.6 Evaluation on large-scale scene recognition

We evaluate LST-Net for large-scale scene recognition on Places365-Standard dataset [209]. We build up LST-Nets w.r.t. ResNet [59], AlexNet [90] and 11-layer VGG [151] for fair comparison. We compare LST-Net with its counterpart networks. Table 3.24 and Table 3.25 present top-5 accuracies obtained using ten-crop estimation.

One can see that LST-Net surpasses its counterparts. This validates that LST-Net is also effective for the large-scale scene recognition task. In particular, an 18-layer LST-Net can even surpass ResNet-50 by 1.27% while saving nearly 70% of the total parameters and 64% of the total FLOPs. Meanwhile, LST-Net under ResNet-50 architecture achieves the best performance on Places365-Standard dataset, 0.96% higher than its closest follower, CBAM-50. Besides, by replacing the last linear layers of AlexNet by GAP, the accuracy drops significantly, while LST-Net(GAP) is robust in this case. AlexNet (BN) only slightly improves AlexNet [90], while LST-Net (FC) built up w.r.t. AlexNet, also using BN and FC, improves much AlexNet (BN). Similarly, the accuracy drops by nearly 1% when the last linear layers of VGG is replaced by GAP, while LST-NET (GAP) constructed under VGG is also robust in the same case. LST-Net (FC) built up w.r.t. VGG improves VGG (BN) by 0.24%.

3.4.7 Evaluation on fine-grained visual recognition

We evaluate the performance of our LST-Net on fine-grained visual recognition. Four datasets are used for evaluation, including FGVC-Aircraft, Birds-CUB200-2011, FGVC-Cars and Stanford Dogs. Neither part annotations nor bounding boxes are used for training or testing. For fair comparison, we follow [194] for experimental setting. For both ResNet-50 and our LST-Net w.r.t. ResNet-50, we report the best performance we achieved.

Table 3.23: Comparison of model robustness to common corruptions on ImageNet-C.

Network	mCE	Noise			Blur			Weather			Digital				Extra		
		Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG	Saturate	Spatter	Gaus.	Blur	Speckle
ResNet-18 [59]	85.29	87	89	89	88	93	90	88	88	87	81	73	81	93	81	89	72	81	86		86
SENet-18 [71]	83.97	85	86	87	87	93	88	88	85	85	79	73	82	92	84	92	71	81	86		82
CBAM-18 [178]	84.97	86	88	87	88	93	89	90	85	86	80	74	82	92	81	89	71	81	87		85
LST-Net-18 (w/o ST)	80.34	81	82	85	85	**91**	**83**	**85**	**82**	83	**75**	**68**	79	**90**	74	**85**	**66**	75	84		78
LST-Net-18 (w/ ST)	**79.89**	**80**	**81**	**83**	**84**	**91**	**83**	**85**	**82**	**82**	**75**	**68**	**78**	**90**	**73**	**85**	**66**	**74**	**83**		**76**
ResNet-50 [59]	77.01	78	80	80	79	90	81	80	80	78	69	62	75	88	76	78	62	74	78		76
SENet-50 [71]	74.47	76	77	76	77	89	79	82	75	76	70	59	75	85	71	74	58	69	76		71
CBAM-50 [178]	72.56	**69**	**71**	**71**	80	86	**77**	78	75	76	69	61	74	85	63	**70**	58	68	78		**66**
LST-Net-50 (w/o ST)	70.85	71	72	**71**	77	85	**77**	**75**	**73**	**72**	66	**58**	**70**	82	**61**	72	**56**	**65**	76		67
LST-Net-50 (w/ ST)	**70.54**	71	72	**71**	**76**	**84**	**77**	**75**	**73**	**72**	**65**	**58**	**70**	**81**	**61**	72	**56**	**65**	**75**		67

Table 3.24: Results of networks under ResNet on Places365-Standard dataset.

Model	Param/FLOPs	Top-5 Acc. (%)
ResNet-50 [59]	25.24M/4.09G	85.08
SENet-50 [71]	26.77M/4.09G	85.86
CBAM-50 [178]	26.79M/4.09G	86.22
LST-Net (ResNet-18)	**7.71M/1.48G**	**86.35**
LST-Net (ResNet-34)	**13.50M/2.56G**	**86.94**
LST-Net (ResNet-50)	**23.01M/4.05G**	**87.18**

Table 3.25: Results of networks under other architectures on Places365-Standard dataset.

Model	Param/FLOPs	Top-5 Acc. (%)
AlexNet [90]	58.50M/0.71G	82.89
AlexNet (BN)	58.50M/0.71G	82.98
AlexNet (GAP)	2.56M/0.66G	77.89
LST-Net (FC)	**57.70M/0.64G**	**83.99**
LST-Net (GAP)	**2.09M/0.62G**	**82.95**
VGG [151]	130.26M/7.61G	84.91
VGG (BN)	130.26M/7.61G	85.09
VGG (GAP)	9.73M/7.49G	83.95
LST-Net (FC)	**127.15M/6.01G**	**85.33**
LST-Net (GAP)	**6.30M/5.89G**	**85.12**

Table 3.26 presents the fine-grained visual recognition results. Our LST-Net outperforms ResNet-50 on all datasets by at least 2% in terms of top-1 accuracy. It is worth noting that LST-Net obtains a considerable gain of 12.37% on Stanford Dogs dataset.

3.4.8 Evaluation on texture classification

We employ DTD [30] and Indoor67 dataset [137] to study the performance of our LST-Net on texture classification. For fair comparison, we follow [109] for experi-

Table 3.26: Fine-grained visual recognition results (top-1 accuracy, %) of LST-Net.

Backbone	FGVC -Aircraft	Birds -CUB200-2011	FGVC -Cars	Stanford Dogs
ResNet-50 [59]	86.70	82.21	88.68	76.23
LST-Net	**90.04**	**85.61**	**91.51**	**88.60**

Table 3.27: Texture classification results (top-1 accuracy, %) of LST-Net.

Backbone	DTD	Indoor67
ResNet-50 [59]	63.68±0.85	79.63
LST-Net	**67.47±1.09**	**81.72**

mental setting to compare ResNet-50 and our LST-Net w.r.t. ResNet-50.

Table 3.27 presents the texture classification results. It can be observed that LST-Net outperforms ResNet-50 on both datasets. Specifically, our method improves the average top-1 accuracy of ResNet-50 by 3.79% on DTD. In addition, it outperforms top-1 accuracy of ResNet-50 by 2.09% on Indoor67 dataset.

3.4.9 Evaluation on object detection and instance segmentation

We report results of object detection and instance segmentation on MS-COCO [107]. We trained all models on COCO-2017 training set and evaluated on COCO-2017 validation set using standard COCO AP metric of single scale. For comparison, we adopted some popular one-stage and two-stage object detection and/or instance segmentation frameworks, including Faster-RCNN [144], RetinaNet [106], FCOS [161], Mask R-CNN [57], and Cascade Mask R-CNN [12]. We replaced vanilla ResNet-50 with our LST-Net as the backbone model of each framework, and the settings for hyper-parameters and detection heads remained unchanged.

Table 3.28 and Table 3.29 present the object detection results and instance seg-

Table 3.28: Object detection results (%) of LST-Net on MS-COCO validation set.

Detector	Backbone	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN [144]	ResNet [59]	37.4	58.1	40.4	21.2	41.0	48.1
	LST-Net	**40.8**	**62.2**	**44.3**	**24.8**	**44.7**	**53.1**
RetinaNet [106]	ResNet [59]	36.5	55.4	39.1	20.4	40.3	48.1
	LST-Net	**38.7**	**58.5**	**41.7**	**22.2**	**42.7**	**51.2**
FCOS [161]	ResNet [59]	36.6	55.7	38.8	20.7	40.1	47.4
	LST-Net	**38.8**	**58.7**	**41.5**	**22.5**	**42.4**	**50.2**
Mask R-CNN [57]	ResNet [59]	38.2	58.8	41.4	21.9	40.9	49.5
	LST-Net	**41.3**	**62.5**	**45.0**	**25.1**	**45.1**	**54.3**
Cascade Mask R-CNN [12]	ResNet [59]	41.2	59.4	45.0	23.9	44.2	54.4
	LST-Net	**43.9**	**62.6**	**47.9**	**26.3**	**47.4**	**57.8**

Table 3.29: Instance segmentation results (%) of LST-Net on MS-COCO validation set.

Detector	Backbone	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN [57]	ResNet [59]	34.7	55.7	37.2	18.3	37.4	47.2
	LST-Net	**37.1**	**59.3**	**39.4**	**20.9**	**40.5**	**50.8**
Cascade Mask R-CNN [12]	ResNet [59]	35.9	56.6	38.4	19.4	38.5	49.3
	LST-Net	**38.1**	**59.7**	**40.9**	**21.4**	**41.3**	**51.9**

mentation results on MS-COCO validation set, respectively. One can clearly see that LST-Net significantly boosts the performance of vanilla ResNet under all settings. mAP is improved by 2.2%∼3.4% in object detection and 2.2%∼2.4% in semantic segmentation. This suggests that our LST-Net is more reliable on object detection and instance segmentation tasks.

Table 3.30: Semantic segmentation results (%) of LST-Net on PASCAL VOC2012.

Seg. Method	Backbone	mIoU	Overall Acc. (%)	Freq. W. Acc.(%)
DeepLab V3 [23]	ResNet [59]	0.767	93.89	88.76
	LST-Net	**0.790**	**94.49**	**90.07**
DeepLab V3+ [24]	ResNet [59]	0.771	94.13	89.14
	LST-Net	**0.792**	**94.58**	**90.17**

3.4.10 Evaluation on semantic segmentation

We report semantic segmentation results on PASCAL VOC [41]. We closely follow the experimental settings of [24] for fair comparison. Specifically, we trained all models on the original PASCAL VOC2012 training set as well as extra annotations provided by SBD [56], and we validated the models on the original PASCAL VOC2012 validation set. Similar to Section 3.4.9, we also substituted vanilla ResNet-50 for our LST-Net as the backbone model. We resorted to two popular semantic segmentation methods for evaluation, including DeepLab V3 [23] and DeepLab V3+ [24]. In favor of both methods, we replaced stride with dilation in the spatial transform T_s of LST bottlenecks at Stage 4, while the rest transforms remained unchanged. The performance of semantic segmentation is measured in terms of mIoU, overall accuracy, and frequency weighted accuracy.

Table 3.30 demonstrates the results. One can see that our LST-Net improves ResNet in terms of the four semantic segmentation indices. It is worthwhile noting that mIoU is significantly improved by more than 0.02 for both DeepLab V3 and DeepLab V3+.

Table 3.31: Human pose estimation results (AP, %) of LST-Net on MPII dataset.

Backbone	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Mean	Mean@0.1
ResNet-50 [59]	96.35	95.33	88.99	83.18	88.42	83.96	79.59	88.53	33.91
LST-Net	**96.76**	**95.35**	**89.42**	**84.00**	**88.56**	**84.95**	**80.63**	**89.05**	**34.41**

Table 3.32: Human pose estimation results (%) of LST-Net on COCO key-points detection dataset.

Backbone	Input Size	AP	$AP_{0.5}$	$AP_{0.75}$	AP_M	AP_L	AR	$AR_{0.5}$	$AR_{0.75}$	AR_M	AR_L
ResNet-50 [59]	256×192	70.4	88.6	78.3	67.1	77.2	76.3	92.9	**83.4**	72.1	**82.4**
LST-Net		**74.1**	**92.5**	**81.5**	**71.1**	**78.7**	**77.0**	**93.4**	**83.5**	**73.7**	82.0
ResNet-50 [59]	384×288	72.2	89.3	78.9	68.1	79.7	77.6	93.2	83.8	72.8	**84.6**
LST-Net		**76.5**	**93.6**	**83.7**	**73.4**	**81.3**	**79.0**	**94.1**	**85.2**	**75.6**	84.2

3.4.11 Evaluation on human pose estimation

To study the performance of LST-Net on human pose estimation, we use SimpleBaseline [183] as the key-points detection method. We only replace the backbone ResNet-50 with our LST-Net, where both backbone models are pre-trained on ImageNet dataset. All implementation remains the same with SimpleBaseline [183]. Both MPII dataset [2] and COCO key-point detection dataset [107] are used for evaluation. For each dataset, we use the training set to train all the models and validation set for test.

Table 3.31 and Table 3.32 show the results using LST-Net on MPII dataset and COCO key-points detection dataset, respectively. LST-Net outperforms mean AP of ResNet-50 by 0.52% on MPII dataset. Meanwhile, on COCO key-points detection dataset, LST-Net significantly improves mean AP of ResNet-50 by 3.7% and 4.3% when input size is 256×192 and 384×288, respectively. In addition, our method also has considerable gain of mean AR by 0.7% and 1.3% on COCO key-points detection dataset when input size is 256×192 and 384×288, respectively.

Table 3.33: Salient object detection results of LST-Net.

Dataset	Backbone	F-measure↑	MAE↓
ECSSD [188]	ResNet-50 [59]	0.940	0.042
	Res2Net [113]	0.947	0.036
	LST-Net	**0.950**	**0.034**
PASCAL-S [104]	ResNet-50 [59]	0.863	0.075
	Res2Net [113]	0.871	0.070
	LST-Net	**0.876**	**0.066**
DUT-OMRON [189]	ResNet-50 [59]	0.830	0.055
	Res2Net [113]	**0.837**	**0.052**
	LST-Net	0.834	**0.052**
HKU-IS [92]	ResNet-50 [59]	0.934	0.032
	Res2Net [113]	0.936	0.031
	LST-Net	**0.941**	**0.028**
SOD [130]	ResNet-50 [59]	0.867	0.100
	Res2Net [113]	**0.885**	0.096
	LST-Net	0.875	**0.093**
DUTS [166]	ResNet-50 [59]	0.886	0.040
	Res2Net [113]	0.892	0.037
	LST-Net	**0.895**	**0.035**

3.4.12 Evaluation on salient object detection

To evaluate LST-Net on salient object detection, we replace the ResNet-50 backbone of PoolNet [113] with our LST-Net w.r.t. the original backbone architecture and keep the remaining network architecture unchanged. In addition, we also compare our LST-Net to state-of-the-art salient object detection backbone Res2Net [113] by using its official implementation [1]. No edge information is leveraged during training.

Table 3.33 presents the results of LST-Net on salient object detection. Our LST-Net consistently achieves better salient object detection results than the vanilla

[1]https://github.com/Res2Net/Res2Net-PoolNet

ResNet-50 in terms of both F-measure and MAE across all the benchmark datasets. Compared to Res2Net, the proposed method obtains comparable results on DUT-OMRON and SOD. Meanwhile, it slightly outperforms the state-of-the-art saliency detection backbone architecture under both metrics across the remaining four benchmarks.

3.5 Conclusion

In this chapter, we propose to train deep CNNs with a learnable sparse transform (LST), which learns to convert the input features into a more compact and sparser domain together with the CNN training process. LST can more effectively reduce the spatial and channel-wise feature redundancies than the conventional Conv2d. It can be efficiently implemented with existing CNN modules, and is portable to existing CNN architectures for seamless training and inference. We further present a hybrid ST-ReLU activation to enhance the robustness of the learned CNN models to common types of corruptions in the input. Extensive experiments validate that the proposed LST-Net achieves even higher accuracy than its counterpart networks of the same family with lower cost.

In the next two chapters, we will discuss the bottleneck of our LST and improve its implementation with the knowledge in traditional signal processing.

Chapter 4

Learning a Fast and Lightweight Transform with Hierarchical Depth-wise Separable Convolution

The conventional 2D convolutional layer (Conv2d) produces redundant features. In Chapter 3, we introduced LST-Net to reduce the redundancies of Conv2d by learning a learnable sparse transform. In the spatial transform step of LST-Net, all channels are expanded several times by using a depth-wise separable convolution (DWConv) kernel to allow reweighting in the associated resize transform step. However, using a single DWConv kernel is not able to effectively model the input features at different frequencies. Besides, the expansion of high frequency features will result in unnecessary channels, which cause unnecessary parameters and computational cost in the resize transform. In this chapter, we propose to mitigate the above issues with the knowledge in traditional signal processing. We develop a fast and lightweight transform with hierarchical DWConv (HDWConv) to build a CNN, namely LST-Net v2. HDWConv allows flexible expansion of the input features based on their frequency. Low frequency features are decomposed into more bands, while high frequency features are decomposed into fewer bands. In this way, we can effectively discard redundant features along channel dimension to relieve the resize transform. In addition, HDWConv partitions the input features into groups by the similarity

of frequency. Each group is assigned one or more unique DWConv kernels to better model the features at different frequencies. Extensive experiments demonstrated that the proposed LST-Net v2 can achieve comparable or even higher accuracy than LST-Net while saving 20% $\sim$ 40% parameters and computational cost.

4.1 Introduction

The 2d convolutional layer (Conv2d) plays a key role in deep convolutional neural networks (CNNs) to extract features for a wide range of computer vision tasks, *e.g.*, image recognition [35, 209], image retrieval [138, 139], image restoration [86, 202], *etc*. Despite its great success, Conv2d has been found redundant in extracting image features [80, 55, 181, 95, 94, 97]. How to design fast and lightweight alternatives of Conv2d remains a heated topic in computer vision and machine learning research.

To reduce the redundancies of Conv2d, existing works can be roughly divided into two categories. Some works train a CNN model built with Conv2d to identify and discard weights of less importance. Representative works in this category can be found in [95, 61, 76, 60, 116]. Though these methods can produce lightweight models for faster inference, it is rather painful to train the whole network from scratch. What's worse, re-training is usually needed to improve the pruned architecture. Other works [68, 149, 119, 67, 91, 53] design Conv2d alternatives by using fast primitive operators such as point-wise convolutional layers (PWConv) [105], depth-wise separable convolutional layers (DWConv) [68], channel shuffle [119], *etc*. In practice of network architecture design, these primitives are widely used as undivided routines. It remains an open question whether some of them, *e.g.*, DWConv, can be further accelerated.

In this chapter, we make an effort along this line and develop a fast and lightweight transform as an alternative of Conv2d. Our work is mainly inspired by the recently

developed LST-Net [97], which learns a set of channel and spatial transforms to convert the given CNN features into a more compact and sparser latent space. The transforms are initialized by discrete cosine transform (DCT) for adaptive decomposition, and a soft thresholding (ST) is applied to remove noises and trivial features in the learned domain. Though LST-Net is effective and efficient, it can still be improved from two aspects. First, in the spatial transform step of LST-Net, all channels are expanded several times by using a DWConv kernel to allow reweighting in the associated resize transform step. However, it is ineffective to model the input features at different frequencies in this way. Second, high frequency features are already sparse enough so that their expansion will easily result in trivial channels, which cause unnecessary parameters and computational cost in the resize transform.

We dedicatedly design a hierarchical depth-wise separable convolutional layer (HDWConv) to mitigate the above issues of LST-Net. To reduce redundant features, HDWConv allows flexible expansion whereas the expansion rate is fixed in DWConv. High frequency features are expanded more times for near-complete expansion, while low frequency features are expanded fewer times to save cost. Notably, based on the prior knowledge of DCT and inspired by the visualization results of LST-Net, the structure of HDWConv can be directly determined before training. This helps to avoid training the whole CNN model. Besides, we partition the input features of HDWConv into a few groups along the channel dimension based on the similarity of frequency. Each feature group is assigned one or more DWConv kernels with identical DCT initialization for better and flexible modeling. Compared to entire bottleneck, the extra parameters introduced to each group are negligible. By replacing DWConv with HDWConv in the spatial transform step, we develop LST v2 bottleneck to construct CNNs, namely LST-Net v2. Our extensive experiments on representative benchmarks show that the proposed LST-Net v2 can obtain comparable or even higher accuracy than LST-Net while saving around 20% $\sim$ 40% parameters and

overhead.

4.2 Review and analysis of LST-Net

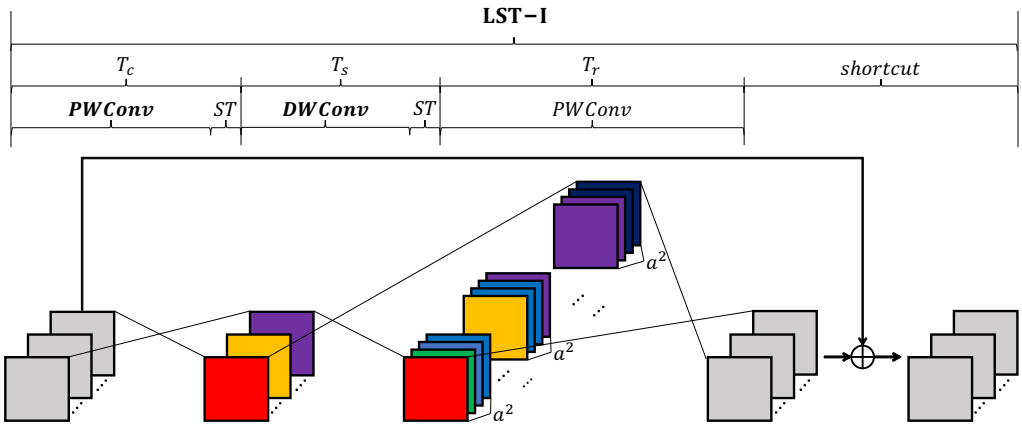
4.2.1 Review of LST-Net

Our work is inspired by LST-Net [97], which is briefly reviewed here.

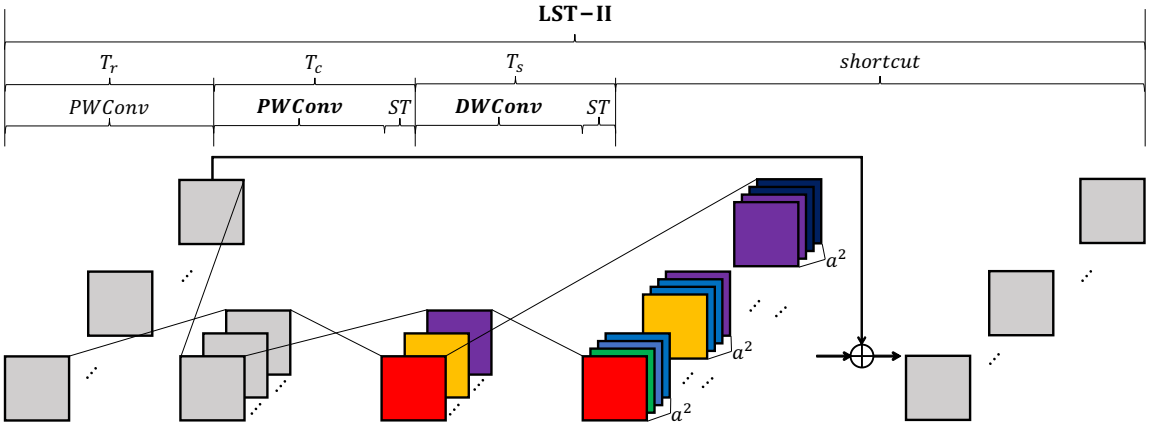
Figure 4.1 illustrates two types of LST bottlenecks used to build LST-Net, *i.e.*, LST-I and LST-II. An LST bottleneck consists of three primitive transforms, including channel transform T_c, spatial transform T_s, and resize transform T_r. The weights of T_c or T_s are initialized as DCT so that the transform input can be decomposed along the channel or spatial dimension at different frequencies. Both T_c and T_s can be implemented by using existing routines to facilitate end-to-end learning, including PWConv and DWConv. Both outputs are arranged along the channel dimension for removal of noise and trivial feature by applying ST. In either type of LST bottleneck, T_s is arranged right after T_c to save parameters and computational cost [98]. In T_r, a PWConv is randomly initialized to learn to reweight all channels for T_c and T_s, where the desired output size can be obtained. Notably, both types of LST bottlenecks are the same except for the location of T_r.

4.2.2 Complexity analysis

To figure out the main constraints of an LST bottleneck, we study the complexity of its three primitive transforms. The computational complexity of the channel transform T_c, spatial transform T_s and resize transform T_r of an LST bottleneck is $\mathcal{O}(HWC^2)$, $\mathcal{O}(HWs^2a^2C)$ and $\mathcal{O}(HWa^2C^2)$, respectively. where C is the input/output channels, H/W is the width/height of the input or output, s is the kernel size, and $a \in Z^+$ is the channel multiplier along the height or the width dimension in T_s. To be clear, the input and output of an LST bottleneck is assumed



(a) LST-I



(b) LST-II

Figure 4.1: Illustration of two types of LST bottlenecks [97]. PWConv/DWConv in bold font indicates initialization as DCT while normal font suggests random initialization. The input and the output channels of the spatial transform T_s are highlighted in prism colorset, where red means low frequency signals while purple means high frequency signals. T_s lies at the core part of either LST bottleneck, closely following the channel transform T_c in either bottleneck. The output of T_s determines the maximum number of channels of an LST bottleneck, where each input channel, regardless of the frequency, is expanded by a fixed number of a^2 times due to the use of DWConv. Meanwhile, the output of T_s also settles the cost of the resize transform T_r, which dominates the entire bottleneck.

to be identical in shape so that subscripts can be omitted where necessary. Since $1 \leqslant a < s << C$ always holds in modern architectures, *e.g.*, AlexNet [90], VGG [151], ResNet [59], *etc.*, the overhead of an LST bottleneck is dominated by the resize transform T_r at $\mathcal{O}(HWs^2a^2C)$. Meanwhile, the number of parameters of T_c, T_s and T_r are C^2, a^2s^2 and a^2C^2, respectively. Similarly, the number of parameters of an entire LST bottleneck is also determined by T_r at a^2C^2.

