Toward Efficient Deep Learning for Computer Vision Applications

Xiangyu Chen

Submitted to the graduate degree program in Department of Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

	,
	,
	,
Committee members	,
	,
	,
	,

Date defended: April 22, 2024

The Dissertation Committee for Xiangyu Chen certifies
that this is the approved version of the following dissertation:

Toward Efficient Deep Learning for Computer Vision Applications

____
,

Date approved: April 22, 2024

Abstract

Deep learning leads the performance in many areas of computer vision. However, after a decade of research, it tends to require larger datasets and more complex models, leading to heightened resource consumption across all fronts. Regrettably, meeting these requirements proves challenging in many real-life scenarios. First, both data collection and labeling processes entail substantial labor and time investments. This challenge becomes especially pronounced in domains such as medicine, where identifying rare diseases demands meticulous data curation. Secondly, the large size of state-of-the-art models, such as ViT, Stable Diffusion, and ConvNext, hinders their deployment on resource-constrained platforms like mobile devices. Research indicates pervasive redundancies within current neural network structures, exacerbating the issue. Lastly, even with ample datasets and optimized models, the time required for training and inference remains prohibitive in certain contexts. Consequently, there is a burgeoning interest among researchers in exploring avenues for efficient artificial intelligence.

This study endeavors to delve into various facets of efficiency within computer vision, including data efficiency, model efficiency, as well as training and inference efficiency. The data efficiency is improved from the perspective of increasing information brought by given image inputs and reducing redundancies of RGB image formats. To achieve this, we propose to integrate both spatial and frequency representations to finetune the classifier. Additionally, we propose explicitly increasing the input information density in the frequency domain by deleting unimportant frequency channels. For model efficiency, we scrutinize the redundancies present in widely used vision transformers. Our investigation reveals that trivial attention in their attention modules covers useful non-trivial attention due to its large amount. We propose mitigating the impact of accumulated trivial attention weights. To increase training efficiency, we propose SuperLoRA, a generation of LoRA adapter, to fine-tune pretrained models with few iterations and extremely-low parameters. Finally, a model simplification pipeline is proposed to further reduce inference time on mobile devices. By addressing these challenges, we aim to advance the practicality and performance of computer vision systems in real-world applications.

Acknowledgements

Many people make this great journey possible for me and I want to express my sincere appreciation to them. To begin with, Prof. Guanghui Wang offered me this valuable opportunity. His thoughtfulness, generousness, support and encouragement provided me with much freedom, space and confidence to explore my interests deeply. I also want to thank my co-advisor, Prof. Cuncong Zhong, for taking care of my progress carefully and timely.

In the early times, I was new to a totally different country and environment, full of confusion, frustration and stress. I want to thank Prof. Andrew B. Williams for caring about me and helping me to finish my first publication by coming up with ideas, reading related papers, conducting experiments and writing the paper. This paper helps to build my confidence.

Besides this, I also want to thank Prof. John Gibbons and Prof. Hongguo Xu, from whom I know what good teachers are like in classes. To know your stuff and explain it to others patiently. This also applies to scenarios whenever I want to share my thoughts with others, either by papers or by teaching.

Additionally, I appreciate my committee members, Prof. Bo Luo, Prof. Fengjun Li, and Prof. Prasad Kulkarni for all the cooperation, valuable suggestions and insightful discussions.

I would like to thank my ex-colleagues and also co-authors, Dr. Ruiwen Zhen, Dr. Shuai Li and Dr. Xiaotian Li from SenseBrain, Dr. Toshiaki Koike-Akino, Dr. Ye Wang, Dr. Jing Liu, Dr. Perry Wang and Dr. Matthew Brand from MERL, for setting good researcher role model to me. Thanks for letting me know what my next step will be like.

I would like to thank my friends and co-authors, Prof. Ying Qin and Prof. Qinghao Hu for their support and advice from research to life, from when I decided to pursue a Ph.D. degree to the present.

I want to thank my lab mates, Kaidong Li, Tianxiao Zhang, Wenju Xu, Xi Mo, Yiju Yang, Usman Sajad and Krushi, for their valuable discussions and generous help.

I would like to thank my dear friends, for supporting me all the time, taking care of my parents, caring for me, and accompanying me remotely. I feel so happy whenever I think of that I still have many years to go with you. I definitely share every moment with you.

I want to express my deepest gratitude to my parents, for being my parents and for simply being who you are. You are a constant source of pride for me, and I love you so much.

Lastly, I would like to acknowledge my own efforts for having the courage to take this crucial step toward pursuing my dreams. I have persevered, believed in myself, and remained true to who I am. For me, research is not just about exploring a specific topic; it is also a journey of self-discovery and self-love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Data Efficient Learning

1.1 Background

With the explosion in the amount of data we generate and are able to access today, deep learning, as a data-driven method, has become the hotspot and achieved significant performance in many computer vision tasks [62, 12, 63, 118, 146]. Deep learning is a branch of machine learning algorithms that feeds data to models and then optimizes the models with some loss functions to estimate patterns of the data, where the models are deep neural networks. Deep neural networks refer to networks with more than three layers, up to 152 layers in [62]. With more data, the feature extraction ability of deep neural networks is extraordinary on large models with many parameters by accepting only unstructured data like text and image pixels to output high-level features for tasks, like classification and regression. And this benefits many fields where large data is accessible like general image classification and segmentation, and both computation resources and speed are not restricted. However, meeting all these requirements is either difficult or unnecessary.

Firstly, collecting so much data to train a good large model is not always realistic and is expensive. For example, identify the dangerous event when a child is playing in the middle of a road with cars passing by. This is a vital ability for autonomous vehicles so that they can recognize it and correspond correctly and immediately to avoid accidents. However, collecting enough data to train a deep learning model in this case is not realistic since this is a rare event in our daily lives. Besides this, both collecting and labeling data can be expensive considering both time and money costs. More importantly, research shows humans can recognize new categories with even a single example [150], which is apparently in the opposite direction of current machine learning algorithms.

