ABSTRACT

LI, XILAI. Neural Architectures Design and Search for Deep Representation Learning and Continual Learning. (Under the direction of Tianfu Wu).

Recent progress of deep neural networks or deep learning has enabled great impact and transformative applications in both computer vision and natural language processing. Amongst all the aspects of deep learning development, neural architecture design is the foundation for improving the performance of deep neural networks. This thesis explores several novel approaches on neural architecture design or search in terms of both network topology and node operations to improve deep neural networks on not only accuracy, but also interpretability, robustness, continual/lifelong learning ability, etc.

I begin by presenting a class of models called *And-Or Grammar networks* (AOGNet) which utilize And-Or grammar as a network structure generator. AOGNets harness the best of both world: grammar models and deep neural networks. It integrates compositionality adn reconfigurability of the former and capability of learning rich features of the latter in a principled way. This family of architectures enable better performance on multiple computer vision benchmarks and also shows better model interpretability and adversarial robustness.

Next, I will introduce a novel feature normalization technique that integrates existing feature normalization with attention. And the resulting *Attentive Normalization* (AN) outperforms the widely used batch normalization (BN) and its variants.

The final chapter explores a potential learning framework to alleviate a widely-known issue of neural networks, called *catastrophic forgetting*, when the learning systems are trained with sequential or streaming tasks. We presents a conceptually simple yet general and effective framework, called *Learn-to-Grow*, from the angle of neural architecture by separating the learning into a neural structure optimization component and a parameter learning component.

Neural Architectures Design and Search for Deep Representation Learning and Continual Learning

by
Xilai Li

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina
2020

APPROVED BY:

DEDICATION

This dissertation is dedicated to my dear parents Hongchao Li and Jiandong Qiu, who have loved and supported me no matter what. I would like to thank my wife Yang Li for our many wonderful years and always being on my side.

BIOGRAPHY

The Author was born on October 4, 1991, in Shenyang, Liaoning, China. In June 2014, he received his B.S. degree from the department of Physics, University of Science and Technology of China (USTC) in Hefei. There, he got the interest on the research. Two months later, he enrolled in the Ph.D. program in the Department of Electrical and Computer Engineering, at North Carolina State University. In the first two and half years of his Ph.D., he worked as a research assistant supervised by Professor Ki Wook Kim. His research aims at designing the next generation of information and signal processing devices, enabled by newly discovered physics phenomena, especially on ultra-low power magnetic devices. In the latter three years, he worked as a research assistant supervised by Professor Tianfu (Matt) Wu. His research focuses on designing universal, efficient and interpretable deep learning architectures that can power computer vision, natural language processing, or general artificial intelligence.

ACKNOWLEDGEMENTS

The 5-year Ph.D. study at NC State was an amazing experience. There are many people that I wish to acknowledge. This dissertation would not have been possible without the support of you guys.

First of all, I would like to thank my advisor, Prof. Tianfu Wu. He is a very good advisor. Most of the work here is the result form our fruitful discussions. He has always been patient when I encounter difficult problems during my research and always motivate me to work on project with big impact.

Also, I would like to thank my previous advisor, Ki Wook Kim. He is a very good physicist. I learned a lot on the mathematical and physical modeling while working with him. And the skills I learned is very helpful for my current research topics.

I would also like to thank my thesis committee, Hamid Krim, Edgar Lobaton and Luo Xiao, for their kind support and advice on my research. I am very honored to have them in my committee.

Also, I would like to thank my group-mates who made my graduate school journey more colorful and fulfilling: Xiaopeng Duan, Rui Mao, Zhenghe Jin, Wei Sun, Liang Dong, Zekun Zhang, Nan Xue, Xianpeng Liu, and Kelvin Cheng.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Recent progress of deep neural networks or deep learning has enabled great impact and transformative applications in both computer vision and natural language processing, because its powerful representation learning capability. However, despite the superior performance of the deep neural networks on some standard tasks with abundant labeled training data, it is still far away from the level of general human intelligence in many aspects.

Deep neural network models such as Convolutional Neural Networks (CNN) are typically contains tens or hundreds of millions parameters, and it often costs huge computation resources to train and high-speed inference still need to be performed on high-end devices, like GPUs or TPUs. Also, deep neural networks are often criticized as being black boxes that lack of interpretability. The interpretablility of AI models is closely relevant to various important issues, especially in some safety-aware areas, like medical image diagnosis and autonomous driving.

Another main issue of current deep neural networks is that most of the networks are designed for a particular single task and trained from random initialization or only partly pretrained. Human intelligence can handle many related or unrelated tasks. And when human learns a new task, we could take advantage from the knowledge or pattern we have learned from previous tasks, to learn faster and better. And most importantly, we won't forget the previous learned knowledge too much after we learn on something knew. In most of the cases, learning on new things will reversely help for better understanding of the previously learned knowledge. However, deep neural networks have severe "catastrophic forgetting" issues when the model is trained on multiple tasks sequentially.

This Ph.D. thesis work explores to alleviate the above issues of current deep neural networks from the angle of neural architecture design and search by seeking for an universal representation learning system with better parameter efficiency, interpretability and lifelong learning capability. For universal architecture, we aim at designing a family of neural architectures that follows certain biological plausible principles or rules, and can be used on many different domains or tasks in computer vision, or natural language processing, or both. We also aim to design a better learning framework that can enable deep neural networks learn multiple tasks in a sequential/online manner with minimum forgetting and maximum inter-task knowledge transfer.

A novel model family introduced in this thesis is summarized under the term *And-Or Grammar Networks*. Although there has been great success using deep learning techniques in image classification and natural language processing, an import aspect of language and the visual world that has not been accounted for in deep neural networks is the pervasiveness of recursive or hierarchical and compositional structure. This motivated us to bridge grammar model with deep neural networks to integrate compositionality and reconfigurability of the former and the capability of learning rich representations of the later in a principled way. Here I would like to cite a sentence said by Dr. David Mumford, *"Grammar in language is merely a recent extension of much older grammars that are built into the brains of all intelligent animals to analyze sensory input, to structure their actions and even formulate their thought"*.

In order to have elegant framework for lifelong/continual learning, there are several requirements. the most important one is that we need to have minimum forgetting. Secondly, learning new knowledge could take advantage of the previously learned ones, so that it will learn better and faster. The model size need to grow but cannot grow linearly as the number of tasks increases. External memory could be used but the size would be limited to reasonable constant maximum. Finally, positive backward transfer is plausible, in other words, learning new task should potentially improve the learning on old tasks instead of forgetting. Considering on these requirements, I proposed our simple yet general and effective *Learn-to-Grow framework*, for handling catastrophic forgetting in continual learning with deep neural networks.

In Chapter 2, I'll review the history of Convolutional neural architecture designs from the original invention of CNNs, basic components of common CNN architecture, to modern deep CNN architecture designs and search. I'll review crucial components that CNNs are built on such as convolution, batch normalization, etc. I'll also review several widely-used architectural designs such as ResNet and DenseNet, and how it is related to AOGNet I will introduce in Chapter 3. And finally review some recent progress on neural architecture search.

In Chapter 3, I'll introduce *And-Or Grammar Networks* for computer vision tasks. It represents a family of deep compositional grammatical architectures which harness the best of two worlds: grammar models and Convolutional Neural Networks (CNNs). AOGNets integrate compositionality and reconfigurability of the former and the capability of learning rich features of the latter in a principled way. We utilize AND-OR Grammar (AOG) as network generator and call the resulting networks AOGNets. An AOGNet consists of a number of stages each of which is composed of a number of AOG building blocks. An AOG building block splits its input feature map into N groups along feature channels and then treat it as a sentence of N words. It then jointly realizes a phrase structure grammar and a dependency grammar in bottom-up parsing the "sentence" for better feature exploration and reuse. It provides a unified framework for the best practices developed in state-of-the-art DNNs. In experiments, AOGNet obtains state-of-the-art results on CIFAR-10, CIFAR-100 and ImageNet-1K classification benchmark and the MS-COCO object detection and segmentation benchmark. AOGNet also obtains the best model interpretability score using network dissection. AOGNet further shows better potential in adversarial defense.

In Chapter 4, I propose a new feature normalization operator called *Mixture Normalization* which is a simple and unified alternative to the widely used Batch Normalization. Mixture normalization absorbed channel-wise feature attention into the affine transformation of vanilla normalization to make the affine transformation more dynamic and instance specific. The overhead of our mixture normalization is neglectable, and could be used for replacing the vanilla normalization in most of the vision tasks. We test our mixture normalization module with multiple computer vision benchmarks with multiple popular network backbones, the results show that our mixture can improve the performance consistently.

In Chapter 5, I'll introduce our *Learn-to-Grow framework*, for handling catastrophic forgetting in continual learning with deep neural networks. It consists of two components: a neural structure optimization component and a parameter learning and/or fine-tuning component. By separating the explicit neural structure learning and the parameter estimation, not only is the proposed method capable of evolving neural structures in an intuitively meaningful way, but also shows strong capabilities of alleviating catastrophic forgetting in experiments. Furthermore, the proposed method outperforms most of the baselines on several datasets in continual learning setting.

In the final chapter, I summarize all the works and discuss about the future directions and remaining challenges.

CHAPTER

2

AN OVERVIEW OF CONVOLUTIONAL NEURAL ARCHITECTURE

2.1 Overview

Convolutional neural networks (71), or CNNs for short, form the backbone of many modern computer vision systems. It enables interesting use-cases such as image classification, object detection and segmentation, video processing, and also areas outside computer vision such as natural language processing and speech processing, etc. CNNs use multiple feature extraction stages that enables its capability of automatically learning intermediate representations from the data. It can be trained in an end-to-end manner with error back-propagation (115) algorithm without any hand-engineered features.

The powerful CNNs are not invented recently. In fact, the CNNs have already been introduced around 30 years ago, however surged until recent years with nowadays big data and high performance computing technology. CNN first came to limelight through the work of LeCuN in 1989 for processing of grid-like topological data (images and time series data). (71; 70) The architectural design of CNN was inspired by Hubel and Wiesel's work and thus largely follows the basic structure of primate's visual cortex (54; 55). During training, CNN learns through back-propagation algorithm, by regulating the change in weights with respect to the target. Optimization of an objective function using backpropagation algorithm is similar to the response based learning of human brain. Multilayered, hierarchical structure of deep CNN, gives it the ability to extract low, mid, and high-level features. High-level features (more abstract features) are a combination of lower and mid-level features. Hierarchical feature extraction ability of CNN emulates the deep and layered learning process of the Neocortex in the human brain, which dynamically learns features from the raw data (30). The popularity of CNN is largely due to its hierarchical feature extraction ability.

With the advancement of modern computing power, various of deep CNN architectures are proposed and the performance improvement from these deep networks are significant among most of the tasks (64; 40; 10). In Section 2.2, the basic components of CNN architectures are reviewed. And In Section 2.3, I will review various manually-designed neural architectures (61) and recently introduced automatically-searched architectures (145).

2.2 Basic Components of CNNs

The success of CNNs are largely due to its capability of learning hierarchical feature representations, from low-level features such as textures and shapes to high-level semantic features. In order to achieve that, A typical CNN architecture consists of alternate layers of the basic components, such as convolution, normalization, non-linear activation function and pooling layer, and then a linear fully-connected classifier is appended to the end (71). Sometimes, inserting some regularization purpose operators, like dropout, can help CNNs avoid overfit to training data and thus improves the generalization ability (129).

Convolution. Convolutional layer is composed of a set of convolutional kernels. Convolutional kernel is simply a small matrix of weights. And This kernel "slides" over the spatial dimension of the input, performing an elementwise multiplication with the part of the input is is currently on, and then summing up the results into s single output pixel (30). This process repeats for every location the kernel slides over, converting the input matrix to the output matrix with the same dimension. For example, in image application, 2D convolution is often used, the input and output of convolution layer is a 3D tensor ($C \times H \times W$), here C is the number of channels, H is the height, W is the width. Usually, padding is needed when one wants to maintain the output dimension to be exactly the same as the input's.

The weight sharing nature of convolutional operation ensures the model to be spatially invariant, and also make it parameter efficient as compared to fully-connected operator. Convolutional operations can be further classified into different types according to the size of the kernel, type of padding, stride size, etc.

Pooling layer. Another operator that makes CNN more translation-invariant is the pooling layer (122). Pooling layers provides an approach to down-sampling feature maps by summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively. Another thing to note is that, convolution with stride larger than one has similar functionality as the pooling operation. Many modern architectures, such as ResNet uses convolution with stride equals 2 to downsample the feature map among the stages (39; 122).

Feature Normalization. Feature Normalization is another important operation in CNN that are proposed to address the issues of internal covariance shift within feature-maps (57). Internal covariance shift is the change of scale and distribution among intermediate neurons values in each layer, which could potentially slow down the optimization or even make the system not trainable. Without feature normalization, deep networks are often hard to train and need to carefully tuning the learning rate and initialization (57). Typically, a normalization layer consists of two step, one is the normalization process with the computed mean and variance, while the second step is the affine transformation which is proved very useful for get good performance. The exact reason of why affine transformation works is still a research topic, but it kind of works as feature re-calibration (121). Different feature normalization schema differ in how the mean and variance are computed, such as Batch Normalization (BN) (57), Layer Normalization (LN) (3), Instance Normalization (IN) (141), Group Normalization (GN) (150), etc. Among them, BN is most widely used normalization operator for CNNs, and also it has been deeply analyzed in terms of how it helps optimization (121). In Chapter 4, different from other normalizations that focus how to compute the mean and variance, I will introduce Attentive Normalization (AN) that learns a mixture of affine transformations and utilizes their weighted-sum as the final affine transformation applied to re-calibrate features in an instance-specific way, instead of use a single affine transformation as other normalizations (76).

Activation function. Activation function is another crucial operation that introduce non-linearity to CNNs so that CNNs can learn complex patterns. Typical activations are sigmoid, tanh, and ReLU (99), and ReLU variants such as ELU, Leaky ReLU, PReLU, etc. One of the recently proposed activation function is "Mish", which has shown better performance than ReLU in most of the recently proposed deep networks on benchmark datasets (93).

Regularization. Dropout is the most widely used regularization for CNN architectures, which simply skipping some units or connections randomly with certain probability (129). And it is very effective for avoid the overfitting issue when training over-parameterized networks. There are other similar regularizations are proposed recently like ZoneOut (65), DropBlock (27), etc.

Fully connected layer. Fully connected layer is mostly used at the end of the network for classification purpose. Unlike pooling and convolution, it is a global operation. It takes input from feature extraction stages and globally analyses output of all the preceding layers (61).

2.3 Modern CNN architecture designs

Altough, many interesting ideas to bring advancements in CNNs have been explored such as the use of different activation and loss functions, parameter optimization, regularization, and architectural innovations. However, the major improvement in representational capacity of the deep CNN is achieved through architectural innovations.

2.3.1 Hand-crafted CNN architectures

AlexNet. The main breakthrough in CNN performance was brought by AlexNet, which showed significant performance improvement in 2012-ILSVRC (reduced error rate from 25.8 to 16.4) as compared to conventional CV techniques. AlexNet was proposed by Krizhevesky et al. (64), and the network had a very similar architecture as LeNet by Yann LeCun et al but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. In addition to this, ReLU was employed as a non-saturating activation function to improve the convergence rate by alleviating the problem of vanishing gradient to some extent. AlexNet was trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs which is the reason for why their network is split into two pipelines.

ZFNet. Not surprisingly, the ILSVRC 2013 winner was also a CNN which became known as ZFNet. It achieved a top-5 error rate of 14.8% which is now already half of the prior mentioned non-neural error rate. It was mostly an achievement by tweaking the hyper-parameters of AlexNet while maintaining the same structure with additional Deep Learning elements as discussed earlier in this essay.

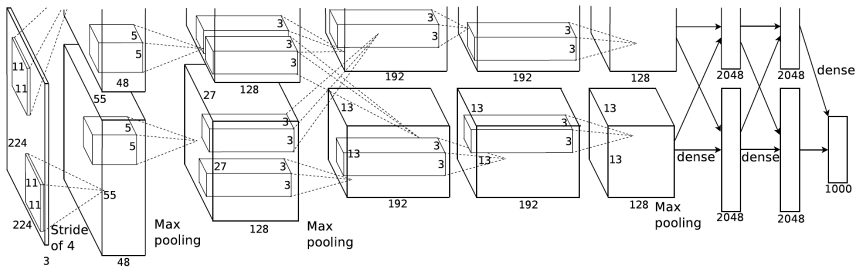


Figure 2.1: AlexNet architecture, image source (64).

GoogLeNet/Inception. In ILSVRC 2014 competition, GoogLeNet (136) won the 1st place and have significant improvement over ZFNet (2013 winner) and AlexNet (2012 winner). The new architectural component of GoogLeNet is that it contains 1x1 convolution at the middle of the network and use global average pooling before the classifier instead of using fully connected layers. These two techniques are form another work called "Network in Network" (82). The main contribution of GoogLeNet is the inception module, which is showed in Fig. 2.2. It has different sizes/types of convolutions for the same input and stacking all the outputs. The whole architecture of GoogLeNet is stack of multiple inception modules, Fig 2.3.

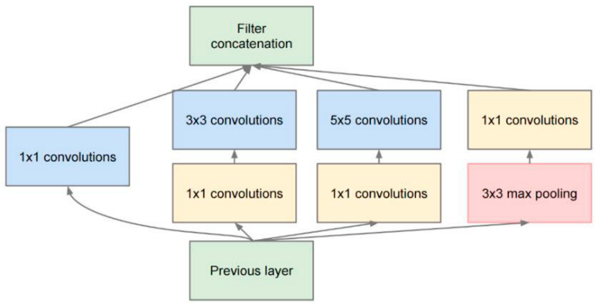


Figure 2.2: Inception module, image source (135).

VGG-Net. VGG-Net is the runner-up at the ILSVRC 2014 competition, and it was developed by Simonyan and Zisserman (10). VGGNet was born out of the need to reduce the number of parameters in the CONV layers and improve on training time. The important idea to note here is that all the convolutional kernels are of fixed size 3x3 as compared to AlexNet that use multiple kernel sizes (11x11, 5x5, 3x3). Fig. 2.4 shows the the architecture

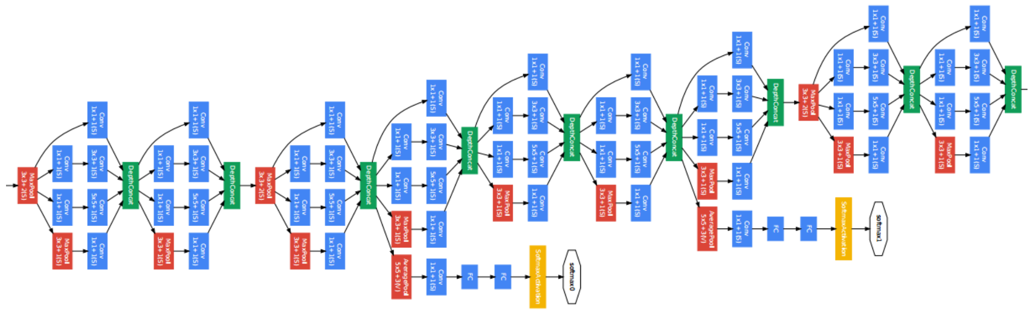


Figure 2.3: GoogLeNet architecture, image source (136).

of VGG-16 architecture.

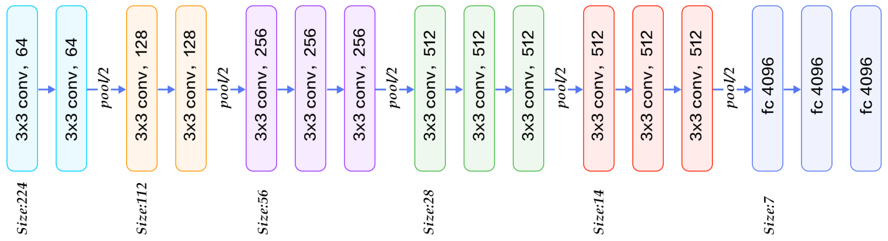


Figure 2.4: VGGNet architecture.

ResNet. Deep Residual Network (ResNet) (40) is one of the first work that successfully adopt skip connections. It provides a simple yet effective solution, inspired by the Highway network (130), that enables networks to enjoy going either deeper or wider without sacrificing the feasibility of optimization, pushes the depth of deep neural networks to hundreds of layers by using skipping connections. Without the residual connection, deep networks often suffer vanishing gradient issues and hard to optimize. The residual/skip connections allow the gradient propagate back to the base layer with shorter/clearer path. Fig. 2.5 shows the simple idea of ResNet basic blocks. The residual path element-wisely adds the input features to the output of the same block, making it a residual unit. In addition to residual connection, the ResNet work introduced a bottleneck design of basic block for better parameter efficiency. It contains three convolutional layers in the block, and the first and third with 1x1 convolution to compress and recover the channel size respectively, while the middle convolution is a 3x3 with compressed channel size (typically ratio of 0.25).

Depending on the inner structure design of the mirco-block, the residual network has

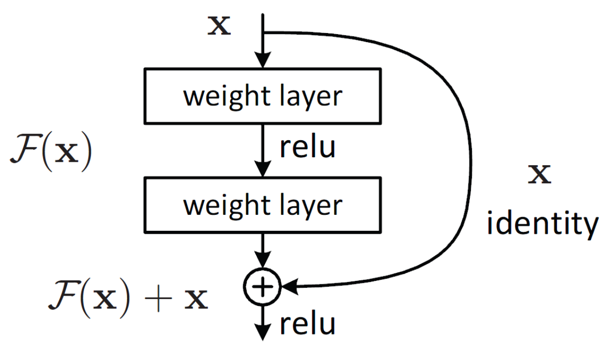

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x})+\mathbf{x}$
relu

Figure 2.5: Illustration of Residual function, image source (40).

developed into a family of various architectures, including Wide-ResNet (155), Inception-resnet (135), and ResNeXt (152), etc.

DenseNet. DenseNet (49) is introduced in CVPR2017 and won the best paper award. Different from ResNet, DenseNet uses feature-maps of all preceding layers as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. Fig. 2.6 shows an example of dense block. In a Dense-block with L layers, there are $\frac{L(L+1)}{2}$ connections. The dense connections provides more diverse information flow that helps optimization and efficiency.

Dual Path Networks (DPN) was proposed to inherits both advantages of residual and densely connected paths, enabling effective feature re-usage and re-exploitation (13). The idea is splting the input channels to two group, one goes through a ResNet-like block and the other goes through a DenseNet-like blcok, and merge them together afterwards. The proposed DPN also enjoys higher parameter and FLOPS efficiency.

SENet. Squeeze-and-Excitation Networks (SENet) (47) introduced a Squeeze and Excitation (SE) block, that explore the channel relationship and adaptively re-calibrates channel-wise feature responses by explicitly modelling inter-dependencies between channels. Fig. 2.7 shows the idea of SE Block. It uses the average-pooled output features to learn a channel-wise attention weight with a set of fully connected layers, and use the weight to re-calibrate the output features by a broadcast multiply. It shows that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different datasets. And SE blocks is demonstrated bring significant improvements in performance for existing state-of-the-art CNNs at slight additional computational cost.

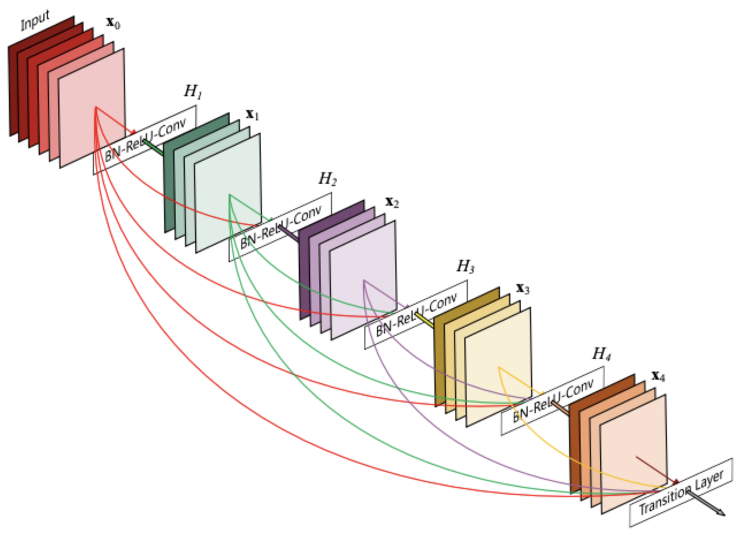


Figure 2.6: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input. Image source (49).

Squeeze-and-Excitation Networks won the 1st place of ILSVRC 2017 classification challenge.

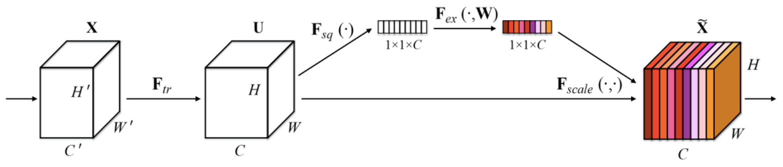


Figure 2.7: SE Block, image source (47).

MobileNet. MobileNet has several versions. MobileNet-V1 (46) replaces conventional convolutional layers with depth-wise Separable Convolution and point-wise convolution (1x1 convolution) to reduce the model size and complexity. It is particularly useful for mobile and embedded vision applications. In MobileNetV2 (120), a better module is introduced with inverted residual structure and non-linearity in narrow layers are removed.

Summary. Neural architectures are the foundation for improving performance of deep neural networks (DNNs). Much of the progress are achieved mainly through engineering network architectures which jointly address two issues: increasing representational power by going either deeper or wider, and maintaining the feasibility of optimization using back-propagation with stochastic gradient descent (i.e., the vanishing and/or exploding gradient problems). As shown in Fig. 2.8, different manually-designed architectures are compared with the most widely used ImageNet-1K benchmark (116).

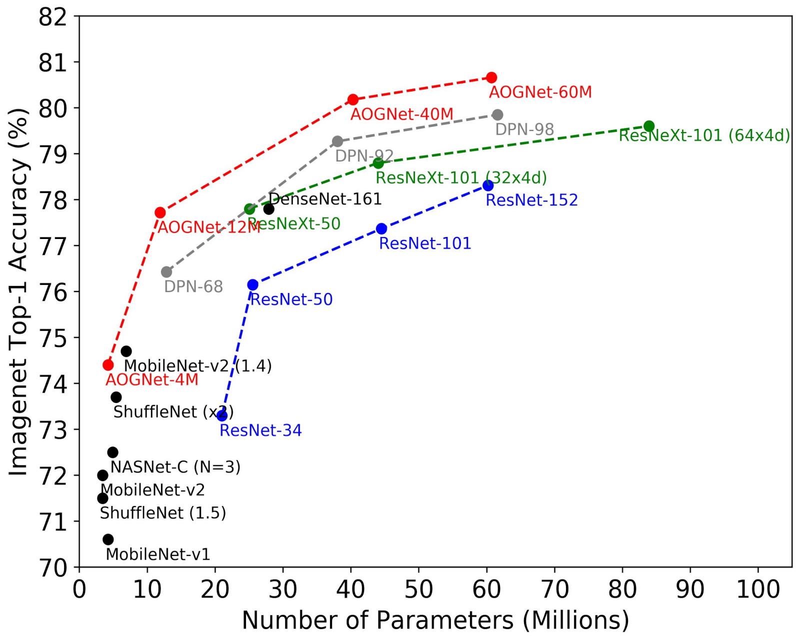


Figure 2.8: Comparison of modern neural architectures on ImageNet-1k benchmark.

In Chapter 3, I'll introduce **And-Or Grammar Networks (AOGNets)** which use a grammar-guided network generators to design networks that unified the so-far best practices of network architecture designs. And I'll demonstrate AOGNets out-performs state-of-the-art architectures on commonly used benchmarks (75).

2.3.2 Neural Architecture Search (NAS)

With network engineering, CNN architectures are improved significantly on different benchmarks across various tasks . However, deep learning techniques are computationally intensive and their application requires a high level of domain knowledge. Thus, Neural Architecture Search (NAS) idea comes in to automate the architecture design process for saving more labor and potentially improving the performance to the next level (18)(145).

In order to perform NAS, a proper search space is needed, and the space needs to contain large enough number of candidate architectures and also feasible in terms of computational time. Normally, there are two level of search space, global search space and cell search space. Global search space covers how blocks are connected and what hyper-parameters are used for each block (e.g. channel size, number of repeat, etc). Cell search spaces defines how each operators are connected within a block.

Without loss of generality, the architecture search space A is represented by a directed acyclic graph (DAG). A network architecture is a sub-graph $a \in \mathscr{A}$, denoted as $\mathscr{N}(a, w)$. with weights w. Neural architecture search aims to solve two related problems. The first is weight optimization of a given network architecture as in standard deep learning,

$$w_a = \operatorname*{argmin}_{w} \mathscr{L}_{train}(\mathscr{N}(a, w)), \tag{2.1}$$

where $\mathscr{L}_{train}(.)$ is the loss function on the training set. The second is architecture optimization. In a general sense, it finds the architecture that is trained on the training set and has best accuracy on the validation set, as

$$a^* = \operatorname*{argmax}_{a \in \mathscr{A}} ACC_{val}(\mathscr{N}(a, w_a)), \tag{2.2}$$

where $ACC_{val}(.)$ is the accuracy on the validation set.

Real world tasks usually have additional requirements on a network's memory consumption, FLOPs, latency, energy consumption, etc. These requirements are up to the architecture a, software and hardware platforms, but irrelevant to the weights w_a. Thus, they are called architecture constraints.

Early NAS approaches perform weight optimization and architecture optimization in a nested manner (164; 166; 4). Numerous architectures are sampled from $\mathscr{A}$ and trained from scratch as in Eq. 2.1. Efficient architecture search algorithms are critical to making Eq. 2.1 affordable. Previous works use reinforcement learning (RL) (164; 4) and evolutionary algorithm (EA) (166; 108; 109) for searching, however the computational cost is still very high, it takes days or weeks of searching on small dataset (e.g. CIFAR-10) with small search space.

The main reason makes the searching process slow is we need to train from scratch for each sampled architecture. To make the search process faster, recent NAS algorithms adopt a common weight sharing strategy (104; 85; 148). The architecture search space $\mathscr{A}$ is encoded in a supernet, denoted as $\mathscr{N}(\mathscr{A}, \mathscr{W})$, where W is the weights in the supernet. The supernet is trained once. All architectures inherit their weights directly from $\mathscr{W}$. Thus, they share the weights in their common graph nodes. Fine tuning of an architecture is performed in need, but no training from scratch is incurred. Therefore, architecture search is fast and suitable for large datasets like ImageNet.