Judging from the decisive part of an LST bottleneck, both the computational complexity $\mathcal{O}(HWs^2a^2C)$ and number of parameters a^2C^2 of T_r are on a basis of a^2, as C is fixed in terms of a bottleneck. The basis of a^2 actually comes from the spatial transform T_s, where each input channel of T_s is expanded a^2 times via a single DWConv kernel, regardless of its exact frequency in the output feature bank of T_c. Thus, we focus on the spatial transform T_s and its use of DWConv to improve an LST bottleneck.

4.2.3 Motivation

From the above analysis, we know that the spatial transform T_s lies at the core of an LST bottleneck. We discuss the shortcomings of T_s from two aspects.

In terms of the parameters and overhead, there still exists a large number of trivial features in the output of T_s because high frequency input features are decomposed into the same number of a^2 frequency bands as their low frequency counterparts. Actually, many high frequency features are already sparse enough, especially in a wider LST bottleneck (*i.e.*, C is larger). It is almost in vain to decompose a sparse high frequency feature into many bands. What's worse, it even increases the burden of the resize transform T_r.

In addition, a single DWConv kernel in T_s is not qualified to simultaneously extract features at different frequencies. High frequency features suggest edge cues while low frequency features reflect smooth region. Therefore, a single kernel should

be only allowed to handle a few features similar in frequency to better model the features of T_s. Meanwhile, it is also desired that a number of DWConv kernels can be learned together to cope with a large number of features at different frequencies.

4.3 Proposed method

In Section 4.3.1, we first present an overview of our proposed HDWConv. Then, we formulate our method in Section 4.3.2. Finally, we discuss some implementation details in Section 4.3.3.

4.3.1 Overview

We design a fast, lightweight yet more powerful substitution of DWConv in the spatial transform T_s of an LST bottleneck, namely Hierarchical DWConv (HDWConv), to mitigate the constraints of the associate DWConv. Below, we brief each constraint and its corresponding solution.

Compactness. To produce a compact feature bank for T_s, it is equivalent to identify and remove those high frequency output features. We have two candidate solutions: (1) drop of last (short for DoL), where we skip expanding the last several high-frequency channels of the input; and (2) flexible expansion (short for F.E.), where all input channels are leveraged for expansion but some are expanded fewer times.

Our HDWConv is designed based on F.E. as it has two merits. First, F.E. preserves all features from the channel transform T_c. In contrast, DoL quadratically discards the input features at the end to achieve the same cost, running a risk of losing critical information before T_s. Second, F.E. allows more flexible expansion in T_s while the expansion rate is discontinuous in DoL due to the use of DWConv. In Section 4.4.2, experimental results will be provided to further validate the advantage of F.E.

Effectiveness. To match features at different frequencies, HDWConv divides input features into several disjoint groups by similarity of their frequency. Therefore, the size of a feature group will not be too large. Unlike one kernel per channel, with the introduction of feature group, we can save parameters. Meanwhile, a feature group allows a small amount of frequency distortions of the input so that training will become more stable. By default, we depict a^2 non-empty feature groups in this chapter so that each group is expanded a unique number of times ranging from 1 to a^2.

Unlike the use of a single DWConv kernel in [97], we assign each feature group one or more separate DWConv kernels to strengthen the power of spatial transform. Even though this requires some additional weights, they are almost negligible when compared to those of the resize transform T_r (see Section 4.2.2). Meanwhile, the overhead and space complexity remain the same after the change.

The topology of HDWConv is deterministic. With the prior knowledge of DCT, we are aware of the general distribution of features at different frequencies, and HDWConv can be accordingly specified even without training. It not only helps to identify and remove high frequency features, but also defines how to partition all feature groups. In this chapter, high frequency input features of T_s are assumed to come after low frequency ones based on two facts. First, DCT initialization of the preceding channel transform T_c arranges the input of T_s in the exact order before training starts. Second, even if there exist some exceptions during training, the expected order remains basically unchanged from the visualization results of an LST-Net [97]. With the assumption, one can directly index the input as contiguous memory along its channel dimension to define the boundaries of all feature groups, which is almost free.

4.3.2 Formulation

Denote the input and the output of HDWConv by $x_s \in \mathcal{R}^{C_i \times H \times W}$ and $y_s \in \mathcal{R}^{C_o \times H \times W}$, respectively, where C_i and C_o are the input and output number of channels. Like DWConv, we always have $C_i \leqslant C_o$ for HDWConv, where the equation only holds when $a = 1$. Notably, we expect that the expansion rate of HDWConv $\epsilon = C_o/C_i \leqslant a^2$ as HDWConv is designed to save the output channels of DWConv, where $\epsilon = a^2$ *iff* $a = 1$.

HDWConv-A.To prune unnecessary high frequency features, we develop the initial version of HDWConv, called HDWConv-A. We later improve the modelling of features in its successors. Figure 4.2(a) and Figure 4.2(b) illustrate the structure of DWConv and HDWConv-A, respectively. Judging from the overall structure, our HDWConv-A uses exactly the same kernel $\theta_A \in \mathcal{R}^{a^2 \times s \times s}$ as the one in DWConv before training. However, HDWConv-A flexibly expands each input channel a number of times from 1 to a^2 according to the channel frequency, *i.e.*, the relative position of the entire input. Given a low frequency channel with high amplitude, it is expanded more times (*i.e.*, close or equivalent to a^2) for near-complete decomposition. In contrast, a high frequency channel with low amplitude is expanded fewer times (*i.e.*, close or equivalent to once) to save cost.

To meet the input and output number of channels, HDWConv-A partitions its input along the channel dimension into a^2 feature groups for flexible expansion as

$$\mathcal{G} = \{G_j\}|_{j=1}^{a^2} = \{(\rho_{s,j}, C_{s,j}, \epsilon_{s,j})\}|_{j=1}^{a^2} \tag{4.1}$$

where G_j is the j-th group, and $\rho_{s,j}$, $C_{s,j}$ and $\epsilon_{s,j}$ are the position of the first feature along the channel dimension, feature group size and expansion rate of G_j, respectively. Let $P = [\rho_{s,1}, \ldots, \rho_{s,a^2}]^T$, $C_s = [C_{s,1}, \ldots, C_{s,a^2}]^T$ and $E = [\epsilon_{s,1}, \ldots, \epsilon_{s,a^2}]^T$. C_i can be written as

$$C_i = \|C_s\|_1, \tag{4.2}$$

and C_o can be computed as

$$C_o = C_s^T E. \tag{4.3}$$

We set $\epsilon_{s,j} = a^2 - j + 1$ $(j = 1, \ldots, a^2)$ to match $\epsilon_{s,j}$ with the general frequencies of $\mathcal{G}_j$. Besides, we conduct DWConv for all input channels of G_j by using the first $\epsilon_{s,j}$ channels of θ_A w.r.t. the formulation of DCT [174].

HDWConv-B. In Figure 4.2(b), the first few unrolled parts of the single DWConv kernel of HDWConv-A are convolved with more input features than the rest parts for expansion. The first kernel part in red is applied to all feature groups, while the last kernel part in blue is only used in G_1. For better modeling, one should apply every part of a DWConv kernel to a limited number of feature groups. To mitigate this issue, we develop HDWConv-B by assigning one unique DWConv kernel to each feature group when $a > 1$. Denote $\theta_B \in \mathcal{R}^{\sum_{j=1}^{a^2}(a^2-j+1)\times s\times s}$ the HDWConv-B kernel. $\theta_{s,j}(j = 1, \ldots, a^2)$ has $\epsilon_{s,j} = a^2 - j + 1$ channels. When $a = 1$, HDWConv-B is the same as HDWConv-A.

Figure 4.2(c) illustrates the structure HDWConv-B. One can see that the output of HDWConv-B is of the same size as that of HDWConv-A, saving the same amount of output features of a corresponding DWConv used in [97]. Each $\theta_{s,j} \in \mathcal{R}^{(a^2-j+1)\times s\times s}(j = 1, \ldots, a^2)$ is initialized as 2D-DCT [174] so that both HDWConv-A and HDWConv-B are created the same for any x_s at the start of training.

HDWConv-C. We propose the last version of HDWConv, HDWConv-C, to further enhance the representation power of HDWConv-B in case of oversized groups. When there are too many channels in one feature group, similar to HDWConv-A, it remains difficult for a single DWConv kernel to model the entire group well.

To avoid larger feature group, we equally partition G_j $(j = 1, \ldots, a^2)$ into $r_j = \lfloor C_{s,j}/C_{s,z} \rfloor$ subgroups along the channel dimension by similarity of frequency, where $C_{s,z}$ denotes the smallest size of all feature groups when $\epsilon > 1$. Other-

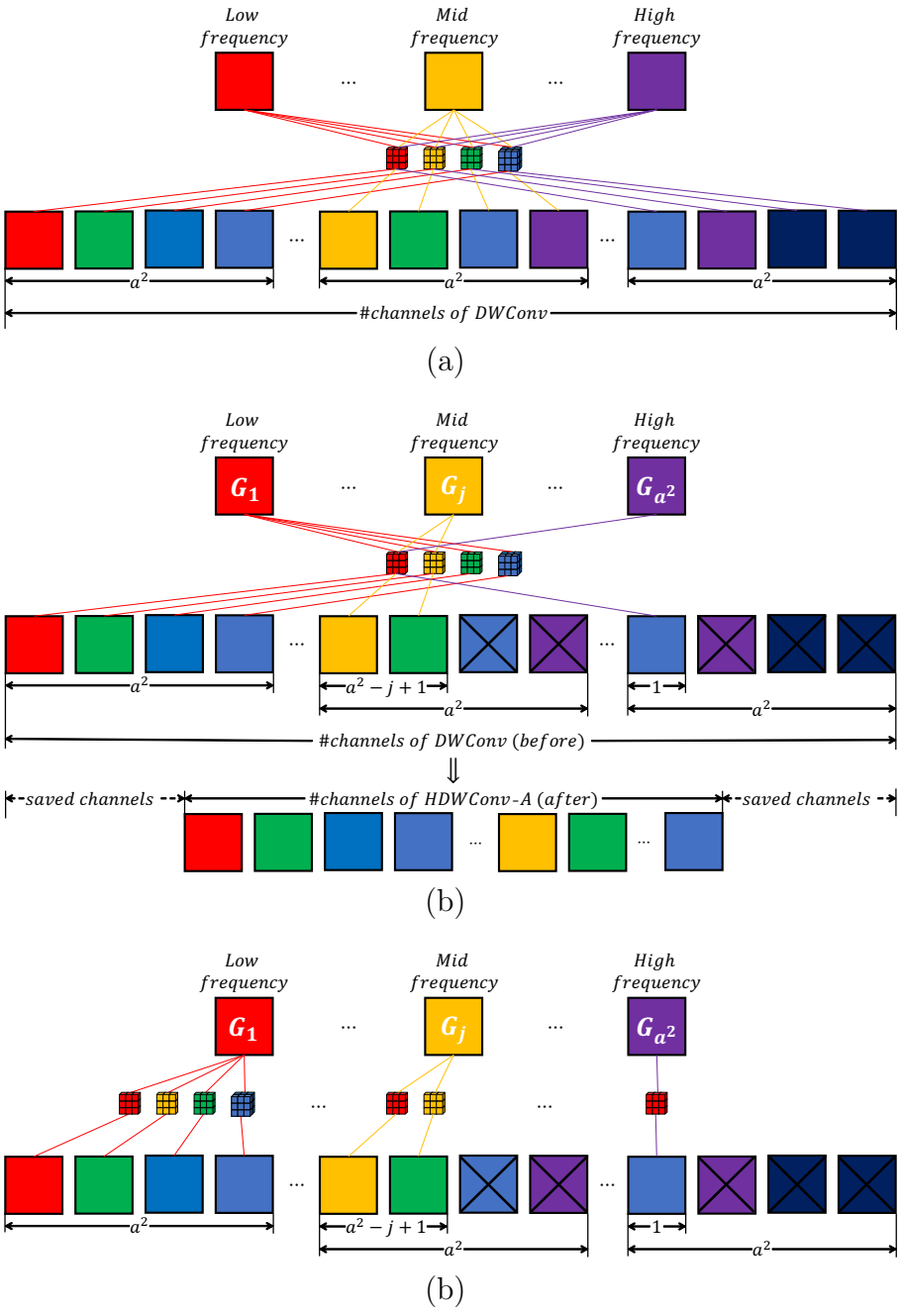
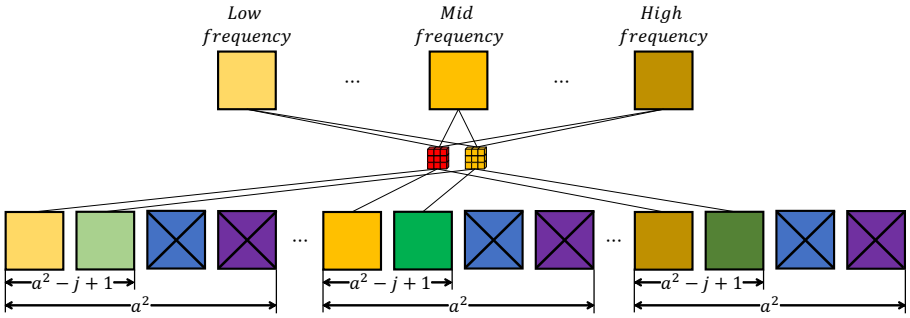
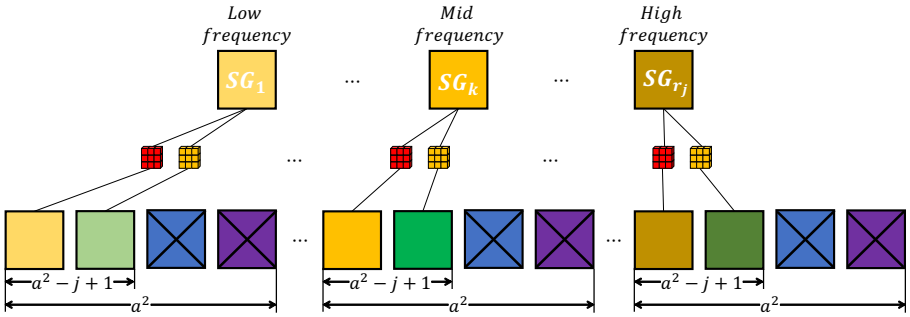


Figure 4.2: Illustration of (a) DWConv, (b) HDWConv-A and (c) HDWConv-B. Each DWConv kernel is unrolled along its channel dimension. In HDWConv-A/B, we stamp a black cross mark on each saved output channel of DWConv. HDWConv-B produces the same number of output features for each input channel of HDWConv-A. However, HDWConv-B is better at modelling the input as it assigns each of its feature group a separate DWConv kernel.



(a) HDWConv-B with a shared DWConv kernel



(b) HDWConv-C with r_j separate DWConv kernels in all feature subgroups

Figure 4.3: Illustration of the j-th ($j = 1, \ldots, a^2$) feature group. Each DWConv kernel is unrolled along its channel dimension. We stamp a black cross mark on every saved output channels of related DWConv.

wise, $C_{s,z} \in (1, C_i)$ is a predefined hyper-parameter. In addition, we assign a separate DWConv kernel $\theta_{s,j,k} \in \mathcal{R}^{(a^2-j+1)\times s\times s}$ to the k-th subgroup ($k = 1, \ldots, r_j$), which is also initialized as 2D-DCT [174]. We have the kernel of HDWConv-C $\theta_C \in \mathcal{R}^{\sum_{j=1}^{a^2}\sum^{r_j}(a^2-j+1)\times s\times s}$.

In HDWConv-C, the j-th group $G_j (j = 1, \ldots, a^2)$ can be written as

$$G_j = \{SG_{j,k}\}|_{k=1}^{r_j} = \{(\rho_{s,j,k}, C_{s,j,k}, \epsilon_{s,j})\}|_{k=1}^{r_j}, \tag{4.4}$$

where $SG_{j,k}$ denotes the k-th subgroup of G_j, and $\rho_{s,j,k}$ and $C_{s,j,k}$ are the position of the first channel and the subgroup size, respectively. We always have $\rho_{s,j} = \rho_{s,j,1}$ and $C_{s,j} = \sum_{k=1}^{r_j} C_{s,j,k}$.

Figure 4.3(a) and Figure 4.3(b) illustrate $G_j(j = 1, \ldots, a^2)$ in HDWConv-B and HDWConv-C, respectively. It can be observed that both versions produce features identical in shape, and they also have the same overhead. As all kernels are initialized the same, all three versions of HDWConv are created the same for x_s at the first iteration of training. The only difference lies in that HDWConv-B uses a shared DWConv kernel for all input channels of the group while HDWConv-C leverages r_j separate DWConv kernels, a very small number independent of $C_{s,j}$. As $r_j << C_{s,j}$, the extra number of parameters introduced in HDWConv-C can be omitted in practice.

Figure 4.4 compares the structure of DWConv and all three versions of the proposed HDWConv. Given the same input x_s, the output of DWConv (with some dark blue rectangles) has more channels than that of any version of our HDWConv (without any dark blue rectangles). All versions of HDWConv have their output y_s same in size but they use different number of DWConv kernels (little cubics in the middle). Specifically, there is only one DWConv kernel in HDWConv-A. In contrast, there are group and subgroup number of separate DWConv kernels in HDWConv-B and HDWConv-C, respectively.

Discussion. By properly applying the proposed HDWConv to an LST bottleneck, one can have an improved version, namely LST v2. With such modification, the computational complexities of the channel transform T_c, spatial transform T_s and resize transform T_r in an LST v2 bottleneck is $\mathcal{O}(HWC^2)$, $\mathcal{O}(HWs^2\epsilon C)$ and $\mathcal{O}(HW\epsilon C^2)$, respectively. Compared with their counterparts in an LST bottleneck, the cost of T_c remains unchanged, while the other two transforms cost only ϵ/a^2 the overhead of their original version. For example, with the default setting of LST (*i.e.*, $a = 2$), T_s and T_r of LST v2 reduce 50% and 25% the overhead for $\epsilon = 2$ and $\epsilon = 3$, respectively.

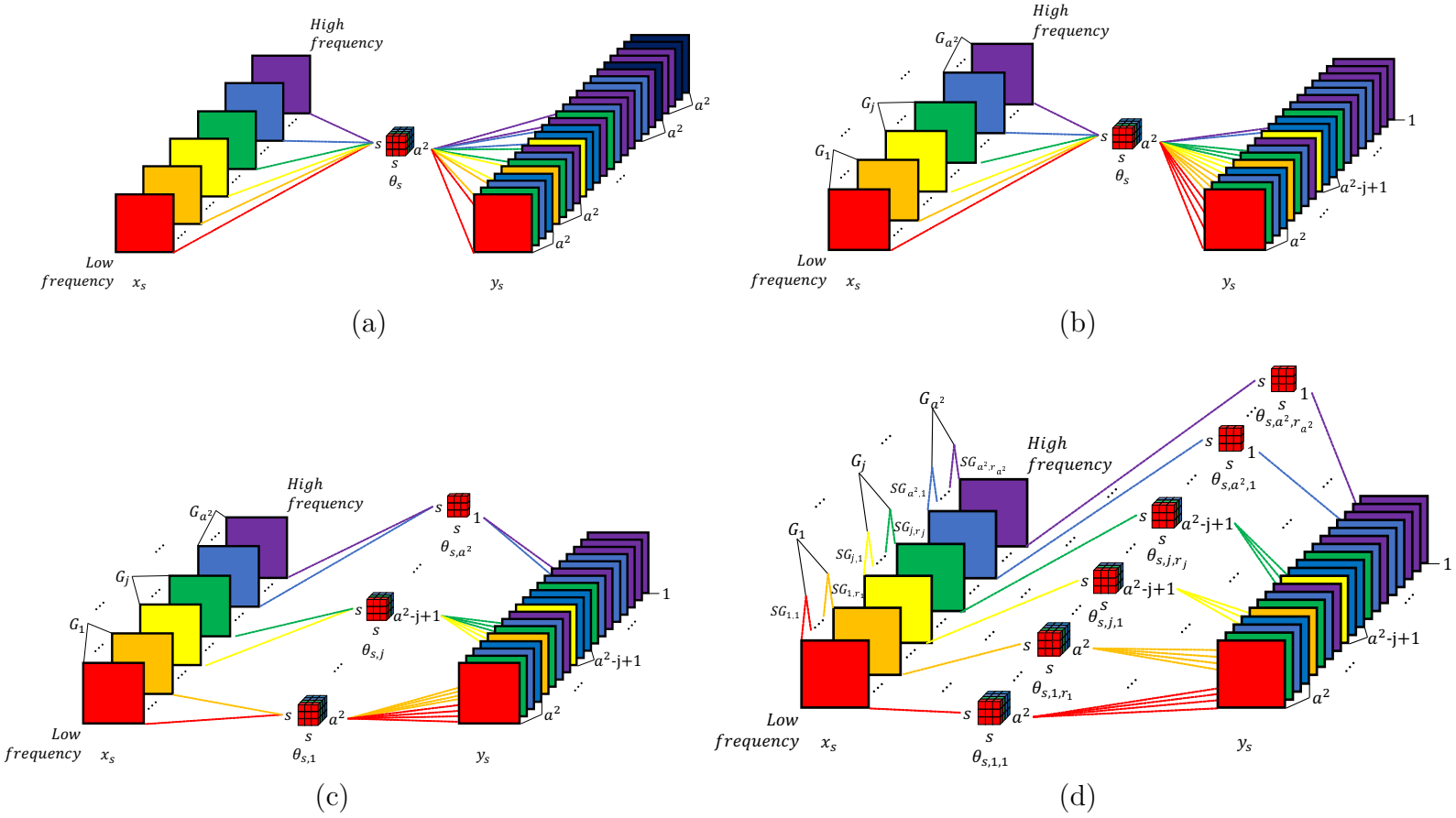


Figure 4.4: Structural comparison of (a) DWConv, (b) HDWConv-A, (c) HDWConv-B and (d) HDWConv-C when $\epsilon > 1$. The input and the output channels are highlighted in prism colorset, where red means low frequency signals while purple means high frequency signals.

4.3.3 Implementation

Given C_i and C_o, it is important to automatically configure the structure of a convolutional layer. So is HDWConv. Below we discuss how to compute $\mathcal{G}$ when $\epsilon > 1$.

As $\epsilon < a^2$, we at first compute a by solving $(a-1)^2 \leqslant C_o/C_i < a^2$ to make sure that there exists at least one solution of $\mathcal{G}$, simultaneously satisfying both Equation 4.2 and Equation 4.3. In this way, both a^2 and E are obtained.

Algorithm 1: Automatic configuration of HDWConv when $a > 1$.

Input: C_i, C_o;
Initialization: $a^2, d_c, d_r, \epsilon, \hat{\epsilon}, s_c, s_r, E, \hat{R} = init(C_i, C_o)$;
if $\hat{\epsilon} \neq \epsilon$ **then**
  if $d_r | d_c$ **then**
    $\hat{R}[-(d_c/d_r)] = \hat{R}[-(d_c/d_r)] + d_r$;
    break;
  else
    $\hat{R} = update\_R(d_c, d_r, a^2, \hat{R})$;
  end
end
$\mathcal{G} = format\_groups(C_i, C_o, E, \hat{R})$;
return $\mathcal{G}$;

Algorithm 2: *init*.

Input: C_i, C_o;
Initialization: $d = gcd(C_i, C_o), \epsilon = C_i/C_o$;
$a^2 = (\lfloor\sqrt{\epsilon}\rfloor + 1)^2$;
$\hat{R}, E = \mathbf{1}^{a^2 \times 1}, [a^2, a^2 - 1, \ldots, 2, 1]^T$;
$s_c, s_r = \hat{R}^T E, \|\hat{R}\|_1$;
$\hat{\epsilon} = s_c/s_r$;
$m, n = C_o/d, C_i/d$;
$t = \lceil \frac{s_c - a^2 \cdot s_r}{m - a^2 \cdot n} \rceil$;
$d_c, d_r = m \cdot t - s_c, n \cdot t - s_r$;
return $a^2, d_c, d_r, \epsilon, \hat{\epsilon}, s_c, s_r, E, \hat{R}$;

Then, we leverage both Equation 4.2 and Equation 4.3 as our criteria and conduct a greedy search for C_s and stop at the first time when the target ϵ is satisfied. Instead

of a direct search, we develop a two-step method detailed in Algorithm 1. At the first step, we explore the relative proportion of each group in C_i, *i.e.*, $R = [r_1, \ldots, r_{a^2}]^T$, by initializing $R = \hat{R} = \mathbf{1}$ and adjusting one element of $\hat{R}$ at a step of 1 until $\hat{R}^T E / \|\hat{R}\|_1 = \epsilon$. Specifically, we aim to reach $\epsilon = C_o/C_i$ from $\hat{\epsilon} = s_c/s_r = \hat{R}^T E/\|\hat{R}\|_1$ at initialization. To facilitate search, we rewrite our goal as

$$\epsilon = \frac{C_o}{C_i} = \frac{m \cdot d}{n \cdot d} = \frac{m \cdot t}{n \cdot t}, \tag{4.5}$$

where d is the greatest common divisor (gcd) of C_i and C_o, and $t \in \mathcal{R}^+$ is the corresponding multiplier when our search finishes, *i.e.*, $\hat{\epsilon} = \epsilon$. As all elements of $\hat{R}$ are considered as positive integers, we introduce two non-negative integers $d_c = m \cdot t - s_c$ and $d_r = n \cdot t - s_r$ to measure the discrepancies at initialization. According to the definition of E, one should be aware that all values of E range from 1 to a^2.In this sense, it is true that

$$d_c \leqslant d_r \leqslant a^2 \cdot d_c. \tag{4.6}$$

By combining Eq. 4.5 and Eq. 4.6, we know that

$$\frac{s_c - a^2 \cdot s_r}{m - a^2 \cdot n} \leqslant t \leqslant \frac{s_c - s_r}{m - n}. \tag{4.7}$$

Algorithm 2 presents our initialization method. One can see that d_c and d_r are initialized by setting t to its lower bound.