In addition to data efficiency, using large models is not always feasible. To bring the power of AI into our daily lives, deep neural network models are normally deployed on edge devices, like smartphones. However, most of those devices have limited memory and computation resources, making it impossible to afford the large amount of computation of deep neural networks. Additionally, researchers found that there is much noise included in large models.

Along with large datasets and large models is much training and inference time, which is not bearable sometimes. For instance, it may take about 355 years to train GPT-4 on a single NVIDIA V100, i.e. forever.

To handle these scenarios, efficient AI has gained increasing attention among researchers, which targeting to increase efficiency in all ways, where efficiency means either achieving comparable or even better quantitative results using fewer resources (data, model size, training time or inference time) or get better results using comparable resource, including data efficiency - to reduce the reliance on a large amount of labeled data by data augmentation and data distillation, model efficiency - to reduce noise in models like searching more compact structures by Neural Architecture Search (NAS) and pruning, training and inference efficiency - to reduce time at training/inference like transfer learning and quantization.

1.2 Challenges

There are some common challenges for efficient deep learning.

1. **Overfitting.** Larger models produce better generalization. However, when data size is decreased largely, e.g., from 14 million images in the ImageNet dataset to 60, 000 images in CIFAR-10, large models cannot work anymore. Instead, with limited data, large models tend to overfit the training set while generalizing badly on the testing set. As a result, small datasets usually mean smaller models to get better convergence at the cost of feature extraction capacity. Besides reducing model size, there are also other methods to mitigate

overfitting, like data augmentation.

2. **Underfitting.** This comes with a greatly decreased model size. When the model becomes much smaller by techniques like pruning, NAS, or knowledge distillation, fewer constraints are set by the model, resulting in a much larger solution space and hence worse generalization result, which is called underfitting.

3. **Domain Generalization.** Another issue is the worse performance on the testing set caused by the domain gap when the dataset is too small or the source domain and target domain differs a lot in transfer learning. Unlike languages, which have little domain gap between datasets, image datasets suffer from domain gaps caused by different view angles, weather, sketch or photograph, light, and so on. Hence, large datasets can cover more domains to get better generalization compared. However, for small datasets, limited labeled examples exaggerate the domain gap between the training set and testing set, even between batches in the training set.

1.3 Related Work

Many researches are conducted around different aspects of efficient AI, including data efficiency, model efficiency, and training/inference efficiency.

1.3.1 Data Efficiency

Firstly, to increase input information. Noticing good performance of data-rich models, the most direct way to increase data efficiency is to generate more data to mimic data-rich scenarios, including inputting data of different modalities (vision and text) and different forms (RGB images and their transformations like Discrete Cosine Transformation and wavelet transformation). In addition, we can generate more data of the same form given a few examples, like getting more RGB features by feature operating on given examples, including intersection, union and subtraction [3]. Mixing foreground and background from given examples is also viable as shown in [201]. We can even learn from those imaginary data that benefit few-shot classification [168].

Besides, to reduce data noise. A commonly used way is to perform data distillation to generate a summary of input data and use the high-quality data to train the model[143]. Actively learning [139, 6, 152] propose to use different strategies to use fewer examples, e.g., using the most important examples first.

1.3.2 Model Efficiency

The most direct way is to reduce the model size. Knowledge distillation [123, 9] is a popular one that distills information from a larger teacher model to a smaller student model by aligning intermediate features. NAS [73, 174, 165] reduces model size by proposing a set of model structure candidates based on the base model, and then selecting the best one with carefully designed metrics after training all the candidates for a few steps.

Moreover, more compact structures are also helpful in increasing model efficiency. For example, depth-wise convolution [31] reduces the connections by 8/9 by removing all relations between depths.

1.3.3 Training Efficiency

To optimize quickly on data-hungry tasks, one widely used technique is fine-tuning, e.g. pretraining model with large-scale data and then using the resulted parameters as initialization of the model in downstream tasks. This becomes even more popular when large language models and foundation models arise. Which parameters to fine-tune and how to fine-tune become the core issues. In this track, adapter-based algorithms, which learn only adapters by freezing all pretrained models and adding adapters into the frozen model, are widely explored as it's convenient for downstream tasks to share pretrained models. In this way, only adapters are necessary to learn and store. Besides, selecting an appropriate task that can benefit current downstream tasks to pretrain is a challenge.

Another interesting way to increase training efficiency is to design task-specific metrics. For example, few-shot learning usually designs a transformation to a lower dimension and then uses a metric function to compare, called metric learning. In other words, it deals with “what to compare” and “how to compare”. Prototypical Network [5] proposes to average the embeddings of given examples from each class as the class prototype, and then compare each query example with class prototypes to get the closest one. TADAM [6] further shaped the class prototype with the task prototype. Both methods mainly answer the “what to compare” question. Different from this, the Relation Network [7] mainly deals with “how to compare” by modeling the metric function instead of using a predefined one like Euclidean distance.

1.3.4 Inference Efficiency

At the deployment stage, there are also some techniques to speed up inference based on the hardware while keeping performance as much as possible. Quantization [109, 124, 185] focuses on replacing high-precision datatype like FP32 with low-precision datatype like int8, which retains the model structure and most model weights, but loses just some precision. In this case, which data to quantize and how to quantize become the core problems.

1.4 Our Contributions

In this dissertation, our contributions to addressing efficient deep learning in computer vision can be summarized as follows:

1. **Data Efficiency - increase input information** Propose to integrate the frequency and spatial representations on few-shot learning, which increases data efficiency by introducing more input information without new labeled data.

2. **Data Efficiency - decrease data noise** Propose to explicitly increase the input information density in the frequency domain, which increases data efficiency by removing noise in data

representations. Specifically, we introduce selecting channels by calculating the channel-wise heatmaps in the frequency domain using Discrete Cosine Transform (DCT), reducing the size of input while keeping most information and hence increasing the information density. As a result, 25% fewer channels are kept while better performance is achieved compared with previous work.