Most weight sharing approaches convert the discrete architecture search space into a continuous one (85; 148). Formally, space $\mathscr{A}$ is relaxed to $\mathscr{A}(\theta)$, where θ denotes the continuous parameters that represent the distribution of the architectures in the space. An evident advantage of the continuous search space is that gradient based methods (85) become feasible for the joint optimization of both weights and architecture distribution parameters. However, there are several optimization challenges with this weight sharing strategy. The weights that compose different architectures in the supernet is strongly coupled with each other during optimization. Also, joint optimization of architecture parameter θ and weights $\mathscr{W}$ introduces further coupling. These coupling might lead to sub-optimal of searching. It is also hard to impose architectural constraint with continuous relexation of architecture parameters, only soft constraint can be applied without guarantee (148).

Recently, another category of NAS is introduced, called One-shot NAS to solve the issue of coupled optimization (8; 33). It decouples the weight optimization and architecture search processes. The supernet is trained only once and architecture search is performed based on the learned supernet.

With NAS techniques, CNN performance is further improved as compared to human-designed architectures. In Chapter 5, I'll introduce Learn-to-Grow framework (77) that utilize NAS technique to help neural network alleviate "catastrophic forgetting" issue under continual/lifelong learning settings.

CHAPTER

3

AND-OR GRAMMAR NETWORKS

Neural architectures are the foundation for improving performance of deep neural networks (DNNs). This chapter presents deep compositional grammatical architectures which harness the best of two worlds: grammar models and DNNs. The proposed architectures integrate compositionality and reconfigurability of the former and the capability of learning rich features of the latter in a principled way.

We utilize AND-OR Grammar (AOG) (128; 163; 162) as network generator and call the resulting networks **AOGNets**. An AOGNet consists of a number of stages each of which is composed of a number of AOG building blocks. An AOG building block splits its input feature map into N groups along feature channels and then treat it as a sentence of N words. It then jointly realizes a phrase structure grammar and a dependency grammar in bottom-up parsing the “sentence" for better feature exploration and reuse. It provides a unified framework for the best practices developed in state-of-the-art DNNs.

In our experiments, AOGNet is tested in the ImageNet-1K classification benchmark and the MS-COCO object detection and segmentation benchmark. In ImageNet-1K, AOGNet obtains better performance than ResNet (40) and most of its variants, ResNeXt (151) and its attention based variants such as SENet (47), DenseNet (49) and DualPathNet (13). AOGNet also obtains the best model interpretability score using network dissection (6). AOGNet further shows better potential in adversarial defense. In MS-COCO object detection and instance segmentation task, AOGNet obtains better performance than the ResNet and ResNeXt backbones with Mask R-CNN framework (36).

In the following sections, I will presents the details of the design and performance of AOGNets. In section 3.1, I will first present the motivation and objective of designing network architecture with And-Or Grammar. Section 3.2 presents some background information on existing CNN architectures and grammar models. Section 3.3 and 3.4 presents detailed designs of our AOGNets. Section 3.5 shows experimental results and comparisons with other networks, as well as ablation studies on different aspects of our AOGNets. Finally, Section 3.6 concludes this work and discusses some on-going and future work.

3.1 Motivation and Objective

Recently, deep neural networks (DNNs) (71; 64) have improved prediction accuracy significantly in many vision tasks, and have obtained superhuman performance in image classification tasks (40; 135; 49; 13). Much of these progress are achieved mainly through engineering network architectures which jointly address two issues: increasing representational power by going either deeper or wider, and maintaining the feasibility of optimization using back-propagation with stochastic gradient descent (i.e., the vanishing and/or exploding gradient problems). The dramatic success does not necessarily speak to its sufficiency, given the lack of theoretical underpinnings of DNNs at present (1). Different methodologies are worth exploring to enlarge the scope of neural architectures for seeking better DNNs. For example, Hinton recently pointed out: a crucial drawback of current convolutional neural networks: according to recent neuroscientific research, these artificial networks do not contain enough levels of structure (44; 118). Thus, in this work, we are interested in **grammar-guided network generators** (Fig. 3.1), which has rich structures due to its compositional and recurrent nature.

Neural architecture design and search can be posed as a combinatorial search problem in a product space comprising two sub-spaces (Fig. 3.2 (a)): (i) The structure space which consists of all directed acyclic graphs (DAGs) with the start node representing input raw data and the end node representing task loss functions. DAGs are entailed for feasible computation. (ii) The node operation space which consists of all possible transformation functions for implementing nodes in a DAG, such as Conv+BN (57)+ReLU (64) with different kernel sizes and feature channels.

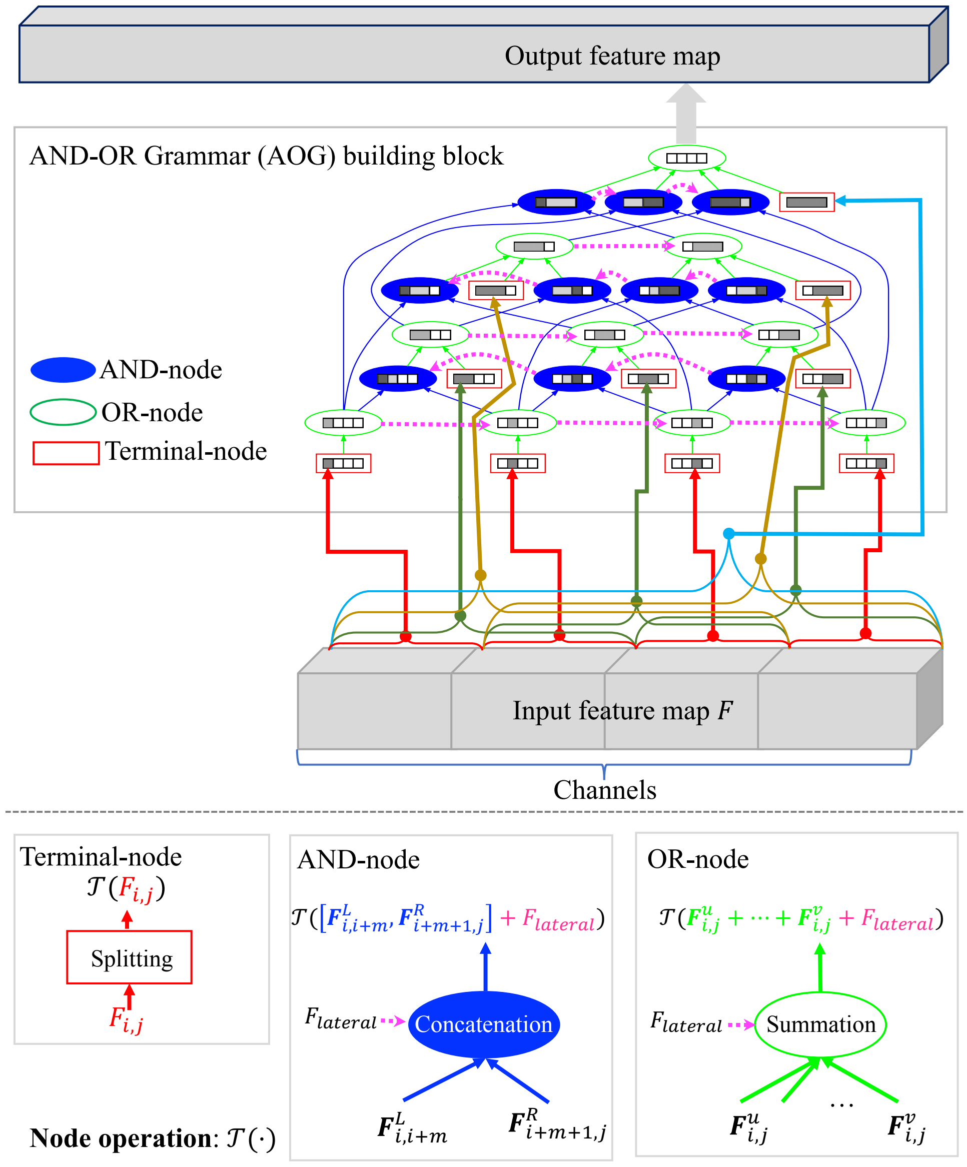

Output feature map
AND-OR Grammar (AOG) building block
AND-node
OR-node
Terminal-node
Input feature map F
Channels
Terminal-node
$\mathcal{T}(F_{i,j})$
Splitting
$F_{i,j}$
AND-node
$\mathcal{T}([\boldsymbol{F}^L_{i,i+m}, \boldsymbol{F}^R_{i+m+1,j}] + F_{lateral})$
$F_{lateral}$
Concatenation
$\boldsymbol{F}^L_{i,i+m}$
$\boldsymbol{F}^R_{i+m+1,j}$
OR-node
$\mathcal{T}(\boldsymbol{F}^u_{i,j} + \cdots + \boldsymbol{F}^v_{i,j} + F_{lateral})$
$F_{lateral}$
Summation
$\boldsymbol{F}^u_{i,j}$
$\cdots$
$\boldsymbol{F}^v_{i,j}$
Node operation: $\mathcal{T}(\cdot)$

Figure 3.1: Illustration of our AOG building block for grammar-guided network generator. The resulting networks, AOGNets obtain 80.18% top-1 accuracy with 40.3M parameters in ImageNet, significantly outperforming ResNet-152 (77.0%, 60.2M), ResNeXt-101 (79.6%, 83.9M), DenseNet-Cosine-264 (79.6%, ∼73M) and DualPathNet-98 (79.85%, 61.6M). See text for details. (Best viewed in color)

The structure space is almost unbounded, and the node operation space for a given structure is also combinatorial. Neural architecture design and search is a challenging problem due to the exponentially large space and the highly non-convex non-linear objective function to be optimized in the search. As illustrated in Fig. 3.2 (b), to mitigate the difficulty, neural architecture design and search have been simplified to design or search a building block structure. Then, a DNN consists of a predefined number of stages each of which has a small number of building blocks. This stage-wise building-block based design is also supported by the theoretical study in (1) under some assumptions. Fig. 3.2 (c) shows examples of some popular building blocks with different structures. Two questions arise naturally:

- Can we unify the best practices used by the popular building blocks in a simple and elegant framework? More importantly, can we generate building blocks and thus networks in a principled way to effectively unfold the space (Fig. 3.2 (a)) ? (If doable)
- Will the unified building block/network generator improve performance on accuracy, model interpretability and adversarial robustness without increasing model complexities and computational costs? If yes, the potential impacts shall be broad and deep for representation learning in numerous practical applications.

To address the above questions, we first need to understand the underlying wisdom in designing better network architectures: It usually lies in finding network structures which can support flexible and diverse information flows for exploring new features, reusing existing features in previous layers and back-propagating learning signals (e.g., gradients). Then, what are the key principles that we need to exploit and formulate such that we can effectively and efficiently unfold the structure space in Fig. 3.2 (a) in a way better than existing networks? *Compositionality, reconfigurability and lateral connectivity* are well-known principles in cognitive science, neuroscience and pattern theory (25; Mumford and Desolneux; Grenander and Miller; 26; 66; 26). They are fundamental for the remarkable capabilities possessed by humans, of learning rich knowledge and adapting to different environments, especially in vision and language. They have not been, however, fully and explicitly integrated in DNNs.

Thus, we presents **compositional grammatical architectures** that realize compositionality, reconfigurability and lateral connectivity for building block design in a principled way. We utilize AND-OR Grammars (AOG) (128; 163; 162) and propose AOG building blocks that unify the best practices developed in existing popular building blocks. Our method deeply integrates hierarchical and compositional grammars and DNNs for harnessing the best of both worlds in deep representation learning.

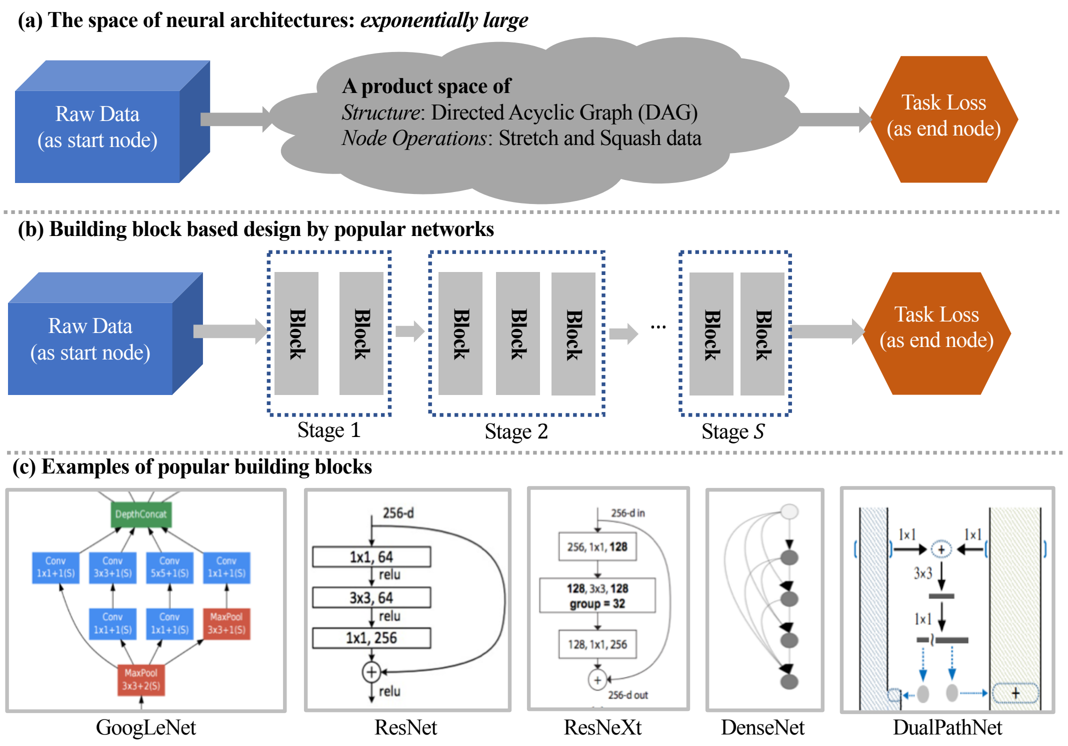


Figure 3.2: Illustration of (a) the space of neural architectures, (b) the building block based design, and (c) examples of popular building blocks in GoogLeNet (135), ResNet (40), ResNeXt (151), DenseNet (49) and DualPathNets (13). See text for details.

Why grammars? Grammar models are well known in both natural language processing and computer vision. Image grammar (163; 21; 162; 25) was one of the dominant methods in computer vision before the recent resurgence in popularity of deep neural networks. With the recent resurgence, one fundamental puzzle arises that grammar models with more explicitly compositional structures and better analytic and theoretical potential, often perform worse than their neural network counterparts. As David Mumford pointed out, "*Grammar in language is merely a recent extension of much older grammars that are built into the brains of all intelligent animals to analyze sensory input, to structure their actions and even formulate their thoughts.*" (Mumford). Our proposed AOG building block is highly expressive for analyzing sensory input and bridges the performance gap between grammars and DNNs. It also enables flexible and diverse network structures to address Hinton's quest on improving structural sufficiency in DNNs (44).

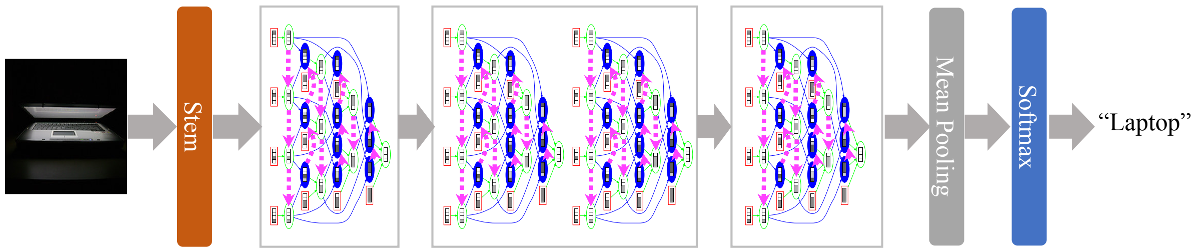


Figure 3.3: Illustration of a 3-stage AOGNet with 1 AOG building bock in the 1st and 3rd stage, and 2 AOG building blocks in the 2nd stage. Note that different stages can use different AOG building blocks. We show the same one for simplicity. The stem can be either a vanilla convolution or convolution+MaxPooling. (Best viewed in color)

3.2 Background

Hand-crafted network architectures. After more than 20 years since the seminal work 5-layer LeNet5 (71) was proposed, the recent resurgence in popularity of neural networks was triggered by the 8-layer AlexNet (64) with breakthrough performance on ImageNet (116) in 2012. Since then, a lot of efforts were devoted to learn deeper AlexNet-like networks with the intuition that deeper is better. The VGG Net (10) proposed a 19-layer network with insights on using multiple successive layers of small filters (e.g., 3×3) A special case, 1×1 convolution, was proposed in the network-in-network (82) for reducing or expanding feature dimensionality between consecutive layers, and have been widely used in many networks. The 22-layer GoogLeNet (136) introduced the first inception module and a bottleneck scheme implemented with 1×1 convolution for reducing computational cost. The main obstacle of going deeper lies in the gradient vanishing issue in optimization, which is addressed with a new structural design, short-path or skip-connection, proposed in the Highway network (131) and popularized by the ResNets (40), especially when combined with the batch normalization (BN) (57). More than 100 layers are popular design in the recent literature (40; 135), as well as even more than 1000 layers trained on large scale datasets such as ImageNet (50; 160). The Fractal Net (69) and deeply fused networks (143) provided an alternative way of implementing short path for training ultra-deep networks without residuals. Complementary to going deeper, width matters in ResNets and inception based networks too (155; 151; 159). Going beyond the first-order skip-connections in ResNets, DenseNets (49) proposed a densely connected network architecture with concatenation scheme for feature reuse and exploration, and DPNs (13) proposed to combine residuals and densely connections in an alternating way for more effective feature exploration and reuse. DLA networks (154) further develop iterative and hierarchical aggregation schema with very good performance obtained. Most work focused on boosting spatial encoding and utilizing spatial dimensionality reduction. The squeeze-and-excitation module (47) is a recently proposed simple yet effective method focusing on channel-wise encoding. The Hourglass network (97) proposed a hourglass module consisting of both subsampling and upsampling to enjoy repeated bottom-up/top-down feature exploration.

Our AOGNet is created by intuitively simple yet principled grammars. It shares some spirit with the inception module (135), the deeply fused nets (143) and the DLA (154).

Grammars. A general framework of image grammar was proposed in (163). Object detection grammar was the dominant approaches for object detection (21; 162; 128; 81; 79; 80), and has recently been integrated with DNNs (138; 139; 12). Probabilistic program induction (133; 67; 68) has been used successfully in many settings, but has not shown good performance in difficult visual understanding tasks such as large-scale image classification and object detection. More recently, recursive cortical networks (26) have been proposed with better data efficiency in learning which adopts the AND-OR grammar framework (163), showing great potential of grammars in developing general AI systems.

3.3 And-Or Grammar Network Overview

We first summarize the best practices in existing building blocks, and then briefly overview our proposed AOG building block (Fig. 3.1) and how it unifies the existing ones.

Existing building blocks usually do not fully implement the three principles (compositionality, reconfigurability and lateral connections).

- InceptionNets or GoogLeNets (135) embodies a split-transform-aggregate heuristic in a shallow feed-forward way for feature exploration, which is inspired by the network-in-network design (82) and the theoretical study on stage-wise design (1). However, the filter numbers and sizes are tailored for each individual transformation, and the modules are customized stage-by-stage. Interleaved group convolutions (159) share the similar spirit, but use simpler scheme.
- ResNets (40) provide a simple yet effective solution, inspired by the Highway network (131), that enables networks to enjoy going either deeper or wider without sacrificing the feasibility of optimization. From the perspective of representation learning, skip-connections within a ResNet (40) contributes to effective feature reuse. They do not, however, realize the split component as done in GoogLeNets.

- ResNeXts (151) add the spit component in ResNets and address the drawbacks of the Inception modules using group convolutions in the transformation.
- Deep Pyramid ResNets (34) gradually increase feature channels between building blocks, instead of increasing feature channels sharply at each residual unit with down-sampling in vanilla ResNets.
- DenseNets (49) explicitly differentiate between information that is added to the network and information that is preserved. Dense connections with feature maps being concatenated together are used, which are effective for feature exploration, but lack the capability of feature reuse as done in ResNets.
- Dual Path Networks (DPN) (13) utilize ResNet blocks and DenseNet blocks in parallel to balance feature reuse and feature exploration.
- Deep Layer Aggregation networks (DLA) (154) iteratively and hierarchically aggregate the feature hierarchy when stacking the building blocks such as the ResNet ones.

Our AOG building block is hierarchical, compositional and reconfigurable with lateral connections by design. As Fig. 3.1 shows, an AOG building block splits its input feature map into N groups along feature channels, and treat it as a sentence of N words. It then jointly realizes a phrase structure grammar (vertical composition) (24; 25; 22; 163; 162; 128) and a dependency grammar (horizontal connections in pink in Fig. 3.1) (35; 163; 26) in bottom-up parsing the "sentence" for better feature exploration and reuse: (i) Phrase structure grammar is a 1-D special case of the method presented in (128; 149). It can also be understood as a modified version of the well-known Cocke-Younger-Kasami (CYK) parsing algorithm in natural language processing according to a binary composition rule. (ii) Dependency grammar is integrated to capture lateral connections and improve the representational flexibility and power.

In an AOG building block, each node applies some basic operation $\mathscr{T}(\cdot)$ (e.g., Conv-BN-ReLU) to its input, and there are three types of nodes:

- A *Terminal-node* takes as input a channel-wise slice of the input feature map (i.e., a k-gram).
- An *AND-node* implements composition, whose input is computed by concatenating features of its syntactic child nodes, and adding the lateral connection if present.
- An *OR-node* represents alternative compositions, whose input is the element-wise sum of features of its syntactic child nodes and the lateral connection if present.

Our AOG building block unifies the best practices developed in popular building blocks in that,

- *Terminal-nodes* implement the split-transform heuristic (or group convolutions) as done in GoogLeNets (135) and ResNeXts (151), but at multiple levels (including overlapped group convolutions). They also implement the skip-connection at multiple levels. Unlike the cascade-based stacking scheme in ResNets, DenseNets and DPNs, Termninal-nodes can be computed in parallel to improve efficiency. Non-terminal nodes implement aggregation.

- *AND-nodes* implement DenseNet-like aggregation (i.e., concatenation) (49) for feature exploration.
- *OR-nodes* implement ResNet-like aggregation (i.e., summation) (40) for feature reuse.
- *The hierarchy* facilitates gradual increase of feature channels as in Deep Pyramid ResNets, and also leads to good balance between depth and width of networks. (34)
- *The compositional structure* provides much more flexible information flows than DPN (13) and the DLA (154).
- *The lateral connections* induce feature diversity and increase the effective depth of nodes along the path without introducing extra parameters.

We stack AOG building blocks to form a deep AOG network, called **AOGNet**. Fig. 3.3 illustrates a 3-stage AOGNet. Our AOGNet utilizes two nice properties of grammars: (i) The flexibility and simplicity of constructing different network structures based on a dictionary of primitives and a set of production rules in a principled way; and (ii) The highly expressive power and the parsimonious compactness of their explicitly hierarchical and compositional structures. Furthermore, the explainable rigor of grammar could be harnessed potentially to address the intepretability issue of deep neural networks.

3.4 The Architecture of AOGNets

In this section, we first present details of constructing the structure of our AOGNets. Then, we define node operation functions for nodes in an AOGNet. We also propose a method of simplifying the full structure of an AOG building block which prunes syntactically symmetric nodes.

3.4.1 The Structure of an AOGNet

An AOGNet (Fig. 3.3) consists of a predefined number of stages each of which comprises one or more than one AOG building blocks. As Fig. 3.1 illustrates, an AOG building block maps an input feature map F_{in} with the dimensions $D_{in} \times H_{in} \times W_{in}$ (representing the number of channels, height and width respectively) to an output feature map F_{out} with the dimensions

$D_{out} \times H_{out} \times W_{out}$. *We split the input feature map into N groups along feature channels, and then treat it as a "sentence of N words"*. Each "word" represents a slice of the input feature map with $\frac{D_{in}}{N} \times H_{in} \times W_{in}$. In implementation, following a common convention, we usually reduce the spatial size and increase the number of channels between consecutive stages for bigger receptive field and greater expressive power. Within a stage, we usually keep the dimensions of input and output same for the AOG building blocks except for the first one (40). Our AOG building block is constructed by a simple algorithm (Algorthm 1) which integrates two grammars.

The phrase structure grammar (24; 25; 22; 163; 162; 128). Let $S_{i,j}$ be a non-terminal symbol representing the sub-sentence starting at the i-th word ($i \in [0, N-1]$) and ending at the j-th word ($j \in [0, N-1], j \geq i$) with the length $k = j - i + 1$. We consider the following three rules in parsing a sentence:

$$\begin{aligned} S_{i,j} &\rightarrow \quad t_{i,j}, & (3.1) \\ S_{i,j}(m) &\rightarrow \quad [L_{i,i+m} \cdot R_{i+m+1,j}], \quad 0 \leq m < k, & (3.2) \\ S_{i,j} &\rightarrow \quad S_{i,j}(0)|S_{i,j}(1)|\cdots|S_{i,j}(j-i). & (3.3) \end{aligned}$$

where we have,

- The first rule is a termination rule which grounds the non-terminal symbol $S_{i,j}$ directly to the corresponding sub-sentence $t_{i,j}$, i.e., a k-gram terminal symbol, which is represented by a **Terminal-node**.
- The second rule is a binary decomposition rule, denoted by $[L \cdot R]$, which decomposes a non-terminal symbol $S_{i,j}$ into two child non-terminal symbols representing a left sub-sentence and a right sub-sentence, $L_{i,i+m}$ and $R_{i+m+1,j}$ respectively. It is represented by an **AND-node**, and entails **the concatenation scheme** in forward computation to match feature channels.
- The third rule represents alternative ways of decomposing a non-terminal symbol $S_{i,j}$, denoted by $A|B|C$, which is represented by an **OR-node**, and can utilize **summation scheme** in forward computation to "integrate out" the decomposition structures.

The dependency grammar (35; 26; 163). We introduce dependency grammar to model lateral connections between non-terminal nodes of the same type (AND-node or OR-node) with the same length k. As illustrated by the arrows in pink in Fig. 3.1, *we add lateral connections in a straightforward way*: (i) For the set of OR-nodes with $k \in [1, N-1]$, we first sort them based on the starting index i; and (ii) For the set of AND-nodes with $k \in [2, N]$, we first sort them based on the lexical orders of the pairs of starting indexes of the two child

```
Input: The total length (or primitive size) $N$.
Output: The AND-OR Graph $\mathcal{G} =< V, E >$
Initialization: Create an OR-node $O_{0,N-1}$ for the entire sentence, $V = \{O_{0,N-1}\}, E = \emptyset$, BFS
  queue $Q = \{O_{0,N-1}\}$;
while $Q$ is not empty do
    Pop a node $v_{i,j}$ from the $Q$ and let $k = j - i + 1$;
    if $v_{i,j}$ is an OR-node then
        i) Add a terminal-node $t_{i,j}$, and update $V = V \cup \{t_{i,j}\}$, $E = E \cup \{< v_{i,j}, t_{i,j} >\}$;
        ii) Create AND-nodes $A_{i,j}(m)$ for all valid splits $0 \le m < k$;
        $E = E \cup \{< v_{i,j}, A_{i,j}(m) >\}$;
        if $A_{i,j}(m) \notin V$ then
            $V = V \cup \{A_{i,j}(m)\}$;
            Push $A_{i,j}(m)$ to the back of $Q$;
        end
    else if $v_{i,j}$ is an AND-node with split index $m$ then
        Create two OR-nodes $O_{i,i+m}$ and $O_{i+m+1,j}$ for the two sub-sentence respectively;
        $E = E \cup \{< v_{i,j}(m), O_{i,i+m} >, < v_{i,j}(m), O_{i+m+1,j} >\}$;
        if $O_{i,i+m} \notin V$ then
            $V = V \cup \{O_{i,i+m}\}$;
            Push $O_{i,i+m}$ to the back of $Q$;
        end
        if $O_{i+m+1,j} \notin V$ then
            $V = V \cup \{O_{i+m+1,j}\}$;
            Push $O_{i+m+1,j}$ to the back of $Q$;
        end
    end
end
Add lateral connections (see text for detail).
```
Algorithm 1: Constructing an AOG building block

nodes. Then, we add sequential lateral connections for nodes in the sorted set either from left to right, or vice versa. We use opposite lateral connection directions for AND-nodes and OR-nodes iteratively to have globally consistent lateral flow from bottom to top in an AOG building block.

The AOG building block is constructed by the Algorithm 1 using the breadth-first search (BFS) order.

3.4.2 Node Operations in an AOGNet

In an AOG building bock, all nodes use the same type of transformation function $\mathcal{T}(\cdot)$ (see Fig. 3.1). For a node v, denote by $f_{in}(v)$ its input feature map, and then its output feature

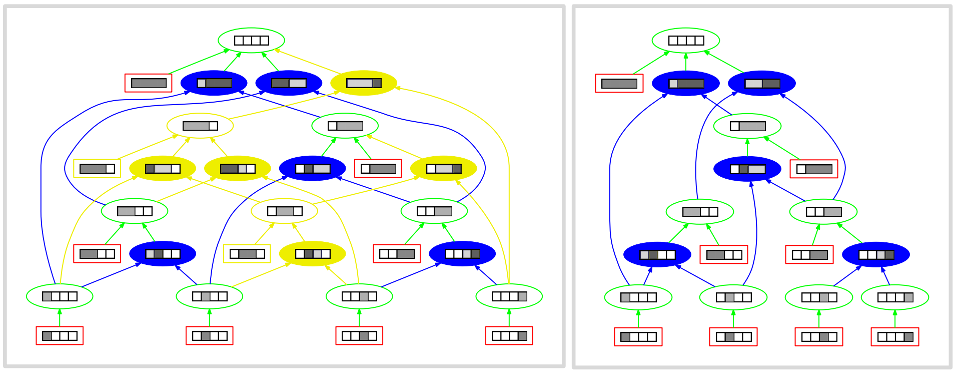


Figure 3.4: Illustration of simplifying the AOG building blocks by pruning syntactically symmetric child nodes of OR-nodes. *Left:* An AOG building block with full structure consisting of 10 Terminal-nodes, 10 AND-nodes and 10 OR-nodes. Nodes and edges to be pruned are plotted in yellow. *Right:* The simplified AOG building block consisting of 8 Terminal-nodes, 5 AND-nodes and 8 OR-nodes. (Best viewed in color)

map is computed by $f_{out}(v) = \mathcal{T}(f_{in}(v))$. For a Terminal-node t, it is straightforward to apply the transformation using $f_{in}(t) = F_{in}(t)$ where $F_{in}(t)$ is the k-gram slice from the input feature map of the AOG building block. For AND-nodes and OR-nodes, we have,

- For a Terminal-node $t_{i,j}$, denote by $F_{i,j}$ the corresponding k-gram chunk in the input feature map F. We have its input $f^t_{i,j} = F_{i,j}$ with the dimensionality $c^v_{i,j} \times H \times W$, and its output $\mathbf{f}^t_{i,j} = \mathcal{T}(F_{i,j})$ with the dimensionality $\mathbf{c}^v_{i,j} \times \mathbb{H} \times \mathbb{W}$, where $c^v_{i,j} = k \times c$ and $\mathbf{c}^v_{i,j} = k \times \frac{\mathbb{C}}{N}$.
- For an AND-node A with two child nodes L and R, its input $f_{in}(A)$ is first computed by the concatenation of the outputs of the two child nodes, $f_{in}(A) = [f_{out}(L) \cdot \lambda_L, f_{out}(R) \cdot \lambda_R]$. If it has a lateral node whose output is denoted by $f_{out}(A_{lateral})$, we add it and get $f_{in}(A) = [f_{out}(L) \cdot \lambda_L, f_{out}(R) \cdot \lambda_R] + f_{out}(A_{lateral}) \cdot \lambda_{lateral}$.
- For an OR-node O, its input is the summation of the outputs of its child nodes (including the lateral node if present), $f_{in}(O) = \sum_{u \in ch(O)} f_{out}(u) \cdot \lambda_u$, where $ch(\cdot)$ represents the set of child nodes.