Once $\hat{R}$ is determined, at the second step, we initialize $C_s = \hat{C}_s = \lfloor C_i/\|R\|_1 \rfloor \cdot R$. Due to the use of floor function, there may be some remaining input features. To address this problem, we first assign all remainders to the group whose expansion rate is the closest to ϵ, and we move its elements one by one to other feature groups where necessary until Equation 4.2 is satisfied. Algorithm 3 demonstrates the approach in Python style, where a zero-based strategy is adopted for indexing and a negative value

Algorithm 3: *update_R.*

Input: d_c, d_r, a^2, R;
Initialization: $idx = -min(d_c, a^2)$;
while $d_r > 0$ **do**
  if $d_c == -idx \times d_r$ **then**
    $R[idx] = R[idx] + d_r$;
    $d_c = d_r = 0$;
  else
    if $d_c + idx + 1 \geqslant d_r$ **then**
      $R[idx] = R[idx] + 1$;
      $d_r = d_r - 1$;
      $d_c = d_c + idx$;
      if $-d_c \geqslant d_r \times (idx + 1)$ **then**
        $idx = idx + 1$;
      end
    else
      $idx = idx + 1$;
    end
  end
end
return R;

suggests reverse indexing. When $\hat{C}_s$ is obtained, R is accordingly solved. Meanwhile, the structure of HDWConv-A or HDWConv-B is also specified.

After R is determined, one can equally partition each feature group of HDWConv-C into relevant number of subgroups. In terms of indivisible cases, as input features at last of a group are less important than the rest, they are put into the subgroup with the highest frequency bands and the lowest amplitude as the remainders in our implementation. To this end, values of any $\rho_{s,j,k}$ and $C_{s,j,k}$ are confirmed. Mathematically, the size $C_{s,j,k}$ can be computed as

$$C_{s,j,k} = \begin{cases} \lfloor \frac{C_{s,j}}{r_j} \rfloor, & 1 \leqslant k < r_j, \\ C_{s,j} - \sum^{r_j - 1} \lfloor \frac{C_{s,j}}{r_j} \rfloor, & k = r_j. \end{cases} \tag{4.8}$$

Table 4.1 lists some typical examples of HDWConv-C, where "A", "R" and "V" stand for AlexNet, ResNet and VGG, respectively.

Table 4.1: Examples of HDWConv-C.

Kernel size	Exp. rate ϵ ($R^T E/\|R\|_1$)	Arch.	#Groups (a^2)	#Subgroups ($\|R\|_1$)	R^T
3×3	2 (20/10)	A/R/V	4	10	$[1, 1, 5, 3]$
3×3	3 (30/10)	A/R/V	4	10	$[3, 5, 1, 1]$
5×5	6 (72/12)	A	9	12	$[4, 1, 1, 1,$ $1, 1, 1, 1, 1]$

4.4 Experiments

In this section, we first brief our experiment setup and datasets. Then, we conduct ablation study of the two candidate pruning methods and different versions of HDWConv. Finally, we evaluate the performance of LST-Net v2 on representative benchmarks.

4.4.1 Experiment setup and datasets

All experiments are conducted on a server equipped with Intel Xeon Gold 6248R CPUs and NVIDIA Quadro RTX 8000 GPU cards. We use PyTorch [133] for implementation.

Methods for comparison. LST-Net [97] is selected as our baseline, to which we report the relative change of other methods in terms of number of parameters and overhead. We build two LST-Net v2 under each architecture w.r.t. the baseline by default except stated otherwise. Specifically, we respectively set $\epsilon = 2$ and $\epsilon = 3$ for each transform in case $s = 3$ and accordingly set $\epsilon = 2$[1] and $\epsilon = 6$ when $s = 5$ (see Table 4.1). We also report the original results of CNNs built with the conventional Conv2d. For better presentation, we compare methods w.r.t. their core building

[1]In case $\epsilon = 2$ and $s = 5$, we keep $a = 3$ to fully match LST-Net under AlexNet and accordingly set $R = [1, 1, 1, 1, 1, 1, 1, 1, 28]^T$.

Table 4.2: Comparison of two candidate pruning methods for the spatial transform T_s.

Dataset	Method	#Param (M)	FLOP	Top-1 E. R. (%)
CIFAR-10	DoL	**0.347**	**55.149**M	6.27
	F.E.	**0.347**	**55.149**M	**6.06**
ImageNet	DoL	**13.844**	**2.528**G	24.01
	F.E.	13.846	2.529G	**23.09**

blocks.

Datasets. We use exactly the same datasets and default experimental settings detailed in [97] for fair comparison. Specifically, CIFAR-10/100 [89] and the ImageNet [35] are used to compare the performance of each method on natural images. The ImageNet-C [63] is employed to evaluate the robustness of each method on images with common corruptions. We study the performance of each method on the Places365-Standard [209] for large-scale scene recognition. We use FGVC-Aircraft dataset [123], Birds-CUB200-2011 dataset [175], FGVC-Cars [88] and Stanford Dogs [85] dataset for fine-grained visual recognition. Describing Textures Dataset (DTD) [30] and Indoor67 dataset [137] are employed for texture classification. We use MS-COCO dataset [107] to study the performance on object detection, instance segmentation and human pose estimation. PASCAL VOC dataset [41] is used to study the performance of each method on semantic segmentation. We conduct experiments on MPII dataset [2] for human pose estimation. DUTS dataset [166], ECSSD [188], PASCAL-S dataset [104], DUT-OMRON dataset [189], HKU-IS dataset [92] and SOD [130] are employed for salient object detection.

4.4.2 Ablation study

Candidate pruning methods. We first investigate the two candidate pruning methods for the spatial transform T_s discussed in Section 4.3.1, including DoL and F.E. For fair comparison, both methods are required to drop half of the output features of T_s in their own way. We implement DoL based on LST [97] by reducing the output channels of PWConv in the channel transform T_c to half of the original size at each layer. For F.E., we adopt HDWConv-A so that both methods leverage an identical shared kernel in T_s at initialization. We build two CNNs under ResNet-56 on CIFAR-10 [89] and ResNet-50 on ImageNet [35], respectively. We use top-1 error rates as our criteria.

Table 4.2 summarizes the results. One can see that DoL uses slightly fewer parameters and less overhead than F.E, which is almost negligible. However, F.E. significantly reduces the top-1 error rates of DoL on both datasets. The gap is 0.21% and 0.92% on CIFAR-10 and ImageNet, respectively. It can be inferred that the last input features of the spatial transform T_s are more critical to those spatially decomposed ones generated from the middle input features. As some valuable cues are missing in the spatial transform T_s of DoL, it eventually harms the performance.

Different versions of HDWConv. We build LST-Net v2 under the architecture of ResNet-18 [59] and AlexNet-GAP [90] on ImageNet to compare different versions of HDWConv when $\epsilon > 1$. In addition, we also build CNNs under the architecture of MobileNet v2 [149] (short for MNet v2) to examine the choice of number of groups when $\epsilon = 1$.

Table 4.3 presents the results. One can have at least three findings. First, given the same ϵ under the same architecture, one can see that HDWConv-C>HDWConv-B>HDWConv-A in terms of performance when $a > 1$. When $\epsilon = 2$, the top-1/5 error rates of LST-Net v2 under AlexNet-GAP with HDWConv-A can be reduced by

Table 4.3: Comparison of different versions of HDWConv on ImageNet.

Arch.	Ver.	#Param (M)	GFLOP	ϵ	#Groups	#Sub-groups	Top-1/5 E. R. (%)
ResNet [59]	A	4.98	0.97	2	1	N.A.	27.33/9.01
	B				4	N.A.	27.07/8.90
	C				4	10	**27.02/8.87**
	A	6.51	1.23	3	1	N.A.	26.83/8.84
	B				4	N.A.	26.62/8.64
	C				4	10	**26.61/8.53**
AlexNet [90]	A	1.42	0.35	2	1	N.A.	42.20/19.50
	B				4	N.A.	41.99/19.40
	C				4	10	**41.59/19.33**
	A	1.84	0.49	3	1	N.A.	40.88/18.59
	B			3/6	4/9	N.A.	40.68/18.57
	C			3/6	4/9	36/12	**40.45/18.31**
MobileNet v2 [149]	A/B	3.4	0.3	1	1	N.A.	27.7/9.4
	C				1	2	**27.6/9.3**
	C				1	3	27.7/**9.3**
	C				1	4	27.7/9.4
	C				1	8	27.8/9.6

0.21%/0.10% and 0.61%/0.17% with HDWConv-B and HDWConv-C, respectively. Second, for the same version of HDWConv under the same architecture when $a > 1$, $\epsilon = 3$ is always better than $\epsilon = 2$ as it preserves more channels for the resize transform T_r. Meanwhile, HDWConv-A is more sensitive to the value of ϵ than HDWConv-B and HDWConv-C. For LST-Net v2 under ResNet-18, the top-1 gap between $\epsilon = 2$ and $\epsilon = 3$ by using HDWConv-A is 0.50% while the corresponding gap by using HDWConv-B and HDWConv-C is reduced to 0.45% and 0.41%, respectively. Third, HDWConv-C can also outperform HDWConv-A/B under the architecture of MobileNet v2 when $\epsilon = 1$. The top-1/5 error rates are both reduced by 0.1% when there are 2 subgroups. However, increase of the feature subgroups does not further

brings performance gains. Instead, it slightly increases error rates. When there are 8 subgroups, the top-1/5 error rates of HDWConv-C is even worse than those of HDWConv-A/B by 0.1%/0.2%. As MobileNet v2 is rather slim, each feature subgroup will easily contain fewer channels with more subgroups. In this sense, HDWConv-C will become vaulnerable to some distortions of frequency from its input channels, which causes performance drop.

Judging from the above results, we use HDWConv-C by default to build LST-Net v2 in the remaining of this chapter.

Choice of number of groups. We study the choice of number of groups for HDWConv-C under the architecture of WRN16-8 [198] on CIFAR-10/100. For comparison, a scale factor m is used to adjust the default number of groups (*i.e.*, a^2). Table 4.4 presents details of all HDWConv-C constructed in the experiment with different number of groups. One can see that increase of groups for the same expansion rate ϵ scales up the size of groups with smaller expansion rates while it reduces the size of groups with larger expansion rates. Notably, a larger number of groups could make HDWConv inapplicable to some slim models. For example, when $m = 2.5$ and $\epsilon = 2$, there are 45 subgroups in total, which is even greater than the channel size of most layers (*e.g.*, 16 or 32 channels) of a vanilla ResNet [59] model. In addition, an HDWConv layer with fewer groups can't produce larger expansion rate as we only allow non-empty groups. For instance, $\epsilon = 3$ can't be obtained from 3 groups as the last two groups are not allowed to be empty. Table 4.5 presents the results. One can see that increase of groups always causes performance drop for the same expansion rate on both CIFAR-10 and CIFAR-100 datasets. Besides, the channel expansion of our HDWConv can be more flexible when $\epsilon = 2$. To be consistent with the case of $\epsilon = 3$, we set the number of groups to a^2 (*i.e.*, $m = 1.0$) for an HDWConv layer by default.

#Parameters reduction vs. performance. LST v2 reduces the feature re-

Table 4.4: Details of HDWConv-C ($s = 3, a^2 = 4$) with different number of groups.

ϵ	m	#Groups ($m \cdot a^2$)	#Subgroups ($\|R\|_1$)	R^T
2	0.75	3	3	$[1, 1, 1]$
	1.0	4	10	$[1, 1, 5, 3]$
	1.5	6	15	$[1, 1, 1, 1, 1, 10]$
	2.0	8	28	$[1, 1, 1, 1, 1, 1, 1, 21]$
	2.5	10	45	$[1, 1, 1, 1, 1, 1, 1, 1, 1, 36]$
3	1.0	4	10	$[3, 5, 1, 1]$
	1.5	6	8	$[1, 1, 1, 1, 2, 2]$
	2.0	8	14	$[1, 1, 1, 1, 1, 1, 1, 7]$
	2.5	10	23	$[1, 1, 1, 1, 1, 1, 1, 1, 2, 13]$

dundancies of LST. A smaller ϵ suggests a smaller number of channels, hence fewer parameters and less overhead. Meanwhile, fewer features may reduce the representation power, causing certain performance drop. It is critical to study the relationship between number of parameters and performance. We build ResNet-18 for ImageNet using Conv2d, LST and the proposed LST v2. The number of channels of Conv2d and LST is multiplied by a decimal to obtain comparable overhead (in terms of FLOPs) to LST v2.

Table 4.6 shows the results in a descending order of FLOPs. One can have at least two findings. First, with comparable overhead, LST v2 requires the fewest number of parameters, while it achieves the lowest error rates. Second, with the reduction of channel number, the performance drops for all three methods, while LST v2 drops the least. In short, LST v2 improves much the accuracy of both Conv2d and LST with similar cost. When C_o is not divisible by C_i in an LST bottleneck due to multiplication of a given decimal (see Figure 3.3(b)), we replace a standard DWConv with the proposed HDWConv-A for its downsample operator, where there are just two groups and one group contains (C_o *mod* C_i) channels and all channels of this

Table 4.5: Comparison (error rates, %) of choice of number of groups for HDWConv-C on CIFAR-10/100.

ϵ	m	#Groups ($m \cdot a^2$)	#Subgroups ($\|R\|_1$)	C10	C100
2	0.75	3	3	**4.50**	**20.98**
	1.0	4	10	4.53	21.05
	1.5	6	15	4.55	21.26
	2.0	8	28	5.16	21.52
	2.5	10	45	5.29	23.56
3	1.0	4	10	**4.51**	**20.09**
	1.5	6	8	4.41	20.36
	2.0	8	14	4.54	20.38
	2.5	10	23	5.87	23.25

group are expanded by one more time than the rest.

4.4.3 Evaluation on image classification

Evaluation on CIFAR-10 and CIFAR-100. To investigate the trade-off between performance and reduction in number of parameters and overhead, we build LST-Net v2 under the architectures of ResNet [59] and WRN [198] for CIFAR-10/100, respectively.

Table 4.7 and Table 4.8 demonstrate the results. Compared to LST [97], the proposed LST v2 reduces around 40% and 20% of parameters and overhead under each architecture when $\epsilon = 2$ and $\epsilon = 3$. The largest gap between LST and LST v2 ($\epsilon = 2$) under ResNet is just 1.0%/1.6% in terms of top-1 error rates while the largest gap between LST and LST v2 ($\epsilon = 3$) is further closed to 0.2%/1.1%. In contrast, our method obtains comparable or even better performance than LST under WRN. This suggests the necessity to partition the input of LST v2 into a few groups and assign separate DWConv kernels for effective modelling. Besides, LST v2 always

Table 4.6: Effect of number of parameters (error rates, %) on ImageNet.

Backbone Method	$\#Ch_{out}$ Mult.	FLOPs (G)	#Param (M)	Top-1	Top-5
Conv2d	×1.0	1.81	11.69	30.24	10.92
Conv2d	×0.9	**1.47**	9.48	31.09	11.28
LST	×1.0	1.48	**8.03**	**26.55**	**8.59**
Conv2d	×0.8	**1.23**	7.55	32.57	12.18
LST	×0.9	**1.23**	6.72	27.58	9.15
LST v2 ($\epsilon = 3$)	N.A.	**1.23**	**6.51**	**26.61**	**8.53**
Conv2d	×0.71	1.00	5.99	33.91	12.90
LST	×0.79	1.00	5.42	28.25	9.61
LST v2 ($\epsilon = 2$)	N.A.	**0.97**	**4.98**	**27.02**	**8.87**

Table 4.7: Results of LST v2 under ResNet on CIFAR-10/100.

Arch.	Method	#Param (M)	FLOP	Top-1 E. R. (%)
ResNet-20 [59]	LST [97]	0.2	34.0M	**6.7/28.2**
	Conv2d	$0.3_{\uparrow 35.0\%}$	$40.8\text{M}_{\uparrow 20.0\%}$	$7.7_{\uparrow 1.0}/30.9_{\uparrow 2.7}$
	LST v2 ($\epsilon = 2$)	$\mathbf{0.1_{\downarrow 40.5\%}}$	$\mathbf{20.6}\text{M}_{\downarrow \mathbf{39.3\%}}$	$7.7_{\uparrow 1.0}/29.8_{\uparrow 1.6}$
	LST v2 ($\epsilon = 3$)	$0.2_{\downarrow 19.9\%}$	$27.5\text{M}_{\downarrow 19.2\%}$	$\mathbf{6.7_{0.0}}/29.1_{\uparrow 0.9}$
ResNet-56 [59]	LST [97]	0.6	94.0M	$5.6/\mathbf{24.1}$
	Conv2d	$0.9_{\uparrow 45.8\%}$	$126.0\text{M}_{\uparrow 34.0\%}$	$6.6_{\uparrow 1.0}/27.6_{\uparrow 3.5}$
	LST v2 ($\epsilon = 2$)	$\mathbf{0.3_{\downarrow 41.1\%}}$	$\mathbf{55.1}\text{M}_{\downarrow \mathbf{41.3\%}}$	$5.9_{\uparrow 0.3}/26.4_{\uparrow 2.2}$
	LST v2 ($\epsilon = 3$)	$0.5_{\downarrow 20.7\%}$	$74.5\text{M}_{\downarrow 20.7\%}$	$\mathbf{5.4}_{\downarrow 0.2}/24.4_{\uparrow 0.3}$
ResNet-110 [59]	LST [97]	1.2	183.0M	**5.0/22.7**
	Conv2d	$1.7_{\uparrow 47.9\%}$	$253.0\text{M}_{\uparrow 38.3\%}$	$6.6_{\uparrow 1.6}/25.2_{\uparrow 2.5}$
	LST v2 ($\epsilon = 2$)	$\mathbf{0.7_{\downarrow 41.0\%}}$	$\mathbf{106.9}\text{M}_{\downarrow \mathbf{41.6\%}}$	$5.6_{\uparrow 0.6}/24.7_{\uparrow 2.0}$
	LST v2 ($\epsilon = 3$)	$0.9_{\downarrow 20.5\%}$	$145.1\text{M}_{\downarrow 20.7\%}$	$5.2_{\uparrow 0.2}/23.8_{\uparrow 1.1}$

Table 4.8: Results of LST v2 under WRN architecture on CIFAR-10/100.

Arch.	Method	#Param (M)	FLOP	Top-1 E. R. (%)
WRN16-8 [198]	LST [97]	7.4	1.3G	4.7/20.9
	Conv2d	$11.0_{\uparrow 47.7\%}$	$2.0\text{G}_{\uparrow 54.3\%}$	$4.8_{\uparrow 0.1}/22.0_{\uparrow 1.2}$
	LST v2 ($\epsilon = 2$)	$\mathbf{4.4_{\downarrow 40.4\%}}$	$\mathbf{0.8}\text{G}_{\downarrow \mathbf{37.6\%}}$	$\mathbf{4.5_{\downarrow 0.2}}/21.1_{\uparrow 0.2}$
	LST v2 ($\epsilon = 3$)	$5.9_{\downarrow 20.2\%}$	$1.1\text{G}_{\downarrow 18.8\%}$	$\mathbf{4.5_{\downarrow 0.2}/20.1_{\downarrow 0.8}}$
WRN16-10 [198]	LST [97]	11.5	2.0G	**4.5**/20.2
	Conv2d	$17.1_{\uparrow 48.5\%}$	$3.1\text{G}_{\uparrow 55.0\%}$	$\mathbf{4.5_{0.0}}/21.5_{\uparrow 1.3}$
	LST v2 ($\epsilon = 2$)	$\mathbf{6.9_{\downarrow 40.4\%}}$	$\mathbf{1.3}\text{G}_{\downarrow \mathbf{37.5\%}}$	$4.6_{\uparrow 0.1}/20.0_{\downarrow 0.2}$
	LST v2 ($\epsilon = 3$)	$9.2_{\downarrow 20.2\%}$	$1.6\text{G}_{\downarrow 18.8\%}$	$\mathbf{4.5_{0.0}/19.9_{\downarrow 0.3}}$
WRN22-8 [198]	LST [97]	10.9	1.8G	4.4/19.3
	Conv2d	$17.2_{\uparrow 56.8\%}$	$2.9\text{G}_{\uparrow 61.0\%}$	$4.6_{\uparrow 0.2}/21.2_{\uparrow 1.9}$
	LST v2 ($\epsilon = 2$)	$\mathbf{6.5_{\downarrow 40.4\%}}$	$\mathbf{1.1}\text{G}_{\downarrow \mathbf{38.4\%}}$	$\mathbf{4.2_{\downarrow 0.2}}/19.7_{\uparrow 0.4}$
	LST v2 ($\epsilon = 3$)	$8.7_{\downarrow 20.2\%}$	$1.5\text{G}_{\downarrow 19.2\%}$	$\mathbf{4.2_{\downarrow 0.2}/19.1_{\downarrow 0.2}}$
WRN22-10 [198]	LST [97]	17.0	2.8G	4.3/**18.6**
	Conv2d	$26.8_{\uparrow 57.5\%}$	$4.5\text{G}_{\uparrow 64.0\%}$	$4.4_{\uparrow 0.1}/20.8_{\uparrow 2.2}$
	LST v2 ($\epsilon = 2$)	$\mathbf{10.2_{\downarrow 40.3\%}}$	$\mathbf{1.7}\text{G}_{\downarrow \mathbf{38.3\%}}$	$4.1_{\downarrow 0.2}/19.0_{\uparrow 0.4}$
	LST v2 ($\epsilon = 3$)	$13.6_{\downarrow 20.2\%}$	$2.3\text{G}_{\downarrow 19.2\%}$	$\mathbf{4.0_{\downarrow 0.3}}/18.7_{\uparrow 0.1}$
WRN28-10 [198]	LST [97]	22.5	3.6G	4.0/18.2
	Conv2d	$36.5_{\uparrow 61.2\%}$	$6.0\text{G}_{\uparrow 64.0\%}$	$4.2_{\uparrow 0.2}/20.5_{\uparrow 2.3}$
	LST v2 ($\epsilon = 2$)	$\mathbf{13.4_{\downarrow 40.3\%}}$	$\mathbf{2.2}\text{G}_{\downarrow \mathbf{38.7\%}}$	$4.0_{\mathbf{0.0}}/19.1_{\uparrow 0.9}$
	LST v2 ($\epsilon = 3$)	$18.0_{\downarrow 20.1\%}$	$2.9\text{G}_{\downarrow 19.4\%}$	$\mathbf{3.8_{\downarrow 0.2}/18.1_{\downarrow 0.2}}$
WRN28-12 [198]	LST [97]	32.3	5.2G	3.9/**17.9**
	Conv2d	$43.4_{\uparrow 61.5\%}$	$8.6\text{G}_{\uparrow 64.6\%}$	$4.3_{\uparrow 0.4}/20.4_{\uparrow 2.5}$
	LST v2 ($\epsilon = 2$)	$\mathbf{19.3_{\downarrow 40.3\%}}$	$\mathbf{3.2}\text{G}_{\downarrow \mathbf{38.7\%}}$	$3.9_{0.0}/18.5_{\uparrow 0.6}$
	LST v2 ($\epsilon = 3$)	$25.8_{\downarrow 20.1\%}$	$4.2\text{G}_{\downarrow 19.4\%}$	$\mathbf{3.7_{\downarrow 0.2}/17.9_{0.0}}$
WRN40-4 [198]	LST [97]	5.5	0.9G	4.3/**19.1**
	Conv2d	$8.9_{\uparrow 66.1\%}$	$1.4\text{G}_{\uparrow 66.8\%}$	$5.0_{\uparrow 0.7}/22.9_{\uparrow 3.8}$
	LST v2 ($\epsilon = 2$)	$\mathbf{3.3_{\downarrow 40.4\%}}$	$\mathbf{0.5}\text{G}_{\downarrow \mathbf{39.5\%}}$	$4.6_{\uparrow 0.3}/20.5_{\uparrow 1.4}$
	LST v2 ($\epsilon = 3$)	$4.4_{\downarrow 20.2\%}$	$0.7\text{G}_{\downarrow 19.8\%}$	$\mathbf{4.1_{\downarrow 0.2}}/19.9_{\uparrow 0.7}$
WRN40-8 [198]	LST [97]	21.5	3.4G	3.8/18.6
	Conv2d	$35.8_{\uparrow 66.1\%}$	$5.6\text{G}_{\uparrow 66.8\%}$	$4.7_{\uparrow 0.9}/19.4_{\uparrow 0.8}$
	LST v2 ($\epsilon = 2$)	$\mathbf{12.9_{\downarrow 40.2\%}}$	$\mathbf{2.1}\text{G}_{\downarrow \mathbf{39.3\%}}$	$3.9_{\uparrow 0.1}/18.4_{\downarrow 0.2}$
	LST v2 ($\epsilon = 3$)	$17.2_{\downarrow 20.1\%}$	$2.7\text{G}_{\downarrow 19.6\%}$	$\mathbf{3.6_{\downarrow 0.2}/18.3_{\downarrow 0.3}}$

Table 4.9: Comparison of state-of-the-art methods for compressing Conv2d on CIFAR-10.