3. **Model Efficiency** Discover that the number of trivial attention weights is far greater than the important ones and the accumulated trivial weights dominate the attention in Vision Transformers due to their large quantity, which is not handled by the attention itself. This will cover useful non-trivial attention and harm the performance when trivial attention includes more noise. To solve this issue, we propose to divide attention weights into trivial and non-trivial ones by thresholds, then Suppressing Accumulated Trivial Attention (SATA) weights by proposed Trivial WeIghts Suppression Transformation (TWIST) to reduce attention noise.

4. **Training Efficiency** Propose a generalized framework called SuperLoRA that unifies and extends different Low-Rank Adaptation (LoRA) variants, which can be realized under different hyper-parameter settings. Introducing new options with grouping, folding, shuffling, projection, and tensor decomposition, SuperLoRA offers high flexibility and demonstrates superior performance, with up to 10-fold gain in parameter efficiency for transfer learning tasks.

5. **Inference Efficiency** Propose a roadmap that can be applied to further accelerate image restoration models prior to deployment while simultaneously increasing PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index). The roadmap first increases the model capacity by adding more parameters to partial convolutions on FLOPs non-sensitive layers. Then, it applies partial depthwise convolution coupled with decoupling upsampling/-downsampling layers to accelerate the model speed.

Chapter 2

Data Efficiency: Increasing Input Information by Integrating Spatial and Frequency Representations

In this chapter, we solve the efficient deep learning tasks, specifically few-shot learning, from the perspective of increasing input information by including representations from another domain, DCT frequency domain [25].

2.1 Introduction

Inspired by the fact that human vision is more sensitive to low-frequency information [83, 178], we assume that human beings are learning some frequency information when classifying images. Results from the recent research [178] also demonstrated impressive performance in the frequency domain where the backbone networks are fed with frequency domain information. To exploit the advantages of both the spatial domain and frequency domain information, we propose to integrate the features extracted from the spatial domain and the frequency domain, aiming to increase the performance of few-shot learning. As illustrated in Figure 2.1, the spatial domain feature module employs CNN to obtain the representation of original images, and the frequency domain feature module utilizes the Discrete Cosine Transformation (DCT) module [**?**] to generate the frequency representation of original images and feed them into CNN to generate frequency features. Both the frequency features and spatial features are then fused together, followed by a classifier.

The main contributions of this study are as follows:

1. This chapter proposes to exploit DCT with static selected channels to few-shot classification

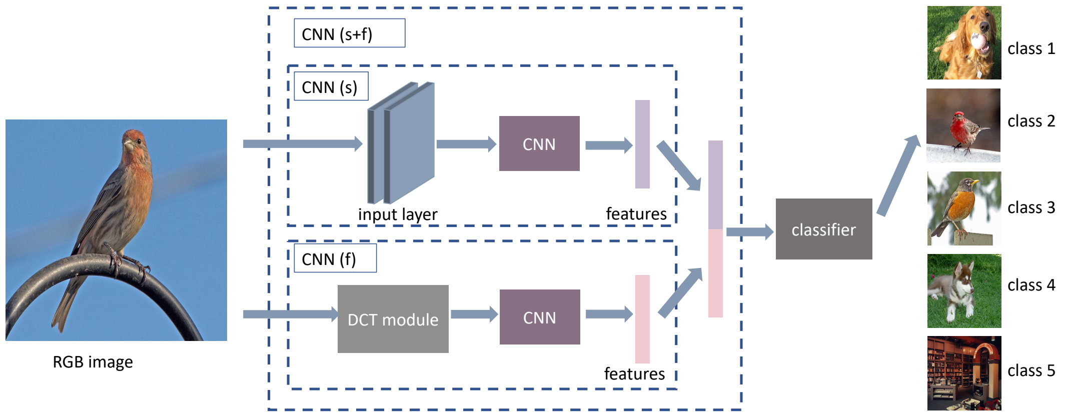


Figure 2.1: The structure of the proposed framework. We concatenate the features from two networks. The upper network, “CNN (s)" denotes a regular image classification network where we draw the input layers before the backbone from the whole CNN and “(s)" means the “spatial domain". In the lower network, “CNN (f)", the images will go through the DCT module first to generate the frequency representation before being supplied to the CNN backbone, and “(f)" represents the “frequency domain". Finally, we concatenate features obtained from both networks to generate the final classification score, which is the whole output network, “CNN (s+f)".

and implements the idea on different FSL models.

2. We investigate the influence of DCT filter size on the few-shot classification and find the relationship between DCT filter size and classification accuracy.

3. Extensive experiments demonstrate that integrating the features from both the spatial and frequency domains can significantly increase the classification accuracy in few-shot classification.

2.2 Related Work

This part introduces some related work on few-shot learning and deep learning methods in the frequency domain.

2.2.1 Few-Shot Learning

Few-shot learning aims to learn to classify query examples from “novel" classes given a few labeled support examples from the “novel" classes and abundant labeled support examples from “base" classes. Recent deep learning based few-shot learning algorithms could be roughly divided into 4 categories: 1) Data augmentation is a direct data-level method to improve classification accuracy in few-shot learning via mimicking large-scale data algorithms [105, 201, 28, 168], which is usually added to meta-learners as auxiliary. By generating more positive and/or negative data according to the given labeled data, more information could be fed to the deep neural network. 2) Metric-learning based method, or learn to compare, is one type of meta-learning based approach in few-shot learning, which focuses on constructing an appropriate embedding space to yield corresponding features of images and then calculating the similarity between the features of given labeled images and test images. Related researches include [97, 134, 122, 98, 164, 154]. 3) Optimization-based meta-learning methods [140, 46, 128], or learn to optimize, usually train another network to get the optimization hyper-parameters to adjust to the few-shot learning scenario that is different from previously fixed hyper-parameters. 4) Another type of meta-learning in few-shot learning is to use external memories [191, 149].

All approaches mentioned above employ the spatial RGB images as the input of backbones. However, none of them make use of the frequency representations. In our work, we preprocess the RGB images with a DCT module to obtain the frequency representations and then input them to the backbone.