Where $\lambda_L, \lambda_R, \lambda_{lateral}$ and λ_u's are weights (see details in Section 3.5.1). Node inputs are computed following the syntactical structure of AOG building block to ensure that feature dimensions and spatial sizes match in the concatenation and summation. In learning and inference, we follow the depth-first search (DFS) order to compute nodes in an AOG building block, which ensures that all the child nodes have been computed when we compute a node v.

3.4.3 Simplifying AOG Building Blocks

The phrase structure grammar is syntactically redundant since it unfolds all possible configurations w.r.t. the binary composition rule. In representation learning, we also want to increase the feature dimensions of different stages in a network for better representational power, but try not to increase the total number of parameters significantly.

To balance the structural complexity and the feature dimensions of our AOG building block, we propose to simplify the structure of an AOG building block by pruning some syntactically redundant nodes. As illustrated in Fig. 3.4, the pruning algorithm is simple: Given a full structure AOG building block, we start with an empty simplified block. We first add the root OR-node into the simplified block. Then, we follow the BFS order of nodes in the full structure block. For each encountered OR-node we only keep the child nodes which do not have left-right syntactically symmetric counterparts in the current set of child nodes in the simplified block. For encountered AND-nodes and Terminal-nodes, we add them to the simplified block. The pruning algorithm can be integrated into Algorithm 1. For example, consider the four child nodes of the root OR-node in the left of Fig. 3.4, the fourth child node is removed since it is symmetric to the second one.

3.5 Experiments

We test our AOGNet on three highly competitive image classification benchmarks: CIFAR-10 and CIFAR-100 (63), and ImageNet-1K (116). For testing the transfer learning ability of our ImageNet-pretrained AOGNets, we also tested our AOGNets as backbone for object detection and instance segmentation tasks on the PASCAL VOC 2007 and 2012 benchmarks (19) and the MS-COCO object detection and segmentation benchmark (84). We implemented our AOGNets using PyTorch. [1]

3.5.1 Implementation Settings and Details

We use simplified AOG building blocks. For the node operation $\mathscr{T}()$, we use the bottleneck variant of Conv-BN-ReLU proposed in ResNets (40), which adds one 1×1 convolution before and after the operation to first reduce feature dimension and then expand it back. More specifically, we have $\mathscr{T}(x) = ReLU(x + T(x))$ for an input feature map x where $T()$

[1]Our code based on PyTorch has been made publicly available at `https://github.com/iVMCL/AOGNets`.

represents a sequence of primitive operations, Conv1x1-BN-ReLU, Conv3x3-BN-ReLU and Conv1x1-BN. If Dropout (64) is used with drop rate $p \in (0,1)$, we add it after the last BN, i.e., $\mathscr{T}(x) = ReLU(x + Dropout(T(x), p))$

Handling double-counting due to the compositional DAG structure and lateral connections. First, in our AOG building block, some nodes will have multiple paths to reach the root OR-node due to the compositional DAG structure. Since we use the skip connection in the node operation $\mathscr{T}()$, the feature maps of those nodes with multiple paths will be double-counted at the root OR-node. Second, if a node v and its lateral node $v_{lateral}$ share a parent node, we also need to handle double-counting in the skip connection. Denote by $n(v)$ the number of paths between v and the root OR-node, which can be counted during the building block construction (Algorithm 1). Consider an AND-node A with two syntactic child node L and R and the lateral node $A_{lateral}$, we compute two different inputs, one for the skip connection, $f_{in}^{skip}(A) = [f_{out}(L) \cdot \frac{n(A)}{n(L)}, f_{out}(R) \cdot \frac{n(A)}{n(R)}]$ if A and $A_{lateral}$ share a parent node and $f_{in}^{skip}(A) = [f_{out}(L) \cdot \frac{n(A)}{n(L)}, f_{out}(R) \cdot \frac{n(A)}{n(R)}] + f_{out}(A_{lateral}) \cdot \frac{n(A)}{n(A_{lateral})}$ otherwise, and the other for $T()$, $f_{in}^{T}(A) = [f_{out}(L), f_{out}(R)] + f_{out}(A_{lateral})$. The transformation for node A is then implemented by $\mathscr{T}(A) = ReLU(f^{skip}(A) + T(f_{in}^{T}(A)))$. Similarly, we can set λ_u's in the OR-node operation. We note that we can also treat λ's as unknown parameters to be learned end-to-end.

let x be the input computed from the syntactic child nodes and $x_{lateral}$ be the output from the lateral node. The input for node v is $x = x_{ch} + x_{lateral}$. If v and its lateral node share a parent node, we omit $x_{lateral}$ in the operation of the skip connection and we implement the node operation for v by $\mathscr{T}(x) = ReLU(x_{ch} + T(x))$.

We notice that we can apply different AOG building blocks and node operations for different types of nodes as long as we can match the dimensions during the computation. We keep them simple in this paper. We leave the exploration of different operators in future work.

3.5.2 Experiments on CIFAR

CIFAR-10 and CIFAR-100 datasets (63), denoted by C10 and C100 respectively, consist of 32×32 color images drawn from 10 and 100 classes. The training and test sets contains $50,000$ and $10,000$ images respectively. We adopt widely used standard data augmentation scheme, random cropping and mirroring, in preparing the training data.

We train AOGNets with stochastic gradient descent (SGD) for 300 epochs with random parameter initialization. The front-end (see Fig. 3.3) uses a single convolution layer. The

Table 3.1: Error rates (%) on the two CIFAR datasets (63). #Params uses the unit of Million. k in DenseNet refers to the growth rate.

Method	Depth	#Params	FLOPs	C10	C100
ResNet (40)	110	1.7M	0.251G	6.61	-
ResNet (reported by (50))	110	1.7M	0.251G	6.41	27.22
ResNet (pre-activation) (42)	164	1.7M	0.251G	5.46	24.33
	1001	10.2M	-	4.62	22.71
Wide ResNet (155)	16	11.0M	-	4.81	22.07
DenseNet-BC (49) ($k=12$)	100	0.8M	0.292G	4.51	22.27
AOGNet-4-(1,1,1)-252d	74	**0.78M**	**0.123G**	**4.37**	**20.95**
DenseNet-BC (49) ($k=24$)	250	15.3M	5.46G	3.62	17.60
AOGNet-4-(1,1,1)-1152d	98	15.8M	**2.4G**	**3.42**	**16.93**
Wide ResNet (155)	28	36.5M	5.24G	4.17	20.50
FractalNet (69)	21	38.6M	-	5.22	23.30
with Dropout/DropPath	21	38.6M	-	4.60	23.73
ResNeXt-29, 8×64d (151)	29	34.4M	3.01G	3.65	17.77
ResNeXt-29, 16×64d (151)	29	68.1M	5.59G	3.58	17.31
DenseNet-BC (49) ($k=40$)	190	25.6M	9.35G	3.46	17.18
AOGNet-4-(1,1,1)-1444d	98	**24.8M**	3.7G	**3.27**	**16.63**

initial learning rate is set to 0.1, and is divided by 10 at 150 and 225 epoch respectively. For CIFAR-10, we chose batch size 64 with weight decay 1×10^{-4}, while batch size 128 with weight decay 5×10^{-4} is adopted for CIFAR-100. The momentum is set to 0.9.

The depth of an AOGNet is defined by the largest number of units which have learnable parameters along the paths from the final output to the input data following BFS order. *E.g.*, the longest path in the simplified AOG building block in Fig. 3.4 is 8, and a bottleneck operation is counted as 3 units, so the depth of the simplified AOG building block is counted as 24. In comparison, to indicate the specifications, AOGNets will be written by AOGNet-*PrimitiveSize-(#AOG blocks per stage)*-[*OutputFeatDim*]. *E.g.*, AOGNet-4-(1,1,1,1)-256d represents a 4-stage AOGNet with 1 AOG building block per stage, primitive size being 4, and the final output feature dimension 256.

After the AOG structure is specified, the number of parameters of an AOGNet is determined by the number of channels of input/output of each stage. Thus, for an M-stage AOGNet, we have an $(M+1)$-tuple specifying the number of channels. For example, we can specify the 4-tuple, $(16,16,32,64)$ or $(16,32,64,128)$ for a 3-stage AOGNet, resulting different number of parameters in total.

Results and Analyses. We summarize the results in Table 3.1. With smaller model sizes

Table 3.2: An ablation study of our AOGNets using the mean error rate across 5 runs. In the first two rows, the AOGNets use full structure, and the pruned structure in the last two rows. The feature dimensions of node operations are accordingly specified to keep model sizes comparable.

Method	#Params	FLOPS	CIFAR10	CIFAR100
AOGNet	4.24M	0.65G	3.75	19.20
AOGNet+LC	4.24M	0.65G	3.70	19.09
AOGNet+RS	4.23M	0.70G	3.57	18.64
AOGNet+RS+LC	4.23M	0.70G	**3.52**	**17.99**

and much reduced computing complexity (FLOPs), our AOGNets obtain better performance than ResNets (40) and some of the variants, ResNeXts (151) and DenseNets (49) consistently on both datasets. Our small AOGNet ($0.78M$) already outperforms the ResNet (40) ($10.2M$) and the WideResNet (155) ($11.0M$). Since the same node operation is used, the improvement must come from the AOG building block structure. Compared with the DenseNets, our AOGNets improve more on C100, and use less than half FLOPs for comparable model sizes. The reason for the reduced FLOPs is that DenseNets apply down-sampling after each Dense block, while our AOGNets sub-sample at Terminal-nodes.

Ablation Study We conduct an ablation study which investigates the effects of *(i) RS:* Removing Symmetric child nodes of OR-nodes in the pruned AOG building blocks, and of *(ii) LC:* adding Lateral Connections. As Table 3.2 shows, the two components, RS and LC, improve performance. The results are consistent with our design intuition and principles. The RS component facilitates higher feature dimensions due to the reduced structural complexity, and the LC component increases the effective depth of nodes on the lateral flows.

3.5.3 Image Classification in ImageNet-1K

The ILSVRC 2012 classification dataset (116) consists of about 1.2 million images for training, and $50,000$ for validation, from $1,000$ classes. We adopt the same data augmentation scheme (random crop and horizontal flip) for training images as done in (40; 49), and apply a single-crop with size 224×224 at test time. Following the common protocol, we evaluate the top-1 and top-5 classification error rates on the validation set.

Model specifications. We test three AOGNets with different model complexities. In comparison, we use the model size as the name tag for AOGNets (e.g., AOGNet-12M means the AOGNet has 12 million parameters or so). The stem (see Fig. 3.3) uses three Conv3x3-BN layers (with stride 2 for the first layer), followed by a 2×2 max pooling layer with stride 2. All the three AOGNets use four stages. Within a stage, we use the same AOG building block, while different stages may use different blocks. A stage is then specified by N_n where N is primitive size (Algorithm 1) and n the number of blocks. The filter channels are defined by a 5-tuple for specifying the input and output dimensions for the 4 stages. The detailed specifications of the three AOGNets are: AOGNet-12M uses stages of $(2_2, 4_1, 4_3, 2_1)$ with filter channels $(32, 128, 256, 512, 936)$, AOGNet-40M uses stages of $(2_2, 4_1, 4_4, 2_1)$ with filter channels $(60, 240, 448, 968, 1440)$, and AOGNet-60M uses stages of $(2_2, 4_2, 4_5, 2_1)$ withe filter channels $(64, 256, 512, 1160, 1400)$.

Training Settings. *i) Common training Setup.*We use 8 GPUs (NVIDIA V100) to train models using the same settings for apple-to-apple comparisons. The method proposed in (38) is used to initialize all convolutions for all models. The batch size is 128 per GPU [2] with FP16 optimization used in training to reduce the training time [3]. The mean and standard deviation for block-wise standardization are computed *within* each GPU. The initial learning rate is 0.4, and the cosine learning rate scheduler (87) is used with 5 warm-up epochs and weight decay 1×10^{-4} and momentum 0.9. In addition to the common settings, we have two different setups in experimental comparisons:

ii) The Vanilla Setup. We adopt the basic data augmentation scheme (random crop and horizontal flip) in training as done in (40). We train the models for 120 epochs. All ResNets (40) use the vanilla stem layer with 7×7 convolution. The MobileNets-v2 uses 3×3 convolution in the stem layer. The AOGNets use two consecutive 3×3 convolution in the stem layer. All the γ and β parameters of the feature normalization backbones are initialized to 1 and 0 respectively.

iii) The State-of-the-Art Setup. There are different aspects in the vanilla setup which have better variants developed with better performance shown (43). *We want to address whether the improvement by our proposed AN are truly fundamental or will disappear with more advanced tips and tricks added in training ConvNets.* First, on top of the basic data augmentation, we also use label smoothing (137) (with rate 0.1) and the mixup (with rate 0.2) (157). We increase the total number of epochs to 200. We use the same stem layer with two consecutive 3×3 convolution for all models. We add the zero γ initialization trick for the feature normalization backbones (not for AN), which uses 0 to initialize the last

[2]Although the best practice is to use 8 GPUs and batch size 32 per GPU for training ConvNets in ImageNet, we do not follow it since it would take too long in order to run all the experiments given the resources we have.

[3]The NVIDIA APEX library is used.

Table 3.3: Comparisons of the top-1 and top-5 error rates (%) in the ImageNet-1000 validation set using *the vanilla setup.*

Method	#Params	FLOPS	top-1	top-5
ResNet-101 (40)	44.5M	8G	23.6	7.1
ResNet-152 (40)	60.2M	11G	23.0	6.7
ResNeXt-50 (151)	25.03M	4.2G	22.2	5.6
ResNeXt-101 (32×4d) (151)	44M	8.0G	21.2	5.6
ResNeXt-101 (64×4d) (151)	83.9M	16.0G	20.4	5.3
ResNeXt-101 + BAM (100)	44.6M	8.05G	20.67	-
ResNeXt-101 + CBAM (147)	49.2M	8.0G	20.60	-
ResNeXt-50+SE (47)	27.7M	4.3G	21.1	5.49
ResNeXt-101+SE (47)	48.9M	8.46G	20.58	5.01
DensetNet-161 (49)	27.9M	7.7G	22.2	-
DensetNet-169 (49)	$\sim 13.5M$	$\sim 4G$	23.8	6.85
DensetNet-264 (49)	$\sim 33.4M$	-	22.2	6.1
DensetNet-cosine-264 (105)	$\sim 73M$	$\sim 26G$	20.4	-
DPN-68 (13)	12.8M	2.5G	23.57	6.93
DPN-92 (13)	38.0M	6.5G	20.73	5.37
DPN-98 (13)	61.6M	11.7G	20.15	5.15
AOGNet-12M	11.9M	2.36G	22.28	6.14
AOGNet-40M	40.3M	8.86G	19.82	4.88
AOGNet-60M	60.7M	14.36G	**19.34**	**4.78**

normalization layer (BN$_3$ in Figure 4.4) to make the initial state of a residual block to be identity.

Results and Analyses: Table 4.2 shows the results, and Fig. 3.5 shows plots for the top-1 error rates and training losses. Our AOGNets are the best among the models with comparable model sizes in comparison in terms of top-1 and top-5 accuracy. Our small AOGNet-12M outperforms ResNets (40) (44.5M and 60.2M) by 1.32% and 0.72% respectively. *We note that our AOGNets use the same bottleneck operation function as ResNets, so the improvement must be contributed by the AOG building block structure.* Our AOGNet-40M obtains better performance than all other methods in comparison, including ResNeXt-101 (151)+SE (47) (48.9M) which represents the most powerful and widely used combination in practice. AOGNet-40M also obtains better performance than the runner-up, DPN-98 (13) (61.6M), which indicates that the hierarchical and compositional integration of the DenseNet- and ResNet-aggregation in our AOG building block is more effective than the cascade-based integration in the DPN (13). Our AOGNet-60M achieves the best results. The FLOPs of our AOGNet-60M are slightly higher than DPN-98 partially because DPN uses ResNeXt

Table 3.4: Comparisons of the top-1 and top-5 error rates (%) in the ImageNet-1000 validation set using *the state-of-the-art setup*. The numbers in brackets show the performance improvement by our proposed AN over the baselines. All models are trained from scratch under the same settings. See text for details.

Method	#Params	FLOPS	top-1	top-5
ResNet-50	25.56M	4.09G	21.08	5.56
ResNet-101	44.57M	8.12G	19.71	4.89
AOGNet-12M	12.26M	2.19G	21.63	5.60
AOGNet-40M	40.15M	7.51G	18.70	4.47

Table 3.5: An ablation study of our AOGNets using different OR-Node merging scheme: elementwise-um, elementwise-average or elementwise-max. This results are preformed with **AOGNet-12m with pruned structure without lateral connections**.

Method	top-1	top-5
elem-sum	**22.59**	**6.12**
elem-avg	**22.60**	**6.14**
elem-max	23.10	6.50

operation (i.e., group conv.).

Mobile settings. We train an AOGNet-4M under the typical mobile settings (46). Table 3.6 shows the comparison results. We obtain performance on par to the popular networks specifically designed for mobile platforms such as the MobileNets (46; 120) and ShuffleNets (161). Our AOGNet also outperforms the auto-searched network, NASNet (166) (which used around 800 GPUs in search). *We note that we use the same AOGNet structure, thus showing promising device-agnostic capability of our AOGNets.* This is potentially important and useful for deploying DNNs to different platforms in practice since no extra efforts of hand-crafting or searching neural architectures are entailed.

3.5.4 Experiment on Model Interpretability

Interpretability has been recognized as a critical concern in developing deep learning based AI systems (DARPA). We use the network dissection metric (6) which compares the number of unique “detectors" (i.e., filter kernels) in the last convolution layer. Our AOGNet obtains the best score in comparison (Fig. 3.10), which indicates the AOG building block has great potential to induce model interpretabilty by design, while achieving the best accuracy

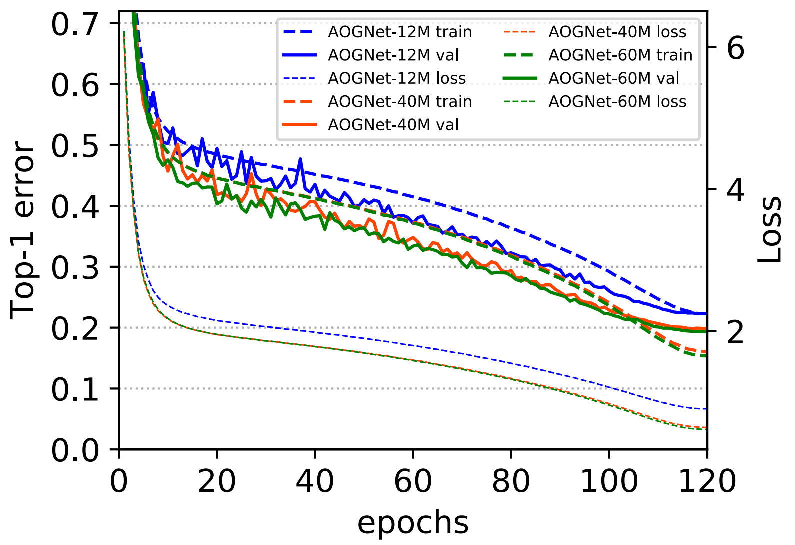


Figure 3.5: Plots of top-1 error rates and training losses of the three AOGNets in ImageNet. (Best viewed in color and magnification)

Table 3.6: The top-1 and top-5 error rates (%) on the ImageNet-1K validation set under mobile settings.

Method	#Params	FLOPS	top-1	top-5
MobileNetV1 (46)	4.2M	575M	29.4	10.5
SqueezeNext (28)	4.4M	-	30.92	10.6
ShuffleNet (1.5) (161)	3.4M	292M	28.5	-
ShuffleNet (x2) (161)	5.4M	524M	26.3	-
CondenseNet (G=C=4) (48)	4.8M	529M	26.2	8.3
MobileNetV2 (120)	3.4M	300M	28.0	9.0
MobileNetV2 (1.4) (120)	6.9M	585M	25.3	7.5
NASNet-C (N=3) (166)	4.9M	558M	27.5	9.0
AOGNet-4M	4.2M	557M	25.6	7.91

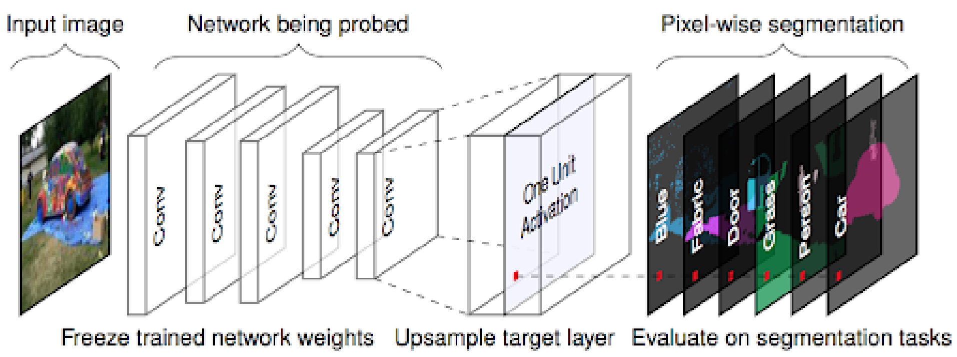


Figure 3.6: Network dissection (6), image source from (6).

Table 3.7: Top-1 accuracy comparisons under white-box adversarial attack using 1-step FGSM (31) with the Foolbox toolkit (107).

Method	#Params	$\epsilon = 0.1$	$\epsilon = 0.3$	clean
ResNet-101	44.5M	12.3	0.40	77.37
ResNet-152	60.2M	16.3	0.85	78.31
DenseNet-161	28.7M	13.0	2.1	77.65
AOGNet-12M	12.0M	18.1	1.4	77.72
AOGNet-40M	40.3M	28.3	2.2	80.18
AOGNet-60M	60.1M	30.2	2.6	80.66

performance.

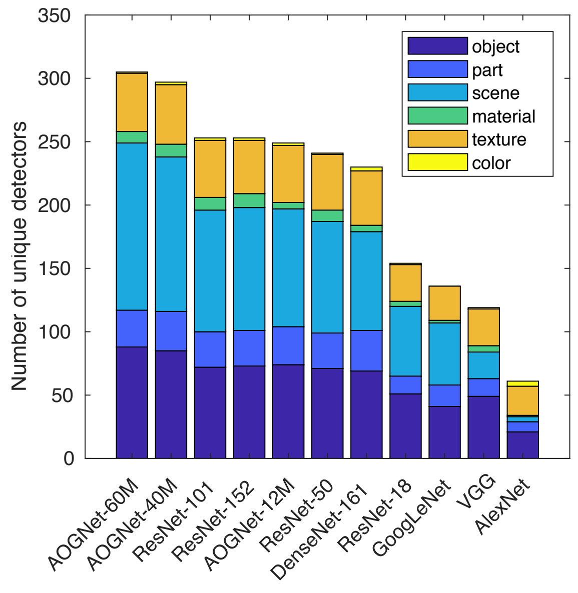
3.5.5 Experiment on Adversarial robustness

Adversarial attack is another crucial issue faced by many DNNs (2). We conduct a simple experiment to compare the out-of-the-box adversarial robustness of different DNNs. Table 3.7 shows the results. Under the vanilla settings, our AOGNets show better potential in adversarial defense, especially when the perturbation energy is controlled relatively low (i.e. $\epsilon = 0.1$). We will investigate this with different attacks and adversarial training in future work.

3.5.6 Object Detection in Pascal VOC

We test our AOGNets in object detection on the PASCAL VOC 2007 and 2012 datasets (19). We adopt the vanilla Faster R-CNN system (113) and reuse the code in PyTorch [4]. We only

[4]`https://github.com/jwyang/faster-rcnn.pytorch`



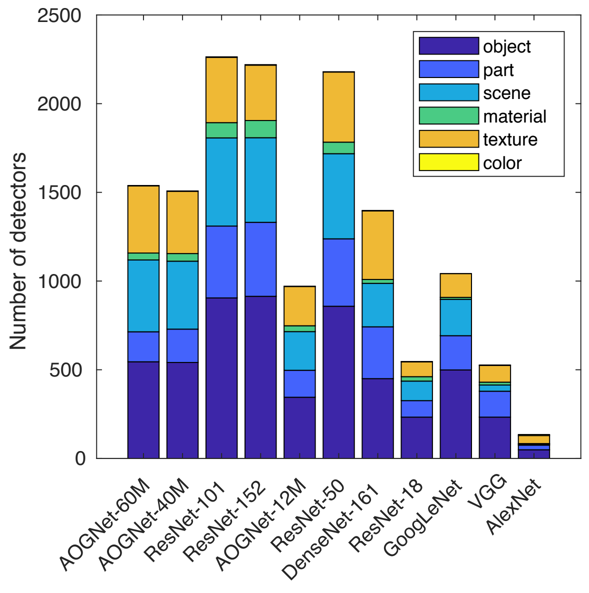


Figure 3.7: Comparisons of model interpretability using the network dissection method (6) on ImageNet pretrained networks.

substitute the ConvNet backbone with our AOGNets in experiments and keep everything else unchanged for fair comparison. We finetue the AOGNets pretrained on ImageNet. We adopt the provided end-to-end training procedure to train the region proposal network (RPN) and R-CNN jiontly. The first three stages are shared by RPN and R-CNN, and the last stage is used as the head classifier for region-of-interest (RoI) prediction. We fix all parameters pretrained on ImageNet before stage 1 (inclusive) in training. We follow the standard evaluation metrics Average Precision (AP) and mean of AP (mAP) in evaluation (19). Table 3.8 shows the detection results and comparisons. Our AOGNets obtain better mAP than ResNet-101 by around 3% consistently. When trained using the 07+12 trainval dataset, our small AOGNet-backboned detector ($13.6M$) already slightly outperforms the ResNet-backboned one ($47.5M$), which further shows the effectiveness of the AOG building blocks in object detection tasks.

3.5.7 Object Detection and Segmentation in COCO

MS-COCO is one the widely used benchmarks for object detection and segmentation (84). It consists of 80 object categories. We train AOGNet in the COCO `train2017` set and evaluate in the COCO `val2017` set. We report the standard COCO metrics of Average Precision (AP), AP_{50}, and AP_{75}, for bounding box detection (AP^{bb}) and instance segmentation, i.e. mask

Table 3.8: Performance comparisons using Average Precision (AP) at the intersection over union (IoU) threshold 0.5 (AP@0.5) in the PASCAL VOC2007 / 2012 dataset. * reported based on our re-implementation using the exactly same PyTorch implementation of Faster R-CNN and PyTorch pretrained ResNet-101 backbone on ImageNet for fair comparisons. The reproduced results of ResNets are better than those reported in the original paper (40).

	07trainval/07test			07+12trainval/12test		
model	ResNet-101*	AOGNet-12M	AOGNet-40M	ResNet-101*	AOGNet-12M	AOGNet-40M
#params	47.5M	13.6M	43.2M	47.5M	13.6M	43.2M
areo	76.2	76.0	77.2	86.1	87.0	88.9
bike	77.9	77.4	84.4	82.5	82.1	82.8
bird	74.6	72.6	78.0	77.6	78.4	81.3
boat	59.9	59.2	64.5	63.4	63.1	66.9
bottle	53.0	55.6	60.6	54.5	56.7	62.4
bus	80.8	80.4	83.5	80.6	80.0	82.6
car	81.7	83.7	87.2	79.9	80.7	83.0
cat	85.3	85.2	85.8	91.0	91.5	92.5
chair	49.0	50.1	55.4	55.8	55.1	59.6
cow	80.2	77.1	85.1	79.9	79.7	82.5
table	64.1	63.7	65.7	56.0	59.9	61.9
dog	83.7	83.8	86.0	89.5	88.8	90.9
horse	83.3	82.8	84.5	82.6	83.8	86.1
mbike	76.5	78.7	80.3	83.3	82.8	84.5
person	77.8	77.8	78.9	83.1	83.4	84.5
plant	45.2	44.7	52.4	53.9	54.3	56.1
sheep	73.0	71.4	75.5	79.8	79.0	83.3
sofa	72.0	72.5	72.6	67.4	65.8	69.7
train	81.7	79.1	82.6	86.3	84.7	87.6
tv	71.3	70.6	75.9	69.5	67.4	71.5
mean AP	72.3	72.1	**75.8**	75.1	75.2	**78.0**

prediction (AP^m). We experiment on the Mask-RCNN system (36) using the state-of-the-art implementation, `maskrcnn-benchmark` (91). We use AOGNets pretrained on ImageNet-1K as the backbones. In fine-tuning for object detection and segmentation, we freeze all the BN parameters as done for the ResNet (40) and ResNeXt (151) backbones. We keep all remaining aspects unchanged. We test both the `C4` and `FPN` settings.

Results. Table 4.4 shows the comparison results. Our AOGNets obtain better results than the ResNet (40) and ResNeXt (151) backbones with smaller model sizes and similar or slightly better inference time. The results show the effectiveness of our AOGNets learning better features in object detection and segmentation tasks.

Table 3.9: Mask-RCNN results on *coco_val2017* using the 1x training schedule. Results of ResNets and ResNeXts are reported by the maskrcnn-benchmark.

