EFFICIENT ARCHITECTURE AND DATA MANIPULATION FOR DEEP LEARNING SYSTEMS

By

Yu Zheng

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Electrical Engineering—Doctor of Philosophy

2024

ABSTRACT

The significant progress of deep learning models in recent years can be attributed primarily to the growth of the model scale and the volume of data on which it was trained. Although scaling up the model with sufficient training data typically provides enhanced performance, the amount of memory and GPU hours used for training provide great challenges for deep learning infrastructures. Another challenge for training a good deep learning model is the quantity of the data it was trained on. To achieve state-of-the-art performance, it has become a standard way to train or fine-tune deep neural networks on a dataset augmented with well-designed augmentation transformations. This introduces difficulties in efficiently identifying the best data augmentation strategies for training. Furthermore, there has been a noticeable increase in the dataset size across many learning tasks, making it the third challenge of modern deep learning systems. The dataset size becomes large and large over the years, posing great burdens on storage and training cost. Moreover, it can be prohibitive to perform hyperparameter optimization and neural architecture search on networks trained on such massive datasets.

In this dissertation, we address the first challenges from a model-centric perspective. We propose MSUNet, which is designed with key techniques: 1) ternary conv layers, 2) sparse conv layers, 3) self-supervised consistency regularizer along with mixed precision training. These techniques allow faster training and inference of deep learning models without sacrificing significant accuracy loss. We then look at deep learning systems from a data-centric perspective. To deal with the second challenge, we propose Deep AutoAugment (DeepAA), a multi-layer data augmentation search method which aims to remove the need of crafting augmentation strategies manually. DeepAA fully automates the data augmentation process by searching for a deep data augmentation policy on an expanded set of transformations. We formulate the search of data augmentation policy as a regularized gradient matching problem by maximizing the cosine similarity of the gradients between augmented data and original data with regularization. To avoid exponential growth of dimensionality of the search space when more augmentation layers are used, we incrementally stack augmentation layers based on the data distribution transformed by all the previous augmentation

layers. DeepAA achieves the best performance compared to existing automatic augmentation search methods evaluated on various models and datasets. To tackle the third challenge, we proposed a dataset condensation method by distilling the information from a large dataset to a small condensed dataset. The data condensation is realized by matching the training trajectories of the original dataset with that of the condensed dataset. Experiments show that our proposed method outperforms the baseline methods.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest appreciation to my academic advisor, Dr. Mi Zhang. He has been supporting me with profound expertise and insightful mentorship throughout my academic journey. His mentorship has not only led to the success of our research project, but has also inspired me to strive for excellence. I would also like to thank Dr. Yiming Deng, Dr. Guan-Hua Tu and Dr. Zhichao Cao who serve as my committee members and provide valuable guidance for my research. I would like to express my profound gratitude to Dr. Tongtong Li for her support on my academic career.

I am also deeply grateful for the opportunities to work alongside my fellows in the lab, including Shen Yan, Biyi Fang, Xiao Zeng, Dong Chen, Jiajia Li, Wei Ao, Zhe Wang, Yuan Liang, Zixiao Yu and Kaixiang Lin. Their encouragement and friendship are a great support during my life at Michigan State University. I am also grateful to all the staff and faculty in the ECE department.

My internships provided me with great learning and networking opportunities. I want to express my sincere appreciation to Dr. Zhi Zhang and Dr. Yi Zhu for hosting me at Amazon Web Services. I am grateful for the valuable experiences and collaboration opportunities they provided.

Last but certainly not least, I must acknowledge the unwavering love and support from my parents. Their constant belief in my abilities and their emotional support have been the cornerstone of my academic and personal achievements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, I first introduce the motivation of this thesis and the challenges for efficient deep learning system. Then I introduce the specific research objectives of this thesis and provide a summary of the key contributions.

1.1 Motivation and Challenges

The remarkable performance of deep learning models has revolutionized a wide spectrum of areas including computer vision, speech recognition, and natural language processing.

The significant progress of deep learning models in recent years can be attributed primarily to the growth of the model scale and the volume of data on which it was trained. Although scaling up the model with sufficient training data typically provides enhanced performance, the amount of memory and GPU hours used for training provide great challenges for deep learning infrastructures. Even for inference purpose, deploying large models on devices with limited resources remains challenging due to their high memory and computational requirements at inference time.

Another challenge for training a good deep learning model is the quantity of the data it was trained on. To achieve state-of-the-art performance, it has become a standard way to train or fine-tune deep neural networks on a dataset augmented with well-designed augmentation transformations. This introduces difficulties in efficiently identifying the best data augmentation strategies for training.

Furthermore, there has been a noticeable increase in the dataset size across many learning tasks, making it the third challenge of modern deep learning systems. The dataset size grows too large over the years posing great burdens on storage and training cost. Moreover, it can be prohibitive to perform hyperparameter optimization and neural architecture search on networks trained on such massive datasets.

1.2 Summary of Research Contributions

This thesis studies efficient architecture and data manipulation techniques for deep learning systems. The principal contributions of this thesis are outlined as follows:

1. A ternary quantization method, which outperforms the binary quantization (https://github.com/AIoT-MLSys-Lab/MSUNet).

2. Self-supervised consistency regularizer, which helps improving the sparsity of pruning and ternary quantization without significant scarifying the test performance (https://github.com/AIoT-MLSys-Lab/MSUNet).

3. An efficient multi-layer data augmentation search method that finds state-of-the-art deep data augmentation policy without using any hand-picked default transformations (https://github.com/AIoT-MLSys-Lab/DeepAA).

4. Importance aware weight factorization, that ranks the weight importance based on its contribution to the layer output.

5. Importance aware trajectory matching method for dataset condensation that builds upon the importance aware weight factorization.

1.3 Thesis Organization

The remainder of this dissertation is organized as follows:

Chapter 2: MSUNet

In this chapter, I propose MSUNet, which is designed with three key techniques: 1) ternary convolution layers, 2) sparse squeeze-excitation layers, and 3) self-supervised consistency regularizer along with mixed precision training. Our framework shows a great improvement in parameter size and computational cost measure by #Parameters and #Flops over the baseline model. Lastly, we show that ternary quantization outperforms binary quantization in all aspects, including accuracy, parameter size, and computation cost.

Chapter 3: Deep AutoAugment

In this chapter, I present Deep AutoAugment (DeepAA), a multi-layer data augmentation search method that finds deep data augmentation policy without using any hand-picked default transformations. We formulate data augmentation search as a regularized gradient matching

problem, which maximizes the gradient similarity between augmented data and original data along the direction with low variance. Our experimental results show that DeepAA achieves strong performance without using default augmentations, indicating that regularized gradient matching is an effective searching method for data augmentation policies.

Chapter 4: Dataset Condensation via Importance Aware Trajectory Matching

Neural networks are typically over parameterized with a lot of redundancy, so the weights are not born equally important. Based on this insight, in this chapter we first proposed an importance aware weight factorization, which factorize the weight matrix based on its influence on the layer output. Building on that we proposed an importance aware trajectory matching algorithm which only match the most important weights early in the training trajectory and gradually adding more weight components until reaching the full weights later in training trajectory. Our methods removes the interfere of the redundancy of the weights when matching the training trajectory. The results show improvements over previous work on CIFAR and Tiny ImageNet datasets.

Chapter 5: Conclusion

This chapter concludes the whole thesis.

CHAPTER 2

MSUNET

2.1 Introduction

The breakthrough of deep learning in recent years can be partially attributed to the growth of model scale along with higher memory and computational requirement. These models have shown remarkable performance in a wide range of applications, such as image classification, object detection, image segmentation and natural language processing. However, deploying these models on devices with limited resources remains challenging due to their high memory and computational requirements.

There has been a significant progress in developing neural network architectures and model compression strategies that enhance the memory and computational efficiency in deep neural networks. Efficient convolutional neural network designs such as SqueezeNet [2] and MobileNet [3] use 1x1 convolutions and depthwise separable convolutions, respectively. The technique of Block-term Tensor Decomposition [4] has been adopted in [5] to compact transformer-based language models. Model compression methods that are agnostic to architecture, including knowledge distillation [6], network pruning [7], and trained quantization [7], are employed to enhance neural network performance or reduce model dimensions. Recently some research topics have also been directed towards automatically generating efficient models [8, 9, 10] and creating specialized hardware to run these compressed models [11, 12].

Network quantization [13] is also a major technique to address the memory and computational challenges for deep neural networks. The key idea of quantization is to convert the network weights or activations from higher precision floating point numbers to lower precision floating point numbers or even lower bit integer numbers. This brings significant reduction of memory overhead and computational cost. While quantization brings light-weight models with enhanced efficiency, it introduced noise to the network, resulting in worse performance than the original one. The challenge is to design a better quantization technique that brings less quantization noise to the model.

In light of the techniques for model compression and quantization, in this chapter, we propose MSUNet, which is designed with key techniques: 1) ternary convolution layers, 2) sparse squeeze-excitation layers, 3) self-supervised consistency regularizer along with mixed precision training.

2.2 Related Work

Efficient CNN Architectures: ResNet proposed in [14] is the pioneering work on efficient convolutional neural network architectures. Compared with prior work such as Alexnet [15] and VGG [16], ResNet achieves better performance with far less memory and FLOPs for processing each image. The key to the reduced parameter size with enhanced performance is to use parameter free global average pooling layers and a skip connections that enable deeper network to be trained. Based on the ResNet architecture, may variants have since been proposed such as Wide-ResNet [17] and ResNeXt [18]. MobileNets [19, 20, 21] transform $k \times k$ convolutions into separate depthwise and pointwise convolutions. ShuffleNets [22, 23] imporves efficiency by incorporating group convolutions and channel shuffling into pointwise convolution. MixConv, introduced in [24], integrates multiple kernel sizes within a single convolutional layer. The work of [25] employs the butterfly transform as a surrogate for pointwise convolution. EfficientNet [26, 27] innovates with a compound scaling technique for balanced scaling across depth, width, and resolution. AdderNet [28] optimizes computational efficiency by substituting multiplications with additions. GhostNet [29] uses inexpensive linear operations to produce additional feature maps. Introduced in [30], a structural revision of the inverted residual block was shown to be effective in reducing the information loss and gradient confusion.

Neural Network Quantization: Depending on whether the quantization is applied during training or at inference time, network quantization can be divided into two categories: 1) quantization aware training (QAT) and 2) post training quantization (PTQ). Half precision floating point numbers has become a standard technique for network training and inference which is supported by NVidia GPUs and major deep learning frameworks [31]. It was shown in [32] that the mixed precision training can achieves lossless performance with speed up of training and inference time. The key technique of mixed precision training is to store weights, activations, and gradients in FP16

while keeping an FP32 copy of weights for gradient accumulation. It was reported in [33, 34, 35] that we can reduce the parameter to 8-bit integers after training without significant drop of inference accuracy. Jacob [36] proposed an 8-bit quantization technique for both training and inference, which shows an accuracy loss of 1.5% on ResNet-50. Xilinx [37] proposed an 8-bit lossless quantization that does not require retraining.

2.3 MSUNet Design

In this section, we present the detailed design of MSUNet. The key components of MSUNet includes 1) ternary convolutional layers, 2) sparse squeeze-excitation layers and 3) self-supervised consistency regularizer.

2.3.1 Ternary Convolutional Layers

For our proposed ternary convolutional layers, the weights are quantized into three level $\{-1, 0, 1\}$ according to a predefined threshold α. For a convolution layer with weight W and a predefined threshold α, we modify the weight as

$$\tilde{W} = \mathsf{Ternary}(W, \alpha) \,, \tag{2.1}$$

where the ternary operator $\mathsf{Ternary}(\cdot, \alpha)$ is a point-wise operation defined as:

$$\mathsf{Ternary}(w, \alpha) = \begin{cases} 1, & \text{if } w > \alpha \\ 0, & \text{if } -\alpha \le w \le \alpha \\ -1, & \text{if } w < -\alpha \end{cases} \tag{2.2}$$

However the quantization leads to a mismatch between the scale of the quantized weight $\tilde{W}$ and the original weight W, which needs to be mitigated. Suppose that the weight W is initialized using normal distribution with standard deviation σ according to [38]. The ternalized weight $\tilde{W}$ has the standard deviation $\tilde{\sigma} = \sqrt{n/N}$ where N and n are the number of all the parameters and the number of *non-zero* parameters of $\tilde{W}$, respectively. Empirically, we found that the mismatch between σ and $\tilde{\sigma}$ will slow down the convergence. Thus we modify the forward pass with a compensation factor $\sigma/\tilde{\sigma}$ for the scale mismatch:

$$\tilde{Y} = \sigma \sqrt{\frac{N}{n}} \tilde{W} X \tag{2.3}$$

Similar to the approach of mixed precision training [32], we keep a copy of the original weights (with FP32 precision) during training, and allow the gradient to be accumulated to the original weights. Different from the normal gradient calculation, we use the ternary weight $\tilde{W}$ during forward and backward, that is the gradient is calculated based on $\tilde{W}$ while accumulated on W instead. The code that reflects the ternary operation is shown in the code snippet below.

```
class ForwardSign(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        global alpha
        x_ternary = (x - x.mean())/x.std()
        ones = (x_ternary > alpha).type(torch.cuda.FloatTensor)
        neg_ones = -1 * (x_ternary < -alpha).type(torch.cuda.FloatTensor)
        x_ternary = ones + neg_ones
        multiplier = math.sqrt(2. / (x.shape[1] * x.shape[2] * x.shape[3]) *
    x_ternary.numel() / x_ternary.nonzero().size(0) )
        if args.amp:
            return (x_ternary.type(torch.cuda.HalfTensor), torch.tensor(
    multiplier).type(torch.cuda.HalfTensor))
        else:
            return (x_ternary.type(torch.cuda.FloatTensor), torch.tensor(
    multiplier).type(torch.cuda.FloatTensor))
```

Listing 2.1 The code snippet that reflects the ternary operation.

2.3.2 Sparse Squeeze-excitation Layers

Empirically, we found that applying quantization on some specific network layers leads to significant accuracy drop, which includes the squeeze-excitation layers and depth-wise convolutional layers. Therefore, we keep the layers that are sensitive to quantization in their original format (FP32) during the initial training. After initial training, we then freeze the quantized layers and conduct pruning on rest layers that are sensitive to quantization using the magnitude pruner provided by the Neural Network Distiller toolbox [39].

2.3.3 Self-supervised Consistency Regularizer

During training, we use mixup [40] as a data augmentation technique. The self-supervised consistency regularizer is built upon mixup [40], enforcing feature-level consistency between mixup data points in the feature space without using the label information. Denoting a pair of sample-label tuples as (x_i, y_i), (x_j, y_j), mixup generates augmented tuples with a simple weighted sum [40]:

$$x' = \lambda x_i + (1 - \lambda) x_j \tag{2.4}$$

$$y' = \lambda y_i + (1 - \lambda) y_j \tag{2.5}$$

where $\lambda \in [0, 1]$. By using the constructed data point (x', y') for training, it encourages the linear behavior of the model where linear interpolation in the raw data leads to linear interpolation of predictions. Denoting $\mathcal{Z}$ as the feature extracted by the network f and $z \in \mathcal{Z}$, we define the regularizer term as:

$$z^{ij} = \lambda f_\theta(x_i) + (1 - \lambda) f_\theta(x_j) \tag{2.6}$$

$$\mathcal{L}_z = \frac{1}{B} \sum_i \left\| z^{ij} - f_\theta(x') \right\|_2^2 \tag{2.7}$$

where x' is from Eq. (2.4). That is, by minimizing $\mathcal{L}_z$ we push the mixed feature z^{ij} closer to the feature of the mixed input $f_\theta(x')$ through MSE loss between the two vectors. In this way, we impose the linearity constraint to be enforced also at the feature level without using the label information.

2.4 Experiment

2.4.1 Experiment Setup

We attend the *NeurIPS 2019 MicroNet Challenge (CIFAR-100 Track)* with our proposed MSUNet. Thus, we evaluate the performance on CIFAR-100 dataset. The backbone architectures for MSUNet is MixNet-M [41] which utilize mixed depthwise convolution (MixConv) and squeeze-excitation layers to boost the performance. MSUNet is a series of efficient neural networks with different building blocks. The configuration for each model is listed in Table 2.1.

	Ternary Convolutional	Sparse Squeeze-excite	Consistency Regularizer	Mixed-precision Training
MSUNet-V0	✓			
MSUNet-V1	✓	✓		
MSUNet-V2	✓	✓	✓	
MSUNet-V3	✓	✓	✓	✓

Table 2.1 Configuration of MSUNet series.

Besides the techniques that we developed in Section 2.3, we also employ mixed-precision training that casts the weights from FP32 to FP16 for training and inference, which further boosts the model efficiency.