2.2.2 Deep Learning in the Frequency Domain

Frequency-domain based algorithms are widely used in network compression [169, 20], focusing on modifying the network to yield better efficiency. However, our DCT-based method focuses on increasing the classifying accuracy by reducing the input size with little modification of the model itself. The band-limited algorithm presented in [42] shows that the model focuses on leveraging lower-frequency components. However, this FFT-based method is more effective on larger kernels instead of the most commonly used smaller filters like 3×3 and 1×1 in most neural networks. [52] uses DCT coefficients during JPEG encoding which helps to improve the efficiency. [178] shows that high-frequency channels of DCT could be removed without accuracy loss or even help to increase the classification accuracy in large-scale image classification. However, none of the above-mentioned frequency-based algorithms try to jointly exploit the features from both the spatial and frequency domains. [183] proposes to add spatial-spectral convolution blocks in convolution layers to learn more powerful representations, while it requires much revision on the network and extra computations. Moreover, it employs DCT in a different way from ours. Based on frequency analysis, we propose to explore its effect on few-shot learning by integrating features from both the spatial and frequency domains.

2.3 Method

2.3.1 Description on Few-shot Learning

Considering a problem of few-shot classification, let C_{source} and C_{target} denote the source classes and target classes, respectively, and these two classes are disjoint. In the source classes C_{source}, we have abundant labeled samples as the training data D_{source}, while only a few labeled data D_{target} are accessible for each class in the target classes C_{target}. For a k-way n-shot learning problem, where we have n labeled support samples for k novel classes in C_{target}, our task is to classify a query sample into one of these k support classes.

2.3.2 Discrete Cosine Transform (DCT)

Inspired by the study [178], we design the DCT pipeline as shown in figure 2.2. To generate the frequency representation, we perform some pre-processing first, including the standard transformation as illustrated in [88], rotating, cropping, and translating, and obtain the image with size S_{image} in step (a), e.g. 448×448. Then, similar to the JPEG compression pipeline, we convert the high-resolution RGB images to YCbCr images, where we follow the 4:2:0 Chroma subsam-

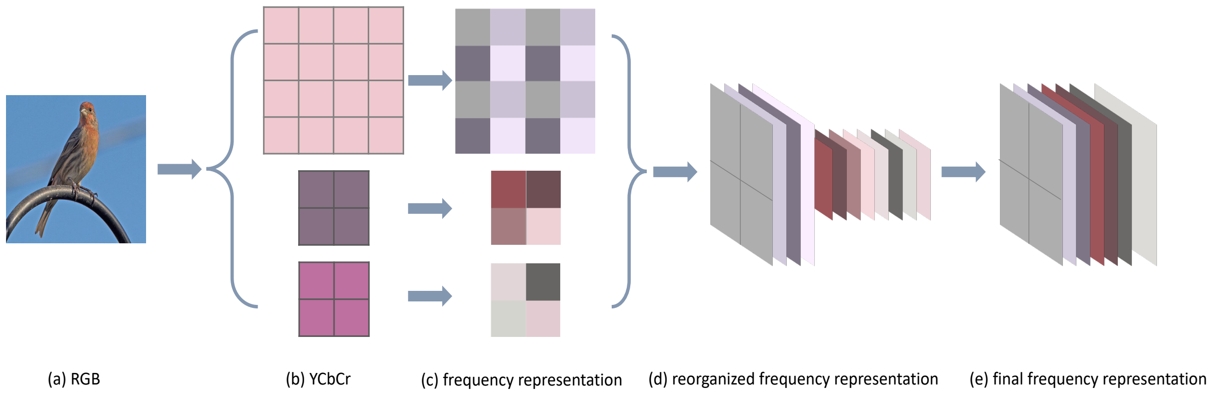
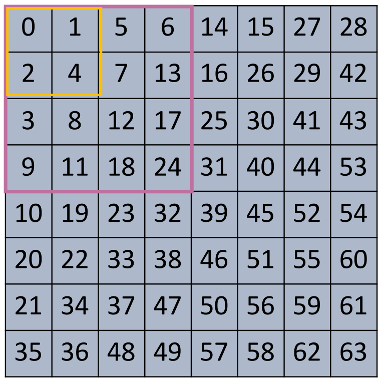

(a) RGB
(b) YCbCr
(c) frequency representation
(d) reorganized frequency representation
(e) final frequency representation

Figure 2.2: (a) images after preprocessing, e.g. $448 \times 448 \times 3$. (b) transform RGB to YCbCr images, Y: 448×448, Cr and Cb: 224×224. (c) frequency representation after DCT transformation with a filter, e.g. 8x8, (d) reorganize frequency representations by frequency channels, Y: $56 \times 56 \times 64$, Cb and Cr: $28 \times 28 \times 64$. (e) upsample all Cr_{dct} and Cb_{dct} frequency channels to the same size with Y_{dct} channels and keep those frequency selected channels. Y: $56 \times 56 \times 16$, Cb and Cr: $28 \times 28 \times 4$.

pling as shown in step (b) considering the human vision system is more sensitive to brightness (Y) than color (Cr and Cb). After that, we divide each channel into $S_{dct} \times S_{dct}$ patches and perform $S_{dct} \times S_{dct}$ DCT transformation on each patch, where S_{dct} is the size of the DCT filter, which is assigned at (e.g. 8×8). In this way, we obtain an 8×8 frequency representation for each 8×8 patch in each YCbCr channel, with a lower frequency in the left top corner and a higher frequency in the right bottom corner as illustrated in step (c). Next, for each channel, we reshape these 8×8 frequency patches to cubes by grouping the same frequency to one sub-channel and yield (d). Finally, from (d) to (e), we first select more impactful frequency sub-channels obtained in [178] from each YCbCr channel, concatenate them into one frequency cube and then upsample CrCb elements to the same size of Y element by interpolation, which would be the input of the following deep neural network. In (e), from left to right, it shows low to high-frequency elements for Y, Cb, and Cr respectively.