Method	#Params	t (s/img)	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{m}	AP^{m}_{50}	AP^{m}_{75}
ResNet-50-C4	35.9M	0.130	35.6	56.1	38.3	31.5	52.7	33.4
ResNet-101-C4	54.9M	0.180	39.2	59.3	42.2	33.8	55.6	36.0
AOGNet-12M-C4	14.6M	0.092	36.8	56.3	39.8	32.0	52.9	33.7
AOGNet-40M-C4	48.1M	0.184	**41.4**	**61.4**	**45.2**	**35.5**	**57.8**	**37.7**
ResNet-50-FPN	44.3M	0.125	37.8	59.2	41.1	34.2	56.0	36.3
ResNet-101-FPN	63.3M	0.145	40.1	61.7	44.0	36.1	58.1	38.3
ResNeXt-101-FPN	107.4M	0.202	42.2	63.9	46.1	37.8	60.5	40.2
AOGNet-12M-FPN	31.2M	0.122	38.0	59.8	41.3	34.6	56.6	36.4
AOGNet-40M-FPN	59.4M	0.147	41.8	63.9	45.7	37.6	60.3	40.1
AOGNet-60M-FPN	78.9M	0.171	**42.5**	**64.4**	**46.7**	**37.9**	**60.9**	**40.3**

3.5.8 Experiment on Chest X-Ray datasets for Medical Image Analysis

3.6 Summary and Discussion

We proposes grammar-guided network generators which construct compositional grammatical architectures for deep learning in an effective way. It presents deep AND-OR Grammar networks (AOGNets). The AOG comprises a phrase structure grammar and a dependency grammar. An AOGNet consists of a number of stages each of which comprises a number of AOG building blocks. Our AOG building block harnesses the best of grammar models and DNNs for deep learning. AOGNet obtains state-of-the-art performance. In experiments, our AOGNet is tested in the ImageNet-1K classification benchmark (116) and the MS-COCO object detection and segmentation benchmark (84). In ImageNet-1K (116), AOGNet obtains better performance than all state-of-the-art networks under fair comparisons. AOGNet also obtains the best model interpretability score using network dissection (6). AOGNet further shows better potential in adversarial defense. In MS-COCO (84), AOGNet obtains better performance than the ResNet and ResNeXt backbones in Mask R-CNN (36).

We hope this work encourages further exploration in integrating compositional grammar models and other structured knowledge representation and deep neural networks end-to-end, especially to harness the explainable rigor of the former and the discriminative power of the latter. In implementation, some interesting aspects that worth investigating include, but not limited to, searching better hyper-parameters in learning AOGNets (e.g., learning rate and schedule, parameter initialization and batch size, etc), conducting

Table 3.10: Results on 14 pathologies in the ChestX-ray14 dataset. CheXNet is using a DenseNet-121 architecture. * CheXNet implementation with PyTorch reported by https://github.com/zoogzog/chexnet.

Pathology	CheXNet	CheXNet*	AOGNet-12M	AOGNet-40M
Athelectasis	0.8094	0.8321	0.8357	**0.8375**
Cardiomegaly	**0.9248**	0.9107	0.9121	0.9186
Effussion	0.8638	0.8860	0.8870	**0.8906**
Infiltration	**0.7345**	0.7145	0.7164	0.7211
Mass	0.8676	0.8653	**0.8734**	0.8714
Nodule	0.7802	0.8037	0.8071	**0.8160**
Pneumonia	0.7680	0.7655	**0.7752**	0.7740
Pneumothorax	0.8887	0.8857	0.8925	**0.8961**
Consolidation	0.7901	0.8157	0.8176	**0.8188**
Edema	0.8878	0.9017	0.9043	**0.9074**
Emphysema	0.9371	0.9422	0.9404	**0.9477**
Fibrosis	0.8047	0.8523	0.8535	**0.8632**
Pleural Thickening	**0.8062**	0.7948	0.7989	0.8059
Hernia	0.9164	**0.9416**	0.9185	0.9287
Mean AUROC	0.8414	0.8508	0.8523	**0.8569**

experiments with much larger AOGNets (deeper and wider with different primitive size per stage), extending binary composition rule in constructing AND-OR graphs, adopting sub-graph based concatenation scheme for AND-nodes with predefined growth rate as done in DenseNet and DPN, and using front-end stages from networks such as ResNet, DenseNet and DPN before the first stage to further boost the experssive power and to save memory footprint.

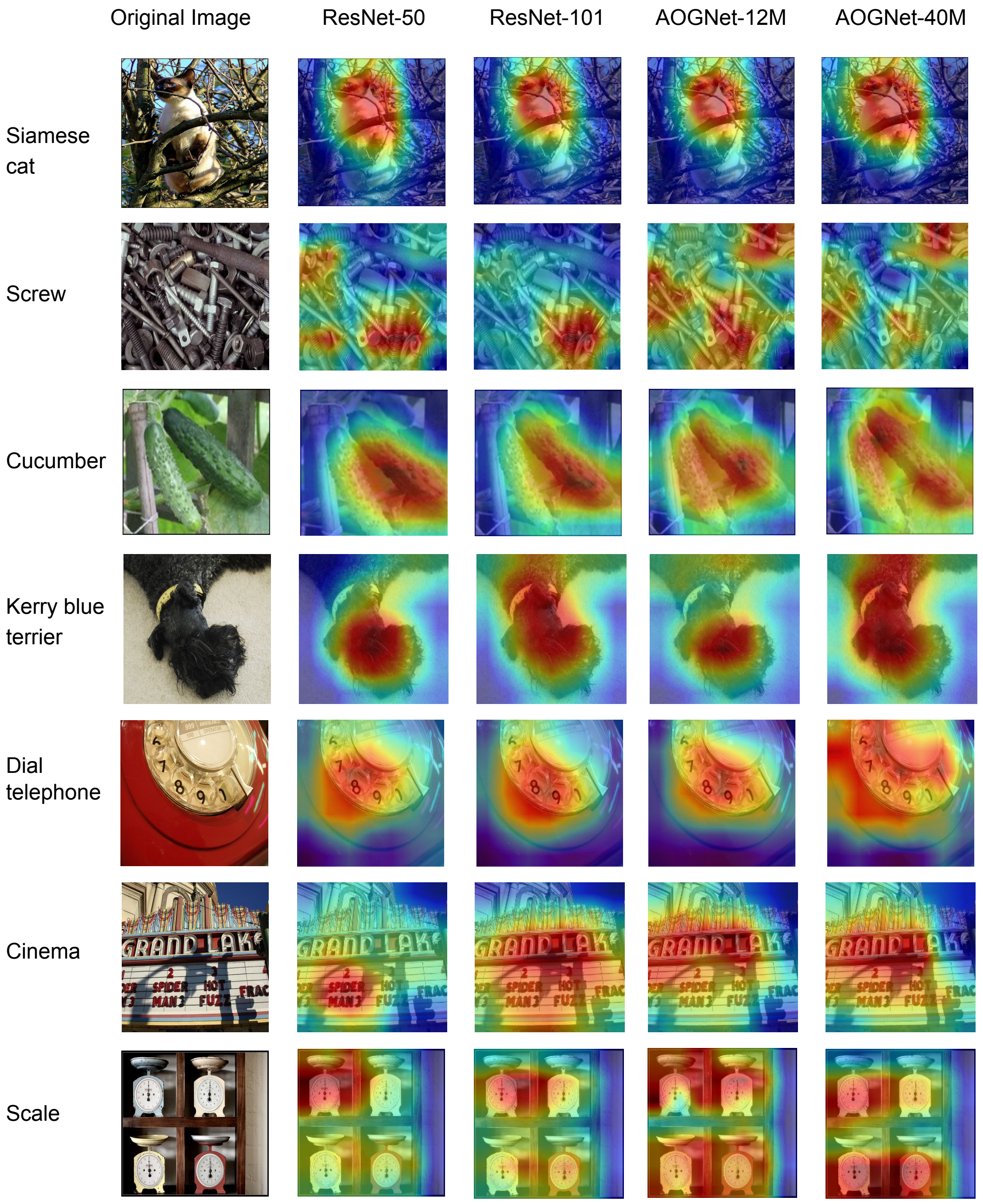


Figure 3.8: Sample grad-cam results from ResNet-50/101, AOGNet-12m/40m.

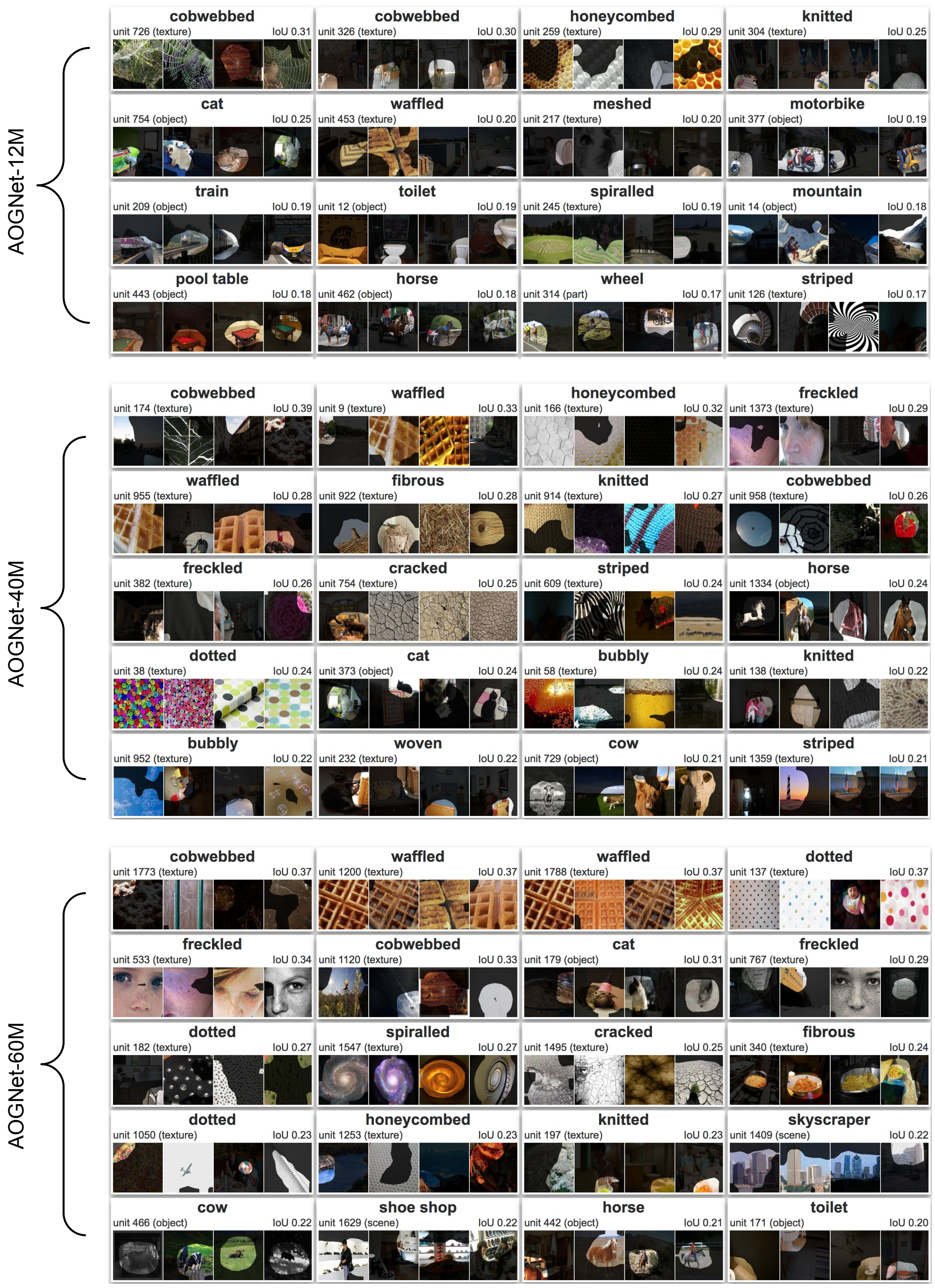

AOGNet-12M
cobwebbed unit 726 (texture) IoU 0.31
cobwebbed unit 326 (texture) IoU 0.30
honeycombed unit 259 (texture) IoU 0.29
knitted unit 304 (texture) IoU 0.25
cat unit 754 (object) IoU 0.25
waffled unit 453 (texture) IoU 0.20
meshed unit 217 (texture) IoU 0.20
motorbike unit 377 (object) IoU 0.19
train unit 209 (object) IoU 0.19
toilet unit 12 (object) IoU 0.19
spiralled unit 245 (texture) IoU 0.19
mountain unit 14 (object) IoU 0.18
pool table unit 443 (object) IoU 0.18
horse unit 462 (object) IoU 0.18
wheel unit 314 (part) IoU 0.17
striped unit 126 (texture) IoU 0.17
AOGNet-40M
cobwebbed unit 174 (texture) IoU 0.39
waffled unit 9 (texture) IoU 0.33
honeycombed unit 166 (texture) IoU 0.32
freckled unit 1373 (texture) IoU 0.29
waffled unit 955 (texture) IoU 0.28
fibrous unit 922 (texture) IoU 0.28
knitted unit 914 (texture) IoU 0.27
cobwebbed unit 958 (texture) IoU 0.26
freckled unit 382 (texture) IoU 0.26
cracked unit 754 (texture) IoU 0.25
striped unit 609 (texture) IoU 0.24
horse unit 1334 (object) IoU 0.24
dotted unit 38 (texture) IoU 0.24
cat unit 373 (object) IoU 0.24
bubbly unit 58 (texture) IoU 0.24
knitted unit 138 (texture) IoU 0.22
bubbly unit 952 (texture) IoU 0.22
woven unit 232 (texture) IoU 0.22
cow unit 729 (object) IoU 0.21
striped unit 1359 (texture) IoU 0.21
AOGNet-60M
cobwebbed unit 1773 (texture) IoU 0.37
waffled unit 1200 (texture) IoU 0.37
waffled unit 1788 (texture) IoU 0.37
dotted unit 137 (texture) IoU 0.37
freckled unit 533 (texture) IoU 0.34
cobwebbed unit 1120 (texture) IoU 0.33
cat unit 179 (object) IoU 0.31
freckled unit 767 (texture) IoU 0.29
dotted unit 182 (texture) IoU 0.27
spiralled unit 1547 (texture) IoU 0.27
cracked unit 1495 (texture) IoU 0.25
fibrous unit 340 (texture) IoU 0.24
dotted unit 1050 (texture) IoU 0.23
honeycombed unit 1253 (texture) IoU 0.23
knitted unit 197 (texture) IoU 0.23
skyscraper unit 1409 (scene) IoU 0.22
cow unit 466 (object) IoU 0.22
shoe shop unit 1629 (scene) IoU 0.22
horse unit 442 (object) IoU 0.21
toilet unit 171 (object) IoU 0.20

Figure 3.9: Sample dissection results of AOGNets generated by network dissection.

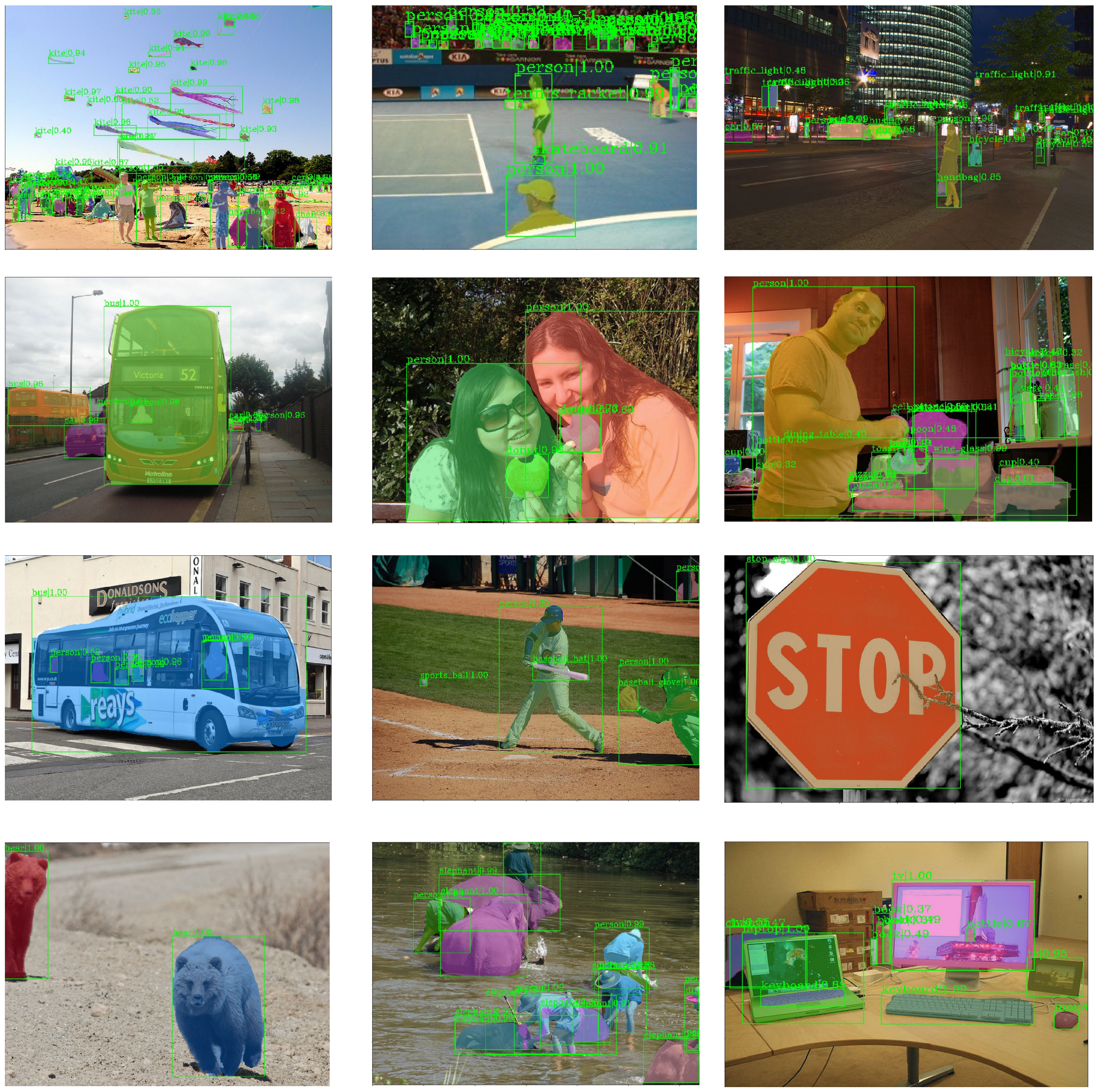


Figure 3.10: Sample COCO object detection and instance segmentation results of Mask-R-CNN model with AOGNet-40m backbone.

CHAPTER

4 ATTENTIVE NORMALIZATION

In state-of-the-art deep neural networks, both feature normalization and feature attention have become ubiquitous with significant performance improvement shown in a vast amount of tasks. They are usually studied as separate modules, In this Chapter, I will propose a light-weight integration between, and thus harness the best of, the two schema. We present Attentive Normalization (AN) which generalizes the common affine transformation component in the vanilla feature normalization. Instead of learning a single affine transformation, AN learns a mixture of affine transformations and utilizes their weighted-sum as the final affine transformation applied to re-calibrate features in an instance-specific way. The weights are learned by leveraging feature attention.

AN introduces negligible extra parameters and computational cost (i.e., light-weight). AN can be used as a drop-in replacement for any feature normalization technique which includes the affine transformation component. In experiments, we test the proposed AN using three representative neural architectures (ResNets (40), MobileNets-v2 (120) and AOGNets (74)) in the ImageNet-1000 classification benchmark (116) and the MS-COCO 2107 object detection and instance segmentation benchmark (84). AN obtains consistent performance improvement for different neural architectures in both benchmarks with absolute increase of top-1 accuracy in ImageNet-1000 between 0.5% and 2.0%, and absolute increase up to 1.8% and 2.2% for bounding box and mask AP in MS-COCO respectively. The source codes are publicly available [1].

In the following sections, I will presents the details of the design and performance of AOGNets. In section 4.1, I will first present the motivation and objective of attentive normalization. Section 4.2 presents some background information on existing feature normalizations and feature attentions. Section 4.3 and 4.4 presents detailed designs of our AOGNets. Section 4.5 shows experimental results and comparisons with other networks, as well as ablation studies on different aspects of our AOGNets. Finally, Section 4.6 concludes this work and discusses some on-going and future work.

4.1 Motivation and Objective

Pioneered by Batch Normalization (BN) (57), feature normalization has become ubiquitous in the development of deep learning. As illustrated in Figure 4.2, feature normalization consists of two components: *feature standardization* and *channel-wise affine transformation* which is introduced to provide the capability of undoing the standardization (by design), and can be treated as *feature re-calibration* in general. Many variants of BN have been proposed for practical deployment in terms of variations of training and testing settings with remarkable progress obtained. They can be roughly divided into two categories:

- *Generalizing feature standardization.* Different methods are proposed for computing the mean(s) and standard deviation(s), or for modeling the data distribution in general, within a min-batch. They include Batch Renormalization (56), Decorrelated BN (52), Layer Normalization (LN) (3), Instance Normalization (IN) (141), Instance-level Meta Normalization (58), Group Normalization (GN) (150), Mixture Normalization (59) and Mode Normalization (16). Switchable Normalization (SN) (88) and its sparse variant (SSN) (126) learn to switch between different vanilla schema. These methods adopt the vanilla channel-wise affine transformation after standardization, and are often proposed for discriminative learning tasks.
- *Generalizing feature re-calibration.* Instead of treating the affine transformation parameters directly as model parameters, different types of task-induced conditions (, class labels in conditional image synthesis using generative adversarial networks) are leveraged and encoded as latent vectors, which are then used to learn the affine transformation parameters, including different conditional BNs (17; 15; 103; 94; 7),

[1] Classification in ImageNet: `https://github.com/iVMCL/AOGNets-v2` and Detection in MS-COCO: `https://github.com/iVMCL/AttentiveNorm_Detection`

style-adaptive IN (60) or layout-adaptive IN (101; 134). These methods have been mainly proposed in generative learning tasks.

In the meanwhile, *feature attention* has also become an indispensable mechanism for improving task performance in deep learning. For computer vision, spatial attention is inherently captured by convolution operations within short-range context, and by non-local extensions (144; 53) for long-range context. Channel-wise attention is relatively less exploited. The squeeze-and-excitation (SE) unit (47) is one of the most popular designs, which learn instance-specific channel-wise attention weights to re-calibrate an input feature map (Figure 4.2). Unlike the affine transformation parameters in feature normalization, the attention weights for re-calibrating an feature map are often directly learned from the input feature map in the spirit of self-attention, and often instance-specific or pixel-specific.

Although both feature normalization and feature attention have become ubiquitous in state-of-the-art DNNs, they are usually studied as separate modules. Therefore, in this paper we address the following problem,

How to learn to re-calibrate feature maps in a way of harnessing the best of feature normalization and feature attention in a single light-weight module?

To that end, we present **Attentive Normalization (AN)**. our proposed AN is both conceptually and computationally simple, which can be used as a drop-in replacement for any feature normalization method that includes the affine transformation component and results in consistent performance improvement (Figure 4.8).

4.2 Background

4.2.1 Feature Normalization.

There are two types of normalization schema, feature normalization (including raw data) (57; 56; 3; 141; 150; 88; 126; 59; 16) and weight normalization (119; 51). Unlike the former stated above, the latter is to normalize model parameters to decouple the magnitudes of parameter vectors from their directions. We focus on feature normalization in this paper.

Different feature normalization schema differ in how the mean and variance are computed. BN (57) computes the channel-wise mean and variance in the entire min-batch which is driven by improving training efficiency and model generalizability. BN has been deeply analyzed in terms of how it helps optimization (121). DecorBN (52) adds a whitening operation after the standardization. BatchReNorm (56) introduces extra parameters to control the pooled mean and variance to reduce BN's dependency on the batch size. IN (141) focuses on channel-wise and instance-specific statistics which stems from the task of artistic image style transfer. LN (3) computes the instance-specific mean and variance from all channels which is designed to help optimization in recurrent neural networks (RNNs). GN (150) stands in the sweet spot between LN and IN focusing on instance-specific and channel-group-wise statistics, especially when only small batches are applicable in practice. In practice, synchronized BN (102) across multiple GPUs becomes increasingly favorable against GN in some applications. SN (88) leaves the design choices of feature normalization schema to the learning system itself by computing weighted sum integration of BN, LN, IN and/or GN via softmax, showing more flexible applicability, followed by SSN (126) which learns to make exclusive selection. Instead of computing one mode (mean and variance), MixtureNorm (59) introduces a mixture of Gaussian densities to approximate the data distribution in a mini-batch. ModeNorm (16) utilizes a general form of multiple-mode computation. Unlike those methods, the proposed AN focuses on generalizing the affine transformation component. Related to our work, Instance-level Meta normalization(ILM) (58) first utilizes an encoder-decoder sub-network to learn affine transformation parameters and then add them together to the model's affine transformation parameters. Unlike ILM, the proposed AN utilizes a mixture of affine transformations and leverages feature attention to learn the instance-specific attention weights.

On the other hand, conditional feature normalization schema (17; 15; 103; 7; 60; 101; 134) have been developed and shown remarkable progress in conditional and unconditional image synthesis. Conditional BN learns condition-specific affine transformations in terms of conditions such as class labels, image style, label maps and geometric layouts. Unlike those methods, the proposed AN learns self-attention data-driven weights for mixture components of affine transformations.

4.2.2 Feature Attention.

Similar to feature normalization, feature attention is also an important building block in the development of deep learning. Residual Attention Network (142) uses a trunk-and-mask joint spatial and channel attention module in an encoder-decoder style for improving performance. To reduce the computational cost, channel and spatial attention are separately applied in (146). The SE module (47) further simplifies the attention mechanism by developing a light-weight channel-wise attention method. The proposed AN leverages the idea of SE in learning attention weights, but formulates the idea in a novel way.

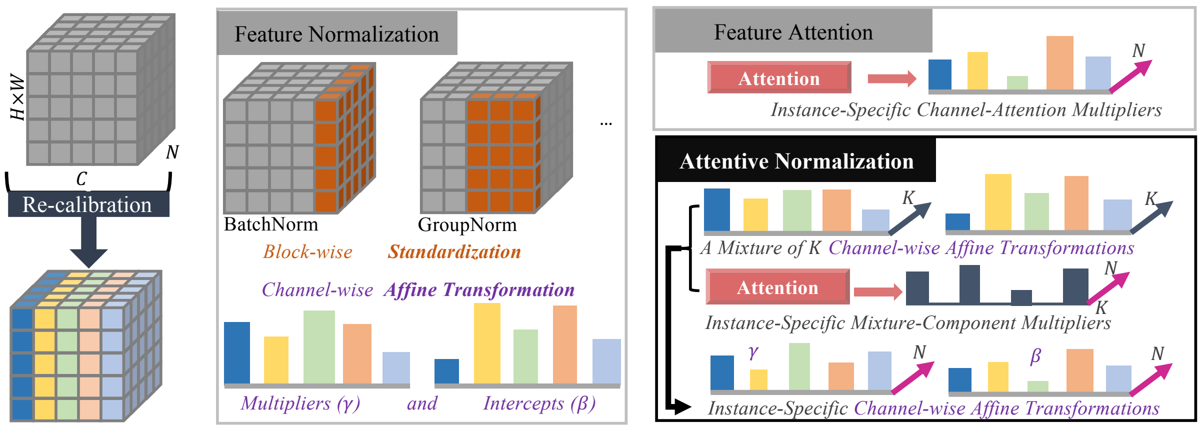


Figure 4.1: Illustration of the proposed Attentive Normalization (AN). AN aims to harness the best of feature normalization and feature attention. AN keeps the block-wise standardization unchanged. AN learns a mixture of K channel-wise affine transformations. AN leverages attention mechanism to learn the instance-specific weights for the mixture components in computing the weighted sum of the mixture as the final affine transformation for re-calibrating the input features. See text for details. Best viewed in color.

4.3 Overview of Attentive Normalization

Figure 4.2 illustrates the proposed AN. The basic idea is straightforward. Conceptually, the affine transformation component in feature normalization (Section 4.4.1) and the re-scaling computation in feature attention play the same role in learning-to-re-calibrate an input feature map, thus providing the foundation for integration (Section 4.4.2). More specifically, consider a feature normalization backbone such as BN or GN, our proposed AN keeps the block-wise standardization component unchanged. Unlike the vanilla feature normalization in which the affine transformation parameters (γ's and β's) are often frozen in testing, we want the affine transformation parameters to be adaptive and dynamic in both training and testing, controlled directly by the input feature map. The intuition behind doing so is that it will be more flexible in accounting for different statistical discrepancies between training and testing in general, and between different sub-populations caused by underlying inter-/intra-class variations in the data.

To achieve the dynamic and adaptive control of affine transformation parameters, the proposed AN utilizes a simple design (Section 4.4). It learns a mixture of K affine transformations and exploits feature attention mechanism to learn the instance-specific weights for the K components. The final affine transformation used to re-calibrate an input feature map is the weighted sum of the learned K affine transformations. We propose a general formulation for the proposed AN (Section 4.4.3). We study how to learn the weights in an efficient and effective way (Section 4.4.3). We study how to deploy the proposed AN to state-of-the-art building blocks such as the Bottleneck block (40) in a light-weight manner in terms of both parameter increase and computational expense (Section 4.4.4).

In experiments, we focus on image classification and object detection and instance segmentation tasks. We test the proposed AN with BN and GN as the feature normalization backbones in three state-of-the-art DNNs: ResNets (40), MobileNets-v2 (120) and AOGNets (74). We test our AN in the ImageNet-1000 dataset (116) and the MS-COCO dataset (84). Our AN significantly outperforms the vanilla feature normalization backbones consistently across the three networks in the two datasets. We also perform an ablation study comparing different design choices in AN.

4.4 Implementation of Attentive Normalization

In this section, we present details of the proposed attentive normalization. Without loss of generality, consider a DNN for 2D images, denote by $\mathbf{x}$ a feature map with axes in the convention order of (N, C, H, W) (i.e., batch, channel, height and width). $\mathbf{x}$ is represented by a 4D tensor. Let $i = (i_N, i_C, i_H, i_W)$ be the address index in the 4D tensor. $\mathbf{x}_i$ represents the feature response at a position i. Note that we can squeeze the last two dimensions to be consistent with Figure 4.2.

4.4.1 Formulation of Feature Normalization

As aforementioned, existing feature normalization schema often consist of two components:

i) Block-wise Standardization. Denote by B_j a block (slice) in a given 4-D tensor $\mathbf{x}$. For example, for BN, we have $j = 1, \cdots, C$ and $B_j = \{\mathbf{x}_i | \forall i, i_C = j\}$. We first compute the empirical mean and standard deviation in B_j, denoted by μ_j and σ_j respectively, and we have,

$$\mu_j = \frac{1}{M} \sum_{x \in B_j} x, \quad \sigma_j = \sqrt{\frac{1}{M} \sum_{x \in B_j} (x - \mu_j)^2 + \epsilon}, \tag{4.1}$$

where $M = |B_j|$ and ϵ is a small positive constant to ensure $\sigma_j > 0$ for the sake of numeric stability. Then, let j_i be the index of the block that the position i belongs to, and we standardize the feature response by,

$$\hat{\mathbf{x}}_i = \frac{1}{\sigma_{j_i}}(\mathbf{x}_i - \mu_{j_i}) \tag{4.2}$$

Remark: As studied in (121), other types of data-statistics can also be exploited in standardization. Since our proposed AN keeps the standardization component unchanged, it will also be applicable to these variants.

ii) Channel-wise Affine Transformation. Denote by γ_c and β_c the scalar coefficient (re-scaling) and offset (re-shifting) parameter respectively for the c-th channel. The re-calibrated feature response at a position i is then computed by,

$$\tilde{\mathbf{x}}_i = \gamma_{i_C} \cdot \hat{\mathbf{x}}_i + \beta_{i_C}, \tag{4.3}$$

where γ_c's and β_c's are shared by all the instances in a min-batch across the spatial domain. They are usually frozen in testing and fine-tuning.