2.4.2 Results on CIFAR-100

The performance of MSUNet series is reported in Table 2.2. In addition to Top-1 accuracy, we also report the #Flops and #Parameters at inference time for comparison. To calculate #Flops, a 32-bit operation counts as one operation. If quantization is performed, an operation on data of less than 32-bits will be counted as a fraction of one operation. To calculate #Parameters, a 32-bit parameter counts as one parameter. Quantized parameters of less than 32-bits will be counted as a fraction of one parameter. As required by *NeurIPS 2019 MicroNet Challenge* the Score is the sum of #Flops and #Parameters normalized relative to WideResNet-28-10, which has 36.5M parameters and 10.49B math operations.

$$\text{Score} = \frac{\#\text{Flops}}{10.49\ \text{B}} + \frac{\#\text{Parameters}}{36.5\ \text{M}} \tag{2.8}$$

As seen from Table 2.2, by applying ternary quantization alone, the MSUNet-V0 could achieve a very large improvement over the base model with normalized score 0.01836. By further adding the sparse sequeeze-excitation component for MSUNet-V1, we see a slight accuracy drop alone with a drop of #Flops and #Parameters. By adding the consistency regularizer component to MSUNet-V2, we did not see an improvement in accuracy, instead we obtained a significant improvement over #Flops and #Parameters. This indicates that the consistency regularizer could help improve the quantization and pruning process. Lastly, as we add the mixed-precision training in MSUNet-V3, an accuracy drop of 0.17% coupled with improvment over #Flops and #Parameters. The

final performance of MSUNet-V3 wins the 4th place in the *NeurIPS 2019 MicroNet Challenge (CIFAR-100 Track)*.

Model	Top-1 acc	#Flops	#Paramters	Score
MSUNet-V0	80.61 %	122.6 M	0.2437 M	0.01836
MSUNet-V1	80.47%	118.6 M	0.2199 M	0.01711
MSUNet-V2	80.30%	97.01 M	0.1204 M	0.01255
MSUNet-V3	80.13%	85.27 M	0.09853 M	0.01083

Table 2.2 The performance of MSUNet series on CIFAR-100 dataset.

2.4.3 Comparison of Ternary and Binary Quantization

Even though the ternary quantization technique proposed in section 2.3 admits binary quantization (i.e., quantized to $\{-1, 1\}$ instead of $\{-1, 0, 1\}$) without significant modifying the implementation, we found that the performance of ternary quantization performs significantly better than binary in terms of #Flops and #Parameters accuracy as shown in Table 2.3. The major reason is that ternarization introduced may 0s which can be efficiently represented by sparse data structures. Moreover, the additional quantization level of ternary weights brings more freedom for performing quantization.

Quantization	Top-1 acc	#Flops	#Paramters
Binary	80.43%	170.0 M	0.2564 M
Ternary	80.61%	122.6 M	0.2437 M

Table 2.3 Comparison between binary and ternary quantization of MixNet-M base model.

2.5 Conclusion

In this chapter, we propose MSUNet, which is designed with three key techniques: 1) ternary convolution layers, 2) sparse squeeze-excitation layers, and 3) self-supervised consistency regularizer along with mixed precision training. Our framework shows a great improvement in parameter size and computational cost measure by #Parameters and #Flops over the baseline model. Lastly, we show that ternary quantization outperforms binary quantization in all aspects, including accuracy, parameter size, and computation cost.

CHAPTER 3

DEEP AUTOAUGMENT

3.1 Introduction

Data augmentation (DA) is a powerful technique for machine learning since it effectively regularizes the model by increasing the number and the diversity of data points [42, 43]. A large body of data augmentation transformations has been proposed [44, 40, 45, 46, 47, 48] to improve model performance. While applying a set of well-designed augmentation transformations could help yield considerable performance enhancement especially in image recognition tasks, manually selecting high-quality augmentation transformations and determining how they should be combined still require strong domain expertise and prior knowledge of the dataset of interest. With the recent trend of automated machine learning (AutoML), data augmentation search flourishes in the image domain [1, 49, 50, 51, 52, 53, 54], which yields significant performance improvement over hand-crafted data augmentation methods.

Although data augmentation policies in previous works [1, 49, 50, 51, 52, 53] contain multiple

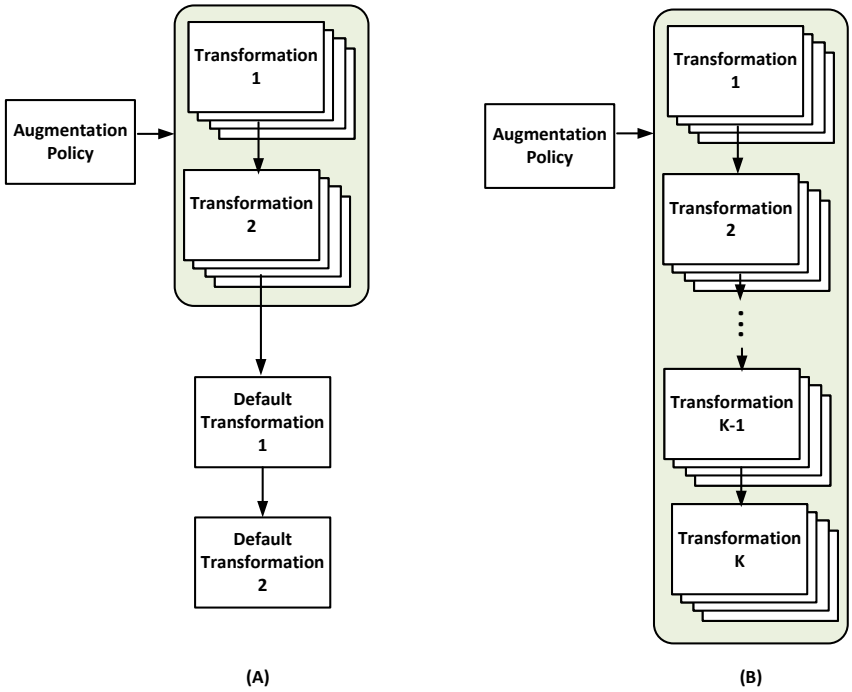


Figure 3.1 (A) Existing automated data augmentation methods with shallow augmentation policy followed by hand-picked transformations. (B) DeepAA with deep augmentation policy with no hand-picked transformations.

transformations applied sequentially, only one or two transformations of each sub-policy are found through searching whereas the rest transformations are hand-picked and applied by default in addition to the found policy (Figure 3.1(A)). From this perspective, we believe that previous automated methods are *not entirely automated* as they are still built upon hand-crafted default augmentations.

In this work, we propose Deep AutoAugment (DeepAA), a multi-layer data augmentation search method which aims to remove the need of hand-crafted default transformations (Figure 3.1(B)). DeepAA fully automates the data augmentation process by searching a deep data augmentation policy on an expanded set of transformations that includes the widely adopted search space and the default transformations (*e.g.* flips, Cutout, crop). We formulate the search of data augmentation policy as a regularized gradient matching problem by maximizing the cosine similarity of the gradients between augmented data and original data with regularization. To avoid exponential growth of dimensionality of the search space when more augmentation layers are used, we incrementally stack augmentation layers based on the data distribution transformed by all the previous augmentation layers.

We evaluate the performance of DeepAA on three datasets – CIFAR-10, CIFAR-100, and ImageNet – and compare it with existing automated data augmentation search methods including AutoAugment (AA) [1], PBA [50], Fast AutoAugment (FastAA) [51], Faster AutoAugment (Faster AA) [52], DADA [53], RandAugment (RA) [49], UniformAugment (UA) [55], TrivialAugment (TA) [56], and Adversarial AutoAugment (AdvAA) [57]. Our results show that, without any default augmentations, DeepAA achieves the best performance compared to existing automatic augmentation search methods on CIFAR-10, CIFAR-100 on Wide-ResNet-28-10 and ImageNet on ResNet-50 and ResNet-200 with standard augmentation space and training procedure.

We summarize our main contributions below:

- We propose Deep AutoAugment (DeepAA), a fully automated data augmentation search method that finds a multi-layer data augmentation policy from scratch.
- We formulate such multi-layer data augmentation search as a regularized gradient matching

problem. We show that maximizing cosine similarity along the direction of low variance is effective for data augmentation search when augmentation layers go deep.

- We address the issue of exponential growth of the dimensionality of the search space when more augmentation layers are added by incrementally adding augmentation layers based on the data distribution transformed by all the previous augmentation layers.

- Our experiment results show that, without using any default augmentations, DeepAA achieves stronger performance compared with prior works.

3.2 Related Work

Automated Data Augmentation. Automating data augmentation policy design has recently emerged as a promising paradigm for data augmentation. The pioneer work on automated data augmentation was proposed in AutoAugment [1], where the search is performed under reinforcement learning framework. AutoAugment requires to train the neural network repeatedly, which takes thousands of GPU hours to converge. Subsequent works [51, 53, 54] aim at reducing the computation cost. Fast AutoAugment [51] treats data augmentation as inference time density matching which can be implemented efficiently with Bayesian optimization. Differentiable Automatic Data Augmentation (DADA) [53] further reduces the computation cost through a reparameterized Gumbel-softmax distribution [58]. RandAugment [49] introduces a simplified search space containing two interpretable hyperparameters, which can be optimized simply by grid search. Adversarial AutoAugment (AdvAA) [57] searches for the augmentation policy in an adversarial and online manner. It also incorporates the concept of Batch Augmentaiton [59, 60], where multiple adversarial policies run in parallel. Although many automated data augmentation methods have been proposed, the use of default augmentations still imposes strong domain knowledge.

Gradient Matching. Our work is also related to gradient matching. In [61], the authors showed that the cosine similarity between the gradients of different tasks provides a signal to detect when an auxiliary loss is helpful to the main loss. In [62], the authors proposed to use cosine similarity as the training signal to optimize the data usage via weighting data points. A similar approach was

proposed in [63], which uses the gradient inner product as a per-example reward for optimizing data distribution and data augmentation under the reinforcement learning framework. Our approach also utilizes the cosine similarity to guide the data augmentation search. However, our implementation of cosine similarity is different from the above from two aspects: we propose a Jacobian-vector product form to backpropagate through the cosine similarity, which is computational and memory efficient and does not require computing higher order derivative; we also propose a sampling scheme that effectively allows the cosine similarity to increase with added augmentation stages.

3.3 Deep AutoAugment

3.3.1 Overview

Data augmentation can be viewed as a process of filling missing data points in the dataset with the same data distribution [52]. By augmenting a single data point multiple times, we expect the resulting data distribution to be close to the full dataset under a certain type of transformation. For example, by augmenting a single image with proper color jittering, we obtain a batch of augmented images which has similar distribution of lighting conditions as the full dataset. As the distribution of augmented data gets closer to the full dataset, the gradient of the augmented data should be steered towards a batch of original data sampled from the dataset. In DeepAA, we formulate the search of the data augmentation policy as a regularized gradient matching problem, which manages to steer the gradient to a batch of original data by augmenting a single image multiple times. Specifically, we construct the augmented training batch by augmenting a single training data point multiple times following the augmentation policy. We construct a validation batch by sampling a batch of original data from the validation set. We expect that by augmentation, the gradient of augmented training batch can be steered towards the gradient of the validation batch. To do so, we search for data augmentation that maximizes the cosine similarity between the gradients of the validation data and the augmented training data. The intuition is that an effective data augmentation should preserve data distribution [64] where the distribution of the augmented images should align with the distribution of the validation set such that the training gradient direction is close to the validation gradient direction.

Another challenge for augmentation policy search is that the search space can be prohibitively large with deep augmentation layers ($K \geq 5$). This was not a problem in previous works, where the augmentation policies is shallow ($K \leq 2$). For example, in AutoAugment [1], each sub-policy contains $K = 2$ transformations to be applied sequentially, and the search space of AutoAugment contains 16 image operations and 10 discrete magnitude levels. The resulting number of combinations of transformations in AutoAugment is roughly $(16 \times 10)^2 = 25,600$, which is handled well in previous works. However, when discarding the default augmentation pipeline and searching for data augmentations from scratch, it requires deeper augmentation layers in order to perform well. For a data augmentation with $K = 5$ sequentially applied transformations, the number of sub-policies is $(16 \times 10)^5 \approx 10^{11}$, which is prohibitively large for the following two reasons. First, it becomes less likely to encounter a good policy by exploration as good policies become more sparse on high dimensional search space. Second, the dimension of parameters in the policy also grows with K, making it more computational challenging to optimize. To tackle this challenge, we propose to build up the full data augmentation by progressively stacking augmentation layers, where each augmentation layer is optimized on top of the data distribution transformed by all previous layers. This avoids sampling sub-policies from such a large search space, and the number of parameters of the policy is reduced from $|\mathbb{T}|^K$ to $\mathbb{T}$ for each augmentation layer.

3.3.2 Search Space

Let $\mathbb{O}$ denote the set of augmentation operations (*e.g.* identity, rotate, brightness), m denote an operation magnitude in the set $\mathbb{M}$, and x denote an image sampled from the space $\mathcal{X}$. We define the set of transformations as the set of operations with a fixed magnitude as $\mathbb{T} := \{t | t = o(\cdot\,;\, m),\ o \in \mathbb{O} \text{ and } m \in \mathbb{M}\}$. Under this definition, every t is a map $t : \mathcal{X} \to \mathcal{X}$, and there are $|\mathbb{T}| = |\mathbb{M}| \cdot |\mathbb{O}|$ possible transformations. In previous works [1, 51, 53, 52], a data augmentation policy $\mathcal{P}$ consists of several sub-policies. As explained above, the size of candidate sub-policies grows exponentially with depth K. Therefore, we propose a practical method that builds up the full data augmentation by progressively stacking augmentation layers. The final data augmentation policy hence consists of K layers of sequentially applied policy $\mathcal{P} = \{\mathcal{P}_1, \cdots, \mathcal{P}_K\}$, where policy $\mathcal{P}_k$ is optimized conditioned

on the data distribution augmented by all previous $(k-1)$ layers of policies. Thus we write the policy as a conditional distribution $\mathcal{P}_k := p_{\theta_k}(n|\{\mathcal{P}_1, \cdots, \mathcal{P}_{k-1}\})$ where n denotes the indices of transformations in $\mathbb{T}$. For the purpose of clarity, we use a simplified notation as p_{θ_k} to replace $p_{\theta_k}(n|\{\mathcal{P}_1, \cdots, \mathcal{P}_{k-1}\})$.

3.3.3 Augmentation Policy Search via Regularized Gradient Matching

Assume that a single data point x is augmented multiple times following the policy p_θ. The resulting average gradient of such augmentation is denoted as $g(x, \theta)$, which is a function of data x and policy parameters θ. Let v denote the gradients of a batch of the original data. We optimize the policy by maximizing the cosine similarity between the gradients of the augmented data and a batch of the original data as follows:

$$\begin{aligned} \theta &= \arg\max_\theta \ \text{cosineSimilarity}(v, g(x, \theta)) \\ &= \arg\max_\theta \ \frac{v^T \cdot g(x, \theta)}{\|v\| \cdot \|g(x, \theta)\|} \end{aligned} \tag{3.1}$$

where $\|\cdot\|$ denotes the L2-norm. The parameters of the policy can be updated via gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta \ \text{cosineSimilarity}(v, g(x, \theta)), \tag{3.2}$$

where η is the learning rate.

3.3.3.1 Policy Search for One layer

We start with the case where the data augmentation policy only contains a single augmentation layer, *i.e.*, $\mathcal{P} = \{p_\theta\}$. Let $L(x; w)$ denote the classification loss of data point x where $w \in \mathbb{R}^D$ represents the flattened weights of the neural network. Consider applying augmentation on a single data point x following the distribution p_θ. The resulting averaged gradient can be calculated analytically by averaging all the possible transformations in $\mathbb{T}$ with the corresponding probability $p(\theta)$:

$$\begin{aligned} g(x; \theta) &= \sum_{n=1}^{|\mathbb{T}|} p_\theta(n) \nabla_w L(t_n(x); w) \\ &= G(x) \cdot p_\theta \end{aligned} \tag{3.3}$$

where $G(x) = \left[\nabla_w L(t_1(x); w), \cdots, \nabla_w L(t_{|\mathbb{T}|}(x); w)\right]$ is a $D \times |\mathbb{T}|$ Jacobian matrix, and $p_\theta = [p_\theta(1), \cdots, p_\theta(|\mathbb{T}|)]^T$ is a $|\mathbb{T}|$ dimensional categorical distribution. The gradient w.r.t. the cosine similarity in Eq. (3.2) can be derived as:

$$\nabla_\theta \text{ cosineSimilarity}(v, g(x; \theta)) = \nabla_\theta p_\theta \cdot r \tag{3.4}$$

where

$$r = G(x)^T \left(\frac{v}{\|g(\theta)\|} - \frac{v^T g(\theta)}{\|g(\theta)\|^2} \cdot \frac{g(\theta)}{\|g(\theta)\|} \right) \tag{3.5}$$

which can be interpreted as a reward for each transformation. Therefore, $p_\theta \cdot r$ in Eq.(3.4) represents the average reward under policy p_θ.

3.3.3.2 Policy Search for Multiple layers

The above derivation is based on the assumption that $g(\theta)$ can be computed analytically by Eq.(3.3). However, when $K \geq 2$, it becomes impractical to compute the average gradient of the augmented data given that the search space dimensionality grows exponentially with K. Consequently, we need to average the gradient of all $|\mathbb{T}|^K$ possible sub-policies.