The final input size after the DCT module, the size of (f), would be S_{image}/S_{dct}, e.g. $448/8 = 56$, and the input size of ResNet is also 56×56. So we could keep the network the same size by adjusting S_{image} and/or S_{dct}. In this way, we can handle images with a vast range of sizes to keep as much information from the original images (e.g. when $S_{dct} = 16$ and the input size of the backbone



0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figure 2.3: The 8×8 frequency index after DCT transformation. Each index represents one frequency component. Low frequencies lie in the top left corner and high frequencies are in the bottom right corner. We choose the top left square frequency elements as selected frequency channels. Specifically, 4×4 square for the Y channel and 2×2 for the Cr and Cb channels.

is 56, we could process images with size $S_{image} = 56 \times 16 = 896$). The DCT transform of an image can be denoted by:

$$D = TMT' \tag{2.1}$$

where M is the $S_{dct} \times S_{dct}$ image patch after subtracting 128 for each pixel. And T is the DCT transformation matrix determined by:

$$T_{i,j} = \begin{cases} \frac{1}{\sqrt{N}}, & i = 0 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N}, & i > 0 \end{cases} \tag{2.2}$$

2.3.3 Frequency Channels Selection

According to the study [178], low-frequency elements would be selected more frequently by designing a channel selection module carefully. In our implementation, we choose the following 24 channels obtained in [178] as shown in Figure 2.3: the top left 4×4 square for Y channel and the top left 2×2 for Cr and Cb channels.

2.3.4 Network

The framework of the proposed network is illustrated in Figure 2.1. The upper network, “CNN (s)" denotes a vanilla image classification network where we draw the input layers before the backbone from the whole CNN and“(s)" means the “spatial domain". In the lower branch, “CNN (f)", images will go through the DCT module first to obtain the frequency representation before being fed to the following CNN backbone, and “(f)" represents the “frequency domain". Finally, we integrate the information from the two domains by concatenating the normalized features obtained from both networks to output the final classification score, which is denoted as the whole outer network, “CNN (s+f)".

2.4 Experiments

2.4.1 Datasets and Setup

Datasets. The proposed framework has been evaluated on three popular few-shot learning datasets: mini-ImageNet, CUB, and CIFAR-FS. 1) **mini-ImageNet**. This is a popular few-shot learning dataset first proposed in [164], which samples 100 classes from the original ILSVRC-12 dataset [142]. For each class, it contains about 600 images. All images are 84×84 RGB colored. In our experiment, we follow the split in [140], with 64 classes for the training set, 16 classes for the validation set, and 20 classes for the testing set. Among these, classes from the training, validation, and testing sets are disjoint. 2) **CUB**. This dataset was introduced in [173] and it contains 6,033 bird images, 130, 20, and 50 classes for training, validation, and testing, respectively. 3) **CIFAR-FS**. This dataset [8] is obtained by randomly splitting 100 classes in CIFAR-100 [87] into 64 training classes, 16 validation classes, and 20 novel classes. All images in this dataset are of the size 32×32.

Implementation details. For CNN (s) where we input the backbones with images, the input samples are resized to 224×224 for ResNet [62] backbones and 84×84 for WRN-28-10 [195] on mini-ImageNet and CUB, 32×32 for CIFAR-FS. For CNN (f), the frequency version, images

backbone	method	accuracy on *mini*ImageNet	
		1-shot	5-shot
ResNet10	MN [164][#]	54.49±0.81	68.82±0.65
	MN (s)[*]	52.98±0.21	72.41±0.16
	MN (f)	55.98±0.20	74.17±0.16
	MN (s+f)	**57.32±0.21**	**76.27±0.16**
		+4.34	*+3.86*
WRN-28-10	S2M2$_R$[122]	64.93±0.18	83.18±0.11
	S2M2$_R$ (s)[*]	63.09±0.17	80.88±0.11
	S2M2$_R$ (f)	63.03±0.18	80.80±0.11
	S2M2$_R$(s+f)	**66.88±0.18**	**84.26±0.10**
		+3.79	*+3.38*
WRN-28-10	PT+MAP [74]	82.92±0.26	88.82±0.13
	PT+MAP (s)[*]	80.73±0.24	87.81±0.13
	PT+MAP (f)	82.04±0.23	88.68±0.12
	PT+MAP (s+f)	**84.81±0.22**	**90.62±0.11**
		+4.08	*+2.81*

Table 2.1: Improvement after integrating features from both the spatial and frequency domains to existing methods on *mini*ImageNet. The highest accuracy (%) with 95% confidence interval is highlighted. # and $*$ denotes results reported in [19] and our reproduced results to the published ones respectively.

are rescaled to $56 \times S_{dct}$ and the input dimension varies from the number of selected frequency channels. For example, when we use 8×8 DCT filters, the original images will be 448×448. To train (s+f), we train (s) and (f) separately first, and then fine-tune the classifier on the integrated features to get the final classifier for (s+f). This framework can also be trained end-to-end.

2.4.2 Application to Existing Learning Models

In this section, we explore the generalization ability of the proposed approach to other few-shot learning frameworks.

2.4.2.1 Influence of the integrated features

We integrate the proposed strategy to the following few-shot learning frameworks on mini-ImageNet: the Matching Network (MN) [164] (metric-based algorithm), S2M2$_R$ [122] (pre-train and fine-tune), and PT+MAP [74] (post-process S2M2$_R$ features). The input is 84×84 for the spatial branch

method	backbone	***mini*ImageNet**		**CUB-200-2011**		**CIFAR−FS**	
		1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
ProtoNet* [154]	ConvNet	50.37±0.83	67.33±0.67	66.36±1.00	82.03±0.59	55.5±0.70	72.0±0.60
MAML* [46]	ConvNet	50.96±0.92	66.09±0.71	66.26±1.05	78.82±0.70	58.9±1.9	71.5±1.0
RelationNet* [156]	ConvNet	51.84±0.88	64.55±0.70	64.38±0.94	80.16±0.64	55.0±1.0	72.0±0.60
S2M2$_R$ [122]	WRN-28-10	64.93±0.18	83.18±0.11	80.68±0.81	90.85±0.44	74.81±0.19	87.47±0.13
AFHN [100]	ResNet18	62.38±0.72	78.16±0.56	70.53±1.01	83.95±0.63	-	-
DPGN [186]	ResNet12	67.77±0.32	84.60±0.43	75.71±0.47	91.48±0.33	77.90±0.50	90.20±0.40
PT+MAP [74]	WRN-28-10	82.92±0.26	88.82±0.13	91.55±0.19	93.99±0.10	87.69±0.23	90.68±0.15
PT+MAP(s+f)	WRN-28-10	**84.81±0.22**	**90.62±0.11**	**95.48±0.13**	**96.70±0.07**	**89.50±0.21**	**92.16±0.15**