4.4.2 Formulation on Feature Attention

We focus on channel-wise attention and briefly review the Squeeze-Excitation (SE) module (47). SE usually takes the feature normalization result (Eqn. 4.3) as its input. SE learns channel-wise attention weights as follows:

i) The squeeze module encodes the inter-dependencies between feature channels in a low dimensional latent space with the reduction rate r (e.g., $r = 16$),

$$S(\tilde{\mathbf{x}}; \theta_S) = v,\ v \in \mathbb{R}^{N \times \frac{C}{r} \times 1 \times 1}, \tag{4.4}$$

which is implemented by a sub-network consisting of a global average pooling layer (AvgPool), a fully-connected (FC) layer and rectified linear unit (ReLU) (64). θ_S collects all the model parameters.

ii) The excitation module computes the channel-wise attention weights, denoted by λ, by decoding the learned latent representations v,

$$E(v; \theta_E) = \lambda,\ \lambda \in \mathbb{R}^{N \times C \times 1 \times 1}, \tag{4.5}$$

which is implemented by a sub-network consisting of a FC layer and a sigmoid layer. θ_E collects all model parameters.

Then, the input, $\tilde{\mathbf{x}}$ is re-calibrated by,

$$\begin{aligned}\tilde{\mathbf{x}}_i^{SE} =& \lambda_{i_N,i_C} \cdot \tilde{\mathbf{x}}_i, \\ =& (\lambda_{i_N,i_C} \cdot \gamma_{i_C}) \cdot \hat{\mathbf{x}}_i + \lambda_{i_N,i_C} \cdot \beta_{i_C},\end{aligned} \tag{4.6}$$

where the second step is obtained by plugging in Eqn. 4.3. *It is straightforward to see the foundation facilitating the integration between feature normalization and channel-wise feature attention.* However, the SE module often entails a significant number of extra parameters (e.g., ∼2.5M extra parameters for ResNet-50 (40) which originally consists of ∼25M parameters, resulting in 10% increase). We aim to design more parsimonious integration that can further improve performance (i.e., favoring "less is more").

4.4.3 Attentive Normalization

The Formulation

Our goal is to generalize Eqn. 4.3 in re-calibrating feature responses to enable dynamic and adaptive control in both training and testing. On the other hand, our goal is to simplify Eqn. 4.6 into a single light-weight module, rather than, for example, the two-module setup using BN+SE. In the most general form, we have,

$$\tilde{\mathbf{x}}_i^{AN} = \Gamma(\mathbf{x};\theta_\Gamma)_i \cdot \hat{\mathbf{x}}_i + \mathbb{B}(\mathbf{x};\theta_\mathbb{B})_i, \tag{4.7}$$

where both $\Gamma(\mathbf{x};\theta_\Gamma)$ and $\mathbb{B}(\mathbf{x};\theta_\mathbb{B})$ are functions of the entire input feature map (without standardization [2]) with parameters θ_Γ and $\theta_\mathbb{B}$ respectively. The two functions compute 4-D tensors of the size same as the input feature map and can be parameterized by some attention guided light-weight DNNs. The subscript in $\Gamma(\mathbf{x};\theta_\Gamma)_i$ and $\mathbb{B}(\mathbf{x};\theta_\mathbb{B})_i$ represents the learned re-calibration weights at a position i.

In this paper, we focus on learning instance-specific channel-wise affine transformations. To that end, we have three components as follows.

i) Learning a Mixture of K Channel-wise Affine Transformations. Denote by $\gamma_{k,c}$ and $\beta_{k,c}$ the re-scaling and re-shifting (scalar) parameters respectively for the c-th channel in the k-th mixture component. They are model parameters learned end-to-end via back-propagation.

[2] We tried the variant of learning $\Gamma()$ and $\mathbb{B}()$ from the standardized features and observed it works worse, so we ignore it in our experiments.

ii) Learning Attention Weights for the K Mixture Components. Denote by $\lambda_{n,k}$ the instance-specific mixture component weight ($n \in [1, N]$ and $k \in [1, K]$), and by λ the $N \times K$ weight matrix. λ is learned via some attention-guided function from the entire input feature map,

$$\lambda = A(\mathbf{x}; \theta_\lambda), \tag{4.8}$$

where θ_λ collects all the parameters.

iii) Computing the Final Affine Transformation. With the learned $\gamma_{k,c}$, $\beta_{k,c}$ and λ, the re-calibrated feature response is computed by,

$$\tilde{\mathbf{x}}_i^{AN} = \sum_{k=1}^{K} \lambda_{i_N,k}[\gamma_{k,i_C} \cdot \hat{\mathbf{x}}_i + \beta_{k,i_C}], \tag{4.9}$$

where $\lambda_{i_N,k}$ is shared by the re-scaling parameter and the re-shifting parameter for simplicity. Since the attention weights λ are adaptive and dynamic in both training and testing, the proposed AN realizes adaptive and dynamic feature re-calibration. Compared to the general form (Eqn. 4.7), we have,

$$\Gamma(\mathbf{x})_i = \sum_{k=1}^{K} \lambda_{i_N,k} \cdot \gamma_{k,i_C}, \; \mathbb{B}(\mathbf{x})_i = \sum_{k=1}^{K} \lambda_{i_N,k} \cdot \beta_{k,i_C}, \tag{4.10}$$

from which **we can see two main advantages of the proposed AN in both training, fine-tuning and testing**:

- γ_{k,i_C}'s and β_{k,i_C}'s are channel-wise and shared across spatial dimensions and by data instances, which can learn population-level knowledge. $\lambda_{i_N,k}$'s are instance specific and learned from features that are not standardized. Combining them together enables paying attention to both the population (what the common and useful information are) and the individuals (what the specific yet critical information are). The latter is particularly useful for testing samples "drifted" from training population, that is to improve generalizability. It also helps establish more direct connections between standardization and affine transformation in feature normalization. Their weighted sum encodes more direct and "actionable" information for re-calibrating standardized features (Eqn. 4.9) without being delayed until back-propagation updates as done in the vanilla feature normalization.
- In fine-tuning, especially between different tasks (, from image classification to object detection), γ_{k,i_C}'s and β_{k,i_C}'s are usually frozen as done in the vanilla feature normalization. They carry information from the source tasks. But, θ_λ (Eqn. 4.8) are

allowed to be fine-tuned, thus potentially better realizing transfer learning for the target tasks. This is a desirable property since we can decouple training correlation between tasks. For example, it is not necessary to train a model in ImageNet with the feature normalization scheme such as GN (150) that is practically feasible and beneficial in object detection in MS-COCO. As we shall show in experiments, the proposed AN facilitates a smoother transition. We can use the proposed AN with BN as normalization backbone in pre-training in ImageNet, and then use AN with GN as normalization backbone for the head classifiers in MS-COCO with significant improvement.

Details of Learning Attention Weights

We present a simple method for computing the attention weights $A(\mathbf{x};\theta_\lambda)$ (Eqn. 4.8). Our goal is to learn a weight coefficient for each component from each individual instance in a mini-batch (i.e, a $N \times K$ matrix). The question of interest is how to characterize the underlying importance of a channel c from its realization across the spatial dimensions (H, W) in an instance, such that we will learn a more informative instance-specific weight coefficient for a channel c in re-calibrating the feature map $\mathbf{x}$.

In realizing Eqn. 4.8, the proposed method is similar in spirit to the squeeze module in SE to maintain light-weight implementation. To show the difference, let's first rewrite the vanilla squeeze module (Eqn. 4.4),

$$v = S(\mathbf{x};\theta_S) = ReLu(fc(AvgPool(\mathbf{x});\theta_S)), \tag{4.11}$$

where the mean of a channel c (via global average pooling, $AvgPool(\cdot)$) is used to characterize its underlying importance. We generalize this assumption by taking into account both mean and standard deviation empirically computed for a channel c, denoted by μ_c and σ_c respectively (μ_c and σ_c are computed using Eqn. 4.1 under the BN setting). More specifically, we compare three different designs using:

i) The mean μ_c only as done in SE.

ii) The concatenation of the mean and standard deviation, (μ_c, σ_c).

iii) The coefficient of variation or the relative standard deviation (RSD), $\frac{\mu_c}{\sigma_c}$.

RSD measures the dispersion of an underlying distribution (i.e., the extent to which the distribution is stretched or squeezed) which intuitively conveys more information in learning attention weights for re-calibration. RSD is indeed observed to work better in our experiments. Eqn. 4.8 is then expanded with two choices,

$$\begin{aligned} &\textit{Choice 1: } A_1(\mathbf{x};\theta_\lambda) = Act(fc(RSD(\mathbf{x});\theta_\lambda)), \\ &\textit{Choice 2: } A_2(\mathbf{x};\theta_\lambda) = Act(BN(fc(RSD(\mathbf{x});\theta_{fc});\theta_{BN})), \end{aligned} \tag{4.12}$$

where $Act(\cdot)$ represents a non-linear activate function for which we compare three designs:

i) The vanilla $sigmoid(\cdot)$ as used in the excitation module of SE.

ii) The channel-wise $softmax(\cdot)$.

iii) the piece-wise linear hard analog of the sigmoid function, so-called h-sigmoid function (45),

$$hsigmoid(a) = ReLu6(a + 3.0)/6.0,$$

where $ReLu6(a) = \min(\max(a, 0), 6.0)$ is a variant of the vanilla ReLu function with a saturation threshold 6.0.

The $hsigmoid(\cdot)$ is observed to work better in our experiments. In the Choice 2 (Eqn. 4.12), we apply the vanilla BN (57) after the FC layer, which normalizes the learned attention weights across all the instances in a mini-batch with the hope of balancing the instance-specific attention weights better. The Choice 2 improves performance in our experiments in ImageNet.

During the learning, we have another hyper-parameter, K. For stage-wise building block based neural architectures such as ResNets (40), we use different K's for different stages with smaller values for early stages. For example, for the 4-stage ResNets, we typically use $K = 10, 10, 20, 20$ for the four stages respectively to balance the expressive power of the mixture of affine transformations and the expense (we are interested in making extra parameters and computation cost negligible).

Learning the attention weights $A(\mathbf{x};\theta_\lambda)$ can adopt other implementations when deploying in different domains such as in NLP. It can also leverage more effective and efficient channel-wise attention mechanism when available in future.

Implementing AN is easy in modern deep learning code framework. For example, Figure 4.3 shows the complete PyTorch code for AN using BN as the backbone.

4.4.4 Integrating AN in Building Blocks of DNNs

In general, the proposed AN can be used as a drop-in replacement for all feature normalization layers in a DNN. Typically, in convolutional neural networks (ConvNets), a feature normalization layer follows a convolution layer. It is not necessary to replace all vanilla

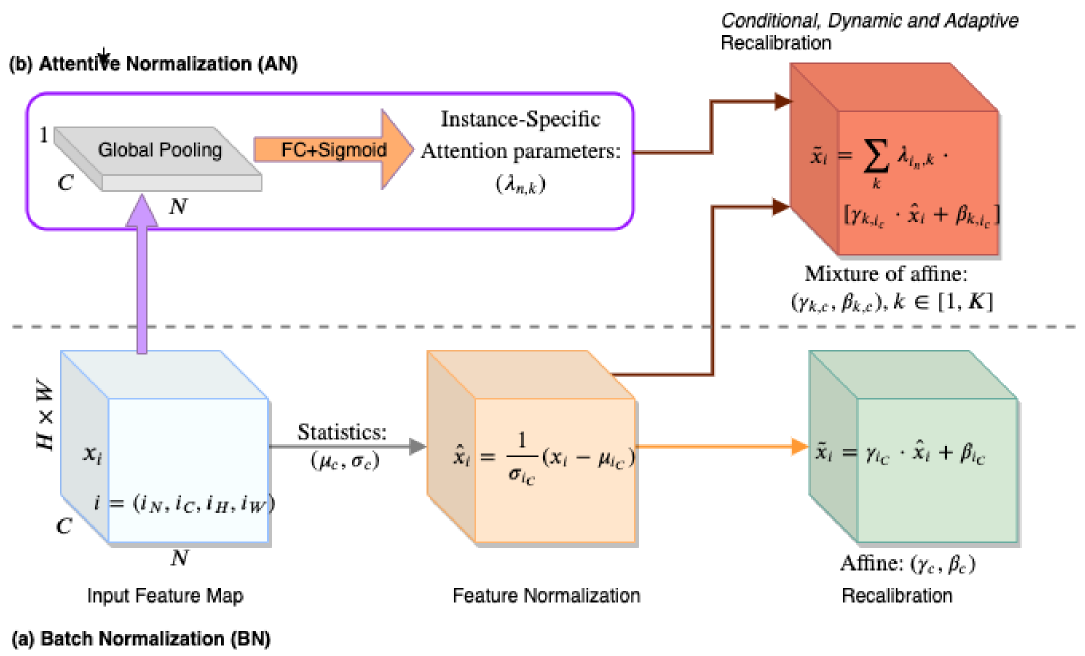

Conditional, Dynamic and Adaptive Recalibration
(b) Attentive Normalization (AN)
Global Pooling
FC+Sigmoid
Instance-Specific Attention parameters: $(\lambda_{n,k})$
$\tilde{x}_i = \sum_k \lambda_{i_n,k} \cdot [\gamma_{k,i_c} \cdot \hat{x}_i + \beta_{k,i_c}]$
Mixture of affine: $(\gamma_{k,c}, \beta_{k,c}), k \in [1, K]$
$H \times W$
x_i
$i = (i_N, i_C, i_H, i_W)$
Statistics: (μ_c, σ_c)
$\hat{x}_i = \frac{1}{\sigma_{i_C}}(x_i - \mu_{i_C})$
$\tilde{x}_i = \gamma_{i_C} \cdot \hat{x}_i + \beta_{i_C}$
Affine: (γ_c, β_c)
Input Feature Map
Feature Normalization
Recalibration
(a) Batch Normalization (BN)

Figure 4.2: Illustration of the proposed Attentive Normalization (AN) in (b) using the vanilla Batch Normalization (BN) (57) as backbone (a). AN shares the feature normalization component with BN, and differs in how the affine transformation is done. AN can also use other variants of BN as backbones. The input feature map is represented using the convention (N, C, H, W) for the batch axis, channel axis, spatial height and width axes respectively. x_i represents a feature response in the input feature map with position index $i = (i_N, i_C, i_H, i_W)$. $\hat{x}_i$ represents the normalized response using the pooled channel-wise mean and variance. $\tilde{x}_i$ is the response after affine transformation with learned scale and offset parameters. See text for details.

feature normalization layers for two reasons: First, our AN exploits feature attention to re-calibrate an input feature map, which is not needed for every convolution layer in a ConvNet, especially for 1×1 convolution which only handles feature channels (increasing or decreasing), and thus channel-wise re-calibration can be tackled by the convolution kernel and the vanilla feature normalization themselves in training. Second, adding too many AN layers may make a ConvNet computationally sloppy. Consider the widespread of the Bottleneck block (40) in ConvNets, we integrate one AN layer into one Bottleneck block, as illustrated and explained in Figure 4.4. The integration of SE adopts the same practice and uses the spot after BN_3 in a Bottleneck block.

```
class AttenNorm(nn.BatchNorm2d):
    def __init__(self, C, K, eps, momentum, running):
        super(AttenNorm, self).__init__(C, eps=eps, momentum=momentum, affine=False,
            track_running_stats=running)

        self.gamma = nn.Parameter(torch.Tensor(K, C))
        self.beta = nn.Parameter(torch.Tensor(K, C))

        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.fc  = nn.Linear(C, K)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        output = super(AttenNorm, self).forward(x)
        size  = output.size()

        b, c, _, _ = x.size()
        y = self.avgpool(x).view(b, c)
        y = self.fc(y)
        y = self.sigmoid(y)

        gamma = y @ self.gamma
        beta = y @ self.beta
        gamma = weight.unsqueeze(−1).unsqueeze(−1).expand(size)
        beta = bias.unsqueeze(−1).unsqueeze(−1).expand(size)

        return gamma  output + beta
```

Figure 4.3: Complete PyTorch code for implementing our AN using BN as the backbone.

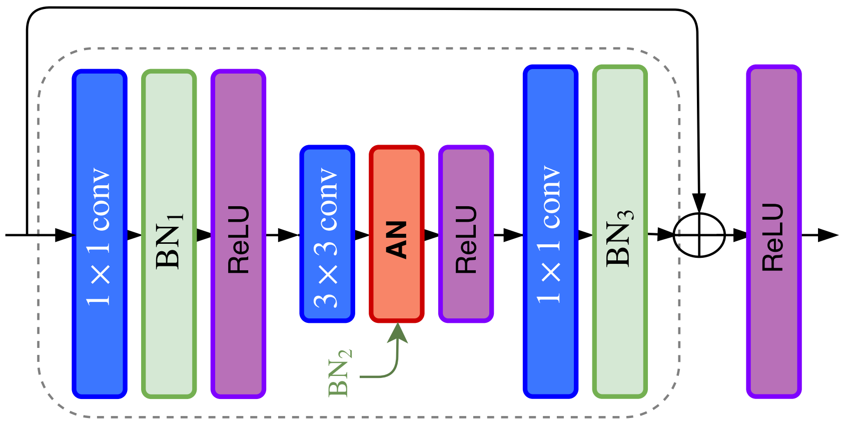


Figure 4.4: Illustration of integrating the proposed AN in the popular Bottleneck building block presented in ResNets (40). Here, the feature normalization backbone is BN. The proposed AN is only used to replace the second one (BN$_2$) followed the 3×3 convolution. This leads to negligible extra parameters. Furthermore, this potentially enables jointly integrating local spatial attention in learning the instance-specific attention weights.

4.5 Experiments

In this section, we first show the ablation study verifying the aforementioned design choices we used in the proposed AN. Then, we present detailed comparisons and results.

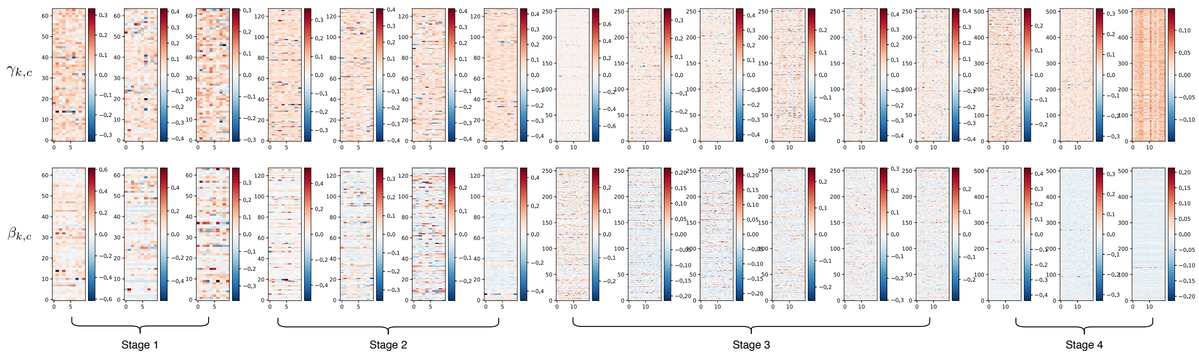


Figure 4.5: Heatmap visualization of γ_{k,i_C}'s and β_{k,i_C}'s in the AN. See text for details. Best viewed in color and magnification.

Data and Evaluation Metric. In experiments, we use two benchmarks, the ImageNet-1000 classification benchmark (ILSVRC2012) (116) and the MS-COCO object detection and instance segmentation benchmark (84). The ImageNet-1000 benchmark consists of about

1.28 million images for training, and 50,000 for validation, from 1,000 classes. We apply a single-crop with size 224×224 in evaluation. Following the common protocol, we report the top-1 and top-5 classification error rates tested using a single model on the validation set. For the MS-COCO benchmark, there are 80 categories of objects. We use `train2017` in training and evaluate the trained models using `val2107`. We report the standard COCO metrics of Average Precision (AP) at different intersection-over-union (IoU) thresholds, , AP_{50} and AP_{75}, for bounding box detection (AP^{bb}_{IoU}) and instance segmentation (AP^{m}_{IoU}), and the mean AP over IoU=$0.5:0.05:0.75$, AP^{bb} and AP^{m} for bounding box detection and instance segmentation respectively.

Neural Architectures and Feature Normalization Backbones. We use three representative neural architectures: (i) ResNets (40) (ResNet-50 and ResNet-101), which are the most widely used architectures in practice, (ii) MobileNet-v2 (120). MobileNets are popular architectures under mobile settings and MobileNet-v2 uses inverted residuals and linear Bottlenecks (for which our AN follows the 3×3 convolution layer in the way same as Figure 4.4), and (iii) AOGNets (74) (AOGNet-12M and AOGNet-40M) are grammar-guided networks with the vanilla Bottleneck building bock, which represent an interesting direction of network architecture engineering with better performance than ResNets and its variants shown. So, the improvement by our AN will be both broadly useful for existing ResNets/MobileNets based deployment in practice and potentially insightful for on-going and future development of more advanced and more powerful DNNs in the community. In classification, we use BN (57) as the feature normalization backbone for our proposed AN, denoted by **AN (w/ BN)**. We compare with the vanilla BN, GN (150) and SN (88). In object detection and instance segmentation, we use the Mask-RCNN framework (37) and its cascade variant (9) in the MMDetection code platform (11). We fine-tune feature backbones pretrained on the ImageNet-1000 dataset. We also test the proposed AN using GN as the feature normalization backbone, denoted by **AN (w/ GN)** in the head classifier of Mask-RCNN.

Initialization of our AN. The initialization of $\gamma_{k,c}$'s and $\beta_{k,c}$'s (Eqn. 4.9) is based on, $\gamma_{k,c} = 1.0 + \mathcal{N}(0,1) \times 0.1$ and $\beta_{k,c} = \mathcal{N}(0,1) \times 0.1$, where $\mathcal{N}(0,1)$ represents the standard Gaussian distribution. This type of initialization is also adopted for conditional BN used in the BigGAN (7).

Table 4.1: Ablation study on different design choices in our proposed AN with BN as feature normalization backbone using ResNet-50 in ImageNet-1000. There are four categories: The first three are detailed in Section 4.4.3. The fourth one refers to the number K of components in the mixture of affine transformation which is used for each of the four stages in ResNet-50 and we empirically select three options for simplicity. The second row shows the best design combination we observed, which will be used in our comparison experiments. In each test (row 3 to 9), we only change one in the best design combination for computational feasibility. During our development, we first observed the best combination based on our intuitive reasoning and small experiments (a few epochs) in the process, and then design this ablation study to verify the design choices. * means AN is applied to all the BatchNorms of the network. The performance of the baseline ResNet-50+BN is in Table 4.2. See text for details.

Design Choices in AN (w/ BN)	#Params	FLOPS	top-1	top-5
RSD(mean/std) + $A_2(\cdot)$ + hsigmoid + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.76M	4.09G	**21.59**	**5.58**
mean + $A_2(\cdot)$ + hsigmoid + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.76M	4.09G	21.85	5.92
concat(mean,std) + $A_2(\cdot)$ + hsigmoid + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.82M	4.09G	21.73	5.85
RSD(mean/std) + $\mathbf{A}_1(\cdot)$ + hsigmoid + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.76M	4.09G	21.76	6.05
RSD(mean/std) + $A_2(\cdot)$ + **softmax** + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.76M	4.09G	21.72	5.90
RSD(mean/std) + $A_2(\cdot)$ + **relu** + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.96M	4.09G	21.89	6.04
RSD(mean/std) + $A_2(\cdot)$ + **sigmoid** + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.76M	4.09G	21.96	5.91
RSD(mean/std) + $A_2(\cdot)$ + hsigmoid + $\mathbf{K} = \begin{pmatrix}5\\5\\10\\10\end{pmatrix}$	25.76M	4.09G	21.92	5.93
RSD(mean/std) + $A_2(\cdot)$ + hsigmoid + $\mathbf{K} = \begin{pmatrix}20\\20\\40\\40\end{pmatrix}$	25.96M	4.09G	21.62	5.63
* RSD(mean/std) + $A_2(\cdot)$ + hsigmoid + $K = \begin{pmatrix}10\\10\\20\\20\end{pmatrix}$	25.76M	4.09G	22.15	6.24

4.5.1 Ablation Study

We compare different design choices in our proposed AN using ResNet-50 in ImageNet-1000. Table 4.1 summarizes the results. The ablation study is in support of the intuitions we have in elaborating the proposed AN in Section 4.4.3. All the models are trained using the same settings (the vanilla setup in Section 4.5.2).

Figure 4.5 shows the learned mixture of affine transformations ($\gamma_{k,c}$'s and $\beta_{k,c}$'s in each building block) in the ResNet-50+AN (the best performer in Table 4.1). We can see there are no degenerated cases observed in the learned affine transformations (i.e., no single mixture component that dominates).

4.5.2 Image Classification in ImageNet-1000

Common Training Settings. We use 8 GPUs (NVIDIA V100) to train models using the same settings for apple-to-apple comparisons. The method proposed in (38) is used to initialize all convolutions for all models. The batch size is 128 per GPU [3] with FP16 optimization used in training to reduce the training time [4]. The mean and standard deviation for block-wise standardization are computed *within* each GPU. The initial learning rate is 0.4, and the cosine learning rate scheduler (87) is used with 5 warm-up epochs and weight decay 1×10^{-4} and momentum 0.9. For AN, the best practice observed in our ablation study (Table 4.1) is used. AN is also used in the stem layer in the models. In addition to the common settings, we have two different setups in experimental comparisons:

i) The Vanilla Setup. We adopt the basic data augmentation scheme (random crop and horizontal flip) in training as done in (40). We train the models for 120 epochs. All ResNets (40) use the vanilla stem layer with 7×7 convolution. The MobileNets-v2 uses 3×3 convolution in the stem layer. The AOGNets use two consecutive 3×3 convolution in the stem layer. All the γ and β parameters of the feature normalization backbones are initialized to 1 and 0 respectively.

ii) The State-of-the-Art Setup. There are different aspects in the vanilla setup which have better variants developed with better performance shown (43). *We want to address whether the improvement by our proposed AN are truly fundamental or will disappear with more advanced tips and tricks added in training ConvNets.* First, on top of the basic data augmentation, we also use label smoothing (137) (with rate 0.1) and the mixup (with rate

[3] Although the best practice is to use 8 GPUs and batch size 32 per GPU for training ConvNets in ImageNet, we do not follow it since it would take too long in order to run all the experiments given the resources we have.

[4] The NVIDIA APEX library is used.

Table 4.2: Comparisons of the top-1 and top-5 error rates (%) in the ImageNet-1000 validation set using *the vanilla setup*. The numbers in brackets show the performance improvement by our proposed AN over the baselines (for ResNet-50, the numbers are shown with the baseline methods since there are multiple ones). † means the model is not trained by us. All other models are trained from scratch by us under the same settings. See text for details.

Method	#Params	FLOPS	top-1	top-5
ResNet-50-BN	25.56M	4.09G	$23.01_{(1.42)}$	$6.68_{(0.80)}$
†ResNet-50-GN (150)	25.56M	4.09G	$23.52_{(1.93)}$	$6.85_{(0.97)}$
†ResNet-50-SN (88)	25.56M	-	$22.43_{(0.83)}$	$6.35_{(0.47)}$
†ResNet-50-SE (47)	28.09M	-	$22.37_{(0.78)}$	$6.36_{(0.48)}$
ResNet-50-AN (w/ BN)	25.76M	4.09G	**21.59**	**5.88**
ResNet-101-BN	44.57M	8.12G	21.33	5.85
ResNet-101-AN (w/ BN)	45.00M	8.12G	$\mathbf{20.61}_{(0.72)}$	$\mathbf{5.41}_{(0.44)}$
MobileNet-v2-BN	3.50M	0.34G	28.69	9.33
MobileNet-v2-AN (w/ BN)	3.56M	0.34G	$\mathbf{26.67}_{(2.02)}$	$\mathbf{8.56}_{(0.77)}$
AOGNet-12M-BN	12.26M	2.19G	22.22	6.06
AOGNet-12M-AN (w/ BN)	12.37M	2.19G	$\mathbf{21.28}_{(0.94)}$	$\mathbf{5.76}_{(0.30)}$
AOGNet-40M-BN	40.15M	7.51G	19.84	4.94
AOGNet-40M-AN (w/ BN)	40.39M	7.51G	$\mathbf{19.33}_{(0.51)}$	$\mathbf{4.72}_{(0.22)}$

0.2) (157). We increase the total number of epochs to 200. We use the same stem layer with two consecutive 3×3 convolution for all models. We add the zero γ initialization trick for the feature normalization backbones (not for AN), which uses 0 to initialize the last normalization layer (BN_3 in Figure 4.4) to make the initial state of a residual block to be identity.

Results Summary. Table 4.2 and Table 4.3 show the comparison results for the two setups respectively. **Our proposed AN consistently obtains the best top-1 and top-5 accuracy results with more than 0.5% absolute top-1 accuracy increase (up to 2.0%) in all models without bells and whistles.** *The improvement is obtained with negligible extra parameters* (e.g., 0.06M parameter increase in MobileNet-v2 for 2.02% absolute top-1 accuracy increase, and 0.2M parameter increase in ResNet-50 with 1.42% absolute top-1 accuracy increase) *at almost no extra computational cost* (up to the precision used in measuring FLOPs). With ResNet-50, our AN also outperforms GN (150) and SN (88) by 1.93% and 0.83% in top-1 accuracy respectively. For GN, it is known that it works (slightly) worse than BN under the normal (big) mini-batch setting (150). For SN, our result shows that it is more beneficial to improve the re-calibration component than to learn-to-switch between

Table 4.3: Comparisons of the top-1 and top-5 error rates (%) in the ImageNet-1000 validation set using *the state-of-the-art setup*. The numbers in brackets show the performance improvement by our proposed AN over the baselines. All models are trained from scratch under the same settings. See text for details.