To reduce the parameters of the policy to $\mathbb{T}$ for each augmentation layer, we propose to incrementally stack augmentations based on the data distribution transformed by all the previous augmentation layers. Specifically, let $\mathcal{P} = \{\mathcal{P}_1, \cdots, \mathcal{P}_K\}$ denote the K-layer policy. The policy $\mathcal{P}_k$ modifies the data distribution on top of the data distribution augmented by the previous $(k-1)$ layers. Therefore, the policy at the k^{th} layer is a distribution $\mathcal{P}_k = p_{\theta_k}(n)$ conditioned on the policies $\{\mathcal{P}_1, \cdots, \mathcal{P}_{k-1}\}$ where each one is a $|\mathbb{T}|$-dimensional categorical distribution. Given that, the Jacobian matrix at the k^{th} layer can be derived by averaging over the previous $(k-1)$ layers of policies as follows:

$$\begin{aligned} G(x)^k = \sum_{n_{k-1}=1}^{|\mathbb{T}|} \cdots \sum_{n_1=1}^{|\mathbb{T}|} p_{\theta_{k-1}}(n_{k-1}) \cdots p_{\theta_1}(n_1) [\nabla_w L((t_1 \circ t_{n_{k-1}} \cdots \circ t_{n_1})(x); w), \cdots, \\ \nabla_w L((t_{|\mathbb{T}|} \circ t_{n_{k-1}} \circ \cdots \circ t_{n_1})(x); w)] \end{aligned} \tag{3.6}$$

where G^k can be estimated via the Monte Carlo method as:

$$\tilde{G}^k(x) = \sum_{\tilde{n}_{k-1} \sim p_{\theta_k}} \cdots \sum_{\tilde{n}_1 \sim p_{\theta_1}} [\nabla_w L((t_1 \circ t_{\tilde{n}_{k-1}} \cdots \circ t_{\tilde{n}_1})(x); w), \cdots, \nabla_w L((t_{|\mathbb{T}|} \circ t_{\tilde{n}_{k-1}} \circ \cdots \circ t_{\tilde{n}_1})(x); w)] \tag{3.7}$$

where $\tilde{n}_{k-1} \sim p_{\theta_{k-1}}(n), \cdots, \tilde{n}_1 \sim p_{\theta_1}(n)$.

The average gradient at the k^{th} layer can be estimated by the Monte Carlo method as:

$$\tilde{g}(x; \theta_k) = \sum_{\tilde{n}_k \sim p_{\theta_k}} \cdots \sum_{\tilde{n}_1 \sim p_{\theta_1}} \nabla_w L\left((t_{\tilde{n}_k} \circ \cdots \circ t_{\tilde{n}_1})(x); w\right). \tag{3.8}$$

Therefore, the reward at the k^{th} layer is derived as:

$$\tilde{r}^k(x) = \left(\tilde{G}^k(x)\right)^T \left(\frac{v}{\|\tilde{g}_k(x; \theta_k)\|} - \frac{v^T \tilde{g}_k(x; \theta_k)}{\|\tilde{g}_k(x; \theta_k)\|^2} \cdot \frac{\tilde{g}_k(x; \theta_k)}{\|\tilde{g}_k(x; \theta_k)\|}\right). \tag{3.9}$$

To prevent the augmentation policy from overfitting, we regularize the optimization by avoiding optimizing towards the direction with high variance. Thus, we penalize the average reward with its standard deviation as

$$r^k = E_x\{\tilde{r}^k(x)\} - c \cdot \sqrt{E_x\{(\tilde{r}^k(x) - E_x\{\tilde{r}^k(x)\})^2\}}, \tag{3.10}$$

where we use 16 randomly sampled images to calculate the expectation. The hyperparameter c controls the degree of regularization, which is set to 1.0. With such regularization, we prevent the policy from converging to the transformations with high variance.

Therefore the parameters of policy $\mathcal{P}_k$ ($k \geq 2$) can be updated as:

$$\theta \leftarrow \theta_k + \eta \nabla_{\theta_k} \text{cosineSimilarity}(v, g(\theta_k)) \tag{3.11}$$

where

$$\nabla_\theta \text{cosineSimilarity}(v, g^k(x; \theta)) = \nabla_\theta p_{\theta_k} \cdot r^k. \tag{3.12}$$

3.4 Experiments and Analysis

Benchmarks and Baselines. We evaluate the performance of DeepAA on three standard benchmarks: CIFAR-10, CIFAR-100, ImageNet, and compare it against a baseline based on

standard augmentations (*i.e.*, flip left-righ, pad-and-crop for CIFAR-10/100, and Inception-style preprocesing [65] for ImageNet) as well as nine existing automatic augmentation methods including (1) AutoAugment (AA) [1], (2) PBA [50], (3) Fast AutoAugment (Fast AA) [51], (4) Faster AutoAugment [52], (5) DADA [53], (6) RandAugment (RA) [49], (7) UniformAugment (UA) [55], (8) TrivialAugment (TA) [56], and (9) Adversarial AutoAugment (AdvAA) [57].

Search Space. We set up the operation set $\mathbb{O}$ to include 16 commonly used operations (identity, shear-x, shear-y, translate-x, translate-y, rotate, solarize, equalize, color, posterize, contrast, brightness, sharpness, autoContrast, invert, Cutout) as well as two operations (*i.e.*, flips and crop) that are used as the default operations in the aforementioned methods. The list of operations and the range of magnitudes in the standard augmentation space are summarized in Table 3.1. Among the operations in $\mathbb{O}$, 11 operations are associated with magnitudes. We then discretize the range of magnitudes into 12 uniformly spaced levels and treat each operation with a discrete magnitude as an independent transformation. Therefore, the policy in each layer is a 139-dimensional categorical distribution corresponding to $|\mathbb{T}| = 139$ {operation, magnitude} pairs.

Operation	Magnitude
Identity	-
ShearX	[-0.3, 0.3]
ShearY	[-0.3, 0.3]
TranslateX	[-0.45, 0.45]
TranslateY	[-0.45, 0.45]
Rotate	[-30, 30]
AutoContrast	-
Invert	-
Equalize	-
Solarize	[0, 256]
Posterize	[4, 8]
Contrast	[0.1, 1.9]
Color	[0.1, 1.9]
Brightness	[0.1, 1.9]
Sharpness	[0.1, 1.9]
Flips	-
Cutout	16 (60)
Crop	-

Table 3.1 List of operations in the search space and the corresponding range of magnitudes in the standard augmentation space. Note that some operations do not use magnitude parameters. We add flip and crop to the search space which were found in the default augmentation pipeline in previous works. Flips operates by randomly flipping the images with 50% probability. In line with previous works, crop denotes pad-and-crop and resize-and-crop transforms for CIFAR10/100 and ImageNet respectively. We set Cutout magnitude to 16 for CIFAR10/100 dataset to be the same as the Cutout in the default augmentation pipeline. We set Cutout magnitude to 60 pixels for ImageNet which is the upper limit of the magnitude used in AA [1].

3.4.1 Performance on CIFAR-10 and CIFAR-100

Policy Search. Following [1], we conduct the augmentation policy search based on Wide-ResNet-40-2 [17]. We first train the network on a subset of $4,000$ randomly selected samples from CIFAR-10. We then progressively update the policy network parameters θ_k $(k = 1, 2, \cdots, K)$ for 512 iterations for each of the K augmentation layers. We use the Adam optimizer [66] and set the learning rate to 0.025 for policy updating.

Policy Evaluation. Using the publicly available repository of Fast AutoAugment [51], we evaluate the found augmentation policy on both CIFAR-10 and CIFAR-100 using Wide-ResNet-28-10 and Shake-Shake-2x96d models. The evaluation configurations are kept consistent with that

of Fast AutoAugment.

Results. Table 3.2 reports the Top-1 test accuracy on CIFAR-10/100 for Wide-ResNet-28-10 and Shake-Shake-2x96d, respectively. The results of DeepAA are the average of four independent runs with different initializations. We also show the 95% confidence interval of the mean accuracy. As shown, DeepAA achieves the best performance compared against previous works using the standard augmentation space. *Note that TA(Wide) uses a wider (stronger) augmentation space on this dataset.*

	Baseline	**AA**	**PBA**	**FastAA**	**FasterAA**	**DADA**	**RA**	**UA**	**TA(RA)**	**TA(Wide)** [1]	**DeepAA**
CIFAR-10											
WRN-28-10	96.1	97.4	97.4	97.3	97.4	97.3	97.3	97.33	97.46	97.46	**97.56** ± 0.14
Shake-Shake (26 2x96d)	97.1	98.0	98.0	98.0	98.0	98.0	98.0	98.1	98.05	98.21	**98.11** ± 0.12
CIFAR-100											
WRN-28-10	81.2	82.9	83.3	82.7	82.7	82.5	83.3	82.82	83.54	84.33	**84.02** ± 0.18
Shake-Shake (26 2x96d)	82.9	85.7	84.7	85.1	85.0	84.7	-	-	-	86.19	**85.19** ± 0.28

Table 3.2 Top-1 test accuracy on CIFAR-10/100 for Wide-ResNet-28-10 and Shake-Shake-2x96d. The results of DeepAA are averaged over four independent runs with different initializations. The 95% confidence interval is denoted by ±.

3.4.2 Performance on ImageNet

Policy Search. We conduct the augmentation policy search based on ResNet-18 [14]. We first train the network on a subset of 200, 000 randomly selected samples from ImageNet for 30 epochs. We then use the same settings as in CIFAR-10 for updating the policy parameters.

Policy Evaluation. We evaluate the performance of the found augmentation policy on ResNet-50 and ResNet-200 based on the public repository of Fast AutoAugment [51]. The parameters for training are the same as the ones of [51]. In particular, we use step learning rate scheduler with a reduction factor of 0.1, and we train and evaluate with images of size 224x224.

Results. The performance on ImageNet is presented in Table 3.3. As shown, DeepAA achieves the best performance compared with previous methods without the use of default augmentation pipeline. In particular, DeepAA performs better on larger models (*i.e.* ResNet-200), as the performance of DeepAA on ResNet-200 is the best within the 95% confidence interval. *Note that*

[1] On CIFAR-10/100, TA (Wide) uses a wider (stronger) augmentation space, while the other methods including TA (RA) uses the standard augmentation space.

while we train DeepAA using the image resolution (224×224), we report the best results of RA and TA, which are trained with a larger image resolution (244×224) on this dataset.

	Baseline	**AA**	**Fast AA**	**Faster AA**	**DADA**	**RA**	**UA**	**TA(RA)**[1]	**TA(Wide)**[2]	**DeepAA**
ResNet-50	76.3	77.6	77.6	76.5	77.5	77.6	77.63	77.85	78.07	**78.30 ± 0.14**
ResNet-200	78.5	80.0	80.6	-	-	-	80.4	-	-	**81.32 ± 0.17**

Table 3.3 Top-1 test accuracy (%) on ImageNet for ResNet-50 and ResNet-200. The results of DeepAA are averaged over four independent runs with different initializations. The 95% confidence interval is denoted by ±.

3.4.3 Performance with Batch Augmentation

Batch Augmentation (BA) is a technique that draws multiple augmented instances of the same sample in one mini-batch. It has been shown to be able to improve the generalization performance of the network [59, 60]. AdvAA [57] directly searches for the augmentation policy under the BA setting whereas for TA and DeepAA, we apply BA with the same augmentation policy used in Table 3.2. Note that since the performance of BA is sensitive to the hyperparameters [67], we have conducted a grid search on the hyperparameters of both TA and DeepAA (details are included in Appendix 3.4.7). As shown in Table 3.4, after tuning the hyperparameters, the performance of TA (Wide) using BA is already better than the reported performance in the original paper. The performance of DeepAA with BA outperforms that of both AdvAA and TA (Wide) with BA.

	AdvAA	**TA(Wide)** (original paper)	**TA(Wide)** (ours)	**DeepAA**
CIFAR-10	98.1 ± 0.15	98.04 ± 0.06	98.06 ± 0.23	**98.21 ± 0.14**
CIFAR-100	84.51 ± 0.18	84.62 ± 0.14	85.40 ± 0.15	**85.61 ± 0.17**

Table 3.4 Top-1 test accuracy (%) on CIFAR-10/100 dataset with WRN-28-10 with Batch Augmentation (BA), where eight augmented instances were drawn for each image. The results of DeepAA are averaged over four independent runs with different initializations. The 95% confidence interval is denoted by ±.

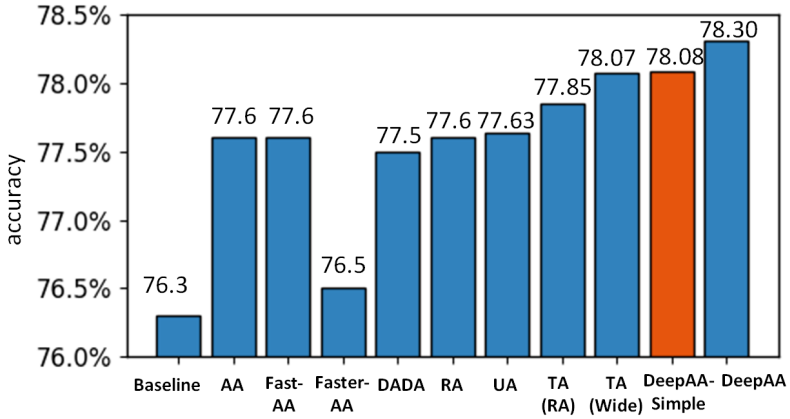


Figure 3.2 Top-1 test accuracy (%) on ImageNet of DeepAA-simple, DeepAA, and other automatic augmentation methods on ResNet-50.

3.4.4 Understanding DeepAA

Effectiveness of Gradient Matching. One uniqueness of DeepAA is the regularized gradient matching objective. To examine its effectiveness, we remove the impact coming from multiple augmentation layers, and only conduct search for a single layer of augmentation policy. When evaluating the searched policy, we apply the default augmentation in addition to the searched policy. We refer to this variant as DeepAA-simple. Figure 3.2 compares the Top-1 test accuracy on ImageNet using ResNet-50 between DeepAA-simple, DeepAA, and other automatic augmentation methods. While there is 0.22% performance drop compared to DeepAA, *with a single augmentation layer*, DeepAA-simple still outperforms other methods and is able to achieve similar performance compared to TA (Wide) but with a standard augmentation space and trains on a smaller image size (224×224 vs 244×224).

Policy Search Cost. Table 3.5 compares the policy search time on CIFAR-10/100 and ImageNet in GPU hours. DeepAA has comparable search time as PBA, Fast AA, and RA, but is slower than Faster AA and DADA. Note that Faster AA and DADA relax the discrete search space to a continuous one similar to DARTS [68]. While such relaxation leads to shorter searching time, it inevitably introduces a discrepancy between the true and relaxed augmentation spaces.

[1]TA (RA) achieves 77.55% top-1 accuracy with image resolution 224×224.
[2]TA (Wide) achieves 77.97% top-1 accuracy with image resolution 224×224.

Dataset	AA	PBA	Fast AA	Faster AA	DADA	RA	DeepAA
CIFAR-10/100	5000	5	3.5	0.23	0.1	25	9
ImageNet	15000	-	450	2.3	1.3	5000	96

Table 3.5 Policy search time on CIFAR-10/100 and ImageNet in GPU hours.

Impact of the Number of Augmentation Layers. Another uniqueness of DeepAA is its multi-layer search space that can *go beyond* two layers which existing automatic augmentation methods were designed upon. We examine the impact of the number of augmentation layers on the performance of DeepAA. Table 3.6 and Table 3.7 show the performance on CIFAR-10/100 and ImageNet respectively with increasing number of augmentation layers. As shown, for CIFAR-10/100, the performance gradually improves when more augmentation layers are added until we reach five layers. The performance does not improve when the sixth layer is added. For ImageNet, we have similar observation where the performance stops improving when more than five augmentation layers are included.

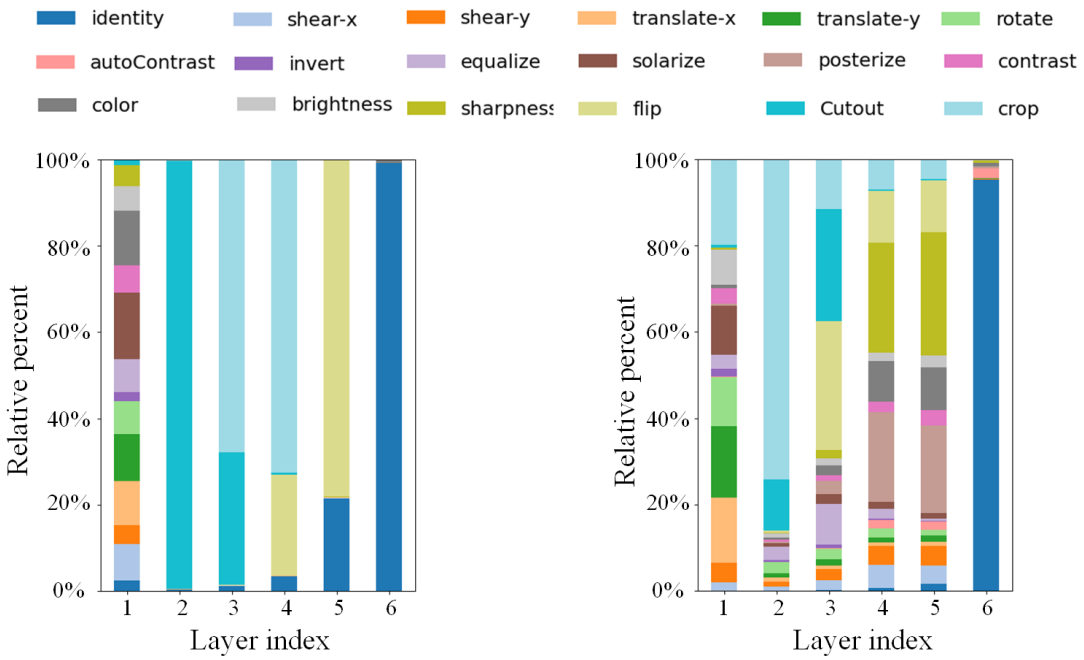
	1 layer	2 layers	3 layers	4 layers	5 layers	6 layers
CIFAR-10	96.3 ± 0.21	96.6 ± 0.18	96.9 ± 0.12	97.4 ± 0.14	**97.56 ± 0.14**	**97.6 ± 0.12**
CIFAR-100	80.9 ± 0.31	81.7 ± 0.24	82.2 ± 0.21	83.7 ± 0.24	**84.02 ± 0.18**	**84.0 ± 0.19**

Table 3.6 Top-1 test accuracy of DeepAA on CIFAR-10/100 for different numbers of augmentation layers. The results are averaged over 4 independent runs with different initializations with the 95% confidence interval denoted by ±.