Table 2.2: Comparison with the state-of-the-art on mini-ImageNet, CUB, and CIFAR-FS. The highest accuracy (%) is highlighted. * means results for miniImageNet and CUB-200-2011 datasets are from [19], and results for CIFAR-FS are from [8].

to yield a fair comparison with previous results and 448×448 for the frequency domain. All images are pre-processed with data augmentation during training. The results are shown in Table 2.1. It is evident that the proposed scheme (s+f) promotes the accuracy by about 2.8-4.3% in all cases compared with the original version (s) of the models. This shows that the integrating features from both domains work for different frameworks and its improvement is not limited to fine-tuning based or metric-learning based methods. Another interesting observation is that when we train S2M2$_R$ (f), the rotation loss decreases to less than 0.1 rapidly within a couple of epochs. This might be because the rotation trick can work for spatial input while failing when it comes to frequency input, which might also explain that S2M2$_R$ (s) works better than S2M2$_R$ (f). This set of experiments demonstrates that the integrated features can work for different few-shot classification frameworks in which the single-frequency branch functions differently for different algorithms.

2.4.2.2 Comparison with the state-of-the-art

We compare the proposed network (s+f) with the state-of-the-art on the benchmarks. PT+MAP [74] proposes to leverage the learned features to Gaussian-like distribution and add it to the network S2M2$_R$ [122]. Since the proposed strategy is designed in a preprocessing way, making it possible to combine it with any network. We implement our method to PT+MAP and name it by adding “(s+f)" to the models we use. The images are resized to 84×84 and 448×448 for the spatial and frequency input respectively. 8×8 DCT filter with static frequency channel selection is employed.

The results are shown in Table 2.2. It is evident that our method can increase the accuracy of the state-of-the-art by a large margin for all datasets we tested, including mini-Imagenet, CUB datasets, and CIFAR-FS. In all three datasets, PT+MAP achieves the best performance in terms of accuracy. For mini-Imagenet, our approach increases the best accuracy by 1.89% and 1.8% for 5-way 1-shot and 5-way 5-shot, respectively. For the CUB dataset, the accuracy is increased by 3.93% and 2.71% respectively for the two tasks. Please note that for the CIFAR-FS dataset, the image size is small, only has 32×32. However, we still observe 1.81% and 1.48% increases for the two tasks, respectively.

2.4.3 Ablation Study

2.4.3.1 Few-shot learning with DCT

In this section, we first validate the effectiveness of DCT module and integrated network by comparing baseline++ (s), baseline++ (f) and baseline++ (s+f) w/o data augmentation during training on *mini*ImageNet, where baseline++ [19] uses all images from base classes to pre-train and then fine-tune with a few support samples from testing set before testing. The only difference between the baseline [136] and baseline++ is that the baseline uses a linear classifier while baseline++ calculates cosine distance. The backbone we choose for feature extraction is ResNet 34. Both 5-way 1-shot and 5-way 5-shot image classification tasks are evaluated. For data augmentation during training, we performed random crop, left-right flip and color jitter as in the paper [19]

method	data augmentation	*mini*ImageNet 5-way 1-shot	*mini*ImageNet 5-way 5-shot
baseline++ (s)	No	48.54±0.17	61.58±0.13
baseline++ (f)	No	53.27±0.19	65.65±0.13
baseline++ (s+f)	No	**54.76±0.18**	**68.76±0.13**
baseline++ (s)	Yes	57.94±0.18	73.98±0.13
baseline++ (f)	Yes	59.22±0.18	76.58±0.13
baseline++ (s+f)	Yes	**62.75±0.18**	**79.73±0.12**

Table 2.3: Results on *mini*ImageNet with different inputs as shown in Figure 2.1. The backbone is ResNet34. Top left square 24 channels as illustrated in Figure 2.3 are selected for frequency versions. The highest accuracy (%) is highlighted.

The experimental results are shown in Table 2.3, from which we can see that the accuracy of baseline++ (f) is higher than baseline++ (s) by 4.73% and 1.28% respectively for 5-way 1-shot classification task without and with data augmentation during the first training phase, 4.07% and 2.6% for the 5-way 5-shot task. The baseline++ (s+f) in all cases further increases the classification accuracy by 1.5-3.5 %, showing that baseline++ (f) is not just an improvement of baseline++ (s) but a complementary method of it and learning from both the spatial and frequency domain could increase the classification accuracy.

2.4.3.2 Influence of original information quantity

For the DCT branch, the network has more flexibility to choose image size with the existence of the DCT module, even taking larger images compared with the spatial version. To explore the effect of information quantity the frequency branch takes, we conducted experiments on *mini*ImageNet. The backbone of the frequency branch is ResNet 10 to save time and data augmentation is employed during training. The results are tabulated in Table 2.4. For the first two parts in the table, we preprocess images for baseline++ (f) with the same data augmentation method as the baseline (s) to generate 84×84 and 224×224 images. Then, we upsample these images to 448×448 to exploit the DCT module. For the bottom part of this table, we resize the images directly to 448×448 to include more original information.