Arch.	Method	#Param (M)	MFLOP	Top-1 E. R. (%)
VGG-16	Conv2d	15.0	313	6.4
	ℓ_1 [95, 116]	5.4	206	6.6
	SBP [61]	N.A.	136	7.5
	Ghost [53]	7.7	158	6.3
	LST v2 ($\epsilon = 2$)	**4.1**	**90**	6.1
	LST v2 ($\epsilon = 3$)	5.9	131	**6.0**
ResNet-56	Conv2d	0.85	125	6.6
	CP [61]	N.A.	63	8.0
	ℓ_1 [95, 116]	0.73	91	7.5
	AMC [60]	N.A.	63	8.1
	Shift [180]	0.55	84	7.3
	SSL [27]	0.55	84	7.2
	Ghost [53]	0.43	63	7.3
	LST v2 ($\epsilon = 2$)	**0.35**	**55**	5.9
	LST v2 ($\epsilon = 3$)	0.47	75	**5.4**

outperforms Conv2d under both architectures while it saves 45% $\sim$ 60% parameters and overhead. Considering the actual parameters and overhead of LST v2, we believe our method is much competitive on CIFAR-10 and CIFAR-100.

We also compare LST v2 with state-of-the-art methods for compressing Conv2d under VGG-16 variant [197] and ResNet-56 [59] on CIFAR-10, including ℓ_1 [95, 116], SBP [61], Ghost [53], CP [61], AMC [60], Shift [180] and SSL [27].

Table 4.9 shows the results. We can have three findings. First, LST v2 takes up comparable or even fewer parameters and less overhead under each architecture. Second, LST v2 obtains the best top-1 error rates. LST v2 ($\epsilon = 3$) outperforms its closest follow-up, Ghost [53], by 0.3% under VGG-16. Meanwhile, it significantly reduces top-1 error rates of its closest follow-up, SSL [27], by 1.8% under ResNet-56.

Table 4.10: Results of LST v2 under ResNet architecture on ImageNet.

Arch.	Method	#Param (M)	GFLOP	Top-1/5 E. R. (%)
ResNet-18 [59]	LST [97]	8.03	1.48	**26.55**/8.59
	Conv2d	$11.69_{\uparrow 45.58\%}$	$1.81_{\uparrow 22.30\%}$	$30.24_{\uparrow 3.69}/10.92_{\uparrow 2.33}$
	LST v2 ($\epsilon = 2$)	$\mathbf{4.98_{\downarrow 37.94\%}}$	$\mathbf{0.97_{\downarrow 34.56\%}}$	$27.02_{\uparrow 0.47}/8.87_{\uparrow 0.28}$
	LST v2 ($\epsilon = 3$)	$6.51_{\downarrow 18.97\%}$	$1.23_{\downarrow 17.13\%}$	$26.61_{\uparrow 0.06}/\mathbf{8.53_{\downarrow 0.06}}$
ResNet-34 [59]	LST [97]	13.82	2.56	23.92/7.24
	Conv2d	$21.79_{\uparrow 57.67\%}$	$3.66_{\uparrow 42.97\%}$	$26.70_{\uparrow 2.78}/8.58_{\uparrow 1.34}$
	LST v2 ($\epsilon = 2$)	$\mathbf{8.45_{\downarrow 38.88\%}}$	$\mathbf{1.61_{\downarrow 37.20\%}}$	$24.08_{\uparrow 0.16}/7.33_{\uparrow 0.09}$
	LST v2 ($\epsilon = 3$)	$11.13_{\downarrow 19.44\%}$	$2.08_{\downarrow 18.66\%}$	$\mathbf{23.86_{\downarrow 0.06}}/\mathbf{7.17_{\downarrow 0.07}}$
ResNet-50 [59]	LST [97]	23.33	4.05	22.78/6.66
	Conv2d	$25.24_{\uparrow 9.69\%}$	$4.09_{\uparrow 0.99\%}$	$23.85_{\uparrow 1.07}/7.13_{\uparrow 0.47}$
	LST v2 ($\epsilon = 2$)	$\mathbf{13.52_{\downarrow 41.24\%}}$	$\mathbf{2.53_{\downarrow 37.56\%}}$	$22.87_{\uparrow 0.09}/\mathbf{6.64_{\downarrow 0.02}}$
	LST v2 ($\epsilon = 3$)	$18.02_{\downarrow 21.69\%}$	$3.32_{\downarrow 18.11\%}$	$\mathbf{22.77_{\downarrow 0.01}}/\mathbf{6.64_{\downarrow 0.02}}$
ResNet-101 [59]	LST [97]	42.36	7.75	21.63/**5.94**
	Conv2d	$44.55_{\uparrow 5.17\%}$	$7.80_{\uparrow 0.65\%}$	$22.63_{\uparrow 1.00}/6.44_{\uparrow 0.50}$
	LST v2 ($\epsilon = 2$)	$\mathbf{25.03_{\downarrow 40.92\%}}$	$\mathbf{4.70_{\downarrow 39.38\%}}$	$21.84_{\uparrow 0.21}/6.16_{\uparrow 0.22}$
	LST v2 ($\epsilon = 3$)	$33.28_{\downarrow 21.43\%}$	$6.22_{\downarrow 19.77\%}$	$\mathbf{21.61_{\downarrow 0.02}}/6.01_{\uparrow 0.07}$

Third, the proposed LST v2 is the only method that improves the performance of Conv2d under the ResNet-56 architecture when $\epsilon = 2$ or $\epsilon = 3$.

Evaluation on ImageNet. We study the performance of LST-Net v2 for large-scale image recognition on ImageNet. For comparison, we build CNNs under the architectures of ResNet [59], WRN [198], VGG-GAP [151] (11-layer) and AlexNet-GAP [90].

Table 4.10, Table 4.11, and Table 4.12 demonstrate the results. One can obtain the following findings. First, under the ResNet architecture, LST v2 ($\epsilon = 3$) achieves comparable or slightly better top-1/5 error rates compared to LST [97] by saving nearly 20% of parameters and overhead. LST v2 ($\epsilon = 2$) further reduces the cost of LST [97] by around 40% while the top-1/5 performance gap is no more than 0.47%/0.28%. Compared to the conventional Conv2d, LST v2 only requires

Table 4.11: Results of LST v2 under WRN architecture on ImageNet.

Arch.	Method	#Param (M)	GFLOP	Top-1/5 E. R. (%)
WRN18-1.5 [198]	LST [97]	17.53	3.21	24.44/7.51
	Conv2d	$25.88_{\uparrow 47.64\%}$	$3.87_{\uparrow 20.59\%}$	$27.06_{\uparrow 2.62}/9.00_{\uparrow 1.49}$
	LST v2 ($\epsilon = 2$)	$\mathbf{10.74_{\downarrow 38.73\%}}$	$\mathbf{2.07_{\downarrow 35.52\%}}$	$24.72_{\uparrow 0.28}/7.58_{\uparrow 0.07}$
	LST v2 ($\epsilon = 3$)	$14.13_{\downarrow 19.37\%}$	$2.64_{\downarrow 17.76\%}$	$\mathbf{24.34_{\downarrow 0.10}/7.44_{\downarrow 0.07}}$
WRN18-2 [198]	LST [97]	30.68	5.59	23.49/6.93
	Conv2d	$45.62_{\uparrow 48.68\%}$	$6.70_{\uparrow 19.87\%}$	$25.58_{\uparrow 2.09}/8.06_{\uparrow 1.13}$
	LST v2 ($\epsilon = 2$)	$\mathbf{18.67_{\downarrow 39.14\%}}$	$\mathbf{3.58_{\downarrow 35.93\%}}$	$23.56_{\uparrow 0.07}/7.12_{\uparrow 0.19}$
	LST v2 ($\epsilon = 3$)	$24.68_{\downarrow 19.57\%}$	$4.59_{\downarrow 17.96\%}$	$\mathbf{23.12_{\downarrow 0.37}/6.86_{\downarrow 0.07}}$
WRN18-3 [198]	LST [97]	67.96	12.31	22.33/6.52
	Conv2d	$101.78_{\uparrow 49.77\%}$	$14.72_{\uparrow 19.53\%}$	$24.06_{\uparrow 1.73}/7.33_{\uparrow 0.81}$
	LST v2 ($\epsilon = 2$)	$\mathbf{41.07_{\downarrow 39.56\%}}$	$\mathbf{7.84_{\downarrow 36.35\%}}$	$22.42_{\uparrow 0.09}/6.41_{\downarrow 0.11}$
	LST v2 ($\epsilon = 3$)	$54.52_{\downarrow 19.78\%}$	$10.08_{\downarrow 18.18\%}$	$\mathbf{22.14_{\downarrow 0.19}/6.36_{\downarrow 0.16}}$
WRN34-1.5 [198]	LST [97]	30.42	5.59	**22.29/6.30**
	Conv2d	$48.61_{\uparrow 59.78\%}$	$8.03_{\uparrow 43.67\%}$	$24.50_{\uparrow 2.21}/7.58_{\uparrow 1.28}$
	LST v2 ($\epsilon = 2$)	$\mathbf{18.46_{\downarrow 39.32\%}}$	$\mathbf{3.49_{\downarrow 37.55\%}}$	$22.55_{\uparrow 0.26}/6.46_{\uparrow 0.16}$
	LST v2 ($\epsilon = 3$)	$24.44_{\downarrow 19.66\%}$	$4.54_{\downarrow 18.78\%}$	$22.43_{\uparrow 0.14}/6.45_{\uparrow 0.15}$
WRN34-2 [198]	LST [97]	53.49	9.79	**21.44**/6.11
	Conv2d	$86.04_{\uparrow 60.87\%}$	$14.09_{\uparrow 43.92\%}$	$23.39_{\uparrow 1.95}/7.00_{\uparrow 0.89}$
	LST v2 ($\epsilon = 2$)	$\mathbf{32.33_{\downarrow 39.55\%}}$	$\mathbf{6.09_{\downarrow 37.77\%}}$	$21.60_{\uparrow 0.16}/6.05_{\downarrow 0.06}$
	LST v2 ($\epsilon = 3$)	$42.91_{\downarrow 19.77\%}$	$7.94_{\downarrow 18.88\%}$	$21.58_{\uparrow 0.14}/\mathbf{6.01_{\downarrow 0.10}}$

approximately 40% $\sim$ 55% overhead under the same architecture but our method significantly reduces the top-1 and top-5 error rates by 0.79% $\sim$ 3.63% and 0.28% $\sim$ 2.39%, respectively. Second, under the WRN architecture, LST v2 outperforms both LST [97] and the conventional Conv2d when $\epsilon = 3$. For example, the top-1 error rate is reduced by 0.37% under WRN18-2. The gap between LST v2 ($\epsilon = 2$) and LST [97] is rather close, ranging from 0.1% to 0.3%. Third, under the VGG-GAP [151] architecture, LST-Net v2 ($\epsilon = 3$) obtains the best results by saving nearly 20% of parameters and overhead. It reduces the top-1/5 error rates of LST [97], conventional Conv2d and Conv2d+BN by 0.54%/0.22%, 4.71%/2.16% and 2.94%/1.72%, respec-

Table 4.12: Results of LST v2 under VGG and AlexNet architecture on ImageNet.

Arch.	Method	#Param (M)	GFLOP	Top-1/5 E. R. (%)
VGG-GAP [151]	LST [97]	6.63	5.89	29.23/10.26
	Conv2d	$9.73_{\uparrow 46.85\%}$	$7.49_{\uparrow 27.15\%}$	$33.40_{\uparrow 4.17}/12.20_{\uparrow 1.94}$
	Conv2d+BN	$9.73_{\uparrow 46.85\%}$	$7.49_{\uparrow 27.15\%}$	$31.63_{\uparrow 2.40}/11.76_{\uparrow 1.50}$
	LST v2 ($\epsilon = 2$)	$\mathbf{4.17_{\downarrow 36.99\%}}$	$\mathbf{3.55_{\downarrow 39.67\%}}$	$29.65_{\uparrow 0.42}/10.63_{\uparrow 0.37}$
	LST v2 ($\epsilon = 3$)	$5.40_{\downarrow 18.50\%}$	$4.72_{\downarrow 19.83\%}$	$\mathbf{28.69_{\downarrow 0.54}/10.04_{\downarrow 0.22}}$
AlexNet-GAP [90]	LST [97]	2.25	0.60	**39.91/17.86**
	Conv2d	$2.73_{\uparrow 21.33\%}$	$0.66_{\uparrow 10.00\%}$	$51.13_{\uparrow 11.22}/26.33_{\uparrow 8.47}$
	Conv2d+BN	$2.73_{\uparrow 21.33\%}$	$0.66_{\uparrow 10.00\%}$	$46.65_{\uparrow 6.74}/23.43_{\uparrow 5.57}$
	LST v2 ($\epsilon = 2$)	$\mathbf{1.42_{\downarrow 36.73\%}}$	$\mathbf{0.35_{\downarrow 42.39\%}}$	$41.59_{\uparrow 1.68}/19.33_{\uparrow 1.47}$
	LST v2 ($\epsilon = 3, 6$)	$1.84_{\downarrow 18.31\%}$	$0.49_{\downarrow 19.13\%}$	$40.45_{\uparrow 0.54}/18.31_{\uparrow 0.45}$

tively. Fourth, under the AlexNet-GAP [90] architecture, the top-1/5 error rates of LST v2 are respectively 0.54%/0.45% and 1.68%/1.47% behind those of LST [97] when $\epsilon = 3$ and $\epsilon = 2$. However, LST v2 still outperforms Conv2d and Conv2d+BN by over 5.06%/4.10% in terms of the top-1/5 error rates while the proposed method only requires nearly half of the parameters and overhead. The gap between LST [97] and LST v2 may result from the fact that AlexNet-GAP [90] is relatively thin and shallow compared to other modern architectures. There may be insufficient number of high frequency features for reduction in LST v2, which leads to the performance drop. However, the proposed method still shows very competitive performance w.r.t. its cost.

Besides, we build up models under ResNet50 and compare the inference time including data loading and pre-processing with a single CPU thread. It takes LST [97] 84.61ms on average to cope with a 224×224 center-cropped image while the proposed LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) only need 53.28ms (37.04% off) and 68.72ms (18.78% off), respectively.

Figure 4.6 and Figure 4.5 compare convergence curves of Conv2d, LST, LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) on ImageNet under ResNet-18 and ResNet-50 [59]

Table 4.13: Comparison of state-of-the-art methods for compressing ResNet-50 on ImageNet.

Method	#Param (M)	GFLOP	Top-1/5 E. R. (%)
Conv2d	25.2	4.1	23.9/7.1
ThiNet [117]	16.9	2.6	27.9/9.7
NISP [196]	14.4	2.3	N.A./9.2
Versatile [172]	**11.0**	3.0	25.5/8.2
SSS [76]	N.A.	2.8	25.8/8.1
Ghost [53]	13.0	2.2	25.0/7.7
LST v2	11.6	**2.1**	**23.0/6.8**

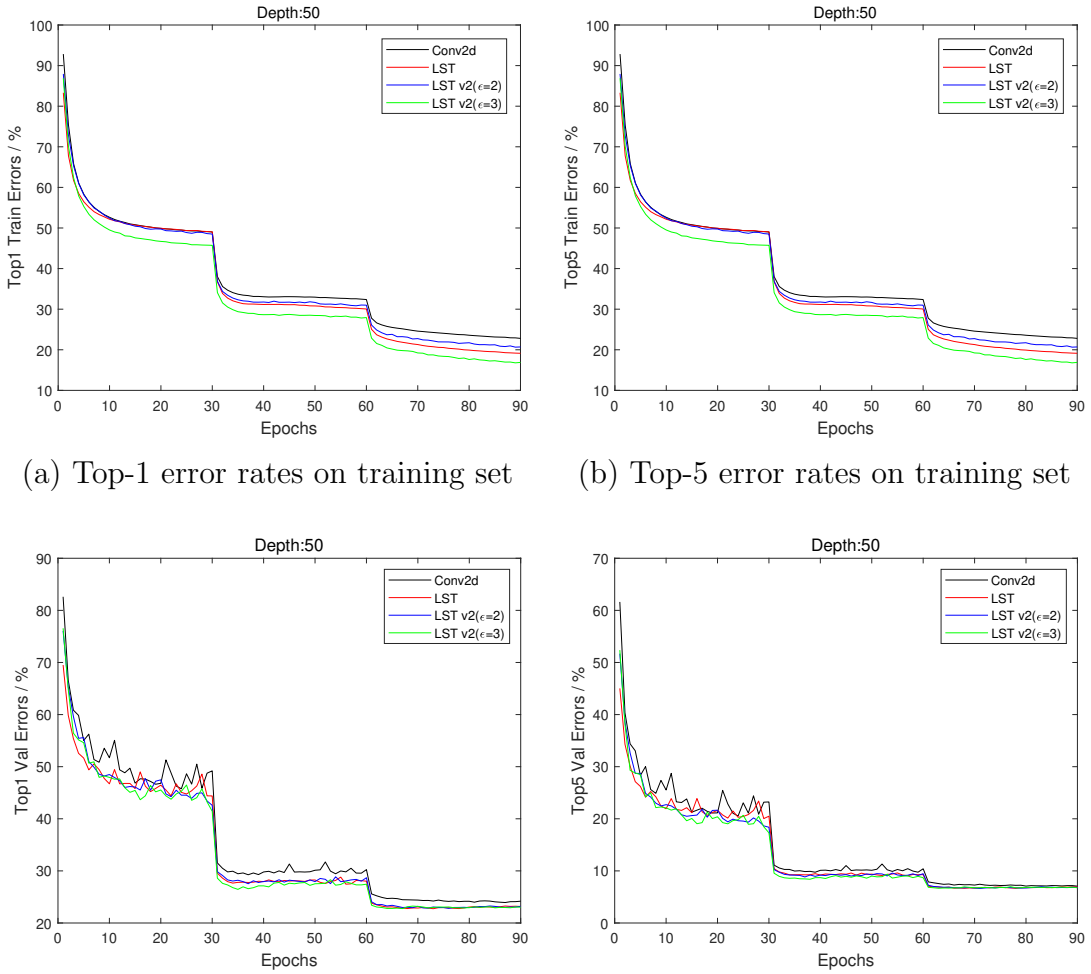
architecture, respectively. One can see that both LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) are similar to LST, achieving more smooth convergence curves than Conv2d under both architectures.

In addition, we compare LST v2 with some state-of-the-art methods for compressing ResNet-50 on ImageNet, including ThiNet [117], NISP [196], Versatile [172], SSS [76] and Ghost [53]. We use LST-II v2 bottleneck to construct CNNs by following the layer configuration of LST-Net [97]. We set $\epsilon = 1.5$ for LST v2 to obtain comparable cost.

Table 4.13 presents the results. We can have at least two findings. First, LST v2 achieves the best top-1/5 error rates, the only one outperforming Conv2d. Second, our method saves 54.0% parameters and 48.8% overhead, taking up the least overhead and the second fewest parameters following SSS [76]. Like Ghost [53], the structure of LST v2 is deterministic. There is no need to train the whole model like ThiNet [117], NISP [196], Versatile [172], and SSS [76].

4.4.4 Evaluation on robustness to common corruptions

We investigate the robustness of LST-Net v2 to common corruptions in input on the ImageNet-C dataset [63]. We perform prediction by using ImageNet pre-trained



(a) Top-1 error rates on training set (b) Top-5 error rates on training set

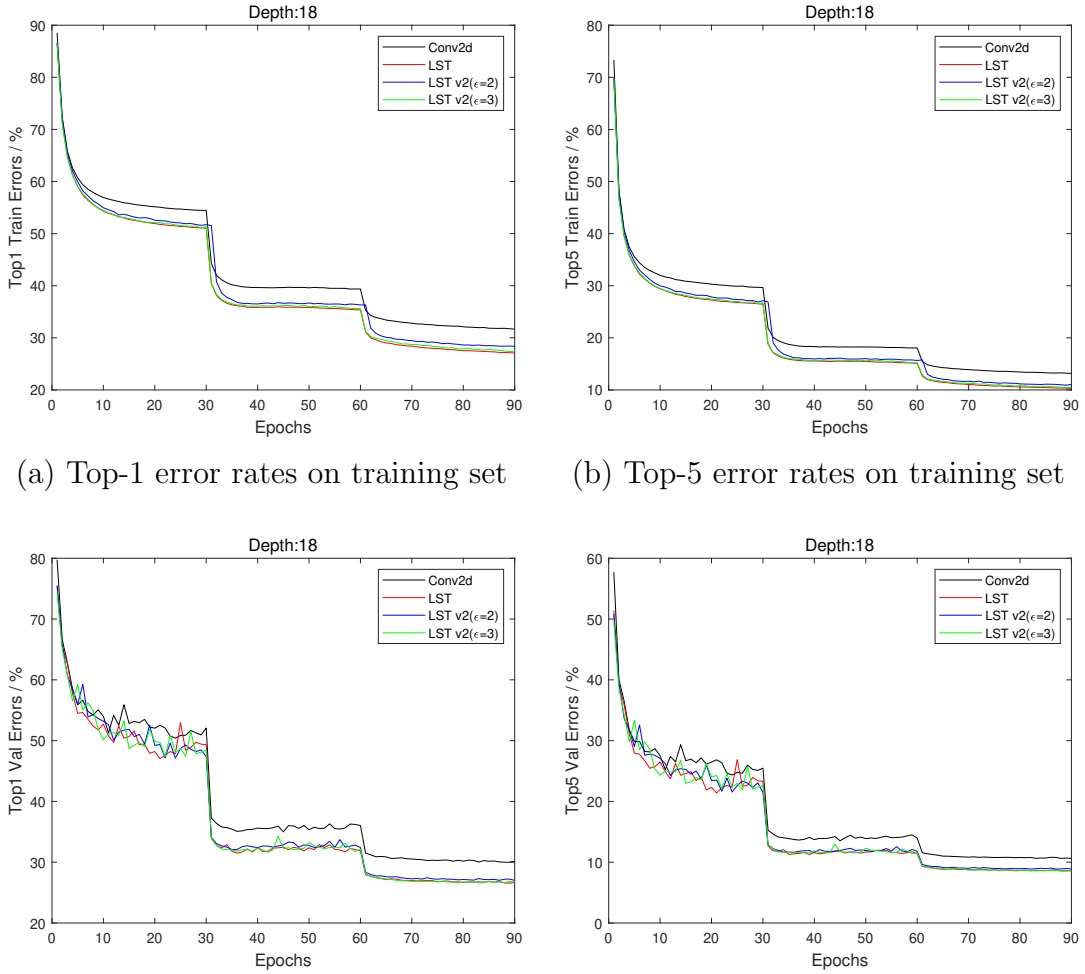
(c) Top-1 error rates on validation set (d) Top-5 error rates on validation set

Figure 4.5: Comparison of convergence curves of Conv2d, LST and LST v2 under ResNet-50 architecture on ImageNet.

models under the architecture of ResNet-18 and ResNet-50 without fine-tuning on [63]. The mean corruption error (mCE) [63] is used as our criteria. We compare LST v2 with the conventional Conv2d and LST [97]. Table 4.14 demonstrates the mCE and corruption errors for each type of corruption. It can be observed that the proposed LST-Net v2 are more robust to common corruptions.

Table 4.14: Results of LST v2 on ImageNet-C.

Arch.	Method	Top-1/5 E.R. (%, clean)	mCE	Noise			Blur				Weather			Digital				Extra		
				Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG	Saturate	Spatter	Gaus. Blur	Speckle
ResNet-18 [59]	Conv2d	30.24/10.92	85.29	87	89	89	88	93	90	88	88	87	81	73	81	93	81	89	72	81	86	86
	LST [97]	26.55/8.59	79.89	80	81	83	**84**	**91**	**83**	85	**82**	82	**75**	**68**	**78**	**90**	**73**	**85**	**66**	**74**	**83**	76
	LST v2 ($\epsilon = 2$)	27.02/8.87	80.16	79	80	80	85	**91**	84	**84**	**82**	82	**75**	**68**	**78**	**90**	76	**85**	**66**	75	85	76
	LST v2 ($\epsilon = 3$)	**26.61/8.53**	**79.85**	**78**	**79**	**79**	85	**91**	84	**84**	**82**	**81**	**75**	**68**	**78**	**90**	74	**85**	**66**	75	85	**75**
ResNet-50 [59]	Conv2d	23.85/7.13	77.01	78	80	80	79	90	81	80	80	78	69	62	75	88	76	78	62	74	78	76
	LST [97]	22.78/6.66	70.54	71	72	71	**76**	**84**	77	**75**	73	72	**65**	**58**	**70**	**81**	61	72	**56**	**65**	75	67
	LST v2 ($\epsilon = 2$)	22.87/**6.64**	70.25	69	**70**	71	77	85	**75**	78	**72**	**72**	67	**58**	71	82	**58**	**69**	**56**	**65**	75	**65**
	LST v2 ($\epsilon = 3$)	**22.77/6.64**	**70.18**	**68**	**70**	**69**	**76**	86	76	77	73	74	67	**58**	**70**	82	**58**	70	57	**65**	**74**	**65**



(a) Top-1 error rates on training set (b) Top-5 error rates on training set

(c) Top-1 error rates on validation set (d) Top-5 error rates on validation set

Figure 4.6: Comparison of convergence curves of Conv2d, LST and LST v2 under ResNet-18 architecture on ImageNet.