	1 layer	3 layers	5 layers	7 layers
ImageNet	75.27 ± 0.19	78.18 ± 0.22	**78.30 ± 0.14**	**78.30 ± 0.14**

Table 3.7 Top-1 test accuracy of DeepAA on ImageNet with ResNet-50 for different numbers of augmentation layers. The results are averaged over 4 independent runs w/ different initializations with the 95% confidence interval denoted by ±.

Figure 3.3 illustrates the distributions of operations in the policy for CIFAR-10/100 and ImageNet respectively. As shown in Figure 3.3(a), the augmentation of CIFAR-10/100 converges to *identity* transformation at the sixth augmentation layer, which is a natural indication of the end of the augmentation pipeline. We have similar observation in Figure 3.3(b) for ImageNet, where the

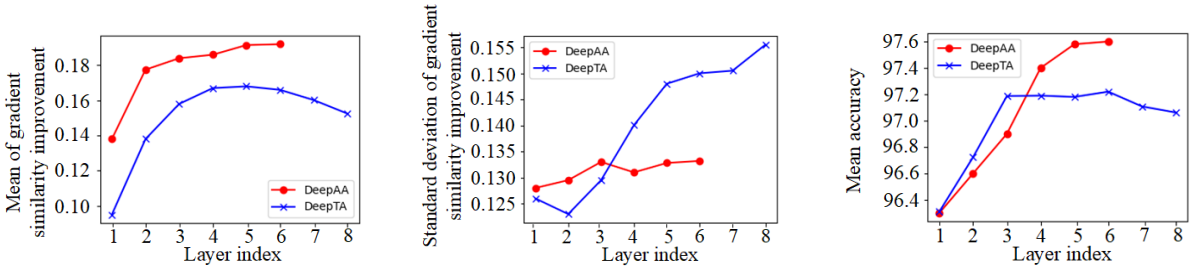


(a) Operation distribution for CIFAR-10/100 (b) Operation distribution for ImageNet

Figure 3.3 The distribution of operations at each layer of the policy for CIFAR-10/100 and ImageNet. The probability of each operation is summed up over all 12 discrete intensity levels (see Appendix 3.4.5 and 3.4.6) of the corresponding transformation.

identity transformation dominates in the sixth augmentation layer. These observations match our results listed in Table 3.6 and Table 3.7. We also include the distribution of the magnitude within each operation for CIFAR-10/100 and ImageNet in Appendix 3.4.5 and Appendix 3.4.6.

Validity of Optimizing Gradient Matching with Regularization. To evaluate the validity of optimizing gradient matching with regularization, we designed a search-free baseline named "DeepTA". In DeepTA, we stack multiple layers of TA on the same augmentation space of DeepAA without using default augmentations. As stated in Eq.(3.10) and Eq.(3.12), we explicitly optimize the gradient similarities with the average reward minus its standard deviation. The first term – the average reward $E_x\{\tilde{r}^k(x)\}$ – *encourages the direction of high cosine similarity*. The second term – the standard deviation of the reward $\sqrt{E_x\{(\tilde{r}^k(x) - E_x\{\tilde{r}^k(x)\})^2\}}$ – acts as a regularization that *penalizes the direction with high variance*. These two terms jointly maximize the gradient similarity along the direction with low variance. To illustrate the optimization trajectory, we design two metrics that are closely related to the two terms in Eq.(3.10): the mean value, and the standard



(a) Mean of the gradient similarity improvement (b) Standard deviation of the gradient similarity improvement (c) Mean accuracy over different augmentation depth

Figure 3.4 Illustration of the search trajectory of DeepAA in comparison with DeepTA on CIFAR-10.

deviation of the improvement of gradient similarity. The improvement of gradient similarity is obtained by subtracting the cosine similarity of the original image batch from that of the augmented batch. In our experiment, the mean and standard deviation of the gradient similarity improvement are calculated over 256 independently sampled original images.

As shown in Figure 3.4(a), the cosine similarity of DeepTA reaches the peak at the fifth layer, and stacking more layers decreases the cosine similarity. In contrast, for DeepAA, the cosine similarity increases consistently until it converges to *identity* transformation at the sixth layer. In Figure 3.4(b), the standard deviation of DeepTA significantly increases when stacking more layers. In contrast, in DeepAA, as we optimize the gradient similarity along the direction of low variance, the standard deviation of DeepAA does not grow as fast as DeepTA. In Figure 3.4(c), both DeepAA and DeepTA reach peak performance at the sixth layer, but DeepAA achieves better accuracy compared against DeepTA. Therefore, we empirically show that DeepAA effectively scales up the augmentation depth by increasing cosine similarity along the direction with low variance, leading to better results.

Comparison with Other Policies. In Figure 3.7 in Appendix 3.4.8, we compare the policy of DeepAA with the policy found by other data augmentation search methods including AA, FastAA and DADA. We have three interesting observations:

- AA, FastAA and DADA assign high probability (over 1.0) on flip, Cutout and crop, as those transformations are hand-picked and applied by default. DeepAA finds a similar pattern that

assigns high probability on flip, Cutout and crop.

- Unlike AA, which mainly focused on color transformations, DeepAA has high probability over both spatial and color transformations.
- FastAA has evenly distributed magnitudes, while DADA has low magnitudes (common issues in DARTS-like method). Interestingly, DeepAA assigns high probability to the stronger magnitudes.

3.4.5 The distribution of magnitudes for CIFAR-10/100

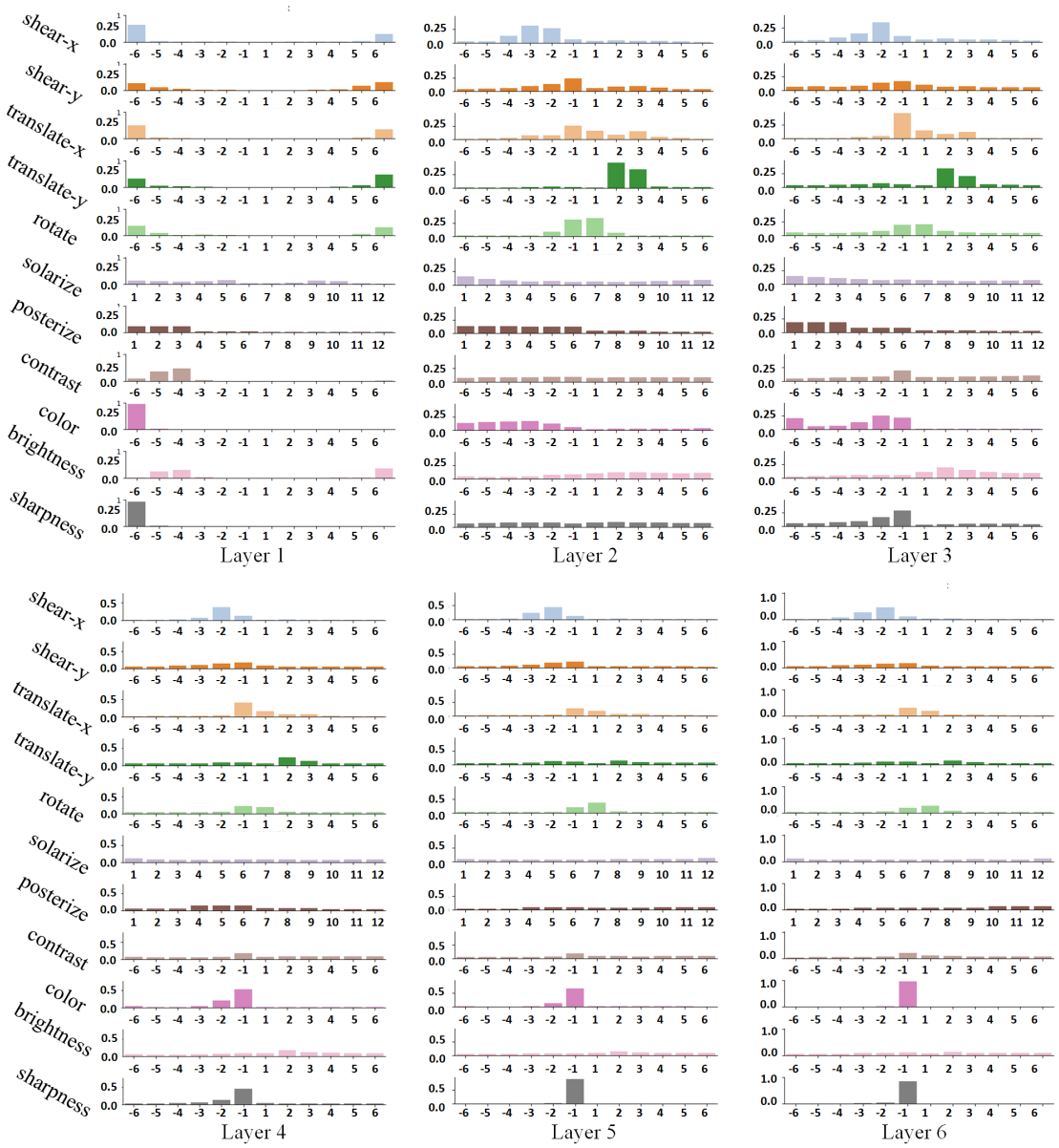


Figure 3.5 The distribution of discrete magnitudes of each augmentation transformation in each layer of the policy for CIFAR-10/100. The x-axis represents the discrete magnitudes and the y-axis represents the probability. The magnitude is discretized to 12 levels with each transformation having its own range. A large absolute value of the magnitude corresponds to high transformation intensity. Note that we do not show identity, autoContrast, invert, equalize, flips, Cutout and crop because they do not have intensity parameters.

3.4.6 The distribution of magnitudes for ImageNet

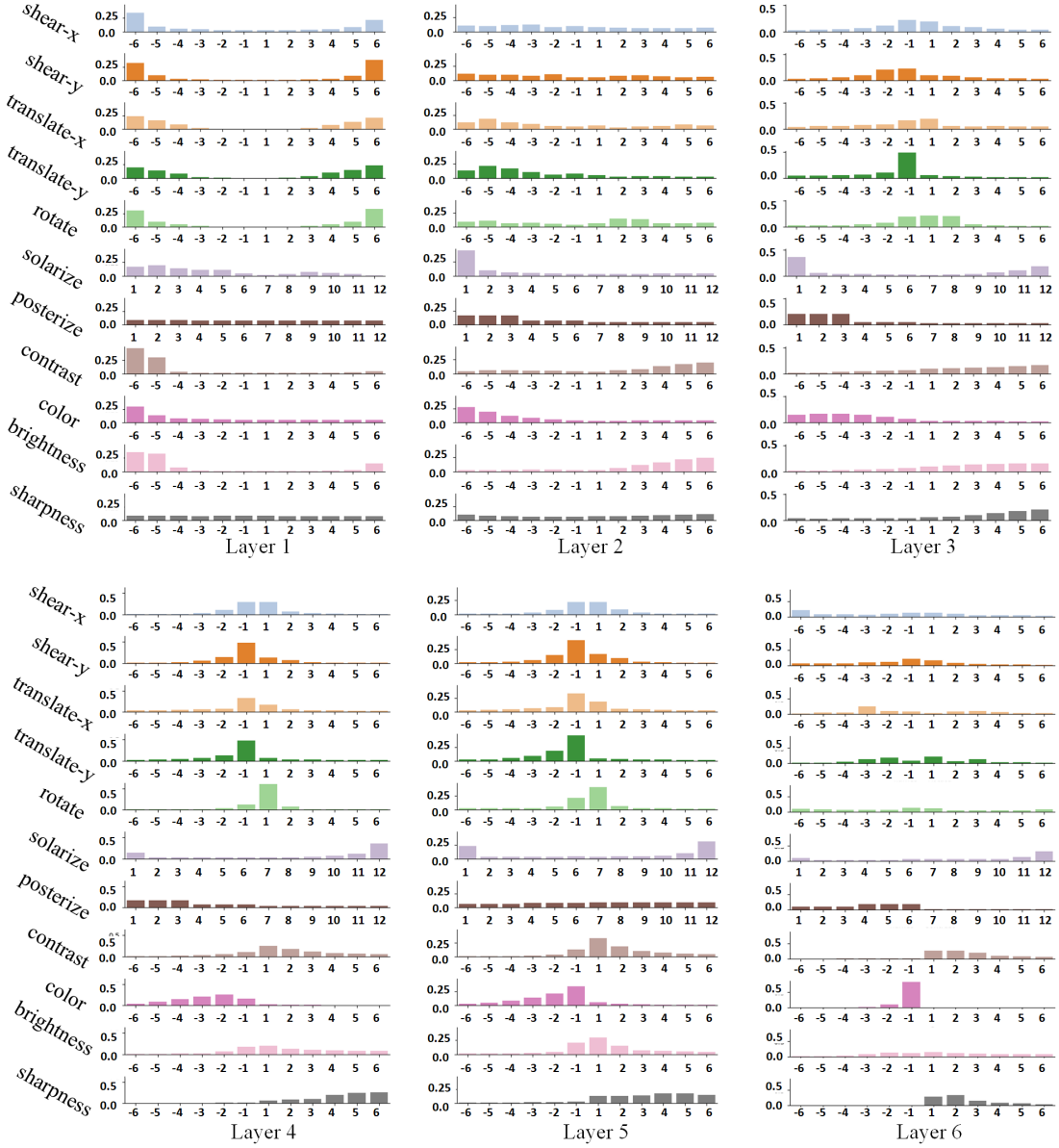


Figure 3.6 The distribution of discrete magnitudes of each augmentation transformation in each layer of the policy for ImageNet. The x-axis represents the discrete magnitudes and the y-axis represents the probability. The magnitude is discretized to 12 levels with each transformation having its own range. A large absolute value of the magnitude corresponds to high transformation intensity. Note that we do not show identity, autoContrast, invert, equalize, flips, Cutout and crop because they do not have intensity parameters.

3.4.7 Hyperparameters for Batch Augmentation

The performance of BA is sensitive to the training settings [67, 69]. Therefore, we conduct a grid search on the learning rate, weight decay and number of epochs for TA and DeepAA with Batch Augmentation. The best-found parameters are summarized in Table 3.8 in Appendix. We did not tune the hyperparameters of AdvAA [57] since AdvAA claims to be adaptive to the training process.

Dataset	Augmentation	Model	Batch Size	Learning Rate	Weight Decay	Epoch
CIFAR-10	TA (Wide)	WRN-28-10	128×8	0.2	0.0005	100
	DeepAA	WRN-28-10	128×8	0.2	0.001	100
CIFAR-100	TA (Wide)	WRN-28-10	128×8	0.4	0.0005	35
	DeepAA	WRN-28-10	128×8	0.4	0.0005	35

Table 3.8 Model hyperparameters of Batch Augmentation on CIFAR10/100 for TA (Wide) and DeepAA. Learning rate, weight decay and number of epochs are found via grid search.