From the table, we can see that, when baseline++ (s) and baseline++ (f) utilize the same information from the original images, baseline++ (s) performs better than baseline++ (f) when the image is 84×84 but worse for image size 224×224, which means (f) holds the potential to perform better with fewer parameters than (s) when inputting enough information. When we input baseline++ (f) with more information than baseline (s), 448 instead of 224, the accuracy for (f) gets slightly improved, 0.9 % for 1 shot and 0.06 % for 5 shots. We think the reason is the information quantity is more and more approaching the amount needed by the current frequency backbone. Moreover, in all cases baseline++ (s+f) performs better than both baseline++ (s) and baseline++ (f), which means that the spatial and frequency representation are complementary to each other.

method	image size	*mini*ImageNet 5-way 1-shot	5-way 5-shot
baseline++ (s)	84	52.32±0.17	68.24±0.14
baseline++ (f)	84→448	49.47±0.17	65.64±0.12
baseline++ (s+f)	N/A	**56.32±0.17**	**75.70±0.13**
baseline++ (s)	224	57.52±0.17	75.56±0.13
baseline++ (f)	224→448	58.71±0.17	76.55±0.12
baseline++ (s+f)	N/A	**62.23±0.18**	**80.08±0.12**
baseline++ (s)	224	57.52±0.17	75.56±0.13
baseline++ (f)	448	59.61±0.18	76.61±0.12
baseline++ (s+f)	N/A	**62.30±0.18**	**79.93±0.11**

Table 2.4: Results on *mini*ImageNet with different information quantity supplied to the frequency channel as shown in Figure 2.1. The backbone is ResNet 10 and data augmentation when training is implemented. The highest accuracy (%) is highlighted.

To conclude, we can use larger images (if we can access them) to increase the accuracy by using the DCT module, and integrated features can always improve the performance no matter whether the frequency branch can access larger images compared to the spatial branch.

2.4.3.3 Different DCT filters and selected channels

In this experiment, we explore the effect of different sizes of DCT filters, 2, 4, 6, and 8, and different selected channels, 24 channels and all frequency channels as shown in Table 2.5. The backbone is ResNet18. For (f) version, the number of channels is 24 and $S_{dct} \times S_{dct} \times 3$ when we select 24 and all channels respectively, e.g. when the DCT filter size is 4, $S_{dct} = 4$, the number of all channels will be $4 \times 4 \times 3 = 48$, where 4×4 is the size of DTC module and 3 is from Y, Cr and Cb channels. For the spatial branch, we use all 224 as the image size. For the DCT branch, we resize images to $56 \times S_{dct}$ directly, e.g. if $S_{dct} = 4$, the input is rescaled to $56 \times 4 = 224$. Experiments with and without data augmentation during training are evaluated.

According to Table 2.5, all baseline++ (f) methods outperform their corresponding baseline++ (s) version without the DCT module, even when the DCT filter size is as small as 2×2. When all frequency channels are employed, we find the accuracy is increased with the increase of the size of the DCT filter no matter there is a data augmentation or not for the 5-way 1-shot classification.

method	train$_{aug}$	DCT filter size	channels	accuracy on *mini*ImageNet 5-way 1-shot	accuracy on *mini*ImageNet 5-way 5-shot
baseline++ (s)	False	-	64	48.40±0.17	62.94±0.13
baseline++ (f)	False	2	all (12)	49.05±0.17	65.45±0.13
baseline++ (f)	False	4	all (48)	49.67±0.18	65.07±0.13
baseline++ (f)	False	6	all (108)	49.70±0.17	64.51±0.13
baseline++ (s)	True	-	64	56.48±0.17	74.00±0.13
baseline++ (f)	True	2	all (12)	57.79±0.17	75.50±0.12
baseline++ (f)	True	4	all (48)	58.41±0.17	76.01±0.12
baseline++ (f)	True	6	all (108)	**58.98±0.17**	75.39±0.12
baseline++ (f)	False	4	24	50.11±0.17	63.91±0.13
baseline++ (f)	False	6	24	50.72±0.18	64.86±0.13
baseline++ (f)	False	8	24	51.04±0.18	65.76±0.13
baseline++ (f)	True	4	24	58.02±0.18	75.73±0.13
baseline++ (f)	True	6	24	57.74±0.17	75.66±0.12
baseline++ (f)	True	8	24	58.25±0.18	**76.23±0.13**

Table 2.5: The effect of different sizes of DCT filters and whether we select frequency channels with backbone ResNet18 on *mini*ImageNet. The number of channels denotes the channels before the backbone with the input layers removed, e.g. for ResNet, the input before the basic backbone is $56 \times 56 \times 64$. The highest accuracy (%) is highlighted.

However, the increase is very small, and for 5-way 5-shot task, we do not observe this trend, and the baseline++ (f) with $S_{dct} = 2$ even outperforms other filter sizes when there is no data augmentation. On the other hand, when only the top left 24 frequency channels are employed, the accuracy increases mildly with that of the filter sizes for both 5-way 1-shot and 5-way 5-shot.

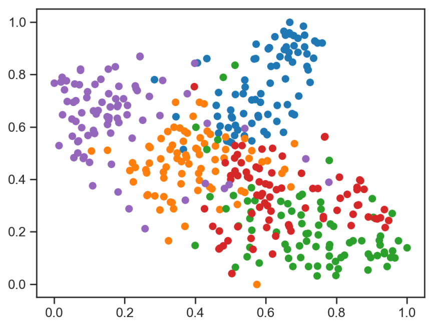
From these observations, we can see that the filter size has little influence on the few-shot classification when we perform static frequency channel selection. The influence is neglectable in comparison with the influence of data augmentation. In this experiment, DCT filter size 6 with all channels and data augmentation achieves the best performance for the 5-way 1-shot task, and DCT filter size 8 with 24 channels and data augmentation achieves the best performance for the 5-way 5-shot task. However, the increase is not significant. In practice, we can simply choose a small filter size to save the computation cost.

backbone	method	accuracy on *mini*ImageNet	
		1-shot	5-shot
ResNet10	baseline++ (s)	57.52±0.17	75.56±0.13
	baseline++ (f)	59.61±0.18	76.61±0.12
	baseline++ (s+f)	**62.30±0.18**	**79.93±0.12**
		+4.78	*+4.37*
ResNet18	baseline++ (s)	56.48±0.17	74.00±0.13
	baseline++ (f)	58.52±0.18	76.23±0.13
	baseline++ (s+f)	**61.66±0.18**	**79.70±0.12**
		+5.18	*+5.70*
ResNet34	baseline++ (s)	57.94±0.18	73.98±0.13
	baseline++ (f)	59.22±0.18	76.58±0.13
	baseline++ (s+f)	**62.75±0.18**	**79.73±0.12**
		+4.81	*+5.75*

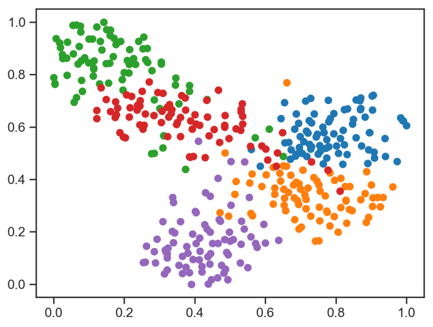
Table 2.6: Results on *mini*ImageNet with different backbone networks when we implement different input versions, (s), (f) and (s+f), as shown in Figure 2.1 to baseline++. The highest accuracy (%) is highlighted.