Method	#Params	FLOPS	top-1	top-5
ResNet-50-BN	25.56M	4.09G	21.08	5.56
ResNet-50-AN (w/ BN)	25.76M	4.09G	**19.92**$_{(1.16)}$	**5.04**$_{(0.52)}$
ResNet-101-BN	44.57M	8.12G	19.71	4.89
ResNet-101-AN (w/ BN)	45.00M	8.12G	**18.85**$_{0.86)}$	**4.63**$_{(0.26)}$
AOGNet-12M-BN	12.26M	2.19G	21.63	5.60
AOGNet-12M-AN (w/ BN)	12.37M	2.19G	**20.57**$_{(1.06)}$	**5.38**$_{(0.22)}$
AOGNet-40M-BN	40.15M	7.51G	18.70	4.47
AOGNet-40M-AN (w/ BN)	40.39M	7.51G	**18.13**$_{(0.57)}$	**4.26**$_{(0.21)}$

different feature normalization schema. It may be useful to integrate our AN and the SN in future work.

Remarks. We observe that the proposed AN is more effective for small ConvNets in terms of performance gain. Intuitively, this makes sense. Small ConvNets usually learn less expressive features. With the mixture of affine transformations and the instance-specific channel-wise feature re-calibration, the proposed AN offers the flexibility of clustering intra-class data better while separating inter-class data better in training. This is potentially very useful in state-of-the-art neural architecture search under mobile settings. The proposed AN can be included as a strong inductive bias in the search space for learning potentially more powerful networks. In the meanwhile, the proposed AN entails a second thought on the Fixup initialization (158) which attempts to remove BN without hurting the performance. Our results show it is beneficial to improve the vanilla feature normalization schema in a light-weigh manner. It will be interesting to investigate the potential integration between our AN and the Fixup. Our AN also provides an alternative for further studying how the normalization schema helps optimization as done in (121). We leave these for the further work.

4.5.3 Object Detection and Segmentation in COCO

In object detection and segmentation, high-resolution input images are beneficial and often entailed for detecting medium to small objects, but limit the batch-size in training

Table 4.4: Detection and segmentation results in MS-COCO `val2017` (84) using the Mask R-CNN (37) framework with the FPN (83) implemented in MMDetection (11). All models use 2x lr scheduling (180k iterations). $\mathbb{BN}$ means BN is frozen in fine-tuning for object detection. † means that models are not trained by us. All other models are trained from scratch by us under the same settings (fine-tuned from pre-trained models under the vanilla setup in Table 4.2 for fair comparisons). The numbers in brackets show the performance improvement by our proposed AN over the baselines (the numbers show sequential improvement in the two AOGNet models).

Architecture	Backbone	Head	#Params	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{m}	AP^{m}_{50}	AP^{m}_{75}
MobileNet-v2	$\mathbb{BN}$	-	22.72M	34.2	54.6	37.1	30.9	51.1	32.6
	AN (w/ BN)	-	22.78M	**36.0**$_{(1.8)}$	**57.0**$_{(2.4)}$	**38.9**$_{(1.8)}$	**32.5**$_{(1.6)}$	**53.8**$_{(2.7)}$	**34.5**$_{(1.9)}$
ResNet-50	$\mathbb{BN}$	-	45.71M	39.2	60.0	43.1	35.2	56.7	37.6
	AN (w/ BN)	-	45.91M	**40.8**$_{(1.6)}$	**62.1**$_{(2.1)}$	**44.5**$_{(1.4)}$	**36.4**$_{(1.2)}$	**58.9**$_{(2.2)}$	**38.7**$_{(1.1)}$
	†GN	GN (150)	45.72M	40.3$_{(1.3)}$	61.0$_{(1.0)}$	44.0$_{(1.7)}$	35.7$_{(1.7)}$	57.9$_{(1.6)}$	37.7$_{(2.2)}$
	†SN	SN (88)	-	41.0$_{(0.6)}$	**62.3**$_{(-0.3)}$	45.1$_{(0.6)}$	36.5$_{(0.9)}$	58.9$_{(0.6)}$	38.7$_{(1.2)}$
	AN (w/ BN)	AN (w/ GN)	45.96M	**41.6**	62.0	**45.7**	**37.4**	**59.5**	**39.9**
ResNet-101	$\mathbb{BN}$	-	64.70M	41.4	62.0	45.5	36.8	59.0	39.1
	AN (w/ BN)	-	65.15M	**43.1**$_{(1.7)}$	**64.1**$_{(2.1)}$	**47.3**$_{(1.8)}$	**38.2**$_{(1.4)}$	**61.0**$_{(2.0)}$	**40.7** $_{(1.6)}$
	†GN	GN (150)	64.71M	41.8	62.5	45.4	36.8	59.2	39.0
	AN (w/ BN)	AN (w/ GN)	65.20M	**43.2**$_{(1.4)}$	**64.0**$_{(1.5)}$	**47.3**$_{(1.9)}$	**38.8**$_{(2.2)}$	**61.3**$_{(2.1)}$	**41.6**$_{(2.6)}$
AOGNet-12M	$\mathbb{BN}$	-	33.09M	40.7	61.4	44.6	36.4	58.4	38.8
	AN (w/ BN)	-	33.21M	42.0$_{(1.3)}$	63.1$_{(1.7)}$	46.1$_{(1.5)}$	37.8$_{(1.4)}$	60.1$_{(1.7)}$	40.4$_{(1.6)}$
	AN (w/ BN)	AN (w/ GN)	33.26M	**43.0**$_{(1.0)}$	**64.2**$_{(1.1)}$	**46.8**$_{(0.7)}$	**38.7**$_{(0.9)}$	**61.1**$_{(1.0)}$	**41.7**$_{(1.3)}$
AOGNet-40M	$\mathbb{BN}$	-	60.73M	43.4	64.2	47.5	38.5	61.0	41.4
	AN (w/ BN)	-	60.97M	44.1$_{(0.7)}$	65.1$_{(0.9)}$	48.2$_{(0.7)}$	39.0$_{(0.5)}$	62.0$_{(1.0)}$	41.8$_{(0.4)}$
	AN (w/ BN)	AN (w/ GN)	61.02M	**44.9**$_{(0.8)}$	**66.2**$_{(1.1)}$	**49.1**$_{(0.9)}$	**40.2**$_{(1.2)}$	**63.2**$_{(1.2)}$	**43.3**$_{(1.5)}$

(often 1 or 2 images per GPU). GN (150) and SN (88) have shown significant progress in handling the applicability discrepancies of feature normalization schema from ImageNet to MS-COCO. *We test our AN in MS-COCO following the standard protocol as done in GN (150) and as implemented in the MMDetection code platform (11) with significant improvement obtained.* The results are summarized in Table 4.4 and Table 4.5.

We first summarize the details of implementation. Following the terminologies used in MMDetection (11), there are four modular components in the R-CNN detection framework (29; 113; 37):

- *Feature Backbone.* We use the pre-trained networks in Table 4.2 (with the vanilla setup) for fair comparisons in detection, since we compare with some models which are not trained by us from scratch and use the feature backbones pre-trained in a way similar to our vanilla setup and with on par top-1 accuracy. In fine-tuning a network with AN (w/ BN) pre-trained in ImageNet such as ResNet-50+AN (w/ BN) in Table 4.2, we freeze the stem layer and the first stage as commonly done in practice (. all the

Table 4.5: Results in MS-COCO using the state-of-the-art cascade variant (9) of Mask R-CNN.

Architecture	Backbone	Head	#Params	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{m}	AP^{m}_{50}	AP^{m}_{75}
ResNet-101	$\mathbb{BN}$	-	96.32M	44.4	62.5	48.4	38.2	59.7	41.3
	AN (w/ BN)	-	96.77M	**45.8**$_{(1.4)}$	**64.3**$_{(1.8)}$	**49.8**$_{(1.4)}$	**39.6**$_{(1.4)}$	**61.7**$_{(2.0)}$	**42.7**$_{(1.4)}$
AOGNet-40M	$\mathbb{BN}$	-	92.35M	45.6	63.9	49.7	39.3	61.2	42.7
	AN (w/ BN)	-	92.58M	**46.5**$_{(0.9)}$	**65.0**$_{(1.1)}$	**50.8**$_{(1.1)}$	**40.0**$_{(0.7)}$	**62.3**$_{(1.1)}$	**43.1**$_{(0.4)}$

models in our experiments). For the remaining stages, we freeze the standardization component only (the learned mixture of affine transformations and the learned running mean and standard deviation), and allow the attention weight sub-network to be fine-tuned.

- *Neck Backbone*: We test the feature pyramid network (FPN) (83) which is widely used in practice.
- *Head Classifiers*. We test two setups: *(i) The vanilla setup* as done in GN (150) and SN (88). In this setup, we further have two settings: with vs without feature normalization in the bounding box head classifier. The former is denoted by "-" in Table 4.4, and the latter is denoted by the corresponding type of feature normalization scheme in Table 4.4 (, GN, SN and AN (w/ GN)). We experiment on using AN (w/ GN) in the bounding box head classifier and keeping GN in the mask head unchanged for simplicity. Adding AN (w/ GN) in the mask head classifier may further help improve the performance. When adding AN (w/ GN) in the bounding box head, we adopt the same design choices except for "Choice 1, $A_1(\cdot)$" (Eqn. 4.12) used in learning attention weights. *(ii) The state-of-the-art setup* which is based on the cascade generalization of head classifiers (9) and does not include feature normalization scheme, also denoted by "-" in Table 4.5.
- *RoI Operation*. We test the RoIAlign operation (37), the default setup in Mask R-CNN.

Compared with the vanilla BN that are frozen in fine-tuning, our AN (w/ BN) improves performance by a large margin in terms of both bounding box AP and mask AP (*1.8% & 1.6%* for MobileNet-v2, *1.6% & 1.2%* for ResNet-50, *1.7% & 1.4%* for ResNet-101, *1.3% & 1.4%* for AOGNet-12M and *0.7% & 0.5%* for AOGNet-40M). It shows the advantages of the self-attention based dynamic and adaptive control of the mixture of affine transformations (although they themselves are frozen) in fine-tuning.

When the AN is further integrated in the bounding box head classifier of Mask-RCNN and trained from scratch, we also obtain better performance than GN and SN. Compared

Table 4.6: Top-1 accuracy of ResNet-50 on ImageNet-1K with adversarial training. The experiment is performed with Adversarial Training for Free with m=4 (cite), which is a fast version of adversarial training, m is PGD attack steps. The model is tested with clean images, and PGD attack with 10 iterations, and 50 iterations, respectively.

Method	#Params	Clean	PGD-10	PGD-50
ResNet-50-BN	25.56M	60.12	32.26	31.40
ResNet-50-AN (w/ BN)	25.76M	**63.05**$_{(2.93)}$	**34.34**$_{(2.08)}$	**33.41**$_{(2.01)}$

with the vanilla GN (150), our AN (w/ GN) improves bounding box and mask AP by 1.3% and 1.7% for ResNet-50, and 1.4% and 2.2% for ResNet-101. Compared with SN (88) which outperforms the vanilla GN in ResNet-50, our AN (w/ GN) is also better by 0.6% bounding box AP and 0.9% mask AP increase respectively. Slightly less improvements are observed with AOGNets.

Similar in spirit to the ImageNet experiments, we want to verify whether the advantages of our AN will disappear if we use state-of-the-art designs for head classifiers of R-CNN such as the widely used cascade R-CNN (9). Table 4.5 shows that similar improvements are obtained with ResNet-101 and AOGNet-40M.

4.5.4 Adversarial training for model robustness

4.6 Conclusion

This work presents Attentive Normalization (AN) that harnesses the best of feature normalization and feature attention in a single lightweight module. AN learns a mixture of affine transformations and uses the weighted sum via a self-attention module for re-calibrating standardized features in a dynamic and adaptive way in training, testing and fine-tuning. AN can be used as a drop-in replacement for existing feature normalization schema. In experiments, the proposed AN is tested in ImageNet and MS-COCO with three representative networks (ResNets, MobileNets-v2 and AOGNets). It consistently obtains better performance, often by a large margin, than the vanilla feature normalization schema and some state-of-the-art variants.

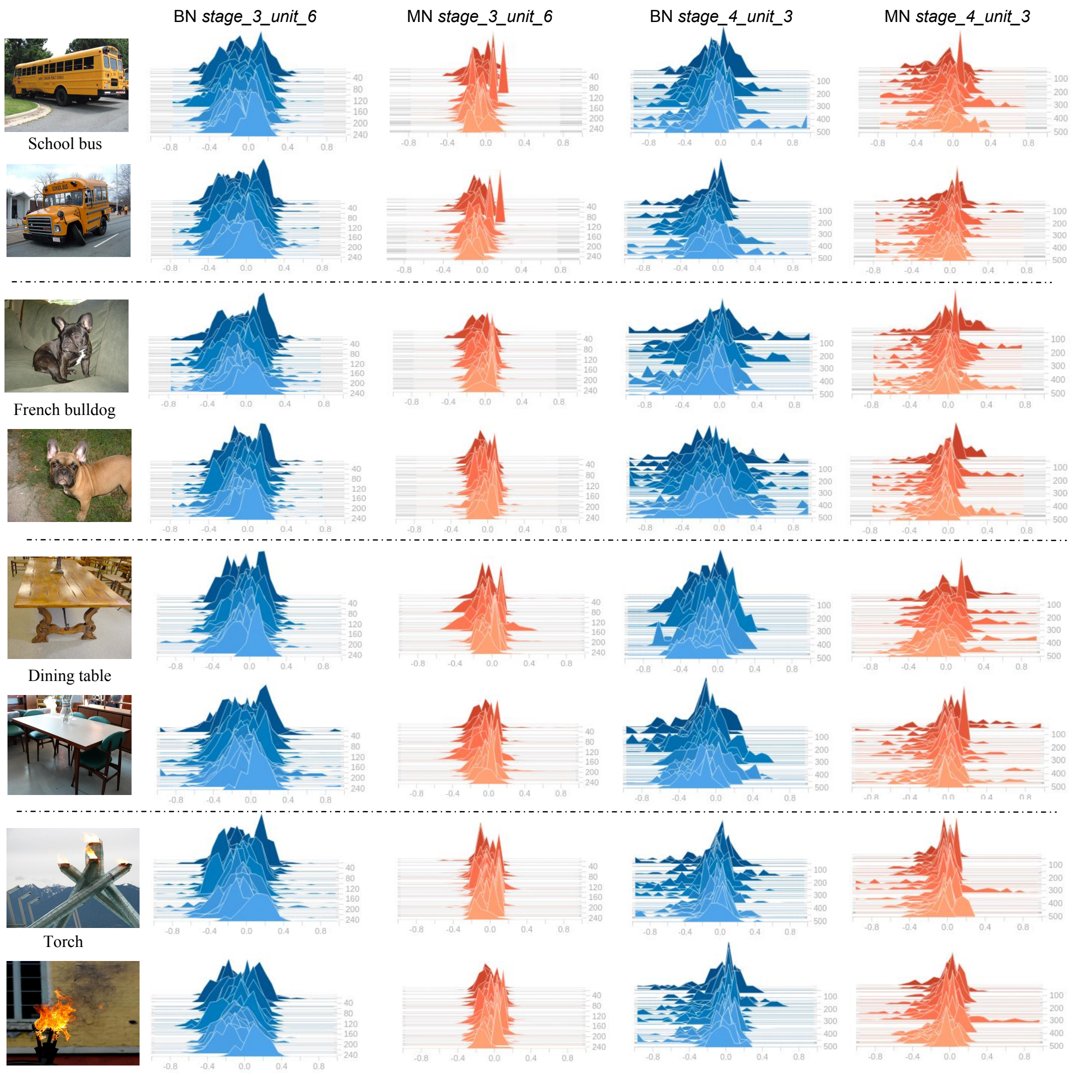


Figure 4.6: Illustration of the effects of AN and BN on filter responses. We show the filter response histograms (marginal distributions) for different images in different categories. Here we show results of a 4-stage ResNet50. $stage\_i\_unit\_j$ means the histograms are plot for the output feature map of the j-th ResBlock in the i-th stage. From the histograms, we observe that for images from the same class (e.g., school bus), the histograms of our AN show higher similarities with smaller variance. This empirically shows that a channel-wise attention guided mixture of affine transformation helps recalibrate the normalized responses in a more meaningful way.

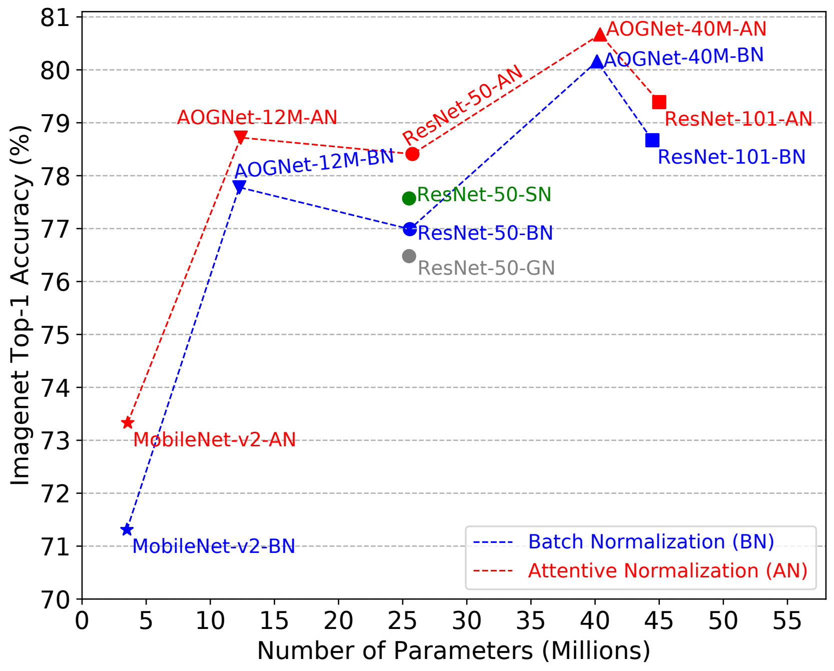


Figure 4.7: Performance plots for the proposed Attentive Normalization (AN) and the vanilla Batch Normalization (BN) (57) across three neural architectures, ResNets (40), MobileNets-v2 (120) and AOGNets (74) in ImageNet-1000 (116). The proposed AN consistently improves performance. It also outperforms other variants of BN tested using ResNet-50: GroupNorm (GN) (150) and SwitchableNorm (SN) (88). See text for details. Best viewed in color.

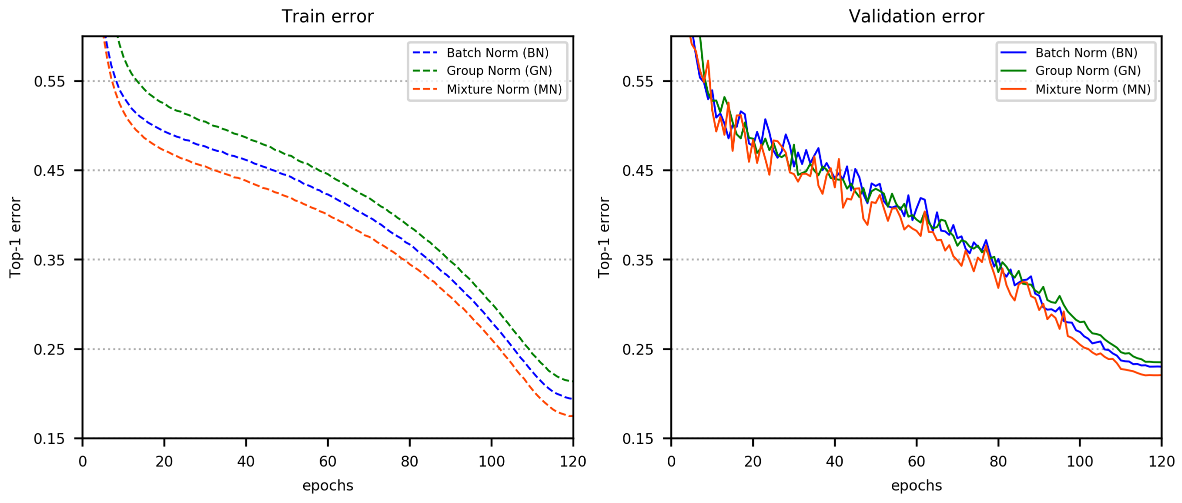


Figure 4.8: Training and validation loss curves of ResNet-50 in ImageNet with BN, GN and AN respectively. Best viewed in color.

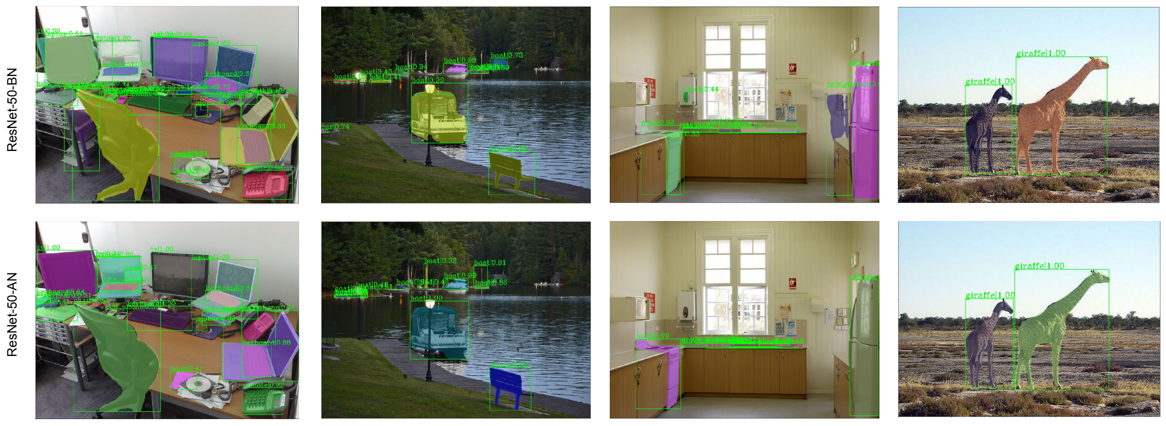


Figure 4.9: Sample comparisons between Mask R-CNN with ResNet-50-AN and ResNet-50-BN backbones on COCO.

CHAPTER

5

LEARN TO GROW: A CONTINUAL STRUCTURE LEARNING FRAMEWORK FOR OVERCOMING CATASTROPHIC FORGETTING

Addressing catastrophic forgetting is one of the key challenges in continual learning where machine learning systems are trained with sequential or streaming tasks. Despite recent remarkable progress in state-of-the-art deep learning, deep neural networks (DNNs) are still plagued with the catastrophic forgetting problem. This chapter presents a conceptually simple yet general and effective framework for handling catastrophic forgetting in continual learning with DNNs. The proposed method consists of two components: a neural structure optimization component and a parameter learning and/or fine-tuning component. By separating the explicit neural structure learning and the parameter estimation, not only is the proposed method capable of evolving neural structures in an intuitively meaningful way, but also shows strong capabilities of alleviating catastrophic forgetting in experiments.

Furthermore, the proposed method outperforms all other baselines on the permuted MNIST dataset, the split CIFAR100 dataset and the Visual Domain Decathlon dataset in continual learning setting.

5.1 Introduction

Learning different tasks continuously is a common and practical scenario that happens all through the course of human learning. The learning of new skills from new tasks usually does not have negative impact on the previously learned tasks. Furthermore, with learning multiple tasks that are highly related, it often helps to advance all related skills. However, this is commonly not the case in current machine learning with deep neural networks (DNNs). When presented a sequence of learning tasks, DNNs experiences so called “catastrophic forgetting" problem (92; 106), where they usually largely “forget" previously learned tasks after trained for a new task. This is an interesting phenomenon that has attracted lots of research efforts recently.

To overcome catastrophic forgetting, approaches such as Elastic Weight Consolidation (EWC 62) and synaptic intelligence (156) introduce constraints to control parameter changes when learning new tasks. However, forgetting is still non-negligible with these approaches, especially when the number of tasks increases. Forgetting is also addressed with memory-based methods, where certain information regarding learned tasks are stored to help retaining the performance of the learned tasks (see 86; 124, for example). Additionally, there are methods (89; 110; 111; 90) that learn multiple domains and completely avoid forgetting by adding a small amount of parameters while the previously estimated parameters are kept fixed. However, these models rely on a strong base network and knowledge transferability is limited mainly between two consecutive tasks.

Most of the current approaches in continual learning with DNNs couple network structure and parameter estimation and usually apply the same model structure for all tasks. Here, *we propose to explore a more intuitive and sensible approach*, that is to learn task specific model structures *explicitly* while retaining model primitives sharing, decoupling from model parameter estimation[1]. Different tasks may require different structures, especially if they are not relevant, so it may not make much sense to employ the same structure in learning. For example, consider the tasks of learning digit and face recognition DNNs, the

[1]The structure that referred here is more fine-grained, such as number of layers, type of operations at each layer, etc. It does not refer to generic structure names like convolutional neural networks or recurrent neural networks.

lower level layers (features) required for the two tasks are likely to be drastically different, thus entailing different overall structures that have task specific low level layers. Forcing the same structure for these tasks is likely to cause catastrophic forgetting for one task (e.g., digit recognition) after the other task (e.g., face recognition) is trained. On the other hand, if different tasks learn to explore different structures and grow their specific components accordingly, it still has the potential to share common feature layers while maximizing the performance for new tasks.

Here, we present a **learn-to-grow framework** that explicitly separates the learning of model structures and the estimation of model parameters. In particular, we employ architecture search to find the optimal structure for each of the sequential tasks. The search accounts for various options, such as sharing previous layers' parameters, introducing new parameters, and so on. After the structure is searched, the model parameters are then estimated. We found that *1) explicitly continual structure learning makes more effective use of parameters among tasks, which leads to better performance and sensible structures for different tasks; 2) separating the structure and parameter learning significantly reduced catastrophic forgetting as compared to other baseline methods with similar model complexities.*

5.2 The Proposed Learn-to-Grow Framework

5.2.1 Problem Definition of Continual Learning

Consider a sequence of N tasks, denoted by $\mathbf{T} = (T_1, T_2, ..., T_N)$. Each task T_t has a training dataset, $\mathcal{D}_{train}^{(t)} = \{(x_i^{(t)}, y_i^{(t)}); i = 1, \cdots, n_t\}$, where $y_i^{(t)}$ is the ground-truth annotation (e.g., a class label) and n_t is the number of training examples. Let $\mathcal{D}_{train} = \cup_{t=1}^{N} \mathcal{D}_{train}^{(t)}$ be the entire training dataset for all tasks. Similarly, denote by $\mathcal{D}_{test}^{(t)}$ the test dataset for a task T_t. Denote by $f(\cdot;\Theta_t)$ the model (e.g., a DNN) in learning where Θ_t collects all learned parameters up to the current task T_t (inclusive). The model gets to observe tasks from 1 to N sequentially. After the model is trained on a task T_t using its training dataset $\mathcal{D}_{train}^{(t)}$, both $\mathcal{D}_{train}^{(t)}$ and $\mathcal{D}_{test}^{(t)}$ will not be available in training tasks from T_{t+1} to T_N. *The main objective of continual learning* is to maximize the performance of $f(\cdot;\Theta_t)$ at the task T_t while minimizing the forgetting for tasks from T_1 to T_{t-1}, all being evaluated in their test datasets $\mathcal{D}_{test}^{(t')}$ $(1 \leq t' \leq t)$. Ideally, we would like to minimize the following objective function in this continual learning

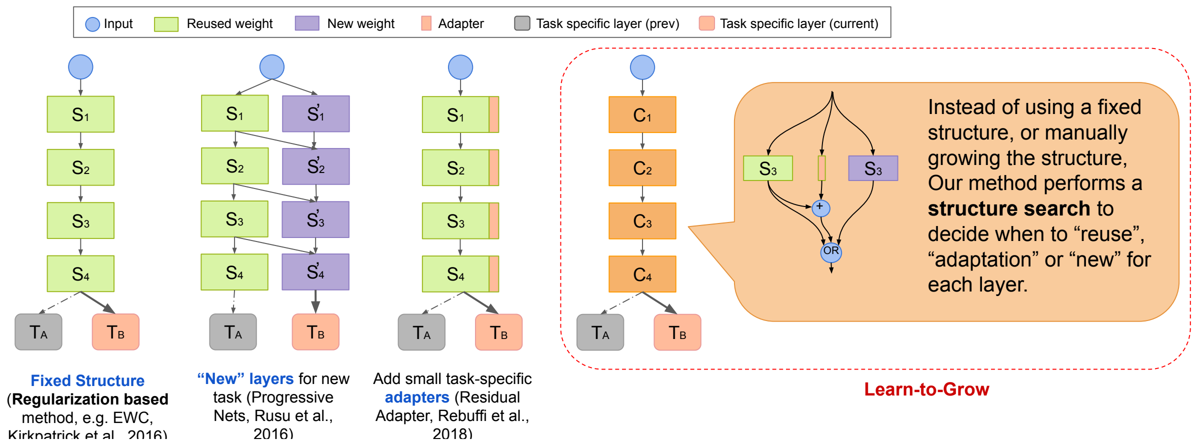

Input
Reused weight
New weight
Adapter
Task specific layer (prev)
Task specific layer (current)
Instead of using a fixed structure, or manually growing the structure, Our method performs a structure search to decide when to "reuse", "adaptation" or "new" for each layer.
Fixed Structure (Regularization based method, e.g. EWC, Kirkpatrick et al., 2016)
"New" layers for new task (Progressive Nets, Rusu et al., 2016)
Add small task-specific adapters (Residual Adapter, Rebuffi et al., 2018)
Learn-to-Grow

Figure 5.1: Illustration of different continual learning approaches. a) All but the task specific layer are shared, catastrophic forgetting is countered by techniques that prevents parameters to move to lower energy regions of previous tasks. b) Each task will add some fixed task specific parameters, all layers' original weights are not tuned, and thus prevents forgetting. c) Our approach, where network structure is learned by architecture search. In this example, the search result decides to "reuse" the first two layer, do "adaptation" for the 3rd layer and allocate "new" weight for the 4th layer.

setting,

$$\mathscr{L}(\Theta_N;\mathscr{D}_{train}) = \sum_{t=1}^{N} \mathscr{L}_t(\Theta_t;\mathscr{D}_{train}^{(t)}) \tag{5.1}$$

$$\mathscr{L}_t(\Theta_t;\mathscr{D}_{train}^{(t)}) = \frac{1}{n_t}\sum_{i=1}^{n_t} \ell_t(f(x_i^{(t)};\Theta_t), y_i^{(t)}) \tag{5.2}$$

where ℓ_t is the loss function for task T_t (e.g., the cross-entropy loss) and the model regularizer term is omitted for notion simplicity. However, since we do not have access to all datasets at the same time, the above objective function (Eqn. 5.1) can not be directly computed and then minimized. The challenge is to maintain $\sum_{t'=1}^{t-1}\mathscr{L}_{t'}(\Theta_{t'};\mathscr{D}_{train}^{(t')})$ not to change too much without explicitly measuring it due to the streaming setting, while estimating Θ_t via minimizing Eqn. 5.2 in isolation.