3.4.8 Comparison of data augmentation policy

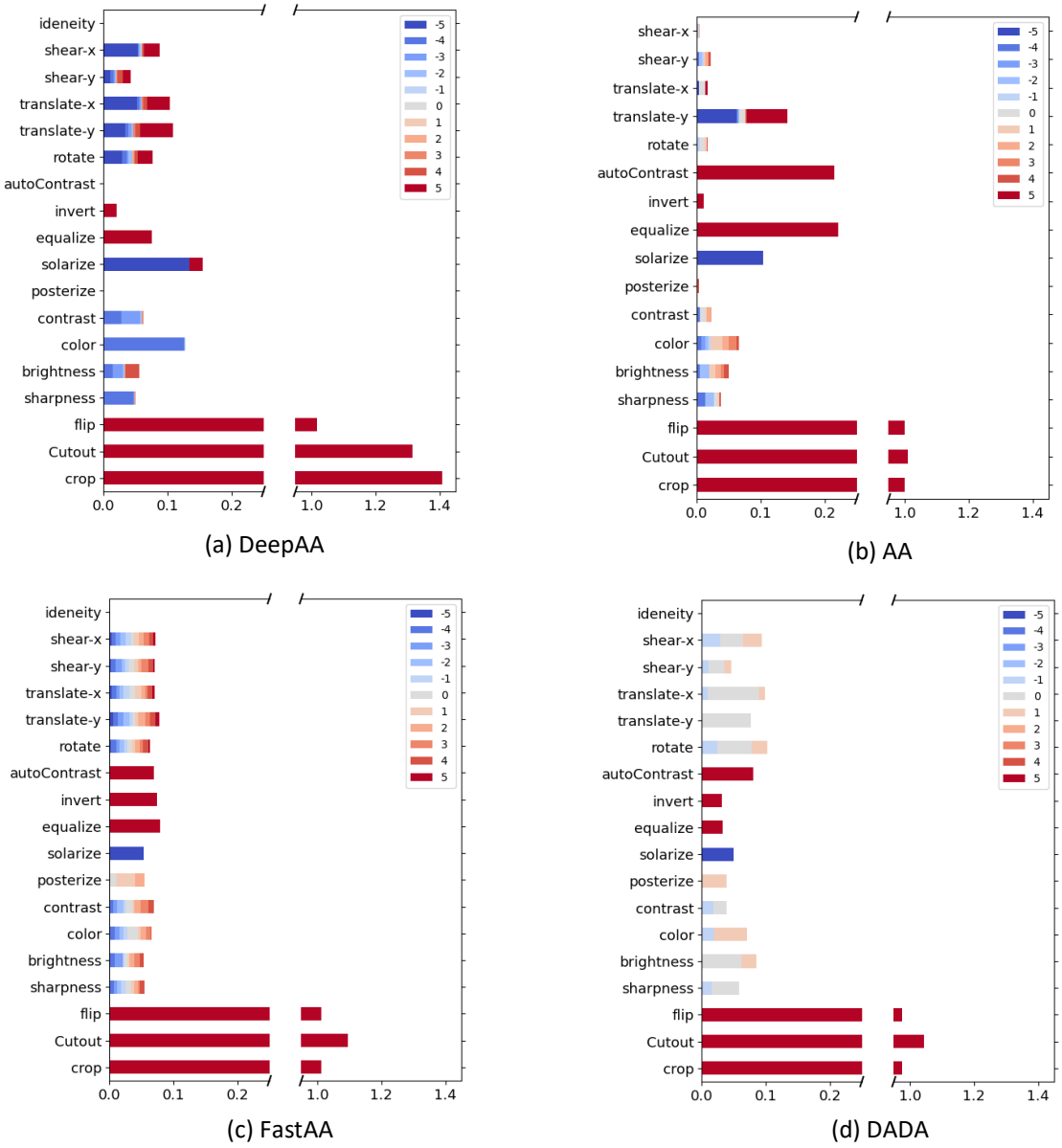


Figure 3.7 Comparison of the policy of DeepAA and some publicly available augmentaiotn policy found by other methods including AA, FastAA and DADA on the CIFAR-10 dataset. Since the compared methods have varied numbers of augmentation layers, we cumulate the probability of each operation over all the augmentation layers. Thus, the cumulative probability can be larger than 1. For AA, Fast AA and DADA, we add additional 1.0 probability to flip, Cutout and Crop, since they are applied by default. In addition, we normalize the magnitude to the range [-5, 5], and use color to distinguish different magnitudes.

3.5 Conclusion

In this chapter, I present Deep AutoAugment (DeepAA), a multi-layer data augmentation search method that finds deep data augmentation policy without using any hand-picked default transformations. We formulate data augmentation search as a regularized gradient matching problem, which maximizes the gradient similarity between augmented data and original data along the direction with low variance. Our experimental results show that DeepAA achieves strong performance without using default augmentations, indicating that regularized gradient matching is an effective search method for data augmentation policies.

CHAPTER 4

DATASET CONDENSATION VIA IMPORTANCE AWARE TRAJECTORY MATCHING

4.1 Introduction of Dataset Condensation

Over recent years, deep learning has achieved significant success across various domains, including computer vision[70], natural language processing[71], and speech recognition[72]. Landmark models such as AlexNet[70] in 2012, ResNet[14] in 2016, Bert[71] in 2018, as well as more recent innovations like ViT[73], CLIP[74], and DALLE[75], all depend on large scale datasets for their training. However, managing such large amounts of data, including collection, storage, transmission, and preprocessing, requires significant effort. Moreover, the computational demands for training on these large datasets often requires large amount of GPU resources for optimal performance. This creates challenges for applications that need to train on a dataset for multiple times such as hyper-parameter optimization[76, 77, 78] and neural architecture search[79, 80, 81]. The situation is exacerbated by the rapid growth of the scale of datasets, where training solely on new data risks catastrophic forgetting[82, 83], and maintaining all historical data becomes impractical. Thus, there is a conflict between the need for high-accuracy models and the limitations of computational and storage resources. A natural solution is dataset condensation, which distill the original datasets into smaller, information-rich subsets to ease storage demands while maintaining model performance at test time.

A direct method to achieve such data condensation is through coreset selection, which selects the most representative samples from original datasets to ensure that models trained on these subsets perform comparably to those trained on the full datasets. Despite its effectiveness, this approach often discards a significant portion of data, that may overlook value training information and lead to suboptimal outcomes. Moreover, using the unmodified original data directly may raise privacy concern.

To address the above challenges, dataset condensation (DC) or dataset distillation (DD) has emerged, focusing on generating new, condensed training data. An overview of dataset condensation

is illustrated in Fig. 4.1 Unlike core-set selection, DD aims to synthesize a limited number of samples encapsulating the essence of the original datasets. Originally proposed by Wang *et al.*[84], this method iteratively updates synthetic samples to ensure model performance well on the original datasets. This foundational work has innovated numerous subsequent studies, significantly advancing DD performance[85, 86, 87, 88, 89, 90] and extending its application to fields like continual learning[91, 92, 93, 94, 95] and federated learning[96, 97, 98, 99, 100, 101, 102].

Neural networks is typically over parameterized with a lot of redundancy, so the weights are not born equally important. Based on this insight, in this chapter we first proposed an importance aware weight factorization, which factorize the weight matrix based on its influence on the layer output. Building on that we proposed an importance aware trajectory matching algorithm which only match the most important weights early in the training trajectory and gradually adding more weight components until reaching the full weights later in training trajectory. Our methods removes the interfere of the redundancy of the weights when matching the training trajectory. The results shows improvements over previous work on various datasets.

4.2 Related Work

Coreset selection. A relatively straight forward way to reduce the dataset size is to maintain a subset that only contains a few most representative samples selected from the original dataset. A key challenge in achieving a good trade-off between the training performance and subset size lies in determining the importance of each sample. This type of method is known as coreset selection [103, 104, 105, 106].

Dataset condensation. The task of dataset condensation is to learn a small synthetic dataset that retains the knowledge the original dataset. The deep neural network trained on the samll synthetic dataset should obtain similar performance to the network trained on the original dataset. A more strict dataset condensation method also requires that the weights of the network trained on the small synthetic dataset be close to the network trained on the original dataset.

- **Performance matching.** The performance matching methods are designed to ensure that neural networks trained on the condensed dataset perform comparable to the network trained

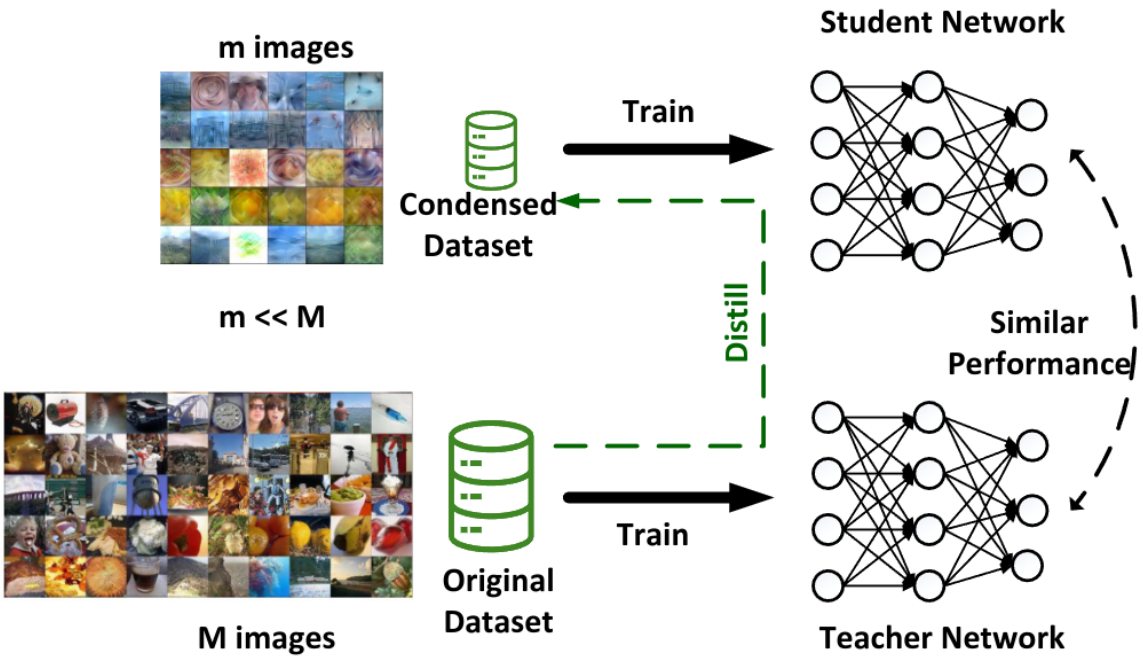


Figure 4.1 An overview of Dataset Condensation. Dataset condensation focuses on generating new, condensed training data such that models trained on such dataset have similar performance to the models trained on the original dataset.

on the original dataset. A simple proof-of-concept algorithm was introduced in [77] where the gradient with respective to the condesed dataset is computed by chaining derivaties backwards through the entire training procedure. In [84] a bilevel optimization technique is used, where the inner loop optimization is training a network on the condensed dataset and outer loop optimize the validation performance of the network trained in the inner loops and backpropage through the unrolled computation graph of inner loop optimization. However the unrolled backpropagation requires to store the entire inner loop training trajectory grows which could easily exceeds the available memory in the hardware. To address the challenge, [78] leverages the Implicit Function Theorem (IFT) in conjection with an efficient Hessian inverse method to approximate the unrolled differentiation which only requires constant amount of memory.

- **Weight space matching.** The methodology of weight space matching was initially introduced

in *et al.*, with subsequent expansions provided through a series of studies [107, 108, 109, 110]. Distinct from performance matching, which aims at optimizing the performance of networks trained on synthetic data, the concept of weight space matching involves training an identical network on both synthetic and original datasets for a predefined number of steps and then aligning the weights of both networks. The early works on weight space matching [88, 107, 108, 111, 112] were focused on matching the trajectory of a single step gradient update. While this strategy is computational efficient, the errors may accumulate when the models are updated by the synthetic datasets for multiple steps. To address this issue, [110] introduced a long range trajectory matching technique that transfers the knowledge from a pre-trained network through multi-step parameter updates.

- **NTK/functional space matching**. The preformance matching and weight space matchign methods with muti-step parameter updates involves unrolling the gradient by traversing back through the entire updating process. It requires higher order gradient calculation that demands considerable computational resources to compute the unrolled gradients. The recent study of neural tangent kernel (NTK) shows that the gradient descent in the infinite-width limit is fully equivalent to kernel gradient descent with the NTK. Inspired by the connections between infinitely-wide neural networks and kernel ridge-regression (KRR), the Kernel Inducing Point (KIP) algorithm is proposed in [86, 87]. It relies on the closed-form solution to linear models to avoid repetitive inner-loop optimization steps that leads to unrolled gradient update. To mitigate the computational complexity associated with calculating the Neural Tangent Kernel [109] introduced RFAD, leveraging the Empirical Neural Network Gaussian Process (NNGP) kernel [113, 114]. To further improve the performance, they adopt platt scaling [115] by applying cross-entropy loss to labels of real data instead of mean square error, which further improves the performance.

- **Feature space matching.** The objective of the feature space matching strategy is to generate synthetic data whose features closely mimics the distribution of real data. Maximum Mean

Discrepancy (MMD) is used in [90] to align the features extracted prior to the final layer between the distilled dataset and the original dataset. Instead of aligning the features prior to the last linear layer, [116] propose CAFE that further improves the performance by ensuring the consistency of features across all intermediate layers.

4.3 Method

In this section, we first provide a brief introduction to the trajectory based condensation method which serves as the background for our method. We then show that the model weights are not equally important which servers as the motivation and mathematical foundation for our proposed method. Finally, we introduce our proposed method named importance aware trajectory matching in details.

4.3.1 Background on Trajectory Matching based Dataset Condensation

Dataset distillation focuses on generating a compact dataset $\mathcal{D}_{\mathsf{syn}}$ from a larger, original dataset $\mathcal{D}_{\mathsf{real}}$, with the goal that models trained on $\mathcal{D}_{\mathsf{syn}}$ exhibit comparable test performance to those trained on $\mathcal{D}_{\mathsf{real}}$.

For methods based on trajectory matching (TM), this process involves aligning the training trajectories of surrogate models trained on both $\mathcal{D}_{\mathsf{real}}$ and $\mathcal{D}\mathsf{syn}$. Specifically, we define τ^* as the expert training trajectories, represented by a sequence of model parameters $\{\theta_t^*\}_0^n$ acquired during the training on $\mathcal{D}_{\mathsf{real}}$. Similarly, $\hat{\theta}t$ represents the parameters at training step t of a network trained on the synthetic dataset $\mathcal{D}_{\mathsf{syn}}$.

During each distillation iteration, θ_t^* and θ_{t+M}^* are randomly chosen from the collection of expert trajectories τ^* as the initial and target parameters for matching, with M being a predetermined hyperparameter. TM-based methods then refine $\mathcal{D}_{\mathsf{syn}}$ by minimizing the loss expressed as:

$$\mathcal{L} = \frac{\|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2}{\|\theta_t^* - \theta_{t+M}^*\|_2^2}, \tag{4.1}$$

where N is a hyper-parameter and $\hat{\theta}t + N$ results from inner optimization using cross-entropy (CE) loss ℓ and a learnable learning rate α:

$$\hat{\theta}_{t+i+1} = \hat{\theta}_{t+i} - \alpha \nabla \ell(\hat{\theta}_{t+i}, \mathcal{D}_{\mathsf{s}yn}), \tag{4.2}$$

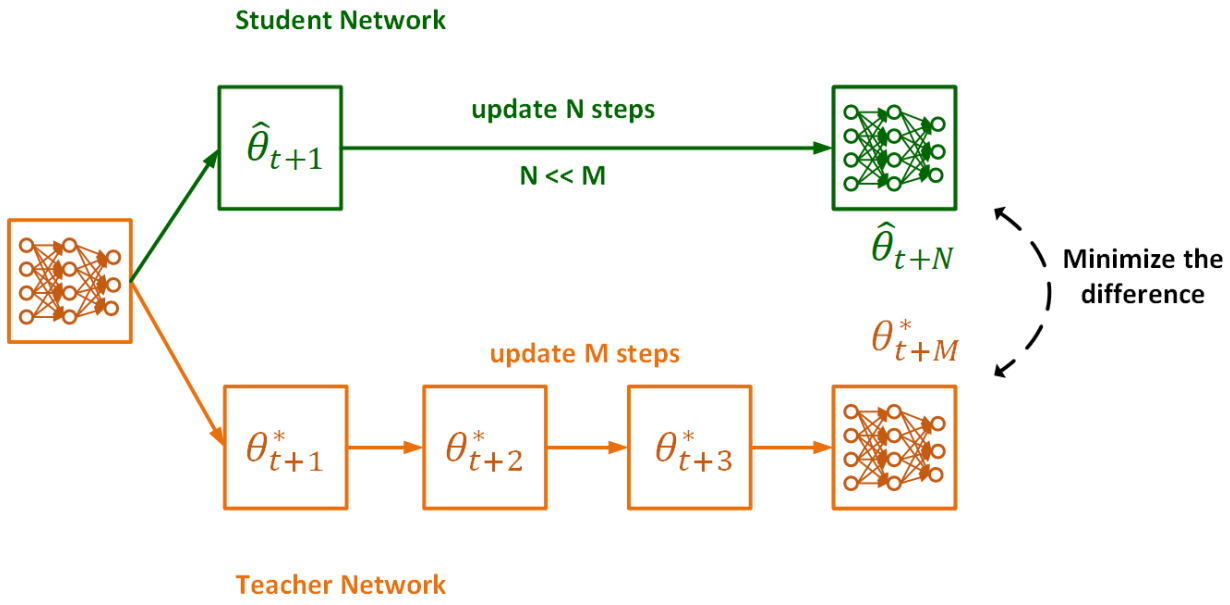


Figure 4.2 Illustration of trajectory matching. The yellow blocks indicates the expert trajectory of the teacher network, while green blocks represent the trajectory of student network. The objective is to minimize the distance between θ^*_{t+M} and $\hat{\theta}_{t+N}$.

where $\hat{\theta}_t := \theta^*_t$.

4.3.2 Importance Aware Weight Factorization

Deep networks with a large number of trainable parameters within each layer are capable achieving remarkable inference accuracy. While a large number of parameters contribute to higher classification precision, it has been discovered that these parameters often exhibit considerable redundancy. This redundancy allows for techniques such as pruning and quantization to be employed in order to decrease the overall size of the network. In this section, we show that the significance of weights within deep networks is not uniform, that is, a subset of these weights plays a far more critical role in determining the ultimate performance of the model than the rest. We hypothesize that matching the most important subset of weighs will lead to good performance, while matching the less important weights would contribute minimally or even negatively to the final performance. Motivated by this, we start our work by exploring the importance of each components of the model weights.