4.4.5 Evaluation on large-scale scene recognition

We study the performance of LST v2 for large-scale scene recognition on Places365-Standard [209]. We construct models under ResNet, VGG-GAP and AlexNet-GAP. Instead of fine-tuning ImageNet pre-trained models like [209, 97], we train each model from scratch for fair comparison. Top-5 accuracy is used as our criteria under the standard 10-crop setting. Table 4.15 presents results. Compared to LST [97], LST

v2 can achieve competitive or even better performance while it saves nearly 20% $\sim$ 40% parameters and overhead. Meanwhile, it always outperforms Conv2d and Conv2d+BN.

4.4.6 Evaluation on fine-grained visual recognition

We follow the experimental setting and protocal in Section 3.4.7 and study the performance of our LST v2 on fine-grained visual recognition. FGVC-Aircraft, Birds-CUB200-2011, FGVC-Cars, and Stanford Dogs are employed for evaluation. ResNet-50 is adopted as the backbone model, where Conv2d, LST [97] and LST v2 are used to build up the model.

Table 4.16 demonstrates the fine-grained visual recognition results. One can see that our LST v2 can achieve comparable results to LST on fine-grained visual recognition. Besides, LST v2 ($\epsilon = 2$) outperforms Conv2d by at least 1% on all four benchmarks. It is interesting to see that LST v2 ($\epsilon = 3$) further improves the top-1 accuracy of LST by 0.4% on Stanford Dog dataset.

4.4.7 Evaluation on texture classification

We study the performance of LST v2 on texture classification using the same setting with Section 3.4.8. Both DTD [30] and Indoor67 dataset [137] are employed for evaluation. All methods, including Conv2d, LST [97] and the proposed LST v2, are used to build up models under the architecture of ResNet-50 [59], and each model is pretrained on ImageNet.

Table 4.17 presents the texture classification results of LST v2. One can see that models constructed using LST v2 with different ϵ values obtain better performance than the one using Conv2d on both DTD and Indoor67 dataset. Average top-1 accuracy of LST ($\epsilon = 2$) and LST ($\epsilon = 3$) on DTD falls behind that of LST [97] by 2.03% and 1.36%, respectively. In fact, textures, unlike natural images, are similar to

Table 4.15: Results of LST v2 on Places365-Standard dataset.

Arch.	Method	#Param (M)	GFLOP	Top-5 Acc. (%)
ResNet-18 [59]	LST [97]	7.71	1.48	86.35
	Conv2d	$11.36_{\uparrow 47.39\%}$	$1.81_{\uparrow 22.55\%}$	$85.78_{\downarrow 0.57}$
	LST v2 ($\epsilon = 2$)	$\mathbf{4.66_{\downarrow 39.59\%}}$	$\mathbf{0.97_{\downarrow 34.58\%}}$	$86.36_{\uparrow 0.01}$
	LST v2 ($\epsilon = 3$)	$6.18_{\downarrow 19.83\%}$	$1.23_{\downarrow 17.15\%}$	$\mathbf{86.38_{\uparrow 0.03}}$
ResNet-34 [59]	LST [97]	13.50	2.56	86.94
	Conv2d	$21.47_{\uparrow 59.05\%}$	$3.66_{\uparrow 43.10\%}$	$86.51_{\downarrow 0.43}$
	LST v2 ($\epsilon = 2$)	$\mathbf{8.12_{\downarrow 39.85\%}}$	$\mathbf{1.61_{\downarrow 37.21\%}}$	$86.79_{\downarrow 0.15}$
	LST v2 ($\epsilon = 3$)	$10.81_{\downarrow 19.94\%}$	$2.08_{\downarrow 18.67\%}$	$\mathbf{86.95_{\uparrow 0.01}}$
ResNet-50 [59]	LST [97]	23.01	4.05	87.18
	Conv2d (fine-tuning)	$25.24_{\uparrow 9.69\%}$	$4.09_{\uparrow 0.99\%}$	$85.08_{\downarrow 2.10}$
	Conv2d	$25.24_{\uparrow 9.69\%}$	$4.09_{\uparrow 0.99\%}$	$86.79_{\downarrow 0.39}$
	LST v2 ($\epsilon = 2$)	$\mathbf{13.52_{\downarrow 41.24\%}}$	$\mathbf{2.53_{\downarrow 37.56\%}}$	$87.15_{\downarrow 0.03}$
	LST v2 ($\epsilon = 3$)	$18.02_{\downarrow 21.69\%}$	$3.32_{\downarrow 18.11\%}$	$\mathbf{87.20_{\uparrow 0.02}}$
VGG-GAP [151]	LST [97]	6.30	5.89	85.12
	Conv2d	$9.41_{\uparrow 49.33\%}$	$7.61_{\uparrow 29.14\%}$	$83.95_{\downarrow 1.17}$
	Conv2d+BN	$9.41_{\uparrow 49.33\%}$	$7.61_{\uparrow 29.14\%}$	$84.76_{\downarrow 0.36}$
	LST v2 ($\epsilon = 2$)	$\mathbf{3.85_{\downarrow 38.91\%}}$	$\mathbf{3.55_{\downarrow 39.66\%}}$	$85.20_{\uparrow 0.08}$
	LST v2 ($\epsilon = 3$)	$5.07_{\downarrow 19.45\%}$	$4.72_{\downarrow 19.83\%}$	$\mathbf{85.29_{\uparrow 0.17}}$
AlexNet-GAP [90]	LST [97]	2.09	0.62	**82.95**
	Conv2d	$2.56_{\uparrow 22.49\%}$	$0.66_{\uparrow 6.45\%}$	$77.89_{\downarrow 5.06}$
	Conv2d+BN	$2.56_{\uparrow 22.49\%}$	$0.66_{\uparrow 6.45\%}$	$79.70_{\downarrow 3.25}$
	LST v2 ($\epsilon = 2$)	$\mathbf{1.26_{\downarrow 39.70\%}}$	$\mathbf{0.35_{\downarrow 44.27\%}}$	$81.69_{\downarrow 1.26}$
	LST v2 ($\epsilon = 3, 6$)	$1.67_{\downarrow 19.87\%}$	$0.49_{\downarrow 21.77\%}$	$82.33_{\downarrow 0.62}$

Table 4.16: Fine-grained visual recognition results (top-1 accuracy, %) of LST v2 under the architecture of ResNet-50.

Backbone Model	FGVC -Aircraft	Birds -CUB200-2011	FGVC -Cars	Stanford Dogs
LST [97]	**90.04**	**85.61**	**91.51**	88.60
Conv2d	$86.70_{\downarrow 3.34}$	$82.21_{\downarrow 3.40}$	$88.68_{\downarrow 2.83}$	$76.23_{\downarrow 12.37}$
LST v2 ($\epsilon = 2$)	$88.03_{\downarrow 2.01}$	$84.47_{\downarrow 1.14}$	$89.73_{\downarrow 1.78}$	$87.52_{\downarrow 1.08}$
LST v2 ($\epsilon = 3$)	$89.11_{\downarrow 0.93}$	$85.21_{\downarrow 0.40}$	$90.51_{\downarrow 1.00}$	$\mathbf{89.00_{\uparrow 0.40}}$

Table 4.17: Texture classification results (top-1 accuracy, %) of LST v2 under the architecture of ResNet-50.

Backbone Method	DTD	Indoor67
Conv2d [59]	63.68±0.85	79.63
LST [97]	**67.47±1.09**	**81.72**
LST v2 ($\epsilon = 2$)	65.71±1.09	80.37
LST v2 ($\epsilon = 3$)	66.14±0.54	81.08

their low frequency features while differentiable with their high frequency features. In our view, the gap between LST and LST v2 on texture classification is still acceptable since LST v2 drops considerable high frequency features to save cost.

4.4.8 Evaluation on object detection and instance segmentation

We report object detection and instance segmentation results on MS-COCO [107]. We follow the data splittion, detectors, and backbone architecture detailed in Section 3.4.9. One should note that the LST v2 produce fewer channels than LST at the same layer due to its use of HDWConv. To make detectors compatible with our LST v2, we accordingly reduced the input channels where necessary in the detectors, so that detectors also require fewer parameters and less overhead when LST v2 is

employed to construct the backbone model.

Table 4.18 and Table 4.19 present the object detection results and instance segmentation results on MS-COCO validation set, respectively. One can see that LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) outperform Conv2d on both tasks. Compared to LST, mAP of LST v2 ($\epsilon = 3$) slightly lacks behind by 0.2% with Faster R-CNN on object detection, while it has the same or even better results in all the remaining cases. Meanwhile, the mAP gap between both LST v2 bottlenecks with various detectors ranges from 0.2% to 0.6%, which we believe is totally acceptable.

4.4.9 Evaluation on semantic segmentation

We report semantic segmentation results on PASCAL VOC [41]. We follow the same experimental settings described in Section 3.4.10. To evaluate LST v2, we reduce input channels where necessary in DeepLab V3 [23] and DeepLab V3+ [24]. To meet the requirement of both segmentation methods, we replace stride with dilation for HDWConv layers at Stage 4 of an ImageNet pretrained model by setting stride to 1 and dilation to 2 before fine-tuning on PASCAL VOC2012 dataset starts.

Table 4.20 demonstrates the results. It can be observed that both LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) achieve very close results of LST with the same segmentation method under all three indices. Meanwhile, our LST v2 are much better than the baseline Conv2d by at least 0.015 in terms of mIoU.

4.4.10 Evaluation on human pose estimation

To study the performance of LST-Net on human pose estimation, we use SimpleBaseline [183] as the key-points detection method. We only replace the backbone ResNet-50 with our LST-Net, where both backbone models are pre-trained on ImageNet dataset. All implementation remains the same with SimpleBaseline [183]. Both MPII dataset [2] and COCO key-point detection dataset [107] are used for

Table 4.18: Object detection results (%) of LST v2 on MS-COCO validation set.

Detector	Backbone Method	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN [144]	LST [97]	**40.8**	**62.2**	**44.3**	**24.8**	**44.7**	**53.1**
	Conv2d	37.4	58.1	40.4	21.2	41.0	48.1
	LST v2 ($\epsilon = 2$)	40.1	61.5	43.8	24.2	43.8	52.2
	LST v2 ($\epsilon = 3$)	40.6	61.8	44.2	23.3	44.4	53.0
RetinaNet [106]	LST [97]	**38.7**	58.5	**41.7**	22.2	42.7	**51.2**
	Conv2d	36.5	55.4	39.1	20.4	40.3	48.1
	LST v2 ($\epsilon = 2$)	38.5	58.4	41.2	22.6	42.3	51.1
	LST v2 ($\epsilon = 3$)	**38.7**	**58.8**	41.4	**22.7**	**42.3**	50.8
FCOS [161]	LST [97]	38.8	58.7	**41.5**	22.5	42.4	50.2
	Conv2d	36.6	55.7	38.8	20.7	40.1	47.4
	LST v2 ($\epsilon = 2$)	38.6	58.3	41.1	22.0	42.3	50.3
	LST v2 ($\epsilon = 3$)	**38.9**	**59.1**	**41.5**	**23.0**	**42.5**	**50.5**
Mask R-CNN [57]	LST [97]	**41.3**	62.5	45.0	**25.1**	**45.1**	**54.3**
	Conv2d	38.2	58.8	41.4	21.9	40.9	49.5
	LST v2 ($\epsilon = 2$)	40.8	61.8	44.6	24.2	44.2	53.3
	LST v2 ($\epsilon = 3$)	**41.3**	**62.6**	**45.1**	**25.1**	44.9	53.8
Cascade Mask R-CNN [12]	LST [97]	43.9	**62.6**	47.9	**26.3**	47.4	**57.8**
	Conv2d	41.2	59.4	45.0	23.9	44.2	54.4
	LST v2 ($\epsilon = 2$)	43.6	62.4	47.6	25.3	47.1	57.2
	LST v2 ($\epsilon = 3$)	**44.1**	**62.6**	**48.2**	**26.3**	**47.7**	57.4

evaluation. For each dataset, we use the training set to train all the models and validation set for test.

Table 4.21 and Table 4.22 show the results using LST v2 on MPII dataset and COCO key-points detection dataset, respectively. One can have at least two findings. First, compared to LST, our LST v2 achieves comparable or slightly better performance on both human pose estimation datasets. For example, LST v2 ($\epsilon = 3$) slightly improves mean AP and mean AR of LST [97] by 0.2% on COCO key-points

Table 4.19: Instance segmentation results (%) of LST v2 on MS-COCO validation set.

Detector	Backbone Method	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN [57]	LST [97]	**37.1**	59.3	39.4	**20.9**	**40.5**	**50.8**
	Conv2d	34.7	55.7	37.2	18.3	37.4	47.2
	LST v2 ($\epsilon = 2$)	36.5	58.6	38.9	20.2	39.8	49.5
	LST v2 ($\epsilon = 3$)	**37.1**	**59.4**	**39.5**	**20.9**	**40.5**	50.2
Cascade Mask R-CNN [12]	LST [97]	38.1	59.7	40.9	**21.4**	41.3	**51.9**
	Conv2d	35.9	56.6	38.4	19.4	38.5	49.3
	LST v2 ($\epsilon = 2$)	37.9	**59.8**	40.4	20.6	41.0	51.0
	LST v2 ($\epsilon = 3$)	**38.3**	59.7	**41.2**	21.3	**41.8**	**51.9**

detection datasets for 256×192 input images. Second, compared to Conv2d, LST v2 obtains considerable gain on both datasets. LST v2 ($\epsilon = 2$) outperforms mean AP of Conv2d by 0.38% on MPII dataset, while it saves nearly 45% parameters and computational cost. Meanwhile, it substantially improves the mean AP/AR values of Conv2d by 2.5%/0.6% and 3.8%/1.2% on COCO key-points detection dataset when input size is 256×192 and 384×288, respectively, .

Table 4.20: Semantic segmentation results of LST v2 on PASCAL VOC2012.

Seg. Method	Backbone Method	mIoU	Overall Acc. (%)	Freq. W. Acc. (%)
DeepLab V3 [23]	LST [97]	**0.790**	**94.49**	**90.07**
	Conv2d	0.767	93.89	88.76
	LST v2 ($\epsilon = 2$)	0.782	94.29	89.37
	LST v2 ($\epsilon = 3$)	0.786	94.30	89.40
DeepLab V3+ [24]	LST [97]	**0.792**	**94.58**	**90.17**
	Conv2d	0.771	94.13	89.14
	LST v2 ($\epsilon = 2$)	0.786	94.37	89.50
	LST v2 ($\epsilon = 3$)	0.790	94.52	89.73

Table 4.21: Human pose estimation results (%) of LST v2 on MPII dataset.

Backbone Method	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Mean	Mean@0.1
LST [97]	96.76	**95.35**	**89.42**	**84.00**	**88.56**	84.95	80.63	**89.05**	34.41
Conv2d	96.35	95.33	88.99	83.18	88.42	83.96	79.59	88.53	33.91
LST v2 ($\epsilon = 2$)	**97.00**	**95.35**	89.21	83.57	87.64	**85.45**	**80.85**	88.91	33.98
LST v2 ($\epsilon = 3$)	96.79	**95.35**	89.14	**84.00**	88.49	85.35	80.73	**89.05**	**34.42**

Table 4.22: Human pose estimation results (%) of LST v2 on COCO key-points detection dataset.

Backbone Method	Input Size	AP	$AP_{0.5}$	$AP_{0.75}$	AP_M	AP_L	AR	$AR_{0.5}$	$AR_{0.75}$	AR_M	AR_L
LST [97]	256×192	74.1	92.5	81.5	71.1	78.7	77.0	**93.4**	83.5	73.7	82.0
Conv2d		70.4	88.6	78.3	67.1	77.2	76.3	92.9	83.4	72.1	**82.4**
LST v2 ($\epsilon = 2$)		73.9	92.5	**81.6**	71.0	78.3	76.9	93.3	83.8	73.7	81.7
LST v2 ($\epsilon = 3$)		**74.3**	**92.6**	**81.6**	**71.3**	**78.8**	**77.2**	**93.4**	**83.9**	**74.0**	82.1
LST [97]	384×288	**76.5**	**93.6**	**83.7**	**73.4**	**81.3**	**79.0**	**94.1**	**85.2**	**75.6**	84.2
Conv2d		72.2	89.3	78.9	68.1	79.7	77.6	93.2	83.8	72.8	**84.6**
LST v2 ($\epsilon = 2$)		76.0	92.6	82.5	72.7	80.9	78.8	93.9	84.6	75.3	83.9
LST v2 ($\epsilon = 3$)		76.3	92.6	83.6	73.1	**81.3**	**79.0**	93.8	**85.2**	75.5	84.3

4.4.11 Evaluation on salient object detection

To study LST v2 on salient object detection, we use the same experiment setting and protocal described in Section 3.4.1 and Section 3.4.12. We replace LST [97] with our LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) to construct the backbone model under ResNet-50 architecture. To enable fusion of intermediate features across different stages, LST v2 ($\epsilon = 2$) and LST v2 ($\epsilon = 3$) only offer half and three quarters of channels of Conv2d or LST at the corresponding layers, respectively. This makes it

really challenging when we examine the representation power of the proposed method on salient object detection.

Table 4.23 presents the results of LST v2 on salient object detection. One can have at least two findings. First, both LST v2 ($\epsilon = 2$) and both LST v2 ($\epsilon = 3$) significantly outperform Conv2d in terms of both F-measure and MAE across all the benchmark datasets. Second, compared to LST, both instances of LST v2 show very close performance on DUT-OMRON, HKU-IS and SOD. It is worth noting that LST v2 ($\epsilon = 3$) is even better than LST on three challenging benchmarks under both metrics on ECSSD, PASCAL-S and DUTS.

4.5 Conclusion

In this chapter, we proposed a fast and lightweight transform with HDWConv based on LST-Net to reduce the redundancy of the conventional Conv2d. To produce a compact feature bank, we allow incomplete yet flexible expansion so that the structure of HDWConv can be completely determined before training. High frequency input channels are expanded more times for near-complete expansion, while low frequency ones are expanded fewer times to save cost. In addition, we partition input channels into groups and assigned one or more unique kernels with identical initialization for better representation. Extensive experiments validate that the proposed method can achieve comparable or even higher accuracy than LST-Net while it only requires approximately 40% $\sim$ 55% parameters and computational cost of Conv2d under the same architecture, saving around 20% $\sim$ 40% overhead of LST-Net.

Table 4.23: Salient object detection results of LST v2 under ResNet-50 architecture.

Dataset	Backbone Method	F-measure↑	MAE↓
ECSSD [188]	Conv2d	0.940	0.042
	LST [97]	0.950	0.034
	LST v2 ($\epsilon = 2$)	0.947	0.035
	LST v2 ($\epsilon = 3$)	**0.952**	**0.031**
PASCAL-S [104]	Conv2d	0.863	0.075
	LST [97]	0.876	0.066
	LST v2 ($\epsilon = 2$)	0.872	0.069
	LST v2 ($\epsilon = 3$)	**0.881**	**0.064**
DUT-OMRON [189]	Conv2d	0.830	0.055
	LST [97]	**0.834**	0.052
	LST v2 ($\epsilon = 2$)	0.830	0.055
	LST v2 ($\epsilon = 3$)	0.830	**0.051**
HKU-IS [92]	Conv2d	0.934	0.032
	LST [97]	0.941	**0.028**
	LST v2 ($\epsilon = 2$)	0.935	0.030
	LST v2 ($\epsilon = 3$)	**0.943**	**0.028**
SOD [130]	Conv2d	0.867	0.100
	LST [97]	0.875	**0.093**
	LST v2 ($\epsilon = 2$)	0.875	0.097
	LST v2 ($\epsilon = 3$)	**0.876**	0.094
DUTS [166]	Conv2d	0.886	0.040
	LST [97]	0.895	0.035
	LST v2 ($\epsilon = 2$)	0.886	0.039
	LST v2 ($\epsilon = 3$)	**0.897**	**0.034**

Chapter 5

Application of LST to Adversarial Attacks

2D convolution (Conv2d) layers are key components of a convolutional neural network (CNN). It is expected that Conv2d can be robust to various types of image corruptions in real-world as well as those manually designed adversarial attacks. Though many efforts have been devoted to improve the robustness of the learned CNN model, seldomly works have been done to study the robustness of Conv2d. Inspired by learnable sparse transform (LST) that learns to convert the CNN features into a compact and sparse latent space, we design a robust layer with multiple kernels as an alternative of Conv2d, namely RConv-MK, to empower LST with much higher robustness to image corruptions and adversarial attacks. RConv-MK employs a set of kernels of different size and flexibly applies them to the input features of different frequencies. It enlarges the receptive fields for low frequency features and saves the overhead for sparse high frequency features. A normalized soft thresholding (NST) operator is introduced to adaptively remove the noise and trivial features in the relevant feature domain. Extensive experiments are performed to validate the effectiveness of RConv-MK under popular CNN architectures for adversarial samples as well as clean input images.

5.1 Introduction

Deep convolutional neural networks (CNNs) have shown their power in a wide range of computer vision tasks, especially in image recognition [35, 209]. Despite the great success, it has been found that a well performed CNN model can be out of work when handling images with various types of corruptions in real world [63]. In addition, a CNN model can be easily fooled by deliberately designed adversarial samples with subtle and unperceivable perturbations to human eyes [157]. Therefore, it is a very practical issue to improve the robustness of CNN models against image corruption and adversarial attacks.

To improve the robustness of CNN models for image recognition with corruptions, most of existing methods aim to improve the quality of input data to CNNs. Based on the priors of image denoising, some pioneer works [44, 65, 187] attempt to transform the input data from spatial (pixel) domain into certain frequency domain for easier noise removal before they are fed into the networks for recognition tasks. Though CNNs adapted to a specific type of corruption can have better robustness, they may be fragile to out-of-box corruptions. Besides, implementation of these methods requires manual setting for each task, which is not practical to use. For example, one needs to manually adjust the noise level to find a good balance between noise removal and image cue preservation. For adversarial attacks, many defense methods [122, 21, 201, 200] have been developed to generate adversarial samples for training robust CNN models. Almost all of them view the CNN model as a "black-box" and focus on the adversarial sample generation process. As a result, it is hard to approach the root cause of poor performance of CNNs under adversarial attacks. Meanwhile, it provides little insight to the development of robust CNN architectures.

Though different methods have been developed to respectively address the problems of corrupted image recognition and adversarial attack, to the best of our knowl-

edge, almost all of them ignored an important perspective: can we and how to improve the robustness of 2D convolution (Conv2d), which is the key component of CNN, for more robust image recognition and anti-attack performance? In this chapter, we make an effort along this line and develop a robust Conv2d layer under existing CNN architectures. Our work is mainly inspired by LST-Net [97] (see Chapter 3), which learns a set of channel and spatial transforms to convert the CNN features into a compact and sparse latent space. The transforms are initialized by discrete cosine transform (DCT), and a soft thresholding (ST) is applied to remove noises and trivial features in the learned domain. Though LST-Net is effective and efficient, it can still be improved from two aspects. First, it is noticed that the output features of the channel transform are basically organized by frequency, which motivates use to develop distinct strategies to match and exploit this property in the following spatial transform. Second, the threshold of ST in LST-Net is fixed for the whole CNN model, which is less accurate and flexible to process the complex input with various corruptions.

We design a robust Conv2d layer with multiple kernels, denoted by RConv-MK, to mitigate the above issues. To make better use of the output of channel transform, a set of kernels of different sizes are adopted in spatial transform. Large kernels can be easily applied to handle low frequency signals and small kernels to high frequency ones. With large kernels, one can avoid misclassifying low frequency signals due to limited receptive field. Meanwhile, small kernels can reduce much the overhead as high frequency signals are usually sparse. By sequentially partitioning the input into suitable groups, RConv-MK can obtain nearly the same overhead as LST-Net at the cost of negligible extra parameters. In addition, considering that the corruption levels always vary from one sample to another, we propose a normalized soft thresholding (NST) operator to effectively control unknown corruption level of each sample. As a result, RConv-MK is more robust than conventional Conv2d as well as LST bot-

tleneck, and the robustness of the entire CNN model is accordingly enhanced. Our extensive experiments on clean images, corrupted images and adversarial samples validate the effectiveness and efficiency of RConv-MK.

5.2 Related Work

5.2.1 Image recognition on corrupted images

In real world, clean images may be easily corrupted due to many reasons, such as improper light condition, defects of imaging devices, bad lighting, defocus blur, *etc*. While a CNN model may perform well on clean images, it can be out of work when handling corrupted photos. The existing image restoration methods [202, 47, 42] are basically developed to improve the image quality according to the criteria such as PSNR or SSIM [173] but not the accuracy in image recognition. Therefore, they are not suitable to be directly used for corrupted image recognition.

It's intuitive to suppress noises of corrupted images for reliable recognition. Algorithms [44, 65, 187] have been developed to convert the input data from spatial domain into certain frequency domain before they are fed into CNNs for two main reasons. First, noises are easy to identify and suppress in frequency domain. Second, it is cost-effective to conduct domain transform and/or its inverted operation by performing classic convolution. Franzen [44] converted gray-scale images into DCT domain and fed the responses into a 2-layer NLP model for classification. Hossain et al. [65] inserted a DCT module before a pre-trained VGG-16 model to fine-tune the model on dataset with various types of common corruptions. Xu et al. [187] converted the color space from RGB space into YCbCr, separately computed DCT for each channel, and aggregated the responses along channel dimension as model's input. However, the robustness of those models may be limited to the specific frequency domain, and the models may suffer from the generalization problem to unseen

corruptions. Meanwhile, these methods require manual adjustment of noise level during domain transform, where one needs to balance a trade-off between noise removal and image cue preservation. Li et al. [97] proposed a novel bottleneck to transform its input features to a compact and sparse domain by using learnable kernels, where soft thresholding (ST) is adopted for noise removal. In this chapter, we use multiple kernels of different sizes w.r.t. frequency in related feature domain. In addition, we improve the model robustness by adaptively estimating the relative corruption level of each sample.