As illustrated in Fig. 5.1 (b), one straightforward solution is to keep Θ_{t-1} fixed when learning $\Theta_t = \Theta_{t-1} \cup \theta_t$ to avoid catastrophic forgetting completely, where θ_t is the new parameters introduced for a new task T_t. How to introduce θ_t for each task sequentially is usually hand-crafted and some simple heuristics are often used, e.g., by adding some extra channels for each layer in a DNN. By doing this, the model will become more and more complicated for incoming tasks and Θ_{t-1} is “artificially" enforced to be reused for a new task T_t without accounting for their potential dissimilarities. So, the computational efficiency and accuracy performance of new tasks are traded-off for avoid catastrophic forgetting.

As illustrated in Fig. 5.1 (a), another way of addressing catastrophic forgetting is to utilize a single set of parameters Θ for all tasks, and control the changes of parameter values from Θ_{t-1} to Θ_t using some statistically inspired functions such as the Fisher information criterion used in EWC (62). Following this direction, the accuracy performance of new tasks are usually suffered from the constrained parameter space and well-designed initial models are entailed for ensuring reasonably good performance across tasks. Furthermore, the effectiveness of the parameter change control functions is often unknown as the number of tasks increases at scale.

5.2.2 Our Proposed Learn-to-Grow Framework

In our learn-to-grow framework (Fig. 5.1 (c)), we adopt the growing strategy as stated above in learning $\Theta_t = \Theta_{t-1} \cup \theta_t$. But, we learn to “grow" θ_t on top the previously trained model Θ_{t-1} and to exploit Θ_{t-1} in a flexible way without enforcing to reuse all of them. Our proposed method is also complementary to the elastic parameter strategy as used in EWC. For the learned-to-reuse parameters in Θ_{t-1}, we can either keep them fixed or allow them to change subject to some elastic penalty functions. So, the proposed method can harness the best of both, and is capable of avoid catastrophic forgetting of old tasks completely without sacrificing the computational efficiency and accuracy performance of new tasks. We introduce $s_t(\Theta_t)$ to indicate the task-specific model for task T_t. The loss function (Eqn. 5.2) is changed to,

$$\mathscr{L}_t(s_t(\Theta_t)) = \frac{1}{n_t}\sum_{i=1}^{n_t} \ell_t(f(x_i^{(t)}; s_t(\Theta_t)), y_i^{(t)}) \tag{5.3}$$

Now the structure is explicitly taken into consideration when learning all the tasks. When optimizing the updated loss function in Eqn. 5.3, one needs to determine the optimal parameter based on the structure s_t. This loss can be viewed in two ways. One can interpret it as selecting a task specific network from a ‘super network’ that has parameter Θ using s_t, or for each task we train a new model with parameter $s_t(\Theta_t)$. There is a subtle difference between this two views. The former one has an constraint on the total model size, while the latter one does not. So, in the worst case scenario of the latter, the model size will grow linearly as we increase the number of tasks. This would lead to a trivial solution – training completely different models for different tasks and is no longer continual learning! To address this problem, we propose the following penalized loss function,

$$\mathscr{L}_t(s_t(\Theta_t)) = \frac{1}{n_t}\sum_{i=1}^{n_t} \ell_t(f(x_i^{(t)}; s_t(\Theta_t)), y_i^{(t)}) + \\ \beta_t \mathscr{R}_t^s(s_t) + \lambda_t \mathscr{R}_t^p(\Theta_t) \tag{5.4}$$

where $\beta_t > 0$, $\lambda_t \geq 0$, $\mathscr{R}_t^s$ and $\mathscr{R}_t^p$ represent the regularizers for the network structure and model parameters respectively. For instance, one can use ℓ_2 regularization for $\mathscr{R}_t^p$ when optimizing model parameters, and $\mathscr{R}_t^s$ can be as simple as the (log) number of parameters. In this way, the total number of parameters are bounded from above, and the degenerate cases are thus avoided.

5.3 Our Implementation

It is a challenging problem to optimize the loss described in Eqn. 5.4, since it involves explicit optimization of the structure of the model. In our implementation, we focus on continual learning with DNNs. The proposed method consists of two components: a neural structure optimization component and a parameter learning and/or fine-tuning component. The former learns the best neural structure for the current task on top of the current DNN trained with previous tasks. It learns whether to reuse or adapt building blocks or layers in the current DNN, or to create new ones if needed under the differentiable neural architecture search framework (85). The latter estimates parameters for newly introduced structures, and fine-tunes the old ones if preferred. We present details in the following sections (see Fig. 5.2).

5.3.1 Structure Optimization

We employ neural architecture search (NAS) for structure optimization. Before we move on to further details, we adopt a further simplification that a global network topology is given and could work for all tasks of interest, such as a ResNet (41). We optimize the wiring pattern between layers and their corresponding operations. It is straightforward to extend this to more complicated cases, e.g., by utilizing NAS at the first task.

Consider a network with L shareable layers and one task-specific layer (i.e. last layer)

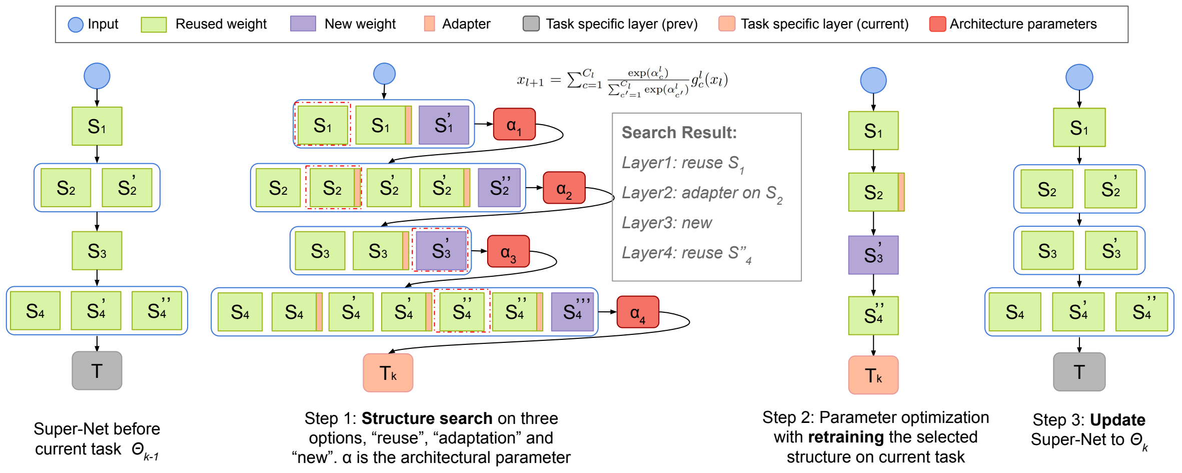

Input
Reused weight
New weight
Adapter
Task specific layer (prev)
Task specific layer (current)
Architecture parameters
$x_{l+1} = \sum_{c=1}^{C_l} \frac{\exp(\alpha_c^l)}{\sum_{c'=1}^{C_l}\exp(\alpha_{c'}^l)} g_c^l(x_l)$
Search Result:
Layer1: reuse S_1
Layer2: adapter on S_2
Layer3: new
Layer4: reuse S''_4
Super-Net before current task Θ_{k-1}
Step 1: Structure search on three options, "reuse", "adaptation" and "new". α is the architectural parameter
Step 2: Parameter optimization with retraining the selected structure on current task
Step 3: Update Super-Net to Θ_k

Figure 5.2: Illustration of the proposed learn-to-grow framework. a) Current state of super model. In this example, the 1st and 3rd layers have single copy of weight, while the 2nd and 4th has two and three respectively. b) During search, three options, "`reuse`", "`adaptation`" and "`new`" are utilized. α is the weight parameters for the architecture. c) Parameter optimization with selected architecture on the current task k. d) Update super model to add the newly created S_3'. See text for details.

for each task. A super network $\mathscr{S}$ is maintained so that all the new task-specific layers and new shareable layers will be stored into $\mathscr{S}$.

The goal of search is to seek the optimal choice for each of the L layers, given the current task data $\mathscr{D}_{train}^{(t)}$ and all the shareable layers' weights stored in $\mathscr{S}$. The candidate choices for each layer are defined by: "`reuse`", "`adaptation`" and "`new`". The `reuse` choice will make new task use the same parameter as the previous task. The `adaptation` option adds a small parameter overhead that trains an additive function to the original layer output. The `new` operator will spawn new parameters of exactly the same size of the current layer parameters. Here, we denote the size of the l_{th} layer in super network $\mathscr{S}$ as $|\mathscr{S}^l|$. The total number of choices in the l_{th} layer C_l is $2|\mathscr{S}^l|+1$, because we will have $|\mathscr{S}^l|$ "`reuse`", $|\mathscr{S}^l|$ "`adaptation`" and l "`new`". Thus, the total search space is $\prod_l^L C_l$. One potential issue here is that, in the worst case, the search space may grow exponentially with respect to the number of tasks. One way of addressing this is to limit the total number of possible choices, and maintain a priority queue for learning the options. We do not find this necessary in all of our experiments.

Similar to DARTS (85), to make the search space continuous, we relax the categorical choices of the l_{th} layer as a `Softmax` over all possible C_l choices, i.e.

$$x_{l+1} = \sum_{c=1}^{C_l} \frac{\exp(\alpha_c^l)}{\sum_{c'=1}^{C_l} \exp(\alpha_{c'}^l)} g_c^l(x_l) \tag{5.5}$$

Here, the vector α^l of dimension C_l is the architecture weights that are used for mixing the choices for each sharable layer. And g_c^l here is the operator for the choice c at layer l which is expressed as:

$$g_c^l(x_l) = \begin{cases} S_c^l(x_l) & \text{if } c \leq |\mathscr{S}^l|, \\ S_c^l(x_l) + \gamma_{c-|\mathscr{S}^l|}^l(x_l) & \text{if } |\mathscr{S}^l| < c \leq 2|\mathscr{S}^l|, \\ o^l(x_l) & \text{if } c = 2|\mathscr{S}^l| + 1 \end{cases} \tag{5.6}$$

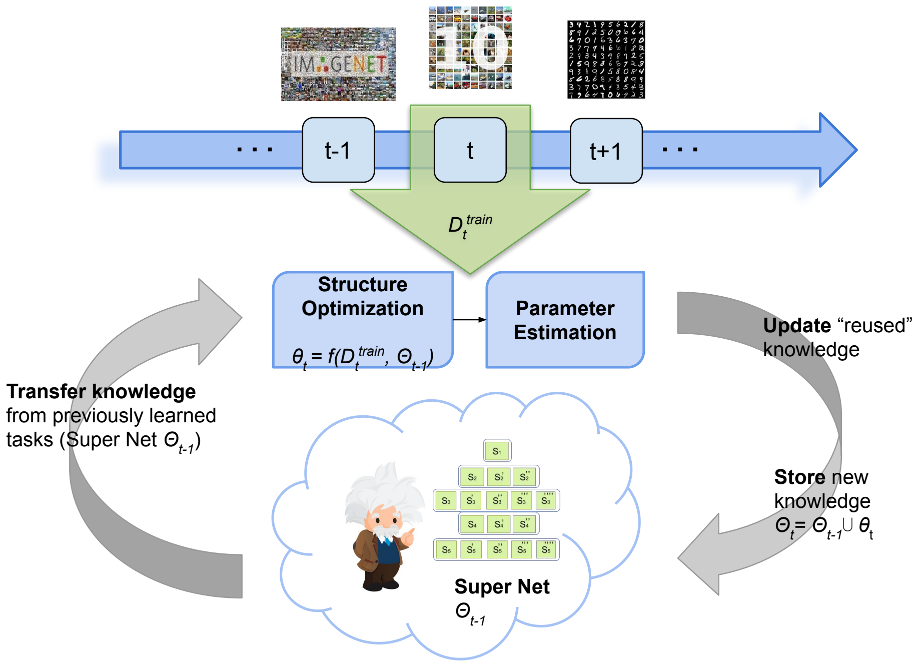
Here, γ is the `adaption` operation and o the `new` operation to be trained from scratch. After this relaxation, the task of discrete search is posed as optimizing a set of continuous weights $\alpha = \{\alpha^l\}$. After the search, the optimal architecture is obtained by taking the index with the largest weight α_c^l for each layer l, i.e. $c_l = \arg\max \alpha^l$.

Adopting the training strategy from DARTS, we split the training dataset $\mathscr{D}_{train}^{(t)}$ into two subsets: a validation subset for NAS, and a training subset for parameter estimation. We use validation loss L_{val} to update the architecture weights α, while the parameters are estimated by the training loss L_{train}. The architecture weights and parameters are updated alternately during the search process. Because it is a nested bi-level optimization problem, the original DARTS provide a second-order approximation for more accurate optimization. In this work, we find it is sufficient to use the simple alternately updating approach, which was referred as the first-order approximation in (85).

To make it clear how “`reuse`", “`adaptation`" and “`new`" operations work, we walk through a concrete example in the following. Let us take a convolutional neural network (CNN) with all the layers using 3×3 kernel size as an example. The choice of “`reuse`" is just using the existing weights and keep them fixed during learning, thus there is no additional parameter cost. For “`adaptation`", it uses a 1×1 convolution layer added to the original 3×3 convolution layer in parallel, similar to the adapter used in (110). During training, the weight of the original 3×3 kernel is kept fixed, while the parameters of the 1×1 adapter is learned. In this case, the additional parameter cost is only $1/9$ of the original parameter size. For the “`new`" operation, it introduces a replicated 3×3 layer that is initialized randomly and trained from scratch. We make use of the loss function L_{val} to implement the regularizer $\mathscr{R}_i^s(s_i)$. The value of the regularizer is set proportional to the product of the additional parameter size z_c^l and its corresponding weight α_c^l (i.e. $\mathscr{R}_i^s(s_i) = \sum_{c,l} \alpha_c^l z_c^l$). The architecture weights α is optimized in terms of both accuracy and parameter efficiency at the same time.

5.3.2 Parameter Optimization

After the search, we retrain the model on the current task. There are two strategies to deal with “`reuse`", we can either fix it unchanged during retraining just as in search, or fine-tune it with some regularization – simple ℓ_2 regularization or more sophisticated methods such as the EWC (62). The former strategy could avoid forgetting completely, however it will lose the chance of getting positive backward transfer, which means the learning of new tasks may help previous tasks' performance. When the search process select “`reuse`" at layer l, it means that the l_{th} layer tends to learn very similar representation as it learned from one of the previous tasks. This indicates semantic similarity learned at this layer l between the two tasks. Thus, we conjecture that fine-tuning the “`reuse`" l_{th} layer with some regularization could also benefit the previous tasks (elaborated in experiments)After retraining on the current task, we need to update/add the created and tuned layers, task-specific adapters and classifiers in the maintained super network.



t-1
t
t+1
D_t^{train}
Structure Optimization
$\theta_t = f(D_t^{train}, \Theta_{t-1})$
Parameter Estimation
Update “reused” knowledge
Transfer knowledge from previously learned tasks (Super Net Θ_{t-1})
Store new knowledge $\Theta_t = \Theta_{t-1} \cup \theta_t$
Super Net Θ_{t-1}

Figure 5.3: Overview of Learn-to-Grow Framework that learning on sequential tasks.

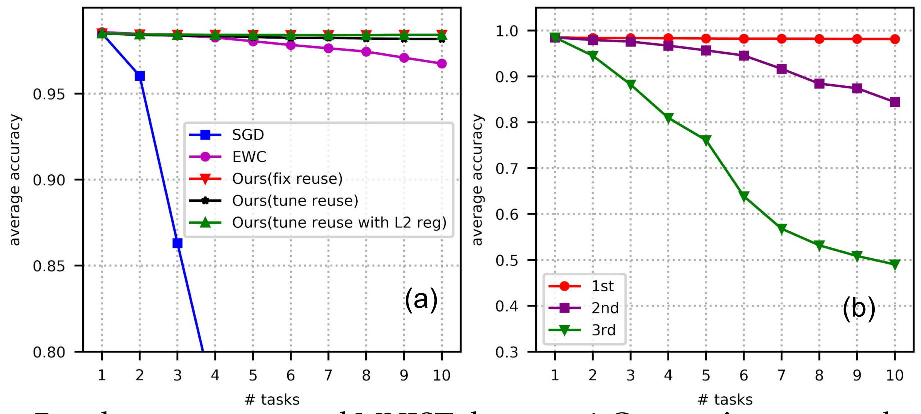


Figure 5.4: Results on permutated MNIST dataset. a) Comparing our method (fix, tune reuse with and without regularization) with SGD and EWC on the average accuracy over the seen tasks. b) Ablation experiments of "new" different layers in terms of average accuracy over the seen tasks.

5.4 Experiments

In this section, we first test two main hypotheses that motivate our proposed learn-to-grow framework and then compare with state-of-the-art continual learning approaches. First, will sensible model architectures be sought via the explicit structure learning for new tasks? Second, will the optimized structure results in better continual learning, i.e., overcoming catastrophic forgetting? We test these two hypotheses on two datasets: the permuted MNIST and the visual domain decathlon dataset (110). **The permuted MNIST dataset** is a simple image classification problem that derived from the MNIST handwritten digit dataset (153), which is commonly used as benchmark in the continual learning literature (62; 86; 156). For each task, a unique fixed random permutation is used to shuffle the pixels of each image, while the annotated label is kept fixed. **The visual decathlon dataset** (VDD) consists of 10 image classification tasks – ImageNet, CIFAR100, Aircraft, DPed, Textures, GTSRB, Omniglot, SVHN, UCF101 and VGG-Flowers. The images of all the tasks are rescaled with the lower-edge being 72 pixels. The tasks are across multiple domains with highly imbalanced dataset, which makes it a good candidate to investigate the continual learning problem and analyze potential inter-task transfer, either forward or backward.

For experiments in permuted MNIST, we use a 4-layer fully-connected networks with 3 feed-forward layers and the $4th$ layer is the *shared* softmax classification layer across all tasks. This corresponds to the so called 'single head' setup (20). We choose to use this setting because for the permuted MNIST dataset all the tasks share the same semantics, and sharing the task classifier is a more reasonable design choice. We test our method in learning the 10 permuted MNIST tasks in sequence. For simplicity, we only use two options, “`reuse`" and “`new`" during the structure optimization.

For experiments in VDD, we use a 26-layer ResNet (41) as the base network to learn the first task. This network consists of 3 *basic* residual blocks with output feature channels being 64, 128 and 256 respectively. At the end of each residual block, the feature resolution is halved by average pooling. We adopt all the three options during the search. For “`adaptation`", a 1×1 convolution layer is used as the adapter.

5.4.1 Are Sensible Structures Sought in Learn-to-Grow?

For the permuted MNIST dataset, we may expect that a sensible evolving architecture for the 10 tasks tends to share higher level layers due to the same task semantics, but to differ at lower layers accounting for the pixel shuffling. Interestingly, our experiments show that the structure optimization indeed selects the same wiring pattern that applies “`new` to the first layer and “`reuse` to the remaining layers in all the 10 tasks. This shows that the structure optimization component is working properly.

Although the learned wiring patterns are intuitive, we perform further experiments to address what if we force to use “`new` for the learned “`reuse`" layers? We enumerate and compare the three settings that the i-th layer is “`new`" and others are “`reuse`" ($i = 1, 2, 3$). In the results, we found that the learned pattern is actually the best choice compared with the other two settings (see Fig. 5.4 b).

In VDD, we test our method between two tasks. As shown in Fig. 5.5 (a), when the two tasks are similar (ImageNet and CIFAR-100, both consisting of natural images), most of the layers are shared for these two tasks. When two drastically different tasks are used, e.g., ImageNet and Omniglot, as Fig. 5.5 (b) shows, most layers in the learned structure select the “`new`" option.

The above experimental results show that the proposed structure optimization does result in sensible task specific structures with proper model primitive sharing. The learned task structure tends to share when the semantic representation of corresponding layers are similar, and spawn new parameters when the required information is very different.

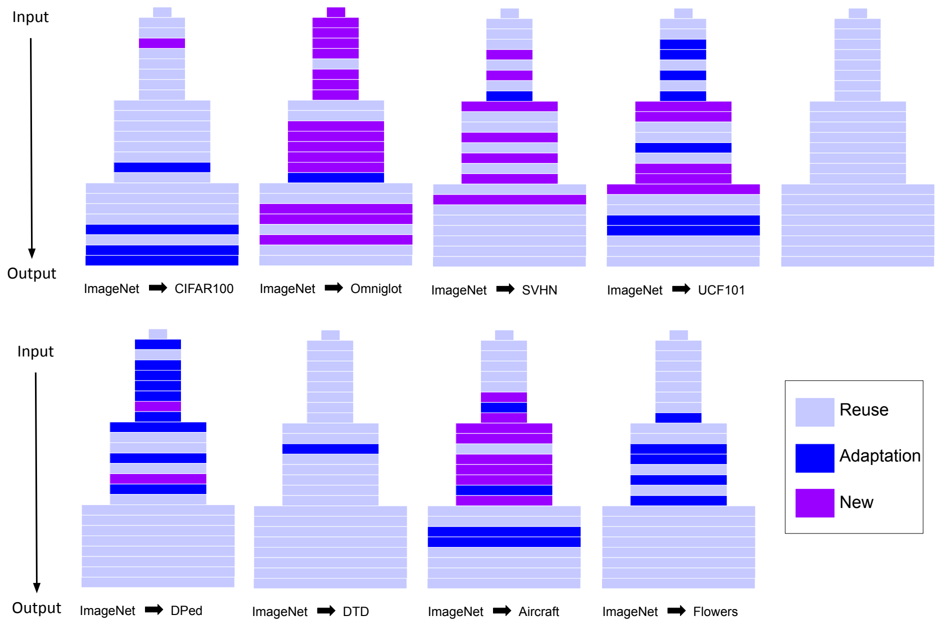


Figure 5.5: Visualization of searched architecture with learning two tasks sequentially. The search are based on the super model obtained by training ImageNet as first task. (a) and (b) shows searched architecture on CIFAR100 and Omniglot task respectively.

5.4.2 Are Forgetting Addressed in Learn-to-Grow?

Obviously, if the "`reuse`" layers are kept fixed in learning, our method does not have any forgetting. We are interest in how significant the forgetting will be when we fine-tune the "`reuse`" layers.

We first test this in the permuted MNIST. As a baseline, we show that simply updating all the layers with stochastic gradient descent (SGD) from task to task (i.e., the setting in Fig. 5.1 (a)) results in catastrophic forgetting (see Fig. 5.4 (a)). After training the 10 tasks sequentially, the average accuracy dropped from 97.9% to 63.0%. With the EWC used in learning (62), the forgetting is alleviated and the average accuracy is 96.8%. For our proposed learn-to-grow approach, we show that tuning the "`reuse`" layers by using a simple l_2 based regularization on previous task parameters (i.e. $||\Theta_i - \Theta_j||_2^2$, where Θ_i is the parameters for the current task and Θ_j is the parameters from the j-th task that selected to reuse) is sufficiently safe in terms of eliminating the forgetting. Both strategies, fixing the "`reuse`" layers or fine-tuning them with simple l_2 regularization can keep the overall accuracy as high as training each task individually (see Fig. 5.4 (a)). Encouraged by the above result, we further conduct experiments by tuning the "`reuse`" layers with smaller learning rate

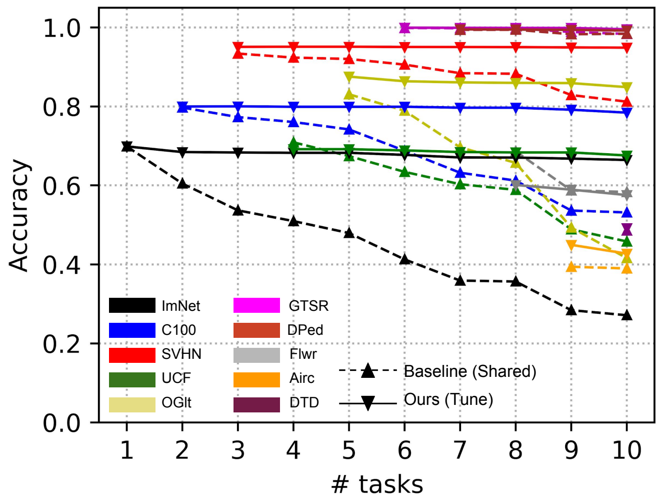


Figure 5.6: Comparisons of the catastrophic forgetting effects between our proposed approach and the baseline in VDD.

without using any regularization. In other words, we do not add any regularization to the parameters to mitigate forgetting among different tasks. The results are shown in Fig. 5.4 (a), which almost have the same behavior compared to the ℓ_2 regularization. This suggests that the learned structures actually make sense in terms of the “`reuse`" decisions, and the reused parameters are near-optimal for specific tasks.

We continue to test this in VDD. A predefined order of the ten tasks is used: ImageNet, CIFAR-100, SVHN, UCF101, Omniglot, GTSR, DPed, Flower, Aircraft and Textures (results for different ordering are provided in the supplementary material). As a baseline, we also train a model that shares all layers in the backbone and updates them from task to task. The baseline model and our learn-to-grow model are trained with similar settings as in the permuted MNIST experiments, and we choose not to use any regularization for fine-tuning the “`reuse`" layers due to the positive results we obtain in the permuted MNIST experiment. As can be seen from Fig.5.6, our learn-to-grow method significantly outperforms the baseline approach.

We also compare with other baselines in VDD and the results are shown in Table. 5.1. Our method obtains the best overall results and the total model size is similar to the “adapter" approach (110)[2]. Our approach obtains the best results in five out of nine tasks. Especially in tasks with small data size, e.g. VGG-Flowers and Aircraft, our method outperforms other

[2]The adapter proposed by Rebuffi et al. is targeted for the VDD dataset, which is not a continual learning method.

baselines by a large margin.

Table 5.1: Results of the (top-1) validation classification accuracy (%) on Visual Domain Decathlon dataset, top-2 performance are highlighted. The total model size (“#params") is the total parameter size (in Million) after training the 10 tasks. Individual indicates separate models trained for different tasks. Classifier denotes that a task specific classifier (i.e. the last softmax layer) is tuned for each task. Adapter refers to methods proposed by Rebuffi et al.(110).

Model	ImNet	C100	SVHN	UCF	OGlt	GTSR	DPed	Flwr	Airc.	DTD	avg.	#params
Individual	69.84	73.96	**95.22**	69.94	**86.05**	**99.97**	**99.86**	41.86	**50.41**	29.88	71.70	58.96M
Classifier	69.84	77.07	93.12	62.37	79.93	99.68	98.92	**65.88**	36.41	48.20	73.14	6.68M
Adapter	69.84	**79.82**	94.21	**70.72**	85.10	**99.89**	**99.58**	60.29	50.11	**50.60**	**76.02**	12.50M
Ours (fix)	69.84	**79.59**	**95.28**	**72.03**	**86.60**	99.72	99.52	**71.27**	**53.01**	**49.89**	**77.68**	14.46M

To analyze why the simple fine-tuning strategies for the “`reuse`" layers work, we calculate the l_2 distance between the parameters before and after fine-tuning for each task in VDD. We want to check if the “`reuse`" layers are almost at an optimal position for the current task to use (i.e., the l_2 distance will be relatively small). Fig. 5.9 (a) and (b) show the ℓ_2 distances between the parameters learned in the very first task and those after tuned in the following tasks for the first and last layers respectively. It is clear that the fine-tuned parameters in our learn-to-grow method do not move far away from the original location in the parameter space as compared to the baseline method, which explains why our method has less forgetting in fine-tuning the “`reuse`" layer[3]. In addition, we notice that the distances in our methods are more or less at the same scale across all layers. This may attribute to the fact that the learn-to-grow of parameters and structures is explicitly optimized, and thus the selected ones are more compatible with a new task. Therefore, less tuning is required for the new task and hence smaller distances.

Experimental results in this section show that the explicitly continual structure learning is important. With the proper structures learned, all the relevant parameters from previous tasks can be exploited. Additionally, since the way to leverage these parameters are learned through the structure optimization, much less tuning is required for better performance on new tasks, and forgetting can thus be overcomed.

[3]Similar trend of the distances between parameters across tasks was found for all layers. In general, the higher a layer is in the network the more the parameter moves for the baseline method, whereas for our learn-to-grow method the distances are typically very small.

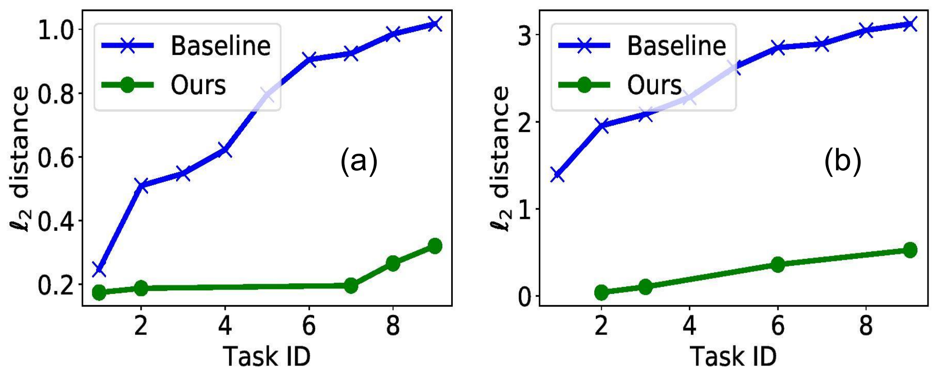


Figure 5.7: Distance between the tuned parameters at each task and the parameters of the very first task in VDD experiments. a) First layer parameter distance, and b) Last layer parameter distance. Baseline indicates the result from tuning all layers using SGD.

5.4.3 Comparisons with State-of-the-Art Methods

We compare our learn-to-grow method with other recent continual learning approaches – Lee et al. (73, DEN), Serrà et al. (125, HAT), Kirkpatrick et al. (62, EWC), Lee et al. (73, IMM), Rusu et al. (117, ProgressiveNet), Fernando et al. (23, PathNet), Nguyen et al. (98, VCL). We compare the performance of various methods in the permuted MNIST dataset with 10 different permutations. Since our model adds more parameters, for fair comparisons we also train other methods with comparable or more parameters. In particular, since our model tends to add new parameters at the first layer, for all methods we increase the number of neurons in the first hidden layer by ten times, so that theoretically they could learn exactly the same structure as our model. We also tried to compare with Shin et al. (127), however, we are unable to get reasonable performance, and hence the results are not included. The results are shown in Fig. 5.8 (a) and Table 5.2. It is clear that our method, either tuned with or without regularization, performs competitive or better than other methods on this task. This result suggests that although theoretically, structure can be learned along with parameter, in practice, the SGD-based optimization schema have a hard time achieving this. This in turn indicates the importance of explicitly taking continual structure learning into account when learning tasks continuously. Although both DEN and our method dynamically expand the network, our performance is much better, which is attributed to the ability of learning new structures for different tasks. Additionally, our model performs competitive as or better than the methods that completely avoids forgetting by fixing the learned weights, such as ProgressNet and PathNet, without enforcing such restrictions.