Given a weight matrix $\Theta \in \mathbb{R}^{M \times N}$ with input activation $X \in \mathbb{R}^{N \times B}$, where M, N and B denotes the input dimension, output feature dimension and batch size, respectively. The output pre-activate

is computed as $Y = \Theta X$ which is equivalent to:

$$\begin{aligned} Y &= (\Theta S)(S^{-1}X) \\ &= \tilde{\Theta}\tilde{X} \end{aligned} \tag{4.3}$$

with $\tilde{\Theta} = \Theta S$ and $\tilde{X} = S^{-1}X$ being the transformed weight and activation, and $S \in \mathbb{R}^{N\times N}$ being an arbitrary inevitable matrix.

We construct the matrix S^{-1} to be an whitening matrix of input activate X, which satisfies $SS^T = XX^T$. Thus each channel of the transformed input activations $\tilde{X}$ are independent from each other, i.e., $\tilde{X}\tilde{X}^T = S^{-1}XX^S(S^{-1})^S = I$. To find the optimal subset of weights that influence the final performance most, we decompose the transformed weight $\tilde{\Theta}$ via singular value decomposition as:

$$\begin{aligned} \tilde{\Theta} &= \Theta S \\ &= \tilde{U}\tilde{\Sigma}\tilde{V}^T \\ &= \sum_{n=1}^{r} \tilde{\sigma}_n \tilde{u}_n \tilde{v}_n^T \end{aligned} \tag{4.4}$$

where $\tilde{U}$, $\tilde{V}$, $\tilde{\Sigma}$ are the left and right singular vectors and singular values. Denoting $\tilde{U} = [\tilde{u_1}, \tilde{u_2}, \tilde{u_3}, ..., \tilde{u_r}]$, $\tilde{\Sigma} = \text{diag}(\tilde{\sigma_1}, \tilde{\sigma_2}, \tilde{\sigma_3}, \cdots, \tilde{\sigma_r})$, and $\tilde{V} = [\tilde{v_1}, \tilde{v_2}, \tilde{v_3}, ..., \tilde{v_r}]$ where $r = \min\{M, N\}$ supposing the weight Θ has full rank.

In the following, we provide the theoretical proofs that are useful to determine the importance of the weight components.

Lemma 1. *The Frobenius norm of matrix A with dimension $m \times n$ can be deduced into the square root of the trace of its gram matrix, which is:*

$$\|A\|_F \triangleq \left(\sum_{j=1}^{n} \sum_{i=1}^{m} \left|a_{ij}\right|^2 \right)^{\frac{1}{2}} = \left[\text{trace}\left(A^T A\right) \right]^{\frac{1}{2}} \tag{4.5}$$

Using lemma 1, we obtain the loss L_i when removing the i^{th} singular value and singular vectors

of $\tilde{W}$:

$$L_i = ||(\tilde{\Theta} - \sum_{n \neq i} \tilde{\sigma}_n \tilde{u}_n \tilde{v}_n^T)\tilde{X}||_F \tag{4.6}$$
$$= ||\tilde{\sigma}_i \tilde{u}_i \tilde{v}_i^T \tilde{X}||_F = \tilde{\sigma}_i \, trace(\tilde{u}_i \tilde{v}_i^T \tilde{X} \tilde{X}^T \tilde{v}_i \tilde{u}_i^T)^{\frac{1}{2}}$$

Since both $\tilde{U} = [\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, ..., \tilde{u}_r]$ and $\tilde{V} = [\tilde{v}_1, \tilde{v}_2, \tilde{v}_3, ..., \tilde{v}_r]$ are orthogonal matrices, we have:

$$\begin{cases} \tilde{v}_i^T \tilde{v}_i = \tilde{u}_i^T \tilde{u}_i = I \\ \tilde{v}_i^T \tilde{v}_j = \tilde{u}_i^T \tilde{u}_j = 0, \forall i \neq j \\ trace(\tilde{v}_i \tilde{v}_i^T) = trace(\tilde{u}_i \tilde{u}_i^T) = 1 \end{cases} \tag{4.7}$$

Theorem 1. *If the whitening matrix S satisfies $SS^T = XX^T$, the compression loss L_i equals to $\tilde{\sigma}_i$.*

Proof. Since $\tilde{X} = S^{-1}X$ and $SS^T = XX^T$, we can further L_i:

$$L_i = ||(\tilde{\Theta} - \sum_{n \neq i} \tilde{\sigma}_n \tilde{u}_n \tilde{v}_n^T)\tilde{X}||_F \tag{4.8}$$
$$= ||\tilde{\sigma}_i \tilde{u}_i \tilde{v}_i^T \tilde{X}||_F$$
$$= \tilde{\sigma}_i \, trace(\tilde{u}_i \tilde{v}_i^T \tilde{X} \tilde{X}^T \tilde{v}_i \tilde{u}_i^T)^{\frac{1}{2}}$$
$$= \tilde{\sigma}_i \, trace(\tilde{u}_i \tilde{v}_i^T S^{-1} X X^T (S^{-1})^T \tilde{v}_i \tilde{u}_i^T)^{\frac{1}{2}}$$
$$= \tilde{\sigma}_i \, trace(\tilde{u}_i \tilde{v}_i^T S^{-1} S S^T (S^T)^{-1} \tilde{v}_i \tilde{u}_i^T)^{\frac{1}{2}}$$
$$= \tilde{\sigma}_i \, trace(\tilde{u}_i \tilde{v}_i^T \tilde{v}_i \tilde{u}_i^T)^{\frac{1}{2}} = \tilde{\sigma}_i$$

We can find such S using the Cholesky decomposition of XX^T, which satisfies $SS^T = XX^T$, the loss L_i of removing $\tilde{\sigma}_i$ equals to the singular value $\tilde{\sigma}_i$ itself. □

Theorem 2. *If the whitening matrix S satisfies $SS^T = XX^T$, removing two components corresponding to the singular values $\tilde{\sigma}_i$ and $\tilde{\sigma}_j$, the squared loss $L_{i,j}^2$ is the summation of $\tilde{\sigma}_i^2$ and $\tilde{\sigma}_j^2$*

Proof. Suppose we remove $\tilde{\sigma}_i$ and $\tilde{\sigma}_j$ from the SVD decompositoin of $\tilde{\Theta}$, we calculate the square of the loss $L_{i,j}$:

$$
\begin{aligned}
L_{i,j}^2 =& ||(\tilde{\Theta} - \sum_{n \notin \{i,j\}} \tilde{\sigma}_n \tilde{u}_n \tilde{v}_n^T)\tilde{X}||_F \\
=& \left\| \tilde{\sigma}_i \tilde{u}_i \tilde{v}_i^T \tilde{X} + \tilde{\sigma}_j \tilde{u}_h \tilde{v}_h^T \tilde{X} \right\|_F^2 \\
=& \tilde{\sigma}_i^2 \; trace(\tilde{u}_i \tilde{v}_i{}^T \tilde{X} \tilde{X}^T \tilde{v}_i \tilde{u}_i{}^T) + \tilde{\sigma}_i \tilde{\sigma}_j \; trace(\tilde{u}_i \tilde{v}_i{}^T \tilde{X} \tilde{X}^T \tilde{v_j} \tilde{u_j}{}^T) \\
& + \tilde{\sigma}_j^2 \; trace(\tilde{u_j} \tilde{v_j}{}^T \tilde{X} \tilde{X}^T \tilde{v}_j \tilde{u_j}{}^T) \\
=& \tilde{\sigma}_i^2 \; trace(\tilde{u}_i \tilde{v}_i{}^T \tilde{X} \tilde{X}^T \tilde{v}_i \tilde{u}_i{}^T) + \tilde{\sigma}_j^2 \; trace(\tilde{u_j} \tilde{v_j}{}^T \tilde{X} \tilde{X}^T \tilde{v}_j \tilde{u_j}{}^T) \\
=& (L_i)^2 + (L_j)^2 \\
=& \tilde{\sigma}_i^2 + \tilde{\sigma}_j^2
\end{aligned}
\tag{4.9}
$$

The squared loss L^2 is equal to the sum of the squared singular values. □

Combining theorem 1 and theorem 2, we can conclude that the value $\tilde{\sigma}_i$ of the transformed weight $\tilde{\Theta} = \Theta S = \sum_{n=1}^{r} \tilde{\sigma}_n \tilde{u}_n \tilde{v}_n^T$ can be used to rank the importance of the SVD component $\tilde{\sigma}_i \tilde{u}_i \tilde{v}_i^T$.

4.3.3 Importance Aware Trajectory Matching

Observing that the network tends to learn easy patterns early in training, then the harder ones later on. The key of our proposed method is to match the few important weight component from the expert trajectory $\tau^* = \{\theta_0^*, \cdots, \theta_t^*, \cdots, \theta_T^*\}$ in the early training stage, then gradually adding more weight components to the matching target when t grows large. In this section θ denotes the vectorized version of the weight matrix Θ, i.e., $\theta = \mathsf{vec}(\Theta)$. With a bit of notation abuse, we use $\theta \cdot S$ to denote the vectorized version of ΘS, that is, $\theta \cdot S = \mathsf{vec}(\Theta S)$.

We rank the singular values in Eqn. 4.4 in descending order $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \tilde{\sigma}_3 \geq \cdots \geq \tilde{\sigma}_r$. For epoch t in the expert trajectory τ^*, we set a threshold $\tau(t) = \sqrt{t/T \sum_{n=1}^{r} \tilde{\sigma}_r^2}$ to truncate the singular values, where T is the total number of epochs in the expert trajectory. We have the truncated weight

$$
\mathsf{Trunc}(\theta \cdot S, t) = \mathsf{vec}\left(\sum_{i=1}^{k(t)} \tilde{\sigma}_i \tilde{u}_i \tilde{v}_i^T \right) \tag{4.10}
$$

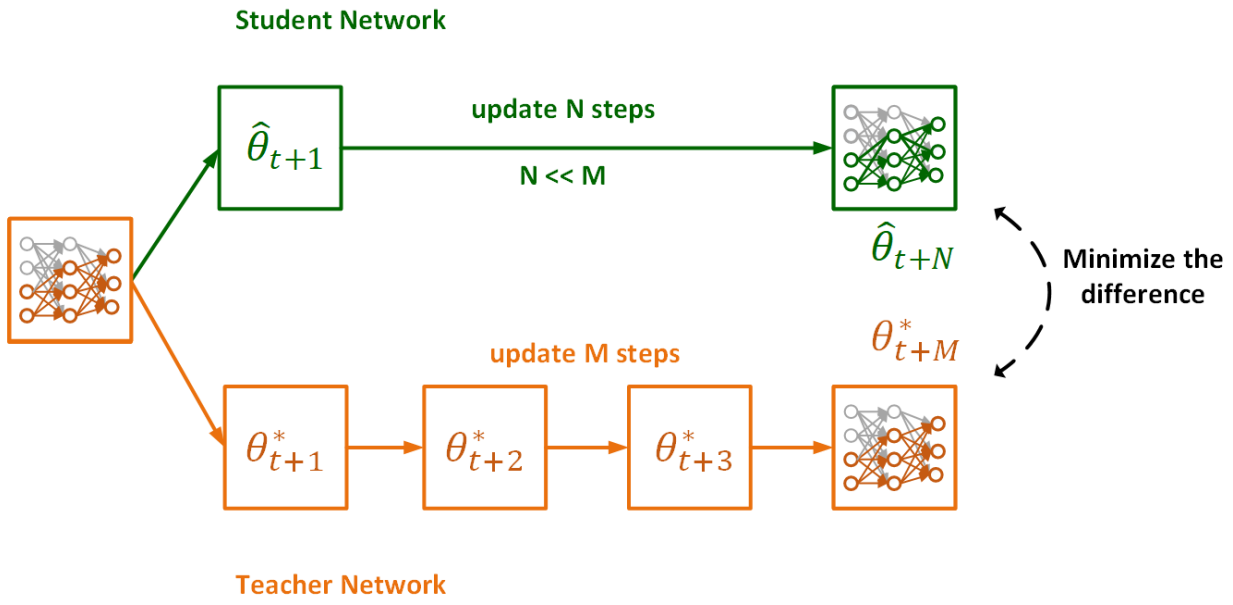


Figure 4.3 Illustration of importance aware trajectory matching. Illustration of trajectory matching. The yellow blocks indicates the expert trajectory of the teacher network, while green blocks represent the trajectory of student network. The gray nodes in the neural network indicates the corresponding weight is truncated by the importance aware factorization. The objective is to minimize the distance between truncated version of θ^*_{t+M} and $\hat{\theta}_{t+N}$.

where $k(t)$ satisfy

$$\begin{cases} \sum_{i=1}^{k(t)} \tilde{\sigma}_i^2 \leq \tau(t)^2 \\ \sum_{i=1}^{k(t)+1} \tilde{\sigma}_i^2 \geq \tau(t)^2 \end{cases} \tag{4.11}$$

Instead of matching the raw weight θ^*_{t+M} in the expert trajectory τ^*, we seek only matching the truncated weight $\mathsf{Trunc}(\theta^*_{t+M} \cdot S, t)$ instead. The trajectory matching target in Eqn.4.1 is modified as:

$$\mathcal{L} = \frac{\|\hat{\theta}_{t+N} \cdot S - \mathsf{Trunc}(\theta^*_{t+M}, t)\|_2^2}{\|\mathsf{Trunc}(\theta^*_t, t) - \mathsf{Trunc}(\theta^*_{t+M}, t)\|_2^2} . \tag{4.12}$$

which ensures that when $t \ll T$, only the most important components of weight is matched, and when $t \to T$ all the weight components are matched. The full algorithm is illustrated in Algorithm. 4.1.

Algorithm 4.1 Importance Aware Trajectory Matching

Require: $\{\tau_i^*\}$: set of expert parameter trajectories trained on $\mathcal{D}_{\text{real}}$.
Require: M: # of updates between starting and target expert params.
Require: N: # of updates to student network per distillation step.
Require: $T^+ < T$: Maximum start epoch.

Initialize distilled data $\mathcal{D}_{\text{syn}} \sim \mathcal{D}_{\text{real}}$
Initialize trainable learning rate $\alpha := \alpha_0$ for apply $\mathcal{D}_{\text{syn}}$
for each distillation step... **do**
    ▷ Sample expert trajectory: $\tau^* \sim \{\tau_i^*\}$ with $\tau^* = \{\theta_t^*\}_0^T$
    ▷ Choose random start epoch, $t \leq T^+$
    ▷ Initialize student network with expert params:
        $\hat{\theta}_t := \theta_t^*$
    for $n = 0 \rightarrow N - 1$ **do**
        ▷ Sample a mini-batch of distilled images:
            $b_{t+n} \sim \mathcal{D}_{\text{syn}}$
        ▷ Update student network w.r.t. classification loss:
            $\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha \nabla \ell(\mathcal{A}(b_{t+n}); \hat{\theta}_{t+n})$
    end for
    ▷ Gather the input features X, compute S:
        $S = \mathsf{cholesky}(XX^T)$
    ▷ Compute loss between ending student and expert params:
        $\mathcal{L} = \dfrac{\|\hat{\theta}_{t+N} \cdot S - \mathsf{Trunc}(\theta^*_{t+M}, t)\|_2^2}{\|\mathsf{Trunc}(\theta_t^*, t) - \mathsf{Trunc}(\theta^*_{t+M}, t)\|_2^2}$
    ▷ Update $\mathcal{D}_{\text{syn}}$ and α with respect to $\mathcal{L}$
end for

Ensure: distilled data $\mathcal{D}_{\text{syn}}$ and learning rate α

4.4 Experiments

4.4.1 Experiments Setup

Datasets. We evaluate the performance of our proposed method on varous datasets, including

- CIFAR-10 dataset which consists of 60000 32×32 images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- CIFAR-100 dataset which consists of 60000 32×32 images in 100 classes, with 600 images per class. There are 50000 training images and 10000 test images.
- Tiny ImageNet dataset which consists of 100,000 64×64 images in 200 classes. For each class, there are 500 training images, 50 validating images, and 50 test images.

Architectures. For a fair comparison, we stay consistent with previous works [117, 118, 119, 87]. The expert and student networks utilize a simple ConvNet architecture introduced in [120], where for CIFAR-10/100 a 3-layer ConvNet is employed and for Tiny ImageNet a depth-4 ConvNet is employed. We use networks with instance normalization by default. We also useAlexNet [15], VGG11 [16], and ResNet18 [14] for cross-architecture experiments.

Baseline methods. We compare our method to recent work on dataset condensation as well as several work on coreset selection.

- **Dataset Condensation:** Dataset Distillatio [121] (`DD`), Flexible Dataset Distillation [122] (`LD`), Dataset Condensation [117] (`DC`), and Differentiable Siamese Augmentation [118] (`DSA`) Matching Training Trajectories [110] (`MTT`), and neural tangent kernel based methods [86, 87] (`KIP`), and feature matching based methods Distribution Matching [119] (`DM`) and Aligning Features [116] (`CAFE`).

- **Coreset selection:** random selection (`random`), herding methods [123] (`herding`), and example forgetting [124] (`forgetting`).

4.4.2 Performance on CIFAR-10 and CIFAR-100

We first evaluate our proposed method on the low resolution datasets (32×32), CIFAR-10 and CIFAR-100. Consistent with prior work, we use a 3-layer ConvNet for both distilation and evaluation. We also employ ZCA whitening [125] on the training dataset as was done previous work [86, 87]. Table 4.1 reports the Top-1 test accuracy on CIFAR-10/100 for ConvNet trained on the condensed dataset with given number of images per class (IPC). We also show the the standard deviation and the mean accuracy. As show, our method achieves the best performance except on CIFAR-10 with IPC=50. The rest of hyperparameters in this expreiemnt can be found in Table 4.3.