5.2.2 Adversarial attacks and defenses

Generally, adversarial attacks refer to using deliberately designing inputs (*a.k.a.* adversarial examples) to fool a trained network model and force it to produce wrong outputs. The purpose of adversarial attacks may vary under different scenarios [17]. In this chapter, we focus on image recognition and the goal of adversarial attacks is to cause misclassification.

Adversarial attack algorithms can be roughly classified into two categories based on whether gradient is adopted. Optimization-based attacks are by far the most powerful methods. Given an input image and its associated ground truth label, these methods generate the adversarial samples by computing the gradients according to the CNN architecture and the pre-defined loss function, such as cross-entropy loss C&W[19], *etc*. Meanwhile, ℓ_1 [25], ℓ_2 [157, 19] and ℓ_∞ [122, 18] distortion metrics are commonly used to measure the budget of adversarial examples, while some emerging work [152] has been reported to investigate the possibility of ℓ_0 distortion. In comparison, gradient-free attacking methods are developed for the cases that the network architecture is unavailable. Some representative works can be found in [26, 163, 79, 8, 115].

To defend against adversarial attacks, adversarial training [4] is one of the most

popular and natural choices. It augments the training data by generating adversarial examples with certain attacking methods. Though the adversarially trained model can deal with unseen data, the model may still fail when facing adversarial samples generated by other attacking methods. This is because one model can hardly cover the entire input space by training with a limited number of searching steps. To narrow the gap, Deng *et al.* [36] modelled the potential adversarial examples from the perspective of distribution for more effective training. Other defense algorithms focus on designing better objective functions. Chen *et al.* [21] proposed a novel loss function to neutralize the probability of wrong predictions and maximize the probability of right predictions. However, these defense methods tend to ignore the role of network architecture.

Different from the above defense methods, we propose to enhance the robustness of CNN architecture (more specifically, the 2D convolution layer) to defend against adversarial attacks. Our method is complementary to current works and it can be readily used to improve the anti-attack performance of existing methods.

5.2.3 Receptive field of CNNs

Receptive field refers to the region in the input space that a CNN can see at a specific layer, determined by the kernel size of the preceding layers. Most existing architectures are manually designed with fixed kernel size at each layer, such as AlexNet [90], VGG [151], ResNet [59], *etc.* These architectures may not work well at capturing features that need large receptive field.

To mitigate this issue, dilated convolution [195] introduces a hyper-parameter, called dilation rate, to control how much the receptive field is expanded by inserting fixed zeros into the spatial dimension of its kernels. However, tuning dilation rate of each layer is time-consuming and requires additional knowledge and expertise. Models from the Inception family [155, 81, 156, 154] are built with bottlenecks of

multiple branches at each layer. Each branch is configured with kernels of different sizes to fit features in various receptive fields. Unfortunately, it is troublesome to match the features with their corresponding kernels when an Inception model is trained from scratch. Consequently, Inception modules are likely to lose cues due to improper matches. In contrast, our proposed method in this chapter can locate features of different frequencies, which can better process different features using kernels of different sizes.

5.2.4 Normalization layers

A normalization layer plays a critical role in the training of a CNN, as it effectively neutralizes the internal covariate shift [81] between input distribution and output distribution. To the best of our knowledge, Batch Normalization (BN) [81] is the first work to mitigate this issue. A BN layer normalizes the whole batch for every single activation, where statistics, such as mean values and standard deviations, are collected for each unit across the batch during mini-batch based training. Unlike BN, a Layer Normalization (LN) [5] layer proposes to normalize all the activations of a single layer of a batch along the channel dimension, where it collects statistics from every unit within the layer. Besides, Instance Normalization (IN) [164] conducts each sample a BN-like computation, where a sample refers to an unit of the space spanned along the batch dimension and the channel dimension. In addition, Group Normalization (GN) [182] improves BN by partitioning channels into groups and computing mean values and standard deviations for each group. As GN is independent of batch size, its accuracy is discovered stable in a wide range of batch sizes.

None of the aforementioned normalization layer is able to handle input with different scales of noise levels in adversarial attack while addressing internal covariate shift. Thus, a CNN model constructed with a single type of normalization layer,

usually BN, are vulnerable to adversarial attacks. In this chapter, we propose normalized soft-thresholding (NST) to simultaneously mitigate input with different noise scales as well as close the discrepancy between input distribution and output distribution for effective removal of noise and redundancies. Besides, our NST is easy to implement as it only leverages two existing methods, including BN and LN.

5.3 Proposed Method

5.3.1 Problem formulation

Denote by $x \in \mathcal{R}^{H\times W\times C}$ the input natural image in the spatial domain and $y = 1, 2, \ldots, m$ its associated class labels in a classification task, where m is the number of classes. A CNN model of L layers can be regarded as a sequence of functions as follows,

$$\hat{y} = f \circ x = f^{(L)} \circ f^{(L-1)} \circ \ldots \circ f^{(2)} \circ f^{(1)} \circ x, \tag{5.1}$$

where $f^{(i)}$ is the function of the i-th layer of the CNN model with its associated parameters $\theta^{(i)}$, $i = 1, 2, \ldots, L$. We further denote the input and the output of $f^{(i)}$ by $x^{(i)} \in \mathcal{R}^{H_{in}^{(i)}\times W_{in}^{(i)}\times C_{in}^{(i)}}$ and $y^{(i)} \in \mathcal{R}^{H_{out}^{(i)}\times W_{out}^{(i)}\times C_{out}^{(i)}}$, respectively. Let $\mathcal{J}_f(y, \hat{y})$ be a loss function measuring the distance between y and $\hat{y}$, e.g. the cross-entropy loss.

Generally, a corrupted sample x' can be modelled as

$$x' = x + \eta, \tag{5.2}$$

where η can be noises caused by some corruptions in the real world, or it can be the perturbation deliberately computed by a specific adversarial algorithm, aiming to enforce the noisy sample resulting in a wrong classification $y' = f(x') \neq y$.

In this chapter, we do not assume any specific distribution on η, but assume that η is a random and high frequency signal. This assumption generally holds for the common image corruptions. When η is generated by adversarial attack, the

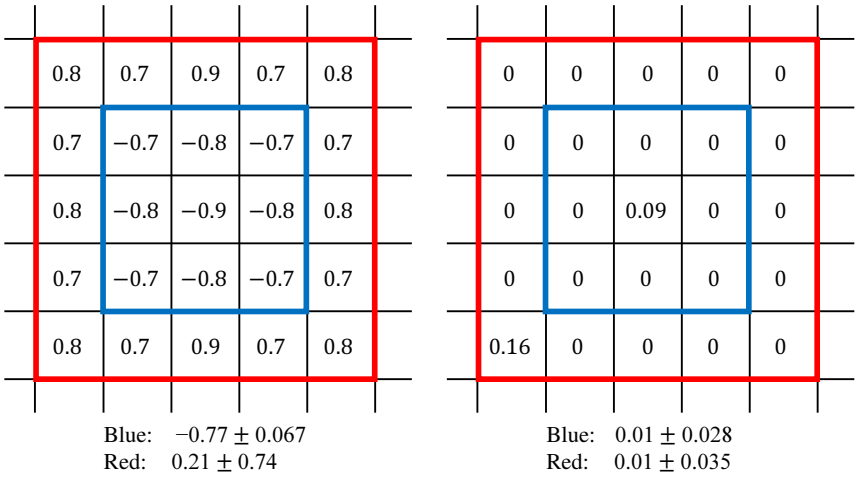
assumption also holds well since the perturbation behaves like noise. A robust CNN model is expected to consistently make correct prediction for either the clean input image x or its corrupted counterparts x'.

5.3.2 Spatial transform with multiple kernels

Given the kernel size $k \times k$ and expansion rate a for DWConv, T_s of LST repeatedly applies a convolutional kernel $\theta_s^{(i)} \in \mathcal{R}^{a^2 \times 1 \times k \times k}$ to each channel of $x_s^{(i)}$. It can be observed that the learned weights of T_c in LST-Net maintain two important properties (which are also possessed by our RConv-MK). First, the transformed features are structured, where the low frequency signals are placed at one end while the high frequency ones are located at the other end in the channel dimension. Second, low frequency features are dense while high frequency features are sparse. Without loss of generality, in the remaining of this chapter, we assume that features of $x_s^{(i)}$ (or $y_c^{(i)}$) are arranged from low to high frequencies in the channel dimension.

Obviously, the existing implementation of T_s ignores the properties of T_c. When we deal with the first few channels, the fixed kernel size (usually 3×3 in most modern architectures) may be too small to identify the genuine low frequency signals. Thus, some high frequency signals will be misclassified as low frequency ones due to limited receptive field. Meanwhile, when we compute the last few channels, the fixed kernel size may not be suitable for sparse high frequency signals.

In Figure 5.1, we use two toy samples to illustrate the effect of using different kernel sizes. When we use an undersized kernel, such as the blue 3×3 kernel in Figure 5.1(a), due to its limited receptive field, the result is very different (by mean value and standard deviation) from the one extracted by a large kernel like the red 5×5 one. In Figure 5.1(b), one can see that both large and small kernels can produce very close results for high frequency signals. This suggests use of small kernels for high frequency signals to reduce redundancies.



(a) Low frequency signals (b) High frequency signals

Figure 5.1: Illustration of the effect (mean±std) of using different kernel sizes.

In this chapter, we are committed to improving the design of T_s in LST. Figure 5.2 illustrates the implementation of T_s in this chapter. By providing kernels of suitable sizes for signals of different frequencies, we can improve T_s in LST to produce better features for low frequency signals and accelerate computation for high frequency ones. The proposed RConv-MK adopts a set of m-kernels of different sizes and sort them by their kernel size in a descending order. The associated weights can be written as $\theta_s^{(i)} = \{\theta_{s,j}^{(i)} | \theta_{s,j}^{(i)} \in \mathcal{R}^{a^2 \times 1 \times k_j \times k_j}, j = 1, \ldots, m\}$, satisfying $\forall p > q, k_p > k_q \geqslant 1$, and $\exists j, k_j = k$. Accordingly, we partition $x_s^{(i)}$ into m groups along the channel dimension to have $x_{s,1}^{(i)}, \ldots, x_{s,m}^{(i)}$, where $x_{s,j}^{(i)} \in \mathcal{R}^{H_{in}^{(i)} \times W_{in}^{(i)} \times C_{s,j}^{(i)}}$, $\forall j = 1, \ldots, m$. Obviously, $\sum_{j=1}^{m} C_{s,j}^{(i)} = C_s^{(i)}$. T_s of RConv-MK applies each $\theta_{s,j}^{(i)}$ to its related group of input channels $x_{s,j}^{(i)}$, $\forall j = 1, \ldots, m$ so that low frequency signals are assigned large kernels while high frequency ones are given small kernels. In this way, we can make good use of signals of different frequencies in relation to the properties of the feature domain. Finally, T_s ends up with concatenating results of all m-groups along the channel dimension for noise removal.

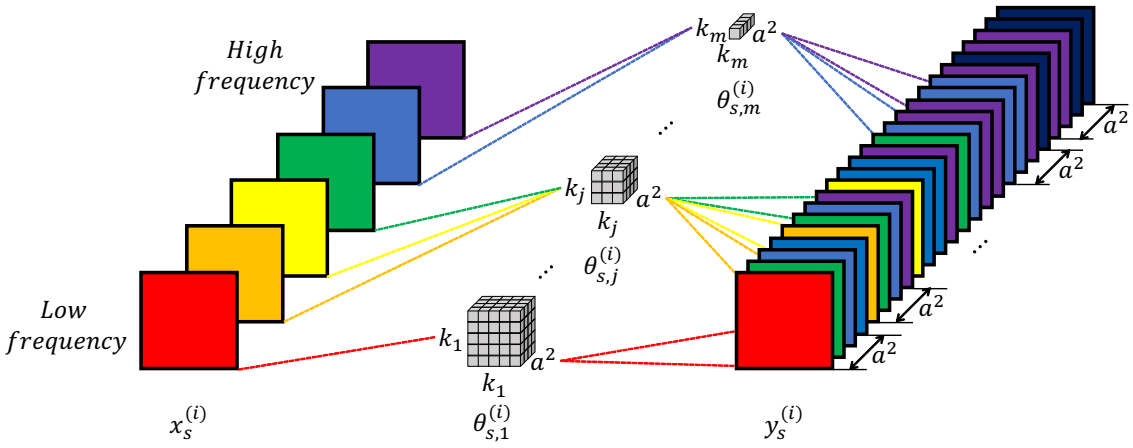


Figure 5.2: Illustration of spatial transform T_s of RConv-MK at the i-th layer. Channels of the input $x_s^{(i)}$ and the output $y_s^{(i)}$ of T_s are highlighted in prism colorset, where red means low frequency signals with high amplitude while purple means high frequency signals with low amplitude. A number of m kernels are used in T_s, including $\theta_{s,1}^{(i)}, \ldots, \theta_{s,m}^{(i)}$. Each kernel has a^2 channels, producing a^2 output channels for each input channel. Kernels of large window size like $\theta_{s,1}^{(i)}$ convolve with low frequency signals, while kernels of small window size like $\theta_{s,m}^{(i)}$ are used for high frequency signals.

5.3.3 Normalized soft thresholding

In LST-Net, soft-thresholding is used to remove noise and trivial features. However, the threshold τ is determined manually based on the noise level in the corresponding feature domain. Mismatch of τ and features may cause performance drop. A large τ value may cut down useful cues, while noises are easy to survive a small τ value. Actually, there are dozens or hundreds of τ to be set when a CNN goes deeper. And it is impossible to separately tune each of them. More importantly, the noise level of x' may vary dramatically from one sample to another and it is hard to determine the threshold in adversarial attack. It is highly desired to develop a more adaptive thresholding scheme to remove noise and trivial features for robust convolution. We develop a normalized soft thresholding (NST) method to this end. Mathematically, NST at first normalizes each sample X_i $(i = 1, \ldots, N)$ from an $N-$sized mini-batch

X as

$$X_{LN,i} = (X_i - \mu(X_i))/\sigma(X_i) \tag{5.3}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ compute the mean and the standard deviation, respectively. In this way, corrupted samples at different noise levels are scaled to nearly the same level in an adaptive manner, so that the normalized corrupted samples are expected to approach the distribution of their corresponding clean samples. In this sense, we are allowed to further mitigate internal corvariate shift of the normalized mini-batch X_{LN} by conducting batch normalization (BN) [81] over it to obtain X_{BN}. The above normalization part of NST can be easily implemented with a sequence of non-parametric layer normalization (LN) [5] plus a standard BN.

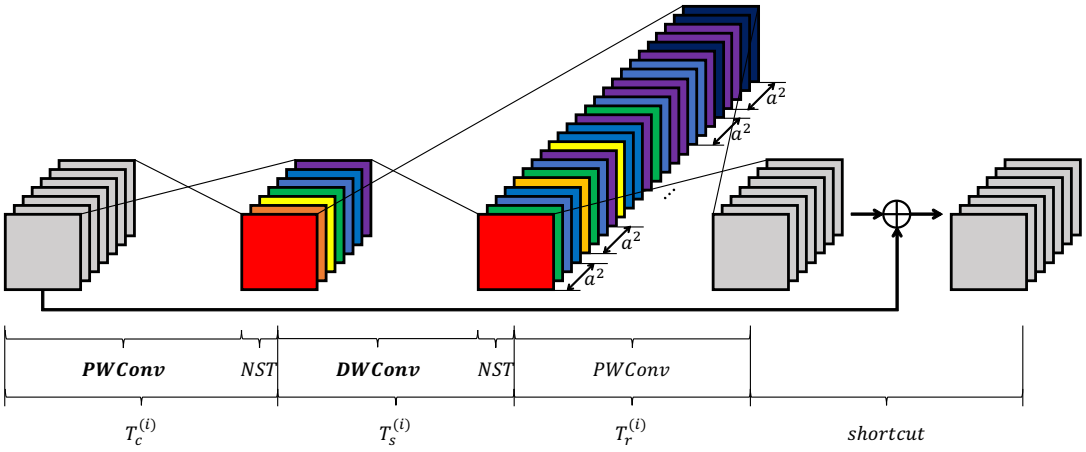
Finally, noises in the normalized feature domain can be suppressed by the classic ST as

$$Y_{NST} = \begin{cases} sgn(X_{BN})(|X_{BN}| - \tau), & |X_{BN}| \geqslant \tau, \\ 0, & otherwise. \end{cases} \tag{5.4}$$

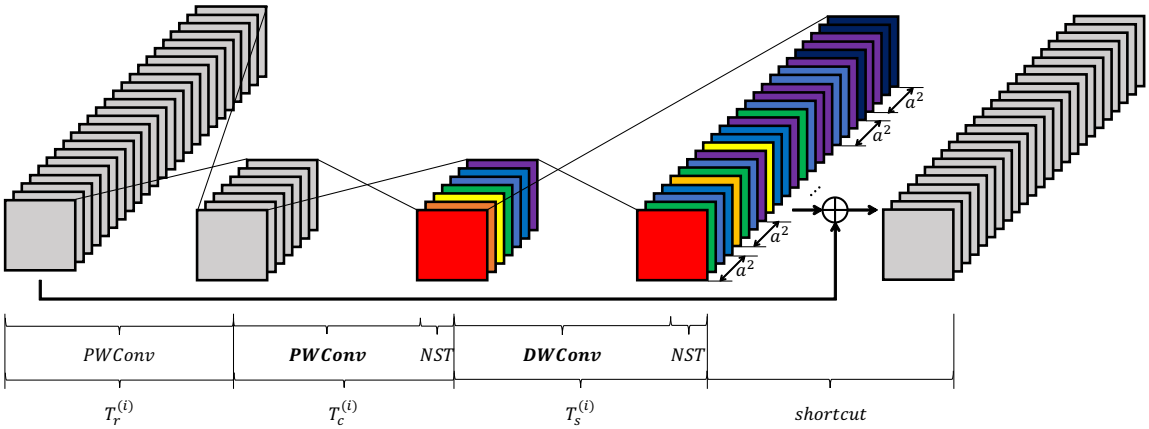
where τ is the threshold and Y_{NST} is the NST output of X. In this chapter, with the introduction of normalization, in NST, we can easily set $\tau = 10^{-4}$ in all experiments.

5.3.4 RConv-MK

By applying spatial transform with multiple kernels and NST to LST bottleneck, we are able to construct a robust convolutional layer, namely RConv-MK, for the application of adversarial attack. It is worthwhile noting that NST can be only found in both channel transform T_c and spatial transform T_s of the proposed RConv-MK, where each ST operator in an LST bottleneck is replaced by an NST operator. According to its design principle, the architecture of resize transform T_r remains unchanged to what it is for an LST bottleneck, *i.e.*, a sequence of Conv1 $\times$ 1, BN and ReLU.



(a) RConv-MK-I



(b) RConv-MK-II

Figure 5.3: Illustration of two types of RConv-MK.

Figure 5.3 illustrates two types of RConv-MK built for LST-I and LST-II (see Section 3.3.3), respectively. It can be observed that none of the changes in RConv-MK is exposed as an outside interface. That is to say, when we are building CNNs on adversarial attack, it needs nothing but shifting from LST to RConv-MK, making it quite convenient in practice.

5.3.5 Implementation details and complexity analysis

Here, we present some implementation details of the proposed RConv-MK. To be comparable with LST, we expect that RConv-MK has almost the same overhead at

the cost of negligible number of extra parameters for the same setting of k and a. We set $m = 3$ and fix $k_1 = k + 2$, $k_2 = k$ and $k_3 = 1$ in this chapter. Besides, we set $C_{s,2}^{(i)}$ to half of $C_s^{(i)}$ so that the kernel of size $k \times k$ will be computed with the majority of the input features. As of DWConv, when we have the same input shape and stride, the overhead is in proportion to the kernel size. Therefore, we have $C_{s,j}^{(i)} \propto k_j^{-2}$, $j = 1, \ldots, m$. With the above settings, we can conduct a grid search to specify the proportion for some popular kernel sizes. In this chapter, we set $C_{s,1}^{(i)} : C_{s,2}^{(i)} : C_{s,3}^{(i)} = 1 : 3 : 2$ when $k = 3$, and $C_{s,1}^{(i)} : C_{s,2}^{(i)} : C_{s,3}^{(i)} = 1 : 2 : 1$ when $k = 5$.

Compared to LST, the number of extra parameters of RConv-MK can be determined by $\{\theta_{s,j}^{(i)} | \theta_{s,j}^{(i)} \in \mathcal{R}^{a^2 \times 1 \times k_j \times k_j}, j = 1, \ldots, m, k_j \neq k\}$. In modern architectures, $C_{in}^{(i)}, C_{out}^{(i)} \gg a, k$ and the total number of parameters are dominated by T_r, proportional to $C_{in}^{(i)} \times C_{out}^{(i)}$. Weights of T_c and T_r discussed in this chapter are just a tiny fraction. Take the ResNet architecture [59] as an example. Given $a = 2$, there are only $a^2 \times (k_1^2 + k_3^2) = 2^2 \times (5^2 + 1^2) = 104$ extra parameters in a RConv-MK. In contrast, $C_{in}^{(i)}$ and $C_{out}^{(i)}$ vary from 64 to 512. Approximately, the number of extra parameters only occupies $0.01\% \sim 1\%$ of the total number of parameters in a RConv-MK.

5.4 Experiments

In this section, we first discuss our experiment setup and datasets. Then, we evaluate the performance of the proposed RConv-MK with clean images as well as its robustness to common types of corruptions and adversarial attacks in order. Finally, we conduct ablation study on the number of multiple kernels and the setting of channel split in a RConv-MK.

Table 5.1: Methods for comparison in this chapter.

Method	Use aux. branch	Noise removal method	Receptive field	Low freq. kernel size	High freq. kernel size
Conv2d	No	N.A.	Uniform	N.A.	N.A.
Conv2d-MK	No	N.A.	Varied	N.A.	N.A.
Conv2d+SE	Yes	N.A.	Uniform	N.A.	N.A.
Conv2d+CBAM	Yes	N.A.	Uniform	N.A.	N.A.
LST	No	ST	Uniform	N.A.	N.A.
RConv-UK	No	NST	Uniform	N.A.	N.A.
RConv-RMK	No	NST	Varied	Small	Large
RConv-DMK	No	N.A.	Varied	Large	Small
RConv-LMK	No	LN	Varied	Large	Small
RConv-SMK	No	ST	Varied	Large	Small
RConv-MK	No	NST	Varied	Large	Small

5.4.1 Experiment setup and datasets

All experiments are conducted on a 10-way GPU server equipped with dual Intel Xeon Gold 6248R@3.0GHz CPUs, 256G DDR4 2933MHz RAM and NVIDIA Quadro RTX 8000 GPU cards and Samsung 860 EVO SSDs. For implementation, we use PyTorch [133] compatible with CUDA 10.2 and cuDNN v7.6.5 on Ubuntu 18.04.

Methods for comparison. Table 5.1 lists key attributes of competing methods. The conventional Conv2d is set as the baseline. Similar to RConv-MK, Conv2d-MK splits input along channel dimension into m-groups and compute Conv2d of different kernel sizes for each group and then concatenates the results of each group. We also study two popular attention modules, *i.e.*, SE [71] and CBAM [178]. In addition to LST [97], RConv-MK and its variants are compared. In terms of arrangement of kernels, RConv-UK adopts a uniform kernel in T_s, and RConv-RMK reverses the order of kernels. For noise removal methods, RConv-LMK replaces NST by LN; RConv-SMK substitutes NST with ST; RConv-DMK removes all NST operators.

The datasets. In this chapter, the ImageNet [35] is employed to evaluate the performance of each method on clean images. The ImageNet-C [63] is employed to evaluate the robustness of each method on images with common corruptions. CIFAR-10/100 [89] are used for the evaluation under white-box adversarial attacks. We closely follow standard experimental settings for fair comparison. Details can be found in Section 3.4.1.

5.4.2 Evaluation on adversarial attacks

We evaluate the robustness of RConv-MK to adversarial attacks on CIFAR-10/100. We compare models built with RConv-MK and other methods under two architectures, *i.e.*, ResNet-18 and WRN34-10. We conducted adversarial training of each model on each dataset under the ℓ_∞ PGD attack for 100 epochs using common hyper-parameter settings. Specifically, the perturbation size $\epsilon = 8/255$, step size $\eta = 2/255$, number of steps is 10. Learning rate started at 0.1 and was reduced by a factor of 10 after 75, 90 and 100 epochs, respectively. We fixed batch size as 128 and weight decay as 0.0002. We tested each trained model under untargeted white-box attacks of four algorithms, including FGSM [50], PGD [122], FFGSM [177] and ODI [160]. We used the official implementation of ODI and advertorch [37] of the rest.