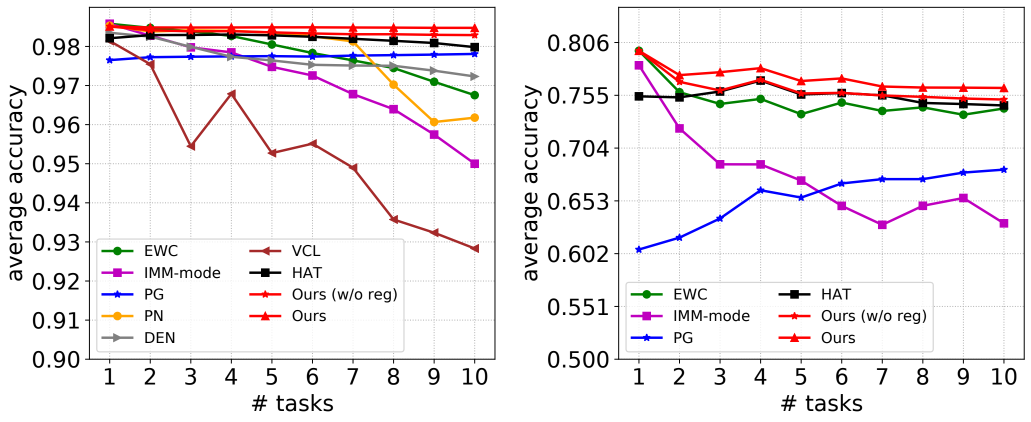


Figure 5.8: Performance comparisons in a) permuted MNIST and b) split CIFAR-100 dataset. Methods include Kirkpatrick et al. (62, EWC), Lee et al. (73, IMM), Fernando et al. (23, PathNet (PN)), Rusu et al. (117, Progressive Net (PG)), Serrà et al. (125, HAT), Lee et al. (73, DEN), Nguyen et al. (98, VCL), ours (w/o reg) denotes the case where finetuning for current tasks is done without using any regularization to prevent forgetting, and ous represents the case where the ℓ_2 regularization is used.

We further compare with other methods in the split CIFAR-100 dataset (86), where we randomly partition the classes of CIFAR-100 into 10 disjoint sets, and regard learning each of the 10-class classification as one task. Different from the permuted MNIST, the split CIFAR-100 presents a continual learning scenario where the input distribution is similar (i.e., natural images) whereas the output distribution is different (disjoint classes). We choose to use Alexnet (64) as the network structure, and all methods are constrained to have comparable model complexities. This network structure contains three convolution and max pooling layers and two fully connected layers before the last classification layer. Comparison results are shown in Fig 5.8 (b) and Table 5.3. Similar results as the MNIST experiment are obtained in this experiment. Interestingly, for all tasks, our method always seeks the structures that use new parameters for the last convolution layer and reuse the rest of the network parameters. It makes sense since the lower layer features are shared accounting for the similar input distribution, and the higher ones need to be specific for different tasks due to different output distribution. The fully connected layers are all selected to be “`reused`" instead of “`new`", and this may be because of the relatively large capacity that is sufficiently powerful to learn the subsequent tasks.

Table 5.2: Results of different continual learning approaches on 10 permutated MNIST datasets. The averaged accuracy after all 10 tasks are learned and total number of parameters are compared.

Method	Acc (%)	#params
SGD	72.51	3.35M
EWC	96.75	3.35M
IMM	95.00	3.35M
VCL	95.32	3.35M
HAT	97.98	3.46M
PN	96.18	3.96M
PG	97.81	3.05M
DEN	97.71	3.53M
Ours (fix)	98.46	2.56M
Ours (tune)	98.29	2.56M
Ours (tune+L2Reg)	98.48	2.56M

5.5 Related Work

Continual learning (140) remains a long standing problem, as models have the tendency to forget previously learned knowledge when trained on new information (140; 92). This is referred as the catastrophic forgetting problem in the literature. Early attempts to alleviate catastrophic forgetting include memory systems that store previous data and replay the stored old examples with the new data (114), and similar approaches are still used in the latest development (112; 78; 86). Shin et al. (127) proposes to learn a generative model to capture the data distribution of previous tasks, and both generated samples and real samples from the current task are used to train the new model so that the forgetting can be alleviated.

On the one hand, a typical class of methods for mitigating catastrophic forgetting relies on regularization which imposes constraints on the update of model parameters. Kirkpatrick et al. (62) proposes elastic weight consolidation (EWC) whose objective is to minimize the change of weights that are important to previous tasks through the use of a quadratic constraint. Zenke et al. (156) proposes to alleviate catastrophic forgetting by allowing individual synapse to estimate their importance for solving learned tasks, then penalizing the change on the important weights. Schwarz et al. (123) presents a method that divides the learning into two phases – progress and compress. During the progress phase, the model makes use of the previous model for learning the new task. In the compress

Table 5.3: Results of different continual learning approaches on split CIFAR100 dataset. The averaged accuracy after all 10 tasks are learned and total number of parameters are compared.

Method	Acc (%)	#params
SGD	21.02	6.55M
EWC	74.23	6.55M
IMM	63.13	6.55M
HAT	74.52	6.82M
PN	60.48	7.04M
PG	68.32	6.80M
Ours (fix)	75.31	6.86M
Ours (tune)	75.05	6.86M
Ours (tune+L2Reg)	76.21	6.86M

phase, the newly learned model is distilled into the old model using EWC to alleviate forgetting. Serrà et al. (125) proposes to use attention mechanism to preserve performance for previous tasks. Other methods could also completely avoid forgetting by preventing changes to previous task weights (see for example 117; 89; 23).

On the other hand, another class of methods for continual learning is allowing the model to expand. Dynamically expandable networks (72) select whether to expand or duplicate layers based on certain criteria for a new task. However, the model for the new task is forced to use the old structure from previous tasks. Similar strategies are adopted in progressive networks (117). Our proposed learn-to-grow framework is more flexible, thanks to the structure optimization via NAS. PathNet (23) selects paths between predefined modules, and tuning is allowed only when an unused module is selected. Our method does not have any restriction on tuning parameters from previous tasks.

Our method also relates to neural architecture search (132; 165; 5; 85), as we employ search methods to implement the structure optimization. In particular, DARTS (85) is used for efficiency, where a continuous relaxation for architecture search is proposed.

5.6 Additional Results

5.6.1 Additional Experimental Details for permuted MNIST

For all MNIST experiment, we use fully connected layer with three hidden layer, each with 300 hidden units, and one shared output layer for our method. For all other methods except pathnet and progressive net we used 3000 units in the first layer and 300 for the rest. For pathnet, each module in the first layer has 300 units, and the result layers has 30 units. We use 16 modules per layer, and 5 layers for pathnet, and restrict each mutation to only use 3 modules besides the output layer. For progressive net, the first layer has 300 units for each task, and the rest layers each has 30 units. Therefore, all competitive methods are having more or the same number of parameters as our methods.

For variational continual learning (VCL 98), we used the official implementation [4]. For fair comparison with other methods, we set the coreset size to zero for VCL.

For (127) we used implementation [5]. We tried various hyper-parameter settings, however, we are unable to get reasonable results on permutated MNIST. Performance was reasonable when the number of tasks is within five (average performance at around 96%). When number of tasks go beyond five, performance drops on previous tasks is quite significant. Reaching 60%

For DEN we use the official implementation [6], and we used Serrà et al. (125) implementation of HAT, EWC, and IMM from [7]. We used our own implemention for for Progressive Network and PathNet. All methods are trained using the same permutations and same subset of training data.

5.6.2 Additional Experimental Details for Split CIFAR-100

For all CIFAR-100 experiment, we use an Alexnet like structure. It contains three convolution and max pooling layers followed by two fully connected layers. The convolution layers are of size (4,4), (3,3) and (2,2) with 64, 128 and 256 filters, respectively. All convolution layers are followed by max pooling layer of size (2,2) and rectified linear activations. The two fully connected layers each have 2048 hidden units.

[4]https://github.com/nvcuong/variational-continual-learning
[5]https://github.com/kuc2477/pytorch-deep-generative-replay
[6]https://github.com/jaehong-yoon93/DEN
[7]https://github.com/joansj/hat

5.6.3 Additional Experiments on Visual Decathlon Dataset

Table 5.4: Comparison of (top-1) validation classification accuracy (%) and total model size (in Million) on Visual Domain Decathlon dataset with parameter loss factor β of 0.01, 0.1, 1.0.

		ImNet	C100	SVHN	UCF	OGlt	GTSR	DPed	Flwr	Airc.	DTD	Tot.
$\beta = 0.01$	acc	69.84	78.50	95.33	72.50	86.41	99.97	99.76	66.01	51.37	50.05	76.97
	#params	6.07	0.15	2.74	2.28	6.17	3.59	1.02	0.19	4.15	0.13	26.49
$\beta = 0.1$	acc	69.84	79.59	95.28	72.03	86.60	99.72	99.52	71.27	53.01	49.89	77.68
	#params	6.07	0.34	1.19	1.32	3.19	0.02	0.27	0.16	1.86	0.04	14.46
$\beta = 1.0$	acc	69.84	78.00	93.40	63.83	84.30	99.78	99.01	65.77	39.27	48.77	74.20
	# params	6.07	0.04	0.03	0.12	0.66	0.02	0.01	0.02	0.35	0.02	7.34

In the multi-task continual learning experiments, the 10 tasks was trained in a random sequence except the first task was fixed to be ImageNet. This is just for fair comparison with other works such as Rebuffi et al. (110) and Mallya and Lazebnik (89), they are all using a light weight module to adapt ImageNet pretrained model to other of the 9 tasks. In real case, the tasks can come in any order, thus our framework would be much more flexible. As the tasks are trained in sequence, a super model is maintained that all the newly created weights and task-specific layers are stored. In this ResNet-26 model, all the Batch Normalization (BN) layers are treated as task-specific, which means each task has its own sets of BNs. Here, we fixed the weight during retraining when "reuse" is selected in the search phase. This means that the results of previous tasks would not be affected, i.e. no forgetting. We leave the evaluation of forgetting in the context of VDD dataset as future work.

In Table 1, we compare the results using our approach with other baselines. "Individual" means that each task is trained individually and weights are initialized randomly. "Classifier" means that only the last layer classifier could be tuned while the former 25 layers are transfer from ImageNet pretrained model and kept fixed during training. In this case, each task only adds a task-specific classifier and BNs, thus the overall model size is small. "Adapter" add a 1x1 conv layer besides each 3x3 conv layer, and the outputs will be added before proceed to the next layer. Due to the lightweight 1x1 conv layer, each task will add approximately 1/9 of the whole model size. As shown in table 1, the results achieved by our framework is

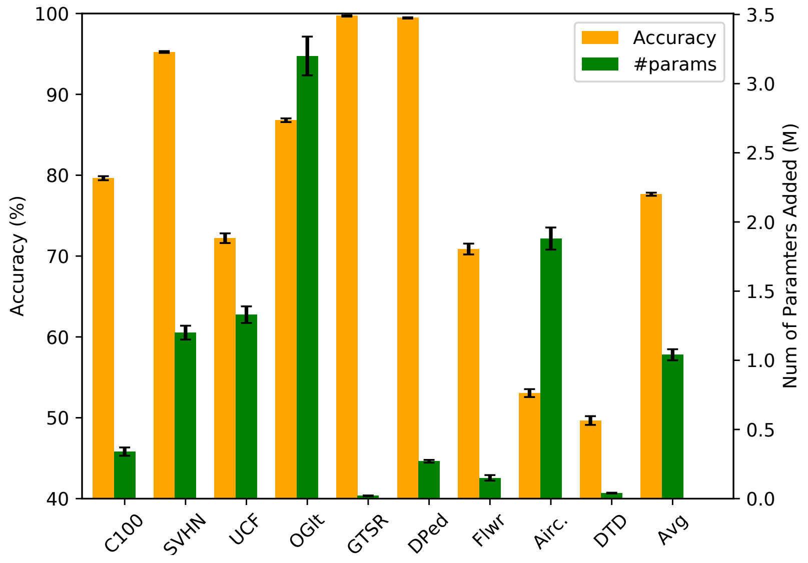


Figure 5.9: Statistics for performance and number of added parameters for each task of VDD dataset with 4 random task ordering. The first task is kept with ImageNet due to its large size and long training time. We observed that both accuracy and parameter growth are robust to different task ordering.

better than other baselines and the total model size is similar to "Adapter" case. We can see that our approach gives best results in five out of nine tasks. Especially in task with small data size, e.g. VGG-Flowers and Aircraft, our method outperforms other baselines by a large margin.

Due to each choice has different parameter cost, we add a parameter loss function to L_{val} to penalize the choices that cost additional parameters. And the value of the loss function is proportional to the product of the additional parameter size and its corresponding weight value α_c^l. In table 2, we test it with three different scaling factor β of the parameter loss. We found that the scaling factor β can control the additional parameter size for each task. And we find that $\beta = 0.1$ gives the best average accuracy and can control the total model size approximate 2.3× compared with the original model size.

5.7 Summary

We present a simple yet effective learn-to-grow framework for overcoming catastrophic forgetting in continual learning with DNNs, which explicitly takes into account continual structure optimization via differentiable neural architecture search. We introduce three intuitive options for each layer in a give model, that is to “`reuse`", “`adapt`" or “`new`" it in learning a new task. In experiments, we observed that the explicit learning of model structures leads to sensible structures for new tasks in continual learning with DNNs. And, catastrophic forgetting can be either completely avoided if no fine-tuning is taken for the “`reuse`" layers, or significantly alleviated even with fine-tuning. The proposed method is thoroughly tested in a series of datasets including the permuted MNIST dataset, the visual decathlon dataset and the split CIFAR-100 dataset. It obtains highly comparable or better performance in comparison with state-of-the-art methods.

CHAPTER

6 CONCLUSION AND DISCUSSION

In this thesis, a list of novel models and frameworks have been proposed and extensively evaluated. This research is intrigue by the need of more powerful and universal representation learning systems that have big impact on nowadays AI research and industrial applications.

In Chapter 2, we introduced a novel family of grammar-guided network architectures which construct compositional grammatical architectures for deep learning in an effective way. The AOG comprises a phrase structure grammar and a dependency grammar. An AOGNet consists of a number of stages each of which comprises a number of AOG building blocks. Our AOG building block harnesses the best of grammar models and DNNs for deep learning. AOGNet obtains state-of-the-art performance on multiple computer vision benchmarks and shows better interpretability and potential adversarial robustness.

We hope this work encourages further exploration in learning grammar-guided network generators. The AOG can be easily extended to adopt k-branch splitting rules with $k > 2$. Other types of edges can also be easily introduced in the AOG such as dense lateral connections and top-down connections. Node operations can also be extended to exploit grammar-guided transformation. And, better parameter initialization methods need to be studied for the AOG structure.

In Chapter 3, we presents a lightweight integration of feature normalization and attention. It connects the affine transformation in feature normalization schema with the channel-wise attention based recalibration of feature responses. It presents Mixture Normalization (MN) as a simple and unified alternative for feature normalization and attention. It also shows a lightweight deployment of the proposed MN in the Bottleneck operation. We validated it by testing MN using Batch Normalization (BN) as the backbone in ImageNet and MS-COCO. It obtains better performance than state-of-the-art variants of BN, Group Normalization (GN) and Switchable Normalization (SN). It also shows interpretable visualization justifying the effectiveness of MN. The proposed method is complementary to existing variants of BN and applicable to extending them to corresponding Mixture versions.

In Chapter 4, we present a simple yet effective learn-to-grow framework for overcoming catastrophic forgetting in continual learning with DNNs, which explicitly takes into account continual structure optimization via differentiable neural architecture search. We introduce three intuitive options for each layer in a give model, that is to “`reuse`", “`adapt`" or “`new`" it in learning a new task. In experiments, we observed that the explicit learning of model structures leads to sensible structures for new tasks in continual learning with DNNs. And, catastrophic forgetting can be either completely avoided if no fine-tuning is taken for the “`reuse`" layers, or significantly alleviated even with fine-tuning. The proposed method is thoroughly tested in a series of datasets including the permuted MNIST dataset, the visual decathlon dataset and the split CIFAR-100 dataset. It obtains highly comparable or better performance in comparison with state-of-the-art methods.

We hope this work encourages further exploration on resolving the "catastrophic forgetting" issue of deep neural networks. One future direction what might be worth trying is that we can combining the Learn-to-Grow framework with grammar architecture like AOGNets. The reason why the grammar architecture has advantage on continual learning is that it has the explicit multi-branch topology with more flexible feature reuse or adaptations.

REFERENCES

[1] Arora, S., Bhaskara, A., Ge, R., and Ma, T. (2014). Provable bounds for learning some deep representations. In *Proceedings of the 31th International Conference on Machine Learning, ICML*, pages 584–592.

[2] Athalye, A. and Sutskever, I. (2017). Synthesizing robust adversarial examples. *CoRR*, abs/1707.07397.

[3] Ba, L. J., Kiros, R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450.

[4] Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

[5] Baker, B., Gupta, O., Raskar, R., and Naik, N. (2017). Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*.

[6] Bau, D., Zhou, B., Khosla, A., Oliva, A., and Torralba, A. (2017). Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[7] Brock, A., Donahue, J., and Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

[8] Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2017). Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*.

[9] Cai, Z. and Vasconcelos, N. (2018). Cascade R-CNN: delving into high quality object detection. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6154–6162.

[10] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference, BMVC*.

[11] Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C. C., and Lin, D. (2019a). MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*.

[12] Chen, T., Chen, R., Nie, L., Luo, X., Liu, X., and Lin, L. (2019b). Neural task planning with AND-OR graph representations. *IEEE Trans. Multimedia*, 21(4):1022–1034.

[13] Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., and Feng, J. (2017). Dual path networks. *arXiv preprint arXiv:1707.01629*.

[DARPA] DARPA. Explainable artificial intelligence (xai) program.

[15] de Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., and Courville, A. C. (2017). Modulating early visual processing by language. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6597–6607.

[16] Deecke, L., Murray, I., and Bilen, H. (2019). Mode normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[17] Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., and Courville, A. C. (2016). Adversarially learned inference. *CoRR*, abs/1606.00704.

[18] Elsken, T., Metzen, J. H., and Hutter, F. (2018). Neural architecture search: A survey. *CoRR*, abs/1808.05377.

[19] Everingham, M., Eslami, S. M., Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vision (IJCV)*, 111(1):98–136.

[20] Farquhar, S. and Gal, Y. (2018). Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*.

[21] Felzenszwalb, P. F. (2011). Object detection grammars. In *IEEE International Conference on Computer Vision Workshops, ICCV*, page 691.

[22] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell. (PAMI)*, 32(9):1627–1645.

[23] Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

[24] Fu, K. S. and Albus, J. E., editors (1977). *Syntactic pattern recognition : applications*. Communication and cybernetics. Springer-Verlag, Berlin, New York.

[25] Geman, S., Potter, D., and Chi, Z. Y. (2002). Composition systems. *Quarterly of Applied Mathematics*, 60(4):707–736.

[26] George, D., Lehrach, W., Kansky, K., Lázaro-Gredilla, M., Laan, C., Marthi, B., Lou, X., Meng, Z., Liu, Y., Wang, H., Lavin, A., and Phoenix, D. S. (2017). A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*.

[27] Ghiasi, G., Lin, T.-Y., and Le, Q. V. (2018). Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10727–10737.

[28] Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., and Keutzer, K. (2018). Squeezenext: Hardware-aware neural network design. *arXiv preprint arXiv:1803.10615*.

[29] Girshick, R. (2015). Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

[30] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[31] Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *ICLR*.

[Grenander and Miller] Grenander, U. and Miller, M. *Pattern Theory: From Representation to Inference*. Oxford University Press.

[33] Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. (2019). Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*.

[34] Han, D., Kim, J., and Kim, J. (2017). Deep pyramidal residual networks. *IEEE CVPR*.

[35] Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

[36] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017a). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

[37] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017b). Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2980–2988.

[38] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034.

[39] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916.

[40] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[41] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[42] He, K., Zhang, X., Ren, S., and Sun, J. (2016c). Identity mappings in deep residual networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 630–645.

[43] He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. (2018). Bag of tricks for image classification with convolutional neural networks. *CoRR*, abs/1812.01187.

[44] Hinton, G. (August 17, 2017). What is wrong with convolutional neural nets? *the 2017 - 2018 Machine Learning Advances and Applications Seminar presented by the Vector Institute at U of Toronto, https://www.youtube.com/watch?v=Mqt8fs6ZbHk.*

[45] Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. (2019). Searching for mobilenetv3. *CoRR*, abs/1905.02244.

[46] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[47] Hu, J., Shen, L., and Sun, G. (2017). Squeeze-and-excitation networks. *CoRR*, abs/1709.01507.

[48] Huang, G., Liu, S., van der Maaten, L., and Weinberger, K. Q. (2017a). Condensenet: An efficient densenet using learned group convolutions. *group*, 3(12):11.

[49] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017b). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[50] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 646–661.

[51] Huang, L., Liu, X., Lang, B., Yu, A. W., Wang, Y., and Li, B. (2018a). Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3271–3278.

[52] Huang, L., Yang, D., Lang, B., and Deng, J. (2018b). Decorrelated batch normalization. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 791–800.

[53] Huang, Z., Wang, X., Huang, L., Huang, C., Wei, Y., and Liu, W. (2018c). Ccnet: Criss-cross attention for semantic segmentation. *CoRR*, abs/1811.11721.

[54] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154.

[55] Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243.

[56] Ioffe, S. (2017). Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1945–1953.

[57] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Blei, D. and Bach, F., editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456. JMLR Workshop and Conference Proceedings.

[58] Jia, S., Chen, D., and Chen, H. (2019). Instance-level meta normalization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4865–4873.

[59] Kalayeh, M. M. and Shah, M. (2019). Training faster by separating modes of variation in batch-normalized models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

[60] Karras, T., Laine, S., and Aila, T. (2018). A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*.

[61] Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. (2019). A survey of the recent architectures of deep convolutional neural networks. *arXiv preprint arXiv:1901.06032*.

[62] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835.

[63] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

[64] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pages 1106–1114.

[65] Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. (2016). Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*.

[66] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015a). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

[67] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015b). Human-level concept learning through probabilistic program induction. *Science*.

[68] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2016). Building machines that learn and think like people. *CoRR*, abs/1604.00289.

[69] Larsson, G., Maire, M., and Shakhnarovich, G. (2016). Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648.

[70] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

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

[72] Lee, J., Yun, J., Hwang, S., and Yang, E. (2017a). Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

[73] Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. (2017b). Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pages 4652–4662.

[74] Li, X., Song, X., and Wu, T. (2019a). Aognets: Compositional grammatical architectures for deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 6220–6230.

[75] Li, X., Song, X., and Wu, T. (2019b). Aognets: Compositional grammatical architectures for deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6220–6230.

[76] Li, X., Sun, W., and Wu, T. (2019c). Attentive normalization. *arXiv preprint arXiv:1908.01259*.

[77] Li, X., Zhou, Y., Wu, T., Socher, R., and Xiong, C. (2019d). Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. *arXiv preprint arXiv:1904.00310*.

[78] Li, Y., Li, Z., Ding, L., Yang, P., Hu, Y., Chen, W., and Gao, X. (2018). Supportnet: solving catastrophic forgetting in class incremental learning with support data. *arXiv preprint arXiv:1806.02942*.

[79] Liang, X., Lin, L., and Cao, L. (2015). Learning latent spatio-temporal compositional model for human action recognition. *CoRR*, abs/1502.00258.

[80] Lin, L., Wang, X., Yang, W., and Lai, J. (2015). Discriminatively trained and-or graph models for object shape detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(5):959–972.

[81] Lin, L., Wu, T., Porway, J., and Xu, Z. (2009). A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition*, 42(7):1297–1307.

[82] Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.

[83] Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J. (2017). Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 936–944.

[84] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

[85] Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

[86] Lopez-Paz, D. et al. (2017). Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476.

[87] Loshchilov, I. and Hutter, F. (2016). SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983.

[88] Luo, P., Ren, J., and Peng, Z. (2018). Differentiable learning-to-normalize via switchable normalization. *CoRR*, abs/1806.10779.

[89] Mallya, A. and Lazebnik, S. (2018). Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *arXiv preprint arXiv:1801.06519*.

[90] Mancini, M., Ricci, E., Caputo, B., and Bulò, S. R. (2018). Adding new tasks to a single network with weight trasformations using binary masks. *arXiv preprint arXiv:1805.11119*.

[91] Massa, F. and Girshick, R. (2018). maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`. Accessed: [Insert date here].

[92] McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

[93] Misra, D. (2019). Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*.

[94] Miyato, T. and Koyama, M. (2018). cgans with projection discriminator. *arXiv preprint arXiv:1802.05637*.

[Mumford] Mumford, D. Grammar isn't merely part of language. `http://www.dam.brown.edu/people/mumford/blog/2016/grammar.html`.

[Mumford and Desolneux] Mumford, D. and Desolneux, A. *Pattern Theory, the Stochastic Analysis of Real World Signals.* AKPeters/CRC Press.

[97] Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. *CoRR*, abs/1603.06937.

[98] Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2018). Variational continual learning. In *International Conference on Learning Representations*.

[99] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

[100] Park, J., Woo, S., Lee, J.-Y., and Kweon, I. S. (2018). Bam: bottleneck attention module. *arXiv preprint arXiv:1807.06514*.

[101] Park, T., Liu, M., Wang, T., and Zhu, J. (2019). Semantic image synthesis with spatially-adaptive normalization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2337–2346.

[102] Peng, C., Xiao, T., Li, Z., Jiang, Y., Zhang, X., Jia, K., Yu, G., and Sun, J. (2018). Megdet: A large mini-batch object detector. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6181–6189.

[103] Perez, E., de Vries, H., Strub, F., Dumoulin, V., and Courville, A. C. (2017). Learning visual reasoning without strong priors. *CoRR*, abs/1707.03017.

[104] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4092–4101.

[105] Pleiss, G., Chen, D., Huang, G., Li, T., van der Maaten, L., and Weinberger, K. Q. (2017). Memory-efficient implementation of densenets. *CoRR*, abs/1707.06990.

[106] Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285.

[107] Rauber, J., Brendel, W., and Bethge, M. (2017). Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*.

[108] Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2018). Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

[109] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q., and Kurakin, A. (2017). Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*.

[110] Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017a). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516.

[111] Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127.

[112] Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017b). icarl: Incremental classifier and representation learning. In *Proc. CVPR*.

[113] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*.

[114] Robins, A. (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

[115] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[116] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision (IJCV)*, 115(3):211–252.

[117] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[118] Sabour, S., Frosst, N., and E Hinton, G. (2017). Dynamic Routing Between Capsules. *ArXiv e-prints*.

[119] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, page 901.

[120] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

[121] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 2488–2498.

[122] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer.

[123] Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*.

[124] Sener, O. and Savarese, S. (2018). Active learning for convolutional neural networks: A core-set approach.

[125] Serrà, J., Surís, D., Miron, M., and Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423*.

[126] Shao, W., Meng, T., Li, J., Zhang, R., Li, Y., Wang, X., and Luo, P. (2019). Ssn: Learning sparse switchable normalization via sparsestmax. *CoRR*, abs/1903.03793.

[127] Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999.

[128] Song, X., Wu, T., Jia, Y., and Zhu, S. (2013). Discriminatively trained and-or tree models for object detection. In *Proceedings of 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3278–3285.

[129] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[130] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015a). Highway networks. *CoRR*, abs/1505.00387.

[131] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015b). Highway networks. *CoRR*, abs/1505.00387.

[132] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

[133] Summers, P. D. (1977). A methodology for LISP program construction from examples. *J. ACM*, 24(1):161–175.

[134] Sun, W. and Wu, T. (2019). Image synthesis from reconfigurable layout and style. In *International Conference on Computer Vision, ICCV*.

[135] Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261.

[136] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015a). Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9.

[137] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015b). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

[138] Tang, W., Yu, P., and Wu, Y. (2018). Deeply learned compositional models for human pose estimation. In *ECCV (3)*, volume 11207 of *Lecture Notes in Computer Science*, pages 197–214. Springer.

[139] Tang, W., Yu, P., Zhou, J., and Wu, Y. (2017). Towards a unified compositional model for visual pattern modeling. In *ICCV*, pages 2803–2812. IEEE Computer Society.

[140] Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. In *The biology and technology of intelligent autonomous agents*, pages 165–196. Springer.

[141] Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. (2016). Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022.

[142] Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., and Tang, X. (2017). Residual attention network for image classification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6450–6458.

[143] Wang, J., Wei, Z., Zhang, T., and Zeng, W. (2016). Deeply-fused nets. *CoRR*, abs/1605.07716.

[144] Wang, X., Girshick, R. B., Gupta, A., and He, K. (2018). Non-local neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7794–7803.

[145] Wistuba, M., Rawat, A., and Pedapati, T. (2019). A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*.

[146] Woo, S., Park, J., Lee, J., and Kweon, I. S. (2018a). CBAM: convolutional block attention module. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, pages 3–19.

[147] Woo, S., Park, J., Lee, J.-Y., and So Kweon, I. (2018b). Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19.

[148] Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742.

[149] Wu, T., Lu, Y., and Zhu, S. (2016). Online object tracking, learning and parsing with and-or graphs. *IEEE Trans. Pattern Anal. Mach. Intell. (PAMI)*, DOI: 10.1109/TPAMI.2016.2644963.

[150] Wu, Y. and He, K. (2018). Group normalization. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, pages 3–19.

[151] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017a). Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500.

[152] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017b). Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500.

[153] Yann LeCun, C. C. (1998). The mnist database of handwritten digits.

[154] Yu, F., Wang, D., and Darrell, T. (2018). Deep layer aggregation. In *CVPR*.

[155] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*.

[156] Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. *arXiv preprint arXiv:1703.04200*.

[157] Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

[158] Zhang, H., Dauphin, Y. N., and Ma, T. (2019). Fixup initialization: Residual learning without normalization. *CoRR*, abs/1901.09321.

[159] Zhang, T., Qi, G., Xiao, B., and Wang, J. (2017a). Interleaved group convolutions. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 4383–4392.

[160] Zhang, X., Li, Z., Loy, C. C., and Lin, D. (2016). Polynet: A pursuit of structural diversity in very deep networks. *CoRR*, abs/1611.05725.

[161] Zhang, X., Zhou, X., Lin, M., and Sun, J. (2017b). Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*.

[162] Zhu, L., Chen, Y., Lu, Y., Lin, C., and Yuille, A. L. (2008). Max margin AND/OR graph learning for parsing the human body. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

[163] Zhu, S. C. and Mumford, D. (2006). A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2(4):259–362.

[164] Zoph, B. and Le, Q. V. (2016a). Neural architecture search with reinforcement learning. In *Proceedings of International Conference on Learning Representations*.

[165] Zoph, B. and Le, Q. V. (2016b). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

[166] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2017). Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6).