4.4.3 Performance on Tiny ImageNet

We further evaluate our proposed method on the higher resolution datasets (64×64), Tiny ImageNet. To account for the large resolution, we use a 4-layer ConvNet for both distilation and evaluation. We do not apply ZCA whitening [125] on the training dataset. Table 4.1 reports

Dataset	CIFAR-10			CIFAR-100		
IPC	1	10	50	1	10	50
Random	14.4 ± 2.0	26.0 ± 1.2	43.4 ± 1.0	4.2 ± 0.3	14.6 ± 0.5	30.0 ± 0.4
Herding	21.5 ± 1.2	31.6 ± 0.7	40.4 ± 0.6	8.4 ± 0.3	17.3 ± 0.3	33.7 ± 0.5
Forgetting	13.5 ± 1.2	23.3 ± 1.0	23.3 ± 1.1	4.5 ± 0.2	15.1 ± 0.3	30.5 ± 0.3
DD	-	36.8 ± 1.2	-	-	-	-
LD	25.7 ± 0.7	38.8 ± 0.4	42.5 ± 0.4	11.5 ± 0.4	-	-
DC	28.3 ± 0.5	44.9 ± 0.5	53.9 ± 0.5	12.8 ± 0.3	25.2 ± 0.3	-
DSA	28.8 ± 0.7	52.1 ± 0.5	60.6 ± 0.5	13.9 ± 0.3	32.3 ± 0.3	42.8 ± 0.4
DM	26.0 ± 0.8	48.9 ± 0.6	63.0 ± 0.4	11.4 ± 0.3	29.7 ± 0.3	43.6 ± 0.4
CAFE	30.3 ± 1.1	40.6 ± 0.6	55.5 ± 0.6	12.9 ± 0.3	27.9 ± 0.3	37.9 ± 0.3
CAFE+DSA	31.6 ± 0.8	50.9 ± 0.5	62.3 ± 0.4	14.0 ± 0.3	31.5 ± 0.2	42.9 ± 0.2
MTT	43.6 ± 0.8	65.3 ± 0.7	**71.6** ± 0.2	24.3 ± 0.3	40.1 ± 0.4	47.7 ± 0.2
Ours	**46.5** ± 0.6	**65.9** ± 0.6	71.2 ± 0.4	**25.2** ± 0.4	**40.9** ± 0.3	**48.4** ± 0.3

Table 4.1 Top-1 test accuracy on CIFAR-10/100 compared with previous work on coreset section and dataset condensation. Consistent with prior work, we use a 3-layer ConvNet for both distilation and evaluation. The results are averaged over four independent runs with different initializations. The standard deviation is denoted by ±.

the Top-1 test accuracy on Tiny ImageNet for ConvNet trained on the condensed dataset with given number of images per class (IPC). We also show the the standard deviation and the mean accuracy. As show, our method achieves significant better performance on Tiny ImageNet with different IPCs. Note that may dataset condensation algorithms that are effective for CIFAR-10/100 are unable to work on Tiny ImageNet dataset due to their high memory and computation resource requirement. Thus we only include DM and MTT in this experiment. The rest of hyperparameters in this expreiemnt can be found in Table 4.3.

4.4.4 Cross Architecture Generalization

Since the distilled dataset is condensed using a simple ConvNet architecture where we match the training trajectory between the teacher and student networks of the same architecture. In this experiment, we evaluate the performance of the condensed dataset on unseen neural architectures. We evaluate the performance on three architectures AlexNet [15], VGG11 [16], and ResNet18 [14] together with the 3-layer ConvNet architecture used for dataset condensation. We synthesis the condensed dataset of CIFAR-100 with IPC=50 and hyperparameters shown in Table 4.3. The resulting Top-1 test accuracy is reported in Table 4.4. Despite the data being distill only from the

Dataset	Tiny ImageNet		
IPC	1	10	50
Random	1.4 ± 0.1	5.0 ± 0.2	1.5 ± 0.4
Herding	2.8 ± 0.2	6.3 ± 0.2	16.7 ± 0.3
Forgetting	1.6 ± 0.1	5.1 ± 0.2	15.0 ± 0.3
DM	3.9 ± 0.2	12.9 ± 0.4	24.1 ± 0.3
MTT	8.8 ± 0.3	23.2 ± 0.2	28.0 ± 0.3
Ours	**12.0** ± 0.3	**27.3** ± 0.3	**33.1** ± 0.2

Table 4.2 Top-1 test accuracy on Tiny ImageNet compared with previous work on coreset section and dataset condensation. For Tiny ImageNet, we use a 4-layer ConvNet for both distillation and evaluation, since it has a larger resolution. The results are averaged over four independent runs with different initializations. The standard deviation is denoted by ±.

Dataset	IPC	N	M	T^-	T	T^+	Interval	Synthetic Batch Size	Learning Rate
CIFAR-10	1	80	2	0	4	4	-	10	100
	10	80	2	0	10	20	100	100	100
	50	80	2	0	20	40	100	500	1000
CIFAR-100	1	40	3	0	10	20	100	100	1000
	10	80	2	0	30	50	100	1000	1000
	50	80	2	20	70	70	-	1000	1000
Tiny ImageNet	1	60	2	0	15	20	400	200	10000
	10	60	2	10	50	50	-	250	100
	50	80	2	40	70	70	-	250	100

Table 4.3 Hyper-parameters for different datasets.

trajectory of 3-layer ConvNet, our synthetic dataset performs best on the other three unseen neural architecture. This indicates that the distilled dataset does not suffer from over-fitting on a particular model.

Method	ConvNet	ResNet18	VGG	AlexNet
Random	30.0	31.9	32.2	26.7
MTT	47.7	42.6	41.2	40.3
ours	**48.4**	**46.3**	**45.1**	**45.4**

Table 4.4 Top-1 test accuracy evaluated on CIFAR-100 with 50 images per class on different architectures.

4.4.5 Visualize the condensed dataset

Figures 4.4 - 4.5 and Figures 4.6 - 4.7 demonstrate the condensed dataset of Tiny ImageNet with IPC=1 and IPC=50 respectively. In the low IPC case (IPC=1), we see that the generated patters are a superposition of many images. This indicates that the algorithm is trying to distill as many patterns as possible into the condensed dataset. However, for the high IPC case (IPC=50), the distilled images have sharp edges and are very close to the images in the original dataset.

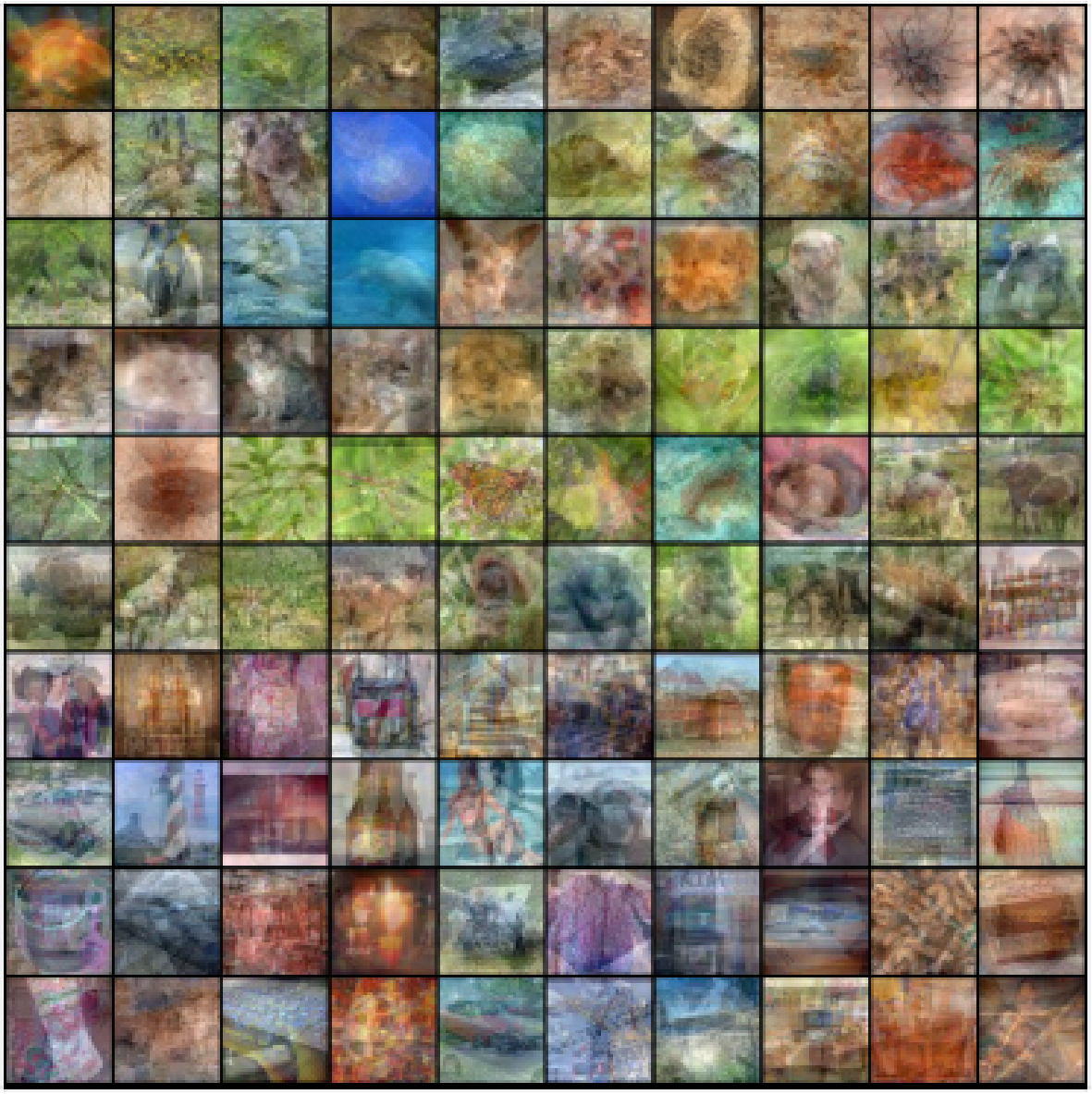


Figure 4.4 Visualization of the distilled Tiny ImageNet dataset with IPC=1 (1/2).

Figure 4.5 Visualization of the distilled Tiny ImageNet dataset with IPC=1 (2/2).

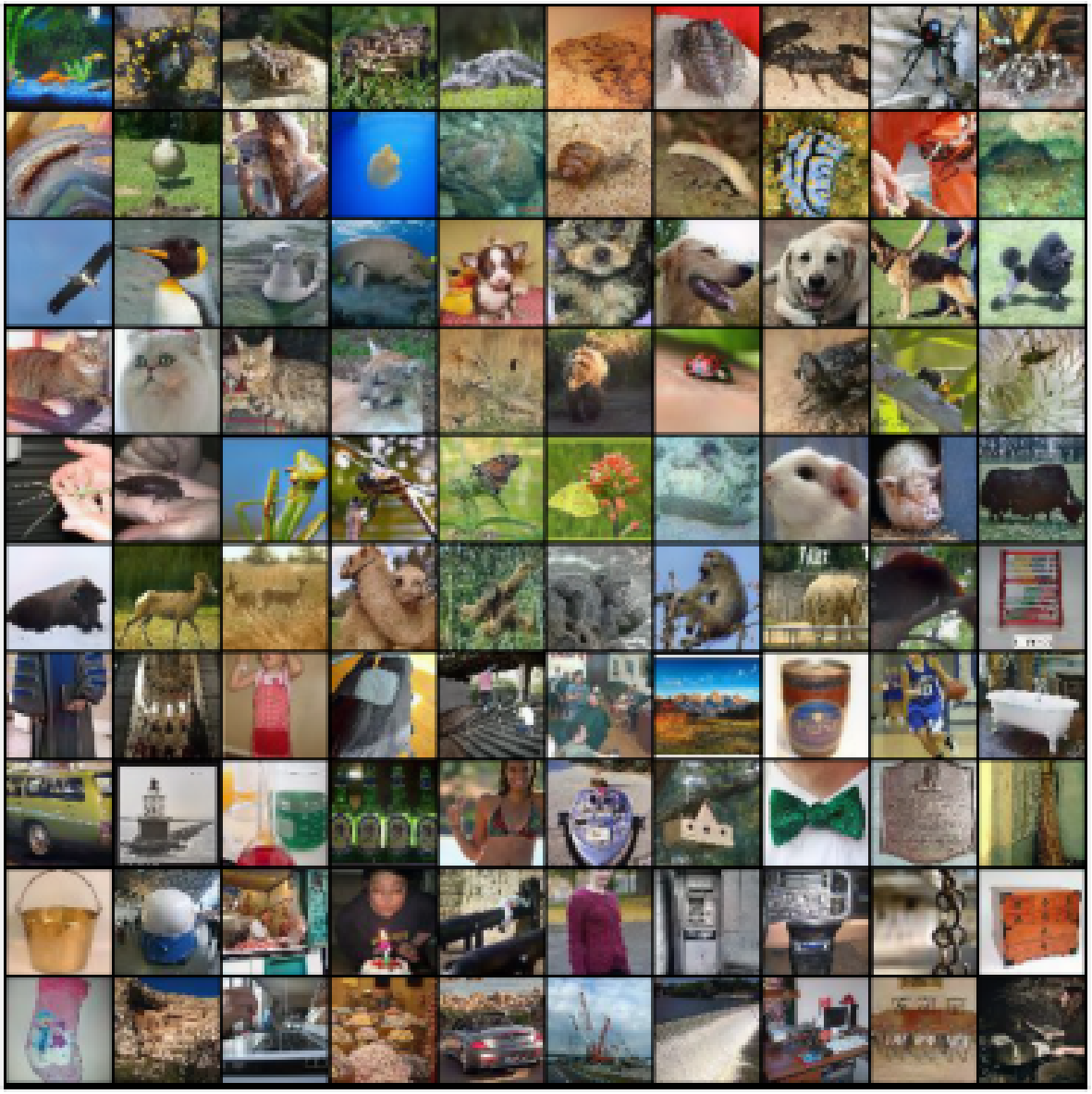


Figure 4.6 Visualization of the distilled Tiny ImageNet dataset with IPC=50 (1/2).

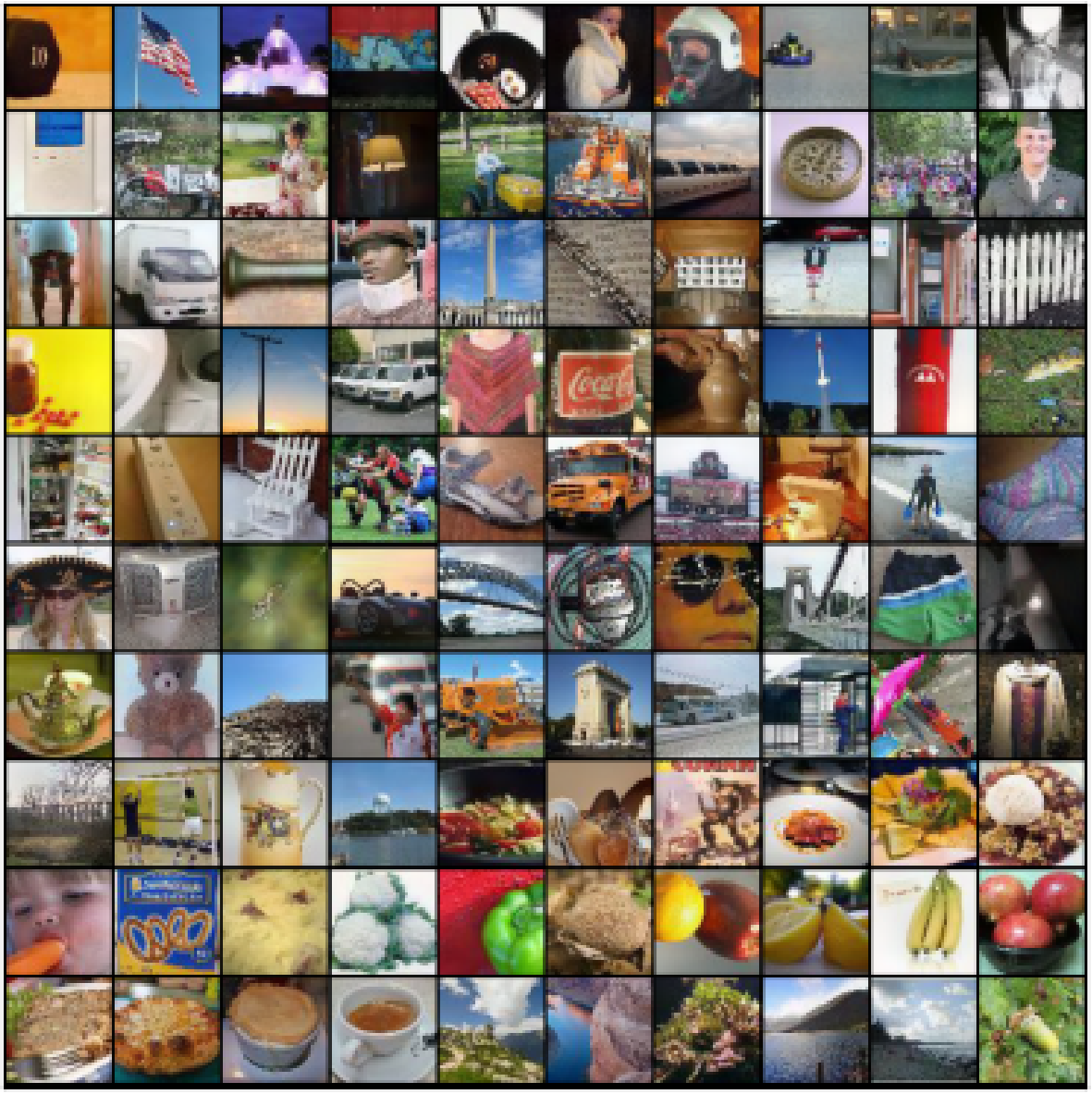


Figure 4.7 Visualization of the distilled Tiny ImageNet dataset with IPC=50 (2/2).