Table 5.2 presents the accuracy obtained by different methods. One can have the following findings. First, the proposed RConv-MK outperforms all its competitors under adversarial attacks. Second, the attention modules, including Conv2d+SE and Conv2d+CBAM, have almost the same performance as the baseline under various untargeted white-box attacks on both datasets. We note that according to the definition, both attention modules show more interest in local patterns while they suppress trivial features for image recognition. Although such kind of mechanism is helpful to the recognition of clean images, it may hurt the backbone model under adversarial attacks because the corrupted local patterns have large chance to

impose uncorrected excitation to the target features. Third, for noise removal methods, RConv-MK (NST) > RConv-LMK (LN) > RConv-SMK (ST) > RConv-DMK (none) on CIFAR-10 and RConv-MK (NST) > RConv-SMK (ST) > RConv-DMK (none) > RConv-LMK (LN) on CIFAR-100. ST improves robustness to adversarial attacks as it actually plays a role of gradient mask under adversarial attacks. LN shows competitive performance on CIFAR-10 but it performs poorly on CIFAR-100. In our view, this may be caused by the over-fitting problem of LN in adversarial training. With the increase of categories, the decision boundaries are expected to be less smooth in the unit space shaped by LN. Therefore, the model becomes vulnerable to unseen adversarial samples during test. Fourth, the arrangement of multiple kernels also matters in adversarial attacks. We can see that RConv-MK (normal order) > RConv-UK (uniform kernel) > RConv-RMK (reversed order). Fifth, Conv2d-MK always obtains worse results than the baseline due to its poor structure in spatial domain for channel split and concatenation. In contrast, our RConv-MK improves RConv-UK under all attacks as the channel operations are conducted in a well structured domain.

5.4.3 Evaluation on images with corruptions

We study the robustness of RConv-MK to common types of corruption on ImageNet-C [63], which contains 19 distinct corruptions. The mean corruption error (mCE) is used as the criteria (the lower the better). We construct CNNs under ResNet-18 and ResNet-50.

Table 5.7 and Table 5.8 show the best top-1/5 error rates on clean ImageNet and the corresponding mCE values on ImageNet-C under ResNet-18 and ResNet-50, respectively. One can have at least four findings. First, RConv-MK obtains lower mCE than its competitors of the same depth. It significantly reduces the mCE of the baseline by 6.55% (18-layer) / 9.10% (50-layer). Second, in terms of

Table 5.2: Results (robust accuracy, %) by different methods under untargeted white-box attacks on CIFAR-10/100.

Arch.	Method	FFGSM	FGSM	PGD	ODI
ResNet-18	Conv2d	59.40/30.05	55.36/27.06	46.30/22.22	44.36/20.92
	Conv2d+SE	59.70/30.41	55.99/27.26	46.34/22.53	44.32/21.42
	Conv2d+CBAM	59.84/30.22	56.28/27.28	46.71/22.37	44.59/21.27
	Conv2d-MK	57.54/30.39	53.86/27.40	45.16/21.98	43.15/21.98
	LST	61.74/32.67	58.01/29.53	49.12/23.86	46.88/22.23
	RConv-LMK	62.64/32.08	58.93/29.40	49.62/24.36	47.64/22.83
	RConv-SMK	61.93/33.21	58.19/30.29	49.65/24.98	48.50/23.96
	RConv-DMK	61.44/32.14	57.64/28.68	48.22/23.12	46.31/21.87
	RConv-RMK	62.15/32.71	58.43/29.63	49.57/24.28	47.80/23.00
	RConv-UK	62.53/33.50	59.09/30.53	50.02/25.02	48.02/23.60
	RConv-MK	**62.68/34.14**	**59.25/31.41**	**50.70/25.81**	**48.85/24.22**
WRN34-10	Conv2d	60.78/32.15	57.31/29.22	47.05/24.00	45.94/22.85
	Conv2d+SE	60.75/32.04	56.70/28.98	46.51/23.34	45.36/22.20
	Conv2d+CBAM	60.49/32.35	56.68/29.62	46.89/24.05	45.62/22.84
	Conv2d-MK	60.87/31.12	57.63/28.29	47.04/22.94	45.64/21.56
	LST	62.80/34.08	59.02/30.25	50.46/24.42	48.21/24.37
	RConv-LMK	64.06/32.48	60.16/30.12	50.92/24.90	49.19/23.81
	RConv-SMK	64.18/33.91	60.18/30.82	50.95/25.42	50.53/24.94
	RConv-DMK	62.32/33.84	57.71/29.90	49.60/25.06	48.08/24.10
	RConv-RMK	63.60/33.66	58.56/30.04	50.24/25.39	49.23/24.93
	RConv-UK	64.05/34.19	59.86/31.07	51.38/25.51	50.29/25.03
	RConv-MK	**64.55/34.55**	**60.67/31.50**	**52.64/26.63**	**51.05/25.39**

noise removal methods, RConv-MK (NST) > RConv-LMK (LN) or RConv-SMK (ST) > RConv-DMK (none). This shows that both LN and ST improves the model robustness to common corruptions, while the rational integration of them as NST can further produce more robust results. Third, when it comes to the arrangement of multiple kernels, RConv-MK (normal order) > RConv-UK (uniform kernel) > RConv-RMK (reversed order). This suggests that signals of different frequencies are sensible to the kernel size. Mismatches make it even worse than the use of a uniform kernel. Fourth, it is critical to group and concatenate channels for multiple kernels in

Table 5.3: Comparison of clean image recognition on ImageNet and robustness to common corruptions on ImageNet-C.

m	kernel sizes	Proportion	Top-1/5 E. R. (%, I)	mCE (I-C)
2	1×1, 5×5	2:1	27.29/9.13	80.19
3 (default)	1×1, 3×3, 5×5	2:3:1	**26.26/8.48**	**78.74**
4	1×1, 3×3, 5×5, 7×7	7:9:1:1	26.64/8.69	79.62

Table 5.4: Comparison of m (robust accuracy, %) under untargeted white-box attacks on CIFAR-10/100.

m	kernel sizes	Proportion	FFGSM	FGSM	PGD	ODI
2	1×1, 5×5	2:1	61.71/32.94	58.14/30.13	50.03/24.88	48.20/23.75
3 (default)	1×1, 3×3, 5×5	2:3:1	**62.68/34.14**	**59.25/31.41**	**50.70/25.81**	**48.85/24.22**
4	1×1, 3×3, 5×5, 7×7	7:9:1:1	62.39/33.41	58.84/30.29	50.28/25.06	48.55/23.81

Table 5.5: Comparison of clean image recognition on ImageNet and robustness to common corruptions on ImageNet-C.

Proportion ($1 \times 1 : 3 \times 3 : 5 \times 5$)	Dominant kernel size	Top-1/5 E. R. (%, I)	mCE (I-C)
2:1:1	1×1	26.59/8.75	79.72
2:2:1	1×1, 3×3	26.46/8.64	79.43
2:3:1 (default)	3×3	**26.26/8.48**	**78.74**
2:4:1	3×3	26.31/8.50	78.96
2:5:1	3×3	26.37/8.52	79.15
N.A. (RConv-UK)	3×3	26.42/8.53	79.38

Table 5.6: Robust accuracy (%) of different kernel proportions ($1 \times 1 : 3 \times 3 : 5 \times 5$) under untargeted white-box attacks on CIFAR-10/100.

Proportion	Dominant kernel size	FFGSM	FGSM	PGD	ODI
2:1:1	1×1	62.44/33.15	58.92/30.39	50.21/25.36	48.49/23.75
2:2:1	1×1, 3×3	62.54/33.62	59.07/30.99	50.48/25.64	48.63/24.10
2:3:1 (default)	3×3	**62.68/34.14**	**59.25/31.41**	**50.70/25.81**	**48.85/24.22**
2:4:1	3×3	62.53/33.74	59.19/31.19	50.59/25.73	48.77/24.09
2:5:1	3×3	62.52/33.60	59.10/30.98	50.42/25.58	48.68/23.93
N.A. (RConv-UK)	3×3	62.53/33.50	59.09/30.53	50.02/25.02	48.02/23.60

frequency domain. We see that RConv-MK > RConv-UK in frequency domain while Conv2d-MK < Conv2d because signals in the spatial domain are not well structured.

Table 5.7: Comparison of robustness to common corruptions under ResNet-18 architecture on ImageNet-C.

Method	Top-1/5 E. R. (%)	mCE
Conv2d	30.24/10.92	85.29
Conv2d+SE	29.41/10.22	83.97
Conv2d+CBAM	29.31/10.17	84.97
Conv2d-MK	37.51/15.84	94.98
LST	26.55/8.59	79.89
RConv-LMK	27.33/8.98	80.48
RConv-SMK	27.40/9.04	80.81
RConv-DMK	27.03/8.81	80.91
RConv-RMK	26.84/8.69	79.99
RConv-UK	26.42/8.53	79.38
RConv-MK	**26.26/8.48**	**78.74**

Table 5.8: Comparison of robustness to common corruptions under ResNet-50 architecture on ImageNet-C.

Method	Top-1/5 E. R. (%)	mCE
Conv2d	23.85/7.13	77.01
Conv2d+SE	23.14/6.70	74.47
Conv2d+CBAM	22.98/6.68	72.56
Conv2d-MK	24.96/7.51	77.17
LST	22.78/6.66	70.54
RConv-LMK	22.76/7.05	70.34
RConv-SMK	22.98/6.64	70.80
RConv-DMK	23.31/6.88	70.93
RConv-RMK	23.10/6.80	70.81
RConv-UK	22.59/6.58	69.79
RConv-MK	**22.22/6.32**	**67.91**

5.4.4 Evaluation on clean images

In addition to adversarial attacks, it is also important to investigate whether RConv-MK is effective to clean images. To this end, we study the performance of our method on some popular visual recognition tasks, including image recognition, object detection, instance segmentation, semantic segmentation and salient object detection.

Image recognition. We discuss the performance of RConv-MK on image recognition with clean images. We first build up CNNs using RConv-MK under ResNet [59] and WRN [198]. CIFAR-10/100 dataset [89] is employed for evaluation.

Table 5.9 and Table 5.10 demonstrate the error rates of RConv-MK under ResNet and WRN on CIFAR-10/100, respectively.

In addition, we construct models under modern network architectures, including ResNet [59], WRN [198], VGG (with 11 layers) [151], AlexNet [90], and ShiftNet [180], and perform experiments on ImageNet [35]. Table 5.11, Table 5.12 and Table 5.13 present the results. One can see that models built with our RConv-MK reduces

Table 5.9: Results (error rates, %) of RConv-MK under ResNet architecture on CIFAR-10/100.

Depth	Method	Param/FLOPs	C10/C100
20	Conv2d	0.27M/40.8M	7.7/30.9
	Conv2d+SE	0.28M/40.8M	7.6/30.5
	Conv2d+CBAM	0.28M/40.8M	7.3/30.3
	LST	**0.20M/34M**	6.7/28.2
	RConv-MK	**0.20M/34M**	**6.4/27.5**
56	Conv2d	0.86M/126M	6.6/27.6
	Conv2d+SE	0.87M/126M	6.4/27.5
	Conv2d+CBAM	0.87M/126M	6.0/27.1
	LST	**0.59M/94M**	5.6/24.1
	RConv-MK	**0.59M/94M**	**5.5/24.0**
110	Conv2d	1.73M/253M	6.6/25.2
	Conv2d+SE	1.74M/253M	5.2/23.9
	Conv2d+CBAM	1.74M/253M	5.1/23.5
	LST	**1.17M/183M**	**5.0**/22.7
	RConv-MK	**1.17M/183M**	**5.0/22.6**

the top-1 error rates of those built with LST by 0.2% $\sim$ 0.5% at almost the same cost. In addition, an 18-layer model built with RConv-MK obtains even lower error rates than a 34-layer model built with Conv2d. Similar results can also be drawn for a 34-layer and a 50-layer RConv-MK ResNet models.

We also compare our RConv-MK with DCTNet [187] with 64 input channels under the same ResNet50 architecture on ImageNet. The top-1/5 error rates of DCTNet are reduced by RConv-MK from 22.84%/6.53% to 22.22%/6.32%. It indicates that noises in the input DCT domain of DCTNet may hurt the backbone model. Besides, RConv-MK runs at 27.78 FPS (including data loading and pre-processing with single CPU thread plus computation on GPU), faster than DCTNet by 3.39 FPS. Even though DCTNet can reduce the latency of data transmission to

Table 5.10: Results (error rates, %) of RConv-MK under WRN architecture on CIFAR-10/100.

Depth	Multiplier	Model	Param/FLOPs	C10/C100
16	8	Conv2d	10.96M/2.00G	4.80/22.03
		LST	**7.42M/1.30G**	4.70/20.88
		RConv-MK	**7.42M/1.30G**	**4.40/20.20**
	10	Conv2d	17.12M/3.12G	4.49/21.52
		LST	**11.53M/2.01G**	4.46/20.21
		RConv-MK	**11.53M/2.01G**	**4.14/20.18**
22	8	Conv2d	17.16M/2.91G	4.56/21.21
		LST	**10.94M/1.82G**	4.40/**19.33**
		RConv-MK	**10.94M/1.82G**	**4.15/19.21**
	10	Conv2d	26.80M/4.54G	4.44/20.75
		LST	**17.01M/2.82G**	4.31/18.57
		RConv-MK	**17.01M/2.82G**	**4.07/18.41**
28	10	Conv2d	36.48M/5.95G	4.17/20.50
		LST	**22.50M/3.63G**	4.03/18.23
		RConv-MK	**22.50M/3.63G**	**3.90/18.07**
	12	Conv2d	43.42M/8.56G	4.33/20.41
		LST	**32.29M/5.20G**	3.94/17.93
		RConv-MK	**32.29M/5.20G**	**3.86/17.83**
40	4	Conv2d	8.91M/1.41G	4.97/22.89
		LST	**5.52M/0.87G**	4.31/19.14
		RConv-MK	**5.52M/0.87G**	**3.92/18.80**
	8	Conv2d	35.75M/5.63G	4.66/19.38
		LST	**21.52M/3.38G**	3.76/18.56
		RConv-MK	**21.52M/3.38G**	**3.70/18.44**

Table 5.11: Results (error rates, %) of RConv-MK under ResNet architecture on ImageNet.

Depth	Method	Param/FLOPs	Top-1/Top-5
18	Conv2d	11.69M/1.81G	30.24/10.92
	LST	**8.03M/1.48G**	26.55/8.59
	RConv-MK	**8.03M/1.48G**	**26.26/8.48**
34	Conv2d	21.79M/3.66G	26.70/8.58
	LST	**13.82M/2.56G**	23.92/7.24
	RConv-MK	**13.82M/2.56G**	**23.54/6.99**
50	Conv2d	25.56M/4.09G	23.85/7.13
	LST	**23.33M/4.05G**	22.78/6.66
	RConv-MK	**23.33M/4.05G**	**22.22/6.32**
101	Conv2d	44.55M/7.80G	22.63/6.44
	LST	**42.36M/7.75G**	21.63/5.94
	RConv-MK	**42.36M/7.75G**	**21.41/5.93**

some extent by performing DCT sequentially on CPU, we note that the cost is still expensive (even with support of advanced CPU instructions). Therefore, it almost neutralizes the latency it actually saves.

Object detection and instance segmentation. We report results of object detection and instance segmentation on MS-COCO [107]. The same dataset partition, detectors, and backbone architecture are used in Section 3.4.9.

Table 5.14 and Table 5.15 demonstrate the object detection results and instance segmentation results on MS-COCO validation set, respectively. One can see that RConv-MK has better mAP results than the other two backbone methods on both tasks. Among all detectors, RConv-MK improves mAP of Conv2d by 2.9%∼3.9% on object detection, while it further boosts mAP of LST by 0.5%∼1.7%. Besides, on instance segmentation, RConv-MK outperforms Conv2d by 2.7%∼2.9% and LST by 0.5% with Mask R-CNN and Cascade Mask R-CNN.

Table 5.12: Results (error rates, %) of RConv-MK under WRN architecture on ImageNet.

Depth	Mulp.	Method	Param/FLOPs	Top-1/Top-5
18	1	Conv2d	11.69M/1.81G	30.24/10.92
		LST	**8.03M/1.48G**	26.55/8.59
		RConv-MK	**8.03M/1.48G**	**26.26/8.48**
	1.5	Conv2d	25.88M/3.87G	27.06/9.00
		LST	**17.53M/3.21G**	24.44/7.51
		RConv-MK	**17.53M/3.21G**	**24.03/7.33**
	2	Conv2d	45.62M/6.70G	25.58/8.06
		LST	**30.68M/5.59G**	23.49/6.93
		RConv-MK	**30.68M/5.59G**	**23.01/6.88**
	3	Conv2d	101.78M/14.72G	24.06/7.33
		LST	**67.96M/12.31G**	22.33/6.52
		RConv-MK	**67.96M/12.31G**	**22.31/6.51**
34	1	Conv2d	21.79M/3.66G	26.70/8.58
		LST	**13.82M/2.56G**	23.92/7.24
		RConv-MK	**13.82M/2.56G**	**23.54/6.99**
	1.5	Conv2d	48.61M/8.03G	24.50/7.58
		LST	**30.42M/5.59G**	22.29/6.30
		RConv-MK	**30.42M/5.59G**	**22.20/6.28**
	2	Conv2d	86.04M/14.09G	23.39/7.00
		LST	**53.49M/9.79G**	21.44/6.11
		RConv-MK	**53.49M/9.79G**	**21.38/6.10**
50	1	Conv2d	25.56M/4.09G	23.85/7.13
		LST	**23.33M/4.05G**	22.78/6.66
		RConv-MK	**23.33M/4.05G**	**22.22/6.32**
	2	Conv2d	68.88M/11.40G	21.90/6.03
		LST	**66.10M/11.09G**	20.89/5.76
		RConv-MK	**66.10M/11.09G**	**20.78/5.72**

Table 5.13: Results (error rates, %) of RConv-MK under AlexNet, VGG and ShiftNet architectures on ImageNet.

Architecture	Method	Param/FLOPs	Top-1/Top-5
AlexNet (FC)	Conv2d w/o BN	61.10M/0.71G	43.45/20.91
	Conv2d w/ BN	61.10M/0.71G	41.93/20.02
	LST	**60.30M/0.62G**	39.32/17.40
	RConv-MK	**60.30M/0.62G**	**38.85/17.22**
AlexNet (GAP)	Conv2d w/o BN	2.73M/0.66G	51.13/26.33
	Conv2d w/ BN	2.73M/0.66G	46.65/23.43
	LST	**2.25M/0.60G**	39.91/17.86
	RConv-MK	**2.25M/0.60G**	**39.31/17.23**
VGG (FC)	Conv2d w/o BN	132.86M/7.61G	30.98/11.37
	Conv2d w/ BN	132.86M/7.61G	29.62/10.19
	LST	**128.63M/5.89G**	28.56/9.79
	RConv-MK	**128.63M/5.89G**	**28.08/9.72**
VGG (GAP)	Conv2d w/o BN	9.73M/7.49G	33.40/12.20
	Conv2d w/ BN	9.73M/7.49G	31.63/11.76
	LST	**6.63M/5.04G**	29.23/10.26
	RConv-MK	**6.63M/5.04G**	**28.20/9.88**
ShiftNet-A	Shift	**4.1M**/1.4G	29.9/10.3
	LST	4.3M/**1.2G**	29.3/10.0
	RConv-MK	4.3M/**1.2G**	**29.2/9.9**
ShiftNet-B	Shift	**1.1M**/N.A.	38.8/16.4
	LST	1.2M/389.5M	36.9/14.8
	RConv-MK	1.2M/389.5M	**36.7/14.7**
ShiftNet-C	Shift	**0.78M**/N.A.	41.2/18.0
	LST	0.84M/342.5M	38.9/16.3
	RConv-MK	0.84M/342.5M	**38.7/16.2**

Table 5.14: Object detection results (%) of RConv-MK on MS-COCO validation set.

Detector	Backbone Method	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN [144]	LST [97]	40.8	62.2	44.3	**24.8**	44.7	53.1
	Conv2d	37.4	58.1	40.4	21.2	41.0	48.1
	RConv-MK	**41.3**	**62.6**	**45.0**	24.6	**44.9**	**54.1**
RetinaNet [106]	LST [97]	38.7	58.5	41.7	22.2	42.7	51.2
	Conv2d	36.5	55.4	39.1	20.4	40.3	48.1
	RConv-MK	**39.4**	**60.0**	**42.0**	**23.2**	**43.3**	**51.5**
FCOS [161]	LST [97]	38.8	58.7	41.5	22.5	42.4	50.2
	Conv2d	36.6	55.7	38.8	20.7	40.1	47.4
	RConv-MK	**39.6**	**60.0**	**42.2**	**23.2**	**43.2**	**52.3**
Mask R-CNN [57]	LST [97]	41.3	62.5	45.0	**25.1**	45.1	54.3
	Conv2d	38.2	58.8	41.4	21.9	40.9	49.5
	RConv-MK	**41.8**	**63.3**	**45.9**	24.6	**45.8**	**54.3**
Cascade Mask R-CNN [12]	LST [97]	43.9	62.6	47.9	**26.3**	47.4	57.8
	Conv2d	41.2	59.4	45.0	23.9	44.2	54.4
	RConv-MK	**44.4**	**63.0**	**48.4**	26.0	**47.8**	**58.2**

Table 5.15: Instance segmentation results (%) of RConv-MK on MS-COCO validation set.

Detector	Backbone Method	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN [57]	LST [97]	37.1	59.3	39.4	**20.9**	40.5	**50.8**
	Conv2d	34.7	55.7	37.2	18.3	37.4	47.2
	RConv-MK	**37.6**	**59.9**	**40.2**	20.6	**41.0**	**50.8**
Cascade Mask R-CNN [12]	LST [97]	38.1	59.7	40.9	**21.4**	41.3	51.9
	Conv2d	35.9	56.6	38.4	19.4	38.5	49.3
	RConv-MK	**38.6**	**60.2**	**41.5**	21.2	**41.7**	**52.6**

Table 5.16: Semantic segmentation results of LST v2 on PASCAL VOC2012.

Seg. Method	Backbone Method	mIoU	Overall Acc. (%)	Freq. W. Acc.(%)
DeepLab V3 [23]	LST [97]	0.790	94.49	90.07
	Conv2d	0.767	93.89	88.76
	RConv-MK	**0.792**	**94.52**	**90.16**
DeepLab V3+ [24]	LST [97]	0.792	94.58	90.17
	Conv2d	0.771	94.13	89.14
	RConv-MK	**0.797**	**94.64**	**90.18**

Semantic segmentation. We report semantic segmentation results of RConv-MK on PASCAL VOC [41]. We set up the backbone model under ResNet-50 architecture by using RConv-MK. More details can be found in Section 3.4.10.

Table 5.16 presents the results. With the same semantic segmentation method, RConv-MK substantially outperforms both Conv2d and LST in terms of all three metrics. For example, RConv-MK improves the mIoU of Conv2d/LST by 0.025/0.002 and 0.026/0.005 in cases of DeepLab V3 and DeepLab V3+, respectively.

Salient object detection. We compare RConv-MK to Conv2d and LST [97] on salient object detection by following the setting in Section 3.4.1 and Section 3.4.12. Specifically, we replace LST [97] with our RConv-MK to build the backbone model under ResNet-50 architecture. For each method, the backbone model is pretrained on ImageNet.

Table 5.17 presents the results of RConv-MK on salient object detection. It can be observed that RConv-MK achieves better results than both Conv2d and LST in terms of F-meansure and MAE across all benchmarks.

Table 5.17: Salient object detection results of RConv-MK under ResNet-50 architecture.

Dataset	Backbone Method	F-measure↑	MAE↓
ECSSD [188]	Conv2d	0.940	0.042
	LST [97]	0.950	0.034
	RConv-MK	**0.953**	**0.032**
PASCAL-S [104]	Conv2d	0.863	0.075
	LST [97]	0.876	0.066
	RConv-MK	**0.882**	**0.063**
DUT-OMRON [189]	Conv2d	0.830	0.055
	LST [97]	0.834	0.052
	RConv-MK	**0.836**	**0.049**
HKU-IS [92]	Conv2d	0.934	0.032
	LST [97]	0.941	0.028
	RConv-MK	**0.944**	**0.026**
SOD [130]	Conv2d	0.867	0.100
	LST [97]	0.875	0.093
	RConv-MK	**0.885**	**0.092**
DUTS [166]	Conv2d	0.886	0.040
	LST [97]	0.895	0.035
	RConv-MK	**0.901**	**0.033**

5.4.5 Ablation study

We explore different settings of multiple kernels in the proposed RConv-MK. Experiments are designed from two aspects, including the effect of m, *i.e.*, total number of kernels used in T_s, and the effect of different settings of channel split when the default $m = 3$ kernels are used.

Regarding the choice of m, Table 5.3 and Table 5.4 present the results. We consider m from 2 to 4 because kernels larger than 7×7 are rare in modern architectures due to computational cost. Thus, we stop at $m = 4$ (*i.e.*, the largest kernel size is

7×7). One can see that a model wins all cases when $m = 3$. When $m = 4$, the results are close to those of $m = 3$. However, there is a clear drop when $m = 2$. In our view, this is caused by the fact that some critical cues may be lost as the 1×1 kernel are applied to convolve with most channels in T_s to balance the cost of 5×5 DWConv.

Table 5.5 and Table 5.6 demonstrate the results of different settings of channel split. We fixed the relative proportion of channels for 1×1 and 5×5 kernels and adjust the proportion of those for the 3×3 kernel. It can be observed that the default setting results in the best performance when we deal with clean images, corrupted images as well as adversarial attacks. Starting from the default setting, with more channels convolved with the 3×3 kernel, the results are closer to the use of a uniform kernel, *i.e.*, RConv-UK.