4.5 Conclusion

Neural networks are typically over parameterized with a lot of redundancy, so the weights are not born equally important. Based on this insight, in this chapter we first proposed an importance aware weight factorization, which factorize the weight matrix based on its influence on the layer output. Building on that we proposed an importance aware trajectory matching algorithm which only match the most important weights early in the training trajectory and gradually adding more weight components until reaching the full weights later in training trajectory. Our methods removes the interfere of the redundancy of the weights when matching the training trajectory. The results show improvements over previous work on CIFAR and Tiny ImageNet datasets.

CHAPTER 5

CONCLUSION

Observing that the significant progress of deep learning models in recent years can be primarily attributed to the growth of the model scale and the volume of data on which it was trained. In this dissertation, my goal is to study efficient architecture and data manipulation techniques for deep learning systems.

Chapter 2 - *MSUNet* deals with the problem of efficient architecture. In this chapter, I propose MSUNet, which is designed with three key techniques: 1) ternary convolution layers, 2) sparse squeeze-excitation layers, and 3) self-supervised consistency regularizer along with mixed precision training. Our framework shows a great improvement in parameter size and computational cost measure by #Parameters and #Flops over the baseline model. Lastly, we show that ternary quantization outperforms binary quantization in all aspects, including accuracy, parameter size and computation cost.

Chapter 3 - *Deep AutoAugment* deals with the problem of efficient data manipulation for deep learning systems. In particular, I focused on automated data augmentation. In this chapter, I present Deep AutoAugment (DeepAA), a multi-layer data augmentation search method that finds deep data augmentation policy without using any hand-picked default transformations. We formulate data augmentation search as a regularized gradient matching problem, which maximizes the gradient similarity between augmented data and original data along the direction with low variance. Our experimental results show that DeepAA achieves strong performance without using default augmentations, indicating that regularized gradient matching is an effective searching method for data augmentation policies.

Chapter 4 - *Dataset Condensation via Importance Aware Trajectory Matching* deals with the problem of efficient data manipulation. Different from Chapter 3, we focus on distilling a large dataset into a condensed dataset. By observing that neural networks are typically over parameterized with a lot of redundancy, so the weights are not born equally important. Based on this, we fist proposed an importance aware weight factorization, which factorize the weight matrix based on

its influence on the layer output. Building on it we proposed an importance aware trajectory matching algorithm which only match the most important weights early in the training trajectory and gradually adding more weight components until reaching the full weights later in training trajectory. Our methods removes the interfere of the redundancy of the weights when matching the training trajectory. The results show improvements over previous work on CIFAR and Tiny ImageNet datasets.

BIBLIOGRAPHY

[1] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123.

[2] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[4] L. Lathauwer, "Decompositions of a higher-order tensor in block terms—part ii: Definitions and uniqueness," *SIAM J. Matrix Analysis Applications*, vol. 30, pp. 1033–1066, 01 2008.

[5] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, D. Song, and M. Zhou, "A tensorized transformer for language modeling," *CoRR*, vol. abs/1906.09777, 2019. [Online]. Available: http://arxiv.org/abs/1906.09777

[6] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016. [Online]. Available: https://arxiv.org/abs/1510.00149

[8] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800. [Online]. Available: https://arxiv.org/pdf/1802.03494.pdf

[9] D. R. So, C. Liang, and Q. V. Le, "The evolved transformer," *arXiv preprint arXiv:1901.11117*, 2019. [Online]. Available: https://arxiv.org/abs/1901.11117

[10] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Conference on Computer Vision and Pattern Recognition*, 2020.

[11] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 27–40, 2017. [Online]. Available: https://arxiv.org/abs/1708.04485

[12] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020. [Online]. Available: https://arxiv.org/abs/2002.08947

[13] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 7308–7316.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

[18] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in *CVPR*, 2017.

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

[21] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[22] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[23] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[24] M. Tan and Q. V. Le, "Mixconv: Mixed depthwise convolutional kernels," in *30th British Machine Vision Conference 2019*, 2019.

[25] K. A. vahid, A. Prabhu, A. Farhadi, and M. Rastegari, "Butterfly transform: An efficient fft based neural architecture design," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[26] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, Long Beach, California, USA, 09–15 Jun 2019, pp. 6105–6114.

[27] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[28] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "Addernet: Do we really need multiplications in deep learning?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[29] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[30] D. Zhou, Q.-B. Hou, Y. Chen, J. Feng, and S. Yan, "Rethinking bottleneck structure for efficient mobile network design," in *ECCV*, August 2020.

[31] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. Pirogov, "Mixed Precision Training of Convolutional Neural Networks using Integer Operations," in *International Conference on Learning Representations(ICLR)*, vol. abs/1802.0, 2 2018, pp. 1–11. [Online]. Available: https://www.anandtech.com/show/11741/hot-chips-intel-knights-mill-live-blog-445pm-pt-1145pm-utc http://arxiv.org/abs/1802.00930

[32] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed Precision Training," in *International Conference on Learning Representations(ICLR)*, 10 2017. [Online]. Available: http://arxiv.org/abs/1710.03740

[33] Y. Ma, N. Suda, Y. Cao, J. S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA," *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*, 2016.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 7, no. 3. IEEE, 12 2015, pp. 171–180. [Online]. Available: http://arxiv.org/abs/1512.03385 http://ieeexplore.ieee.org/document/7780459/

[35] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," 2011. [Online]. Available: https://research.google/pubs/pub37631/

[36] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1712.0. IEEE, 6 2018, pp. 2704–2713. [Online]. Available: https://ieeexplore.ieee.org/document/8578384/

[37] S. O. Settle, M. Bollavaram, P. D'Alberto, E. Delaye, O. Fernandez, N. Fraser, A. Ng, A. Sirasao, and M. Wu, "Quantizing Convolutional Neural Networks for Low-Power High-Throughput Inference Engines," *ArXiv preprint*, 5 2018. [Online]. Available: http://arxiv.org/abs/1805.07941

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[39] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural network distiller: A python package for dnn compression research," October 2019. [Online]. Available: https://arxiv.org/abs/1910.12232

[40] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *International Conference on Learning Representations*, 2018.

[41] M. Tan and Q. V. Le, "Mixconv: Mixed depthwise convolutional kernels," *arXiv preprint arXiv:1907.09595*, 2019.

[42] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[43] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *International Conference on Learning Representations*, 2017.

[44] H. Inoue, "Data augmentation by pairing samples for images classification," *arXiv preprint arXiv:1801.02929*, 2018.

[45] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[46] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6023–6032.

[47] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," *International Conference on Learning Representations*, 2020.

[48] S. Yan, H. Song, N. Li, L. Zou, and L. Ren, "Improve unsupervised domain adaptation with mixup training," in *arXiv preprint arXiv: 2001.00677*, 2020.

[49] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 702–703.

[50] D. Ho, E. Liang, X. Chen, I. Stoica, and P. Abbeel, "Population based augmentation: Efficient learning of augmentation policy schedules," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2731–2741.

[51] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, "Fast autoaugment," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[52] R. Hataya, J. Zdenek, K. Yoshizoe, and H. Nakayama, "Faster autoaugment: Learning augmentation strategies using backpropagation," in *European Conference on Computer Vision*. Springer, 2020, pp. 1–16.

[53] Y. Li, G. Hu, Y. Wang, T. Hospedales, N. M. Robertson, and Y. Yang, "Differentiable automatic data augmentation," in *European Conference on Computer Vision*. Springer, 2020, pp. 580–595.

[54] A. Liu, Z. Huang, Z. Huang, and N. Wang, "Direct differentiable augmentation search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 219–12 228.

[55] T. C. LingChen, A. Khonsari, A. Lashkari, M. R. Nazari, J. S. Sambee, and M. A. Nascimento, "Uniformaugment: A search-free probabilistic data augmentation approach," *arXiv preprint arXiv:2003.14348*, 2020.

[56] S. G. Müller and F. Hutter, "Trivialaugment: Tuning-free yet state-of-the-art data augmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 774–782.

[57] X. Zhang, Q. Wang, J. Zhang, and Z. Zhong, "Adversarial autoaugment," in *International Conference on Learning Representations*, 2019.

[58] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017.

[59] M. Berman, H. Jégou, A. Vedaldi, I. Kokkinos, and M. Douze, "Multigrain: a unified image embedding for classes and instances," *arXiv preprint arXiv:1902.05509*, 2019.

[60] E. Hoffer, T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry, "Augment your batch: Improving generalization through instance repetition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8129–8138.

[61] Y. Du, W. M. Czarnecki, S. M. Jayakumar, M. Farajtabar, R. Pascanu, and B. Lakshminarayanan, "Adapting auxiliary losses using gradient similarity," *arXiv preprint arXiv:1812.02224*, 2018.

[62] X. Wang, H. Pham, P. Michel, A. Anastasopoulos, J. Carbonell, and G. Neubig, "Optimizing data usage via differentiable rewards," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9983–9995.

[63] S. Müller, A. Biedenkapp, and F. Hutter, "In-loop meta-learning with gradient-alignment reward," *arXiv preprint arXiv:2102.03275*, 2021.

[64] S. Chen, E. Dobriban, and J. H. Lee, "A group-theoretic framework for data augmentation," *Journal of Machine Learning Research*, vol. 21, no. 245, pp. 1–71, 2020.

[65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[66] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[67] S. Fort, A. Brock, R. Pascanu, S. De, and S. L. Smith, "Drawing multiple augmentation samples per image during training efficiently decreases test error," *arXiv preprint arXiv:2105.13343*, 2021.

[68] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.

[69] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm," vol. 34, 2021.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[71] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[72] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*. PMLR, 2016, pp. 173–182.

[73] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[74] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8748–8763.

[75] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022.

[76] C. Chen, Y. Zhang, J. Fu, X. Liu, and M. Coates, "Bidirectional learning for offline infinite-width model-based optimization," in *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. [Online]. Available: https://openreview.net/forum?id=_j8yVIyp27Q

[77] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International conference on machine learning*. PMLR, 2015, pp. 2113–2122.

[78] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 1540–1552.

[79] F. P. Such, A. Rawal, J. Lehman, K. Stanley, and J. Clune, "Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data," in *ICML*, 2020.

[80] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 367–377.

[81] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[82] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[83] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *CVPR*, 2017.

[84] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *arXiv preprint arXiv:1811.10959*, 2018.

[85] Y. Zhou, E. Nezhadarya, and J. Ba, "Dataset distillation using neural feature regression," in *NeurIPS*, 2022.

[86] T. Nguyen, Z. Chen, and J. Lee, "Dataset meta-learning from kernel ridge-regression," *arXiv preprint arXiv:2011.00050*, 2020.

[87] T. Nguyen, R. Novak, L. Xiao, and J. Lee, "Dataset distillation with infinitely wide convolutional networks," in *NeurIPS*, 2021.

[88] B. Zhao and H. Bilen, "Dataset condensation with differentiable siamese augmentation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 674–12 685.

[89] ——, "Dataset condensation with differentiable siamese augmentation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 674–12 685.

[90] Bo Zhao and Hakan Bilen, "Dataset condensation with distribution matching," *CoRR*, vol. abs/2110.04181, 2021.

[91] Y. Liu, Y. Su, A.-A. Liu, B. Schiele, and Q. Sun, "Mnemonics training: Multi-class incremental learning without forgetting," in *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 245–12 254.

[92] A. Rosasco, A. Carta, A. Cossu, V. Lomonaco, and D. Bacciu, "Distilled replay: Overcoming forgetting through synthetic samples," in *International Workshop on Continual Semi-Supervised Learning*. Springer, 2022, pp. 104–117.

[93] M. Sangermano, A. Carta, A. Cossu, and D. Bacciu, "Sample condensation in online continual learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 01–08.

[94] F. Wiewel and B. Yang, "Condensed composite memory continual learning," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[95] W. Masarczyk and I. Tautkute, "Reducing catastrophic forgetting with learning on synthetic data," in *CVPR Workshop*, 2020.

[96] J. Goetz and A. Tewari, "Federated learning via synthetic data," *arXiv preprint arXiv:2008.04489*, 2020.

[97] Y. Zhou, G. Pu, X. Ma, X. Li, and D. Wu, "Distilled one-shot federated learning," *arXiv preprint arXiv:2009.07999*, 2020.

[98] Y. Xiong, R. Wang, M. Cheng, F. Yu, and C.-J. Hsieh, "Feddm: Iterative distribution matching for communication-efficient federated learning," *arXiv preprint arXiv:2207.09653*, 2022.

[99] R. Song, D. Liu, D. Z. Chen, A. Festag, C. Trinitis, M. Schulz, and A. Knoll, "Federated learning via decentralized dataset distillation in resource-constrained edge environments," *arXiv preprint arXiv:2208.11311*, 2022.

[100] P. Liu, X. Yu, and J. T. Zhou, "Meta knowledge condensation for federated learning," *arXiv preprint arXiv:2209.14851*, 2022.

[101] S. Hu, J. Goetz, K. Malik, H. Zhan, Z. Liu, and Y. Liu, "Fedsynth: Gradient compression via synthetic data in federated learning," *arXiv preprint arXiv:2204.01273*, 2022.

[102] R. Pi, W. Zhang, Y. Xie, J. Gao, X. Wang, S. Kim, and Q. Chen, "Dynafed: Tackling client data heterogeneity with global dynamics," *arXiv preprint arXiv:2211.10878*, 2022.

[103] M. Welling, "Herding dynamical weights to learn," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 1121–1128.

[104] Y. Chen, M. Welling, and A. Smola, "Super-samples from kernel herding," *arXiv preprint arXiv:1203.3472*, 2012.

[105] D. Feldman, M. Faulkner, and A. Krause, "Scalable training of mixture models via coresets," *Advances in neural information processing systems*, vol. 24, 2011.

[106] O. Bachem, M. Lucic, and A. Krause, "Coresets for nonparametric estimation-the case of dp-means," in *International Conference on Machine Learning*. PMLR, 2015, pp. 209–217.

[107] S. Lee, S. Chun, S. Jung, S. Yun, and S. Yoon, "Dataset condensation with contrastive signals," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2022, pp. 12 352–12 364.

[108] Z. Jiang, J. Gu, M. Liu, and D. Z. Pan, "Delving into effective gradient matching for dataset condensation," *arXiv preprint arXiv:2208.00311*, 2022.

[109] N. Loo, R. Hasani, A. Amini, and D. Rus, "Efficient dataset distillation using random feature approximation," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[110] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4750–4759.

[111] J.-H. Kim, J. Kim, S. J. Oh, S. Yun, H. Song, J. Jeong, J.-W. Ha, and H. O. Song, "Dataset condensation via efficient synthetic-data parameterization," *arXiv preprint arXiv:2205.14959*, 2022.

[112] L. Zhang, J. Zhang, B. Lei, S. Mukherjee, X. Pan, B. Zhao, C. Ding, Y. Li, and X. Dongkuan, "Accelerating dataset distillation via model augmentation," *arXiv preprint arXiv:2212.06152*, 2022.

[113] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[114] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep neural networks as gaussian processes," *arXiv preprint arXiv:1711.00165*, 2017.

[115] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[116] K. Wang, B. Zhao, X. Peng, Z. Zhu, S. Yang, S. Wang, G. Huang, H. Bilen, X. Wang, and Y. You, "Cafe: Learning to condense dataset by aligning features," *arXiv preprint arXiv:2203.01531*, 2022.

[117] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," in *ICLR*, 2020.

[118] B. Zhao and H. Bilen, "Dataset condensation with differentiable siamese augmentation," in *ICML*, 2021.

[119] ——, "Dataset condensation with distribution matching," in *WACV*, 2023.

[120] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," in *CVPR*, 2018.

[121] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *arXiv preprint arXiv:1811.10959*, 2018.

[122] O. Bohdal, Y. Yang, and T. Hospedales, "Flexible dataset distillation: Learn labels instead of images," in *NeurIPS Workshop*, 2020.

[123] Y. Chen, M. Welling, and A. Smola, "Super-samples from kernel herding," in *UAI*, 2010.

[124] M. Toneva, A. Sordoni, R. T. des Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An empirical study of example forgetting during deep neural network learning," in *ICLR*, 2018.

[125] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski, "Kornia: an open source differentiable computer vision library for pytorch," in *WACV*, 2020. [Online]. Available: https://arxiv.org/pdf/1910.02190.pdf