5.5 Conclusion

In this chapter, we proposed RConv-MK to make LST architecturally robust to adversarial attacks. RConv-MK employs a set of kernels of different size and flexibly applies them to the input features of different frequencies. It enlarges the receptive fields for low frequency features and saves the overhead for sparse high frequency features. We further introduce a normalized soft thresholding (NST) operator to adaptively address input samples with different corruption scales for effective removal of noise and trivial features in the relevant feature domain. Extensive experiments validate the effectiveness of the proposed RConv-MK under popular CNN architectures to corrupted samples as well as clean images.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Visual recognition has a wide range of real applications in our daily life. With the rapid development of deep convolutional neural networks (CNNs) in the past years, remarkable progress has been made on many tasks. While design of CNN architecture today becomes increasingly convenient by reusing some common modules and well-proved network architecture, it leaves an open question whether these modules and network architecture can be still improved in some other way to make substantial contribution to CNNs. In this thesis, we made some attempts to design reliable CNN architecture for visual recognition. To satisfy the genuine need of real applications, we value the overhead and number of parameters and are committed to obtaining performance gain at comparable or even lower cost.

First-order CNNs are widely used for various visual recognition tasks. It is highly valued to improve their performance whlie keeping the same cost at testing stage. In Chapter 2, we presented detachable second-order pooling networks (DSoP-Net) to improve the performance of first-order CNNs in image classification. We carefully designed auxiliary branches to transfer knowledge to the backbone first-order networks during training, which can be removed before inference. As a result, DSoP-Net leverages the advantages of second-order pooling networks while keeping similar

complexity to first-order networks during inference. To our best knowledge, this is the first attempt to use higher-order statistics in knowledge distillation. Experiments conducted on benchmarks demonstrated the effectiveness of our network.

2D convolutional layers play an important role in the success of CNNs. In Chapter 3, we proposed to train deep CNNs with a learnable sparse transform (LST), which learns to convert the input features into a more compact and sparser domain together with the CNN training process. LST can more effectively reduce the spatial and channel-wise feature redundancies than the conventional Conv2d. It can be efficiently implemented with existing CNN modules, and is portable to existing CNN architectures for seamless training and inference. We further presented a hybrid ST-ReLU activation to enhance the robustness of the learned CNN models to common types of corruptions in the input. We validated that LST-Net can achieve even higher accuracy than its counterpart networks of the same family with lower cost on a wide range of visual recognition tasks.

Albeit with less overhead and fewer parameters, LST can still be improved to faithfully build CNNs for visual recognition. In Chapter 4, we proposed LST v2 to reduce its redundancy. To produce a compact feature bank, we allowed incomplete yet flexible expansion. The structure of HDWConv can be completely determined with no need for extra fine-tuning. High frequency input channels were expanded more times for near-complete expansion, while low frequency ones were expanded fewer times to save cost. In addition, we partitioned input channels into groups and assigned one or more unique kernels with identical initialization for better representation. We demonstrated that LST v2 can achieve comparable or even higher accuracy than LST-Net on a wide range of visual recognition tasks while it only requires approximately 40% $\sim$ 55% parameters and computational cost of Conv2d under the same architecture, saving around 20% $\sim$ 40% overhead of LST-Net.

Finally, in Chapter 5, we discussed the application of LST to adversarial attacks.

We proposed RConv-MK to improve the architecture of LST against various types of image corruptions and manually designed adversarial attacks. The novel transform contains a set of kernels of different size. The kernels at each layer are flexibly applied to the given features of different frequencies to enlarge the receptive fields for low frequency features and saves the overhead for sparse high frequency features. We further introduce a normalized soft thresholding (NST) operator to adaptively address input samples with different corruption scales for effective removal of noise and trivial features in the relevant feature domain. Extensive experiments conducted on a number of popular visual recognition tasks demonstrated the effectiveness of RConv-MK.

6.2 Future Work

The proposed algorithms in this thesis advance reliable CNN architecture design for visual recognition. In future work, we will expand our study in the following directions:

- Depth-wise separable convolutional layers are widely used in the spatial transform of LST-Net and its variants. However, a large number of independent operations, like HDWConv, are exectued in sequence, making little use of parallel devices, such as GPU cards. We will implement it with our own kernel function to boost the runtime performance on GPU.

- Our current implementation of DSoP-Net, LST-Net and its variants is based on single float. However, it is more desired to perform low precision calculation, like half float, integer, *etc.*, on resource-limited devices. In the future, we will investigate the performance of the proposed methods for such application.

- The proposed methods in this thesis can fill the pool of search space for a large

number of existing Neural Architecture Search (NAS) strategies [211, 212, 134, 40]. We are looking forward to discovering more powerful neural networks constructed with our methods by leveraging NAS.

- To strengthen an LST v2 bottleneck, we will enlarge its receptive field by using different kernel sizes for its spatial transform in a similar way of RConv-MK.

- We will investigate our methods on more visual recognition tasks, including facial attribute analysis, visual tracking, industrial defect detection, action recognition, crowd counting, *etc*.

We will explore the above research directions in the future.

Bibliography

[1] Keivan Alizadeh vahid, Anish Prabhu, Ali Farhadi, and Mohammad Rastegari. Butterfly transform: An efficient FFT based neural network architecture design. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[2] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, June 2014.

[3] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM J. Matrix Anal. Appl.*, 29(1):328–347, 2007.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Int. Conf. Mach. Learn.*, 2018.

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

[6] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Int. Conf. Learn. Represent.*, 2016.

[7] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. Attention augmented convolutional networks. In *Int. Conf. Comput. Vis.*, 2019.

[8] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *Eur. Conf. Comput. Vis.*, 2018.

[9] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

[10] Jian-Feng Cai, Bin Dong, Stanley Osher, and Zuowei Shen. Image restoration: Total variation, wavelet frames, and beyond. *J. Am. Math. Soc.*, 25(4):1033–1089, 2012.

[11] Sijia Cai, Wangmeng Zuo, and Lei Zhang. Higher-order integration of hierarchical convolutional activations for fine-grained visual categorization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.

[12] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: High quality object detection and instance segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[13] Emmanuel J. Candes, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inf. Theory*, 52(2):489–509, 2006.

[14] Emmanuel J. Candes, Michael B. Wakin, and Stephen Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *J. Fourier Anal. Appl.*, 14:877–905, 2008.

[15] Bing Cao, Nannan Wang, Jie Li, and Xinbo Gao. Data augmentation-based joint learning for heterogeneous face recognition. *IEEE Trans. Neural Netw. Learn. Syst.*, 30(6):1731–1743, June 2019.

[16] Yue Cao, Jiarui Xu, Stephen Lin, Fangyun Wei, and Han Hu. GCNet: Non-local networks meet squeeze-excitation networks and beyond. In *Int. Conf. Comput. Vis. Worksh.*, 2019.

[17] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[18] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Provably minimally-distorted adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[19] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symp. Secur. Priv.*, 2017.

[20] Tianhorng Chang and Chung-Chieh Jay Kuo. Texture analysis and classification with tree-structured wavelet transform. *IEEE Trans. Image Process.*, 2(4):429–441, 1993.

[21] Haoyun Chen, Jhaohong Liang, Shihchieh Chang, Jiayu Pan, Yuting Chen, Wei Wei, and Dacheng Juan. Improving adversarial robustness via guided complement entropy. In *Int. Conf. Comput. Vis.*, 2019.

[22] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue

Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open MMLab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[23] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[24] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Eur. Conf. Comput. Vis.*, 2018.

[25] Pinyu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Chojui Hsieh. EAD: Elastic-net attacks to deep neural networks via adversarial examples. In *AAAI*, 2018.

[26] Pinyu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Chojui Hsieh. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. *arXiv preprint arXiv:1708.03999*, 2017.

[27] Weijie Chen, Di Xie, Yuan Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[28] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[29] De Cheng, Yihong Gong, Sanping Zhou, Jinjun Wang, and Nanning Zheng. Person re-identification by multi-channel parts-based CNN with improved triplet loss function. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1335–1344, 2016.

[30] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2014.

[31] Djorkarne Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *Int. Conf. Learn. Represent.*, 2016.

[32] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Int. Conf. Comput. Vis.*, 2017.

[33] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Xiaoping Zhang. Second-order attention network for single image super-resolution. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[34] Sophie Denève, Alireza Alemi, and Ralph Bourdoukan. The brain as an efficient and robust adaptive learner. *Neuron*, 94(5):969–977, 2017.

[35] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conf. Comput. Vis. Pattern Recog.* IEEE, 2009.

[36] Zhijie Deng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Adversarial distributional training for robust deep learning. *arXiv preprint arXiv:2002.05999*, 2020.

[37] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.

[38] David L. Donoho. De-noising by soft-thresholding. *IEEE Trans. Inf. Theory*, 41(3):613–627, 1995.

[39] David L. Donoho and Jain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[40] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:1–21, 2019.

[41] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The PASCAL visual object classes challenge: A retrospective. *Int. J. Comput. Vis.*, 111(1):98–136, January 2015.

[42] Yuchen Fan, Jiahui Yu, Ding Liu, and Thomas S. Huang. Scale-wise convolution for image restoration. In *AAAI*, 2020.

[43] Giulia Fracastoro, Sophie M. Fosson, and Enrico Magli. Steerable discrete cosine transform. *IEEE Trans. Image Process.*, 26(1):303–314, Jan 2017.

[44] Florian Franzen. Image classification in the frequency domain with neural networks and absolute value DCT. In *Int. Conf. Image Signal Process.*, 2018.

[45] Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual attention network for scene segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[46] Tommaso Furlanello, Zachary C. Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-again neural networks. In *Int. Conf. Mach. Learn.*, 2018.

[47] Ryosuke Furuta, Naoto Inoue, and Toshihiko Yamasaki. Fully convolutional network with multi-step reinforcement learning for image processing. In *AAAI*, 2019.

[48] Shang-Hua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr. Res2net: A new multi-scale backbone architecture. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2020.

[49] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[50] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Int. Conf. Learn. Represent.*, 2015.

[51] Qiushan Guo, Xinjiang Wang, Yichao Wu, Zhipeng Yu, Ding Liang, Xiaolin Hu, and Ping Luo. Online knowledge distillation via collaborative learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[52] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[53] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. GhostNet: More features from cheap operations. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[54] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model Rubik's Cube: Twisting resolution, depth and width for tinynets. In *Adv. Neural Inform. Process. Syst.*, 2020.

[55] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Int. Conf. Learn. Represent.*, 2016.

[56] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *Int. Conf. Comput. Vis.*, 2011.

[57] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Grishick. Mask R-CNN. In *Int. Conf. Comput. Vis.*, 2017.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Int. Conf. Comput. Vis.*, 2015.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Eur. Conf. Comput. Vis.* Springer, 2016.

[60] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *Eur. Conf. Comput. Vis.*, 2018.

[61] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Int. Conf. Comput. Vis.*, 2017.

[62] Christopher Heil and David F. Walnut. Continuous and discrete wavelet transforms. *SIAM Rev.*, 31(4):628–666, 1989.

[63] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Int. Conf. Learn. Represent.*, 2019.

[64] Geoffrey E. Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[65] Tahmid Hossain, Shyh Wei Teng, Dengsheng Zhang, Suryani Lim, and Guojun Lu. Distortion robust image classification using deep convolutional neural network with discrete cosine transform. In *IEEE Int. Conf. Image Process.*, 2019.

[66] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip Torr. Deeply supervised salient object detection with short connections. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(4):815–828, 2019.

[67] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3. In *Int. Conf. Comput. Vis.*, 2019.

[68] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[69] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *Int. Conf. Comput. Vis.*, 2019.

[70] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-Excite: Exploiting feature context in convolutional neural networks. In *Adv. Neural Inform. Process. Syst.*, 2018.

[71] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[72] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.

[73] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *Eur. Conf. Comput. Vis.* Springer, 2016.

[74] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[75] Ke Huang and Selin Aviyente. Wavelet feature selection for image classification. *IEEE Trans. Image Process.*, 17(9):1709–1720, 2008.

[76] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Eur. Conf. Comput. Vis.*, 2018.

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

[78] Raoul Huys, Viktor K. Jirsa, Ziauddin Darokhan, Sonata Valentiniene, and Per E. Roland. Visually evoked spiking evolves while spontaneous ongoing dynamics persist. *Front. Syst. Neurosci.*, 9:183, 2016.

[79] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Int. Conf. Mach. Learn.*, 2018.

[80] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training CNNs with low-rank filters for efficient image classification. In *Int. Conf. Learn. Represent.*, 2016.

[81] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. Mach. Learn.*, 2015.

[82] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *Int. Conf. Comput. Vis.*, 2015.

[83] Ira Kemelmacher-Shlizerman, Steven M. Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[84] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from Adam to SGD. *arXiv preprint arXiv:1712.07628*, 2017.

[85] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2011.

[86] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[87] Gunter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Adv. Neural Inform. Process. Syst.*, 2017.

[88] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *IEEE 3D Represent. Recog.*, Sydney, Australia, 2013.

[89] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009.

[90] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Adv. Neural Inform. Process. Syst.*, 2012.

[91] Duo Li, Aojun Zhou, and Anbang Yao. HBONet: Harmonious bottleneck on two orthogonal dimensions. In *Int. Conf. Comput. Vis.*, Oct 2019.

[92] Guanbin Li and Yizhou Yu. Visual saliency based on multiscale deep features. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2015.

[93] Guilin Li, Junlei Zhang, Yunhe Wang, Chuanjian Liu, Matthias Tan, Yunfeng Lin, Wei Wang, Jiashi Feng, and Tong Zhang. Residual distillation, towards portable deep neural networks without shortcuts. In *Adv. Neural Inform. Process. Syst.*, 2020.

[94] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In *Adv. Neural Inform. Process. Syst.*, 2017.

[95] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *Int. Conf. Learn. Represent.*, 2017.

[96] Jun Li, Xue Mei, Danil Prokhorov, and Dacheng Tao. Deep neural network for structural prediction and lane detection in traffic scene. *IEEE Trans. Neural Netw. Learn. Syst.*, 28(3):690–703, March 2017.

[97] Lida Li, Kun Wang, Shuai Li, Xiangchu Feng, and Lei Zhang. LST-Net: Learning a convolutional neural network with a learnable sparse transform. In *Eur. Conf. Comput. Vis.*, 2020.

[98] Lida Li, Kun Wang, Shuai Li, Xiangchu Feng, and Lei Zhang. Remarks on Tc and Ts, 2020. `https://github.com/lld533/LST-Net/blob/master/Remarks_on_Tc_and_Ts.txt`.

[99] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[100] Peihua Li, Jiangtao Xie, Qilong Wang, and Wangmeng Zuo. Is second-order information helpful for large-scale visual recognition? In *Int. Conf. Comput. Vis.*, 2017.

[101] Shuai Li, Lingxiao Yang, Jianqiang Huang, Xian-Sheng Hua, and Lei Zhang. Dynamic anchor feature selection for single-shot object detection. In *Int. Conf. Comput. Vis.*, 2019.

[102] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[103] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Factorized bilinear models for image recognition. In *Int. Conf. Comput. Vis.*, 2017.

[104] Yin Li, Xiaodi Hou, Christof Koch, James M. Rehg, and Alan L. Yuille. The secrets of salient object segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2014.

[105] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *Int. Conf. Learn. Represent.*, 2014.

[106] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Int. Conf. Comput. Vis.*, 2017.

[107] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common objects in context. In *Eur. Conf. Comput. Vis.*, 2014.

[108] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. In *Brit. Mach. Vis. Conf.*, 2017.

[109] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear CNN models for fine-grained visual recognition. In *Int. Conf. Comput. Vis.*, 2015.

[110] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear convolutional neural networks for fine-grained visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(6):1309–1322, 2018.

[111] Drew Linsley, Dan Scheibler, Sven Eberhardt, and Thomas Serre. Global-and-local attention networks for visual recognition. In *Int. Conf. Learn. Represent.*, 2018.

[112] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Int. Conf. Learn. Represent.*, 2019.

[113] Jiang-Jiang Liu, Qibin Hou, Ming-Ming Cheng, Jiashi Feng, and Jianmin Jiang. A simple pooling-based design for real-time salient object detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[114] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *Int. Conf. Learn. Represent.*, 2020.

[115] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *Int. Conf. Learn. Represent.*, 2017.

[116] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *Int. Conf. Learn. Represent.*, 2019.

[117] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A filter level pruning method for deep neural network compression. In *Int. Conf. Comput. Vis.*, 2017.

[118] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In *Int. Conf. Learn. Represent.*, 2019.

[119] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient cnn architecture design. In *Eur. Conf. Comput. Vis.*, 2018.

[120] Xu Ma, Jingda Guo, Sihai Tang, Zhinan Qiao, Qi Chen, Qing Yang, and Song Fu. DCANet: Learning connected attentions for convolutional neural networks. In *Eur. Conf. Comput. Vis.*, 2020.

[121] Andrew Maas, Awni Hannun, and Andrew Ng. Rectifier nonlinearities improve neural network acoustic models. In *Int. Conf. Mach. Learn.*, 2013.

[122] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Int. Conf. Learn. Represent.*, 2018.

[123] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[124] John Makhoul. A fast cosine transform in one and two dimensions. *IEEE Trans. Acoust., Speech, Signal Process*, 28(1):27–34, 1980.

[125] David R. Martin, Charless C. Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Int. Conf. Comput. Vis.*, 2001.

[126] Sachin Mehta, Hannaneh Hajishirzi, and Mohammad Rastegari. DiCENet: Dimension-wise convolutions for efficient networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2020.

[127] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. ESPNet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Eur. Conf. Comput. Vis.*, 2018.

[128] Sachin Mehta, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi. ESPNetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[129] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *Int. Conf. Learn. Represent.*, 2018.

[130] Vida Movahedi and James H. Elder. Design and perceptual validation of performance measures for salient object segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2010.

[131] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Int. Conf. Mach. Learn.*, 2010.

[132] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[133] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang,

Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Adv. Neural Inform. Process. Syst.*, pages 8024–8035. Curran Associates, Inc., 2019.

[134] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Int. Conf. Mach. Learn.*, 2018.

[135] Harry Pratt, Bryan Williams, Frans Coenen, and Yalin Zheng. FCNN: Fourier convolutional neural networks. In Michelangelo Ceci, Jaakko Hollmén, Ljupčo Todorovski, Celine Vens, and Sašo Džeroski, editors, *Mach. Learn. Knowl. Discov. Databases*, volume 10534, pages 786–798. Springer International Publishing, 2017.

[136] Zequn Qin, Pengyi Zhang, Fei Wu, and Xi Li. FcaNet: Frequency channel attention networks. *arXiv preprint arXiv:2012.11879*, 2020.

[137] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2009.

[138] Filip Radenović, Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondřej Chum. Revisiting Oxford and Paris: Large-scale image retrieval benchmarking. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[139] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Fine-tuning CNN image retrieval with no human annotation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(7):1655–1668, 2018.

[140] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Swish: A self-gated activation function. *arXiv preprint arXiv:1710.05941*, 2017.

[141] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[142] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Int. Conf. Mach. Learn.*, 2017.

[143] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[144] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Adv. Neural Inform. Process. Syst.*, 2015.

[145] Olivier Rioul and Pierre Duhamel. Fast algorithms for discrete and continuous wavelet transforms. *IEEE Trans. Inform. Theory*, 38(2):569–586, 1992.

[146] Oren Rippel, Jasper Snoek, and Ryan P. Adams. Spectral representations for convolutional neural networks. In *Adv. Neural Inform. Process. Syst.*, 2015.

[147] Per E. Roland. Space-time dynamics of membrane currents evolve to shape excitation, spiking, and inhibition in the cortex at small and large scales. *Neuron*, 94(5):934–942, 2017.

[148] Dongsheng Ruan, Jun Wen, Nenggan Zheng, and Min Zheng. Linear context transform block. In *AAAI*, 2020.

[149] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[150] Viktor Shipitsin, Iaroslav Bespalov, and Dmitry V. Dylov. Adaptive neural layer for global filtering in computer vision. *arXiv preprint arXiv:2010.01177*, 2020.

[151] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2014.

[152] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.*, 23(5):828–841, 2019.

[153] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Int. Conf. Mach. Learn.*, 2013.

[154] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[155] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2015.

[156] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[157] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Int. Conf. Learn. Represent.*, 2014.

[158] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[159] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Int. Conf. Mach. Learn.*, 2019.

[160] Yusuke Tashiro, Yang Song, and Stefano Ermon. Output diversified initialization for adversarial attacks. *arXiv preprint arXiv:2003.06878*, 2020.

[161] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Int. Conf. Comput. Vis.*, 2019.

[162] Robert Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Series B Stat. Methodol.*, 58(1):267–288, 1996.

[163] Jonathan Uesato, Brendan Odonoghue, Pushmeet Kohli, and Aaron Van Den Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *Int. Conf. Mach. Learn.*, 2018.

[164] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[165] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *Eur. Conf. Comput. Vis.*, 2020.

[166] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, Baocai Yin, and Xiang Ruan. Learning to detect salient objects with image-levels supervision. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.

[167] Qilong Wang, Peihua Li, and Lei Zhang. G^2DeNet: Global gaussian distribution embedding network and its application to visual recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.

[168] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. ECA-Net: Efficient channel attention for deep convolutional neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[169] Shui-Hua Wang, Preetha Phillips, Yuxiu Sui, Bin Liu, Ming Yang, and Hong Cheng. Classification of Alzheimer's disease based on eight-layer convolutional neural network with leaky rectified linear unit and max pooling. *J. Med. Syst.*, 42(5):85, 2018.

[170] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[171] Yan Wang, Lingxi Xie, Chenxi Liu, Siyuan Qiao, Ya Zhang, Wenjun Zhang, Qi Tian, and Alan L Yuille. SORT: Second-order response transform for visual recognition. In *Int. Conf. Comput. Vis.*, 2017.

[172] Yunhe Wang, Chang Xu, Chunjing Xu, Chao Xu, and Dacheng Tao. Learning versatile filters for efficient convolutional neural networks. In *Adv. Neural Inform. Process. Syst.*, 2018.

[173] Zhou Wang, Alan C. Bovik, H. R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004.

[174] Andrew B. Watson. Image compression using the discrete cosine transform. *Math. J.*, 4(1):81, 1994.

[175] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[176] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Adv. Neural Inform. Process. Syst.*, 2016.

[177] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *Int. Conf. Learn. Represent.*, 2020.

[178] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional block attention module. In *Eur. Conf. Comput. Vis.*, 2018.

[179] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[180] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[181] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[182] Yuxin Wu and Kaiming He. Group normalization. *arXiv preprint arXiv:1803.08494*, 2018.

[183] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Eur. Conf. Comput. Vis.*, 2018.

[184] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *Int. J. Comput. Vis.*, 119(1):3–22, 2016.

[185] Lingxi Xie and Alan L. Yuille. Genetic CNN. In *Int. Conf. Comput. Vis.*, 2017.

[186] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.* IEEE, 2017.

[187] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yenkuang Chen, and Fengbo Ren. Learning in the frequency domain. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[188] Qiong Yan, Li Xu, Jianping Shi, and Jiaya Jia. Hierarchical saliency detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2013.

[189] Chuan Yang, Lihe Zhang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Saliency detection via graph-based manifold ranking. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2013.

[190] Xun Yang, Peicheng Zhou, and Meng Wang. Person reidentification via structural deep metric learning. *IEEE Trans. Neural Netw. Learn. Syst.*, 2018.

[191] Yanchao Yang and Stefano Soatto. FDA: Fourier domain adaptation for semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[192] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W. Mahoney. ADAHESSIAN: An adaptive second order optimizer for machine learning. *arXiv preprint arXiv:2006.00719*, 2020.

[193] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.

[194] Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. In *Eur. Conf. Comput. Vis.*, 2020.

[195] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *Int. Conf. Learn. Represent.*, 2016.

[196] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: Pruning networks using neuron importance score propagation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[197] Sergey Zagoruyko. 92.45% on cifar-10 in torch. `https://github.com/szagoruyko/cifar.torch`, 2015.

[198] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Brit. Mach. Vis. Conf.*, 2016.

[199] Yann Zerlaut and Alain Destexhe. Enhanced responsiveness and low-level awareness in stochastic network states. *Neuron*, 94(5):1002–1009, 2017.

[200] Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training. In *Adv. Neural Inform. Process. Syst.*, 2019.

[201] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *Int. Conf. Mach. Learn.*, 2019.

[202] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Trans. Image Process.*, 26(7):3142–3155, 2017.

[203] Lei Zhang, Paul Bao, and Xiaolin Wu. Multiscale LMMSE-based image denoising with optimal wavelet selection. *IEEE Trans. Circuit Syst. Video Technol.*, 15(4):469–481, April 2005.

[204] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[205] Zizhao Zhang, Han Zhang, Sercan Ö. Arik, Honglak Lee, and Tomas Pfister. Distilling effective supervision from severe label noise. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[206] Haiyu Zhao, Maoqing Tian, Shuyang Sun, Jing Shao, Junjie Yan, Shuai Yi, Xiaogang Wang, and Xiaoou Tang. Spindle Net: Person re-identification with human body region guided feature decomposition and fusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1077–1085, 2017.

[207] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Trans. Neural Netw. Learn. Syst.*, pages 1–21, 2019.

[208] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[209] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(6):1452–1464, 2018.

[210] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable ConvNets v2: More deformable, better results. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[211] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[212] Barret Zoph and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.