2.4.3.4 Integrated features with different backbone

In this section, to verify the impact of integrated features under different backbones, we implement different versions of baseline++, (f) and (s+f), with the backbone ResNet10, ResNet18, and ResNet34 as shown in Table 2.6. Image size for (s) and (f) is 224 and 448 respectively. 8×8 DCT filters and static channel selection are implemented. In all cases, baseline++(f), when we preprocess images with the DCT module, outperforms all baseline++ (s) when there is no DCT module. Furthermore, in both tasks (5-way 1-shot and 5-way 5-shot classification) and for all backbones (ResNet10, ResNet18, and ResNet34), the baseline++ (s+f) version achieves the best performance compared with baseline++(f) and baseline++(s), and the accuracy is improved by a margin of 4-6% compared with their vanilla versions, baseline++(s). This further verifies that learning from the frequency domain is a complementary method of learning from the spatial domain and integrating both features could further increase the classification accuracy. The experiment also demonstrates the effectiveness of the proposed approach on different backbone networks.



(a)



(b)

Figure 2.4: t-SNE feature visualization results. (a): baseline++(s). (b): baseline++(s+f).

2.4.4 Feature Visualization

To visually understand the feature we learned from the whole framework, the t-SNE visualization [121] is shown in Figure 2.4. The features for (a) and (b) are learned from baseline++ with backbone ResNet 10, specifically baseline++(s) and baseline++(s+f). 5 novel categories are selected randomly and 80 samples for each class are employed. It can be observed from Figure 2.4 that the clustering results by integrating features from both the spatial and frequency domains are more compact than those from only the spatial domain, which further verifies the effectiveness of integrating both domains in improving the clustering ability.

2.5 Conclusion

In this chapter, we have proposed to integrate the representations from both the spatial and frequency domains to increase the performance of few-shot classification. Through extensive experiments, we have demonstrated that the frequency information is complementary to feature representation, and integrating the features learned from both the spatial and frequency domains can significantly increase the performance of few-shot learning. The proposed strategy can act as a plug-in module for other few-shot learning models to increase their classification accuracy.

Chapter 3

Data Efficiency: Explicitly Increasing Input Information Density by Decreasing Input Noise

Besides increasing input information, data efficiency can be achieved by reducing data noise and hence increasing data quality. Data quality for machine learning models is different from the image quality for human eyes. For models, high-quality data should contain less noise or duplicate images to make the model more robust to various data, e.g. higher information density. This chapter gets inspiration from image compression to increase information density and then feeds the dense input into vision transformers directly to mimic language models [23].

3.1 Introduction

ResNet [62] and its variants have been widely used in computer vision tasks in the recent decade due to their superior performance from introducing inductive bias [12, 132, 188]. However, convolutional layers on the other hand limit its ability to capture global relations. Vision Transformer (ViT) [40], as an emerging alternative backbone to ResNet in computer vision, expands the network representation capacity by including global attention with multi-head self-attention modules, which shows better performance compared with ResNet in large-scale datasets like ImageNet [37] after pretraining on larger datasets JFT-300M [155]. However, the lack of locality in ViT results in worse generalization on small datasets compared with ResNet [24, 95, 49]. To release its dependency on more data and accelerate convergence, researchers try to add convolutional layers to transformers [51, 55, 175, 177]. However, none of them solve this by increasing input density directly.

As noticed in [61], transformers originate from Natural Language Processing and the main difference between languages and images is information density. This reminds us to increase the token information density of image input to close the gap and better generalize vision transformers on image tasks. To achieve this, the principle of image compression can be utilized. Image compression is a widely explored task, aiming at transforming images from spatial domain to another domain to reduce storage and transmission costs, while keeping as much information as possible. JPEG is one of the popular Discrete Cosine Transform (DCT) based image compression algorithms, keeping about 95% information while high compressibility by reducing a large amount of less informative high-frequency details in its quantization process, increasing the information density in the frequency domain. Inspired by this, we propose to feed vision transformers with dense information in the frequency domain to approach the dense word embedding in NLP. Specifically, we perform block DCT on YCbCr images and then remove up to 174 non-functional frequency channels (192 in total) based on the channel-wise heatmaps. This channel selection increases input density greatly by decreasing the size of input greatly while keeping most information, making it closer to word embedding in NLP and better generalizing vision Transformers on small datasets.

The major contributions of this chapter are as below:

1. We first propose to **explicitly** increase the input information density in vision Transformers to close the gap between language and image input.

2. To form the dense input, we design a simple yet effective channel-wise heatmap-based strategy to select useful DCT frequency channels. Compared with the static shape-based channel selection strategy in previous work, our selection keeps fewer channels while showing better results for the Swin Transformer.

3. Experiments on Tiny ImageNet [93], CIFAR-10/100 [87], Flowers-102 [129] and SVHN [127] with backbones Swin Transformer [108] and Focal Transformer [184] demonstrate the effectiveness of learning in the DCT frequency domain with vision Transformers.

3.2 Related Work

This section includes some most related work regarding vision transformers and deep learning in the non-RGB domain.

3.2.1 Vision Transformers

Vision Transformers have received a lot of attention and applications in recent years [48, 120, 131, 145]. However, the original Vision Transformer, ViT, requires a large amount of data to learn local properties. To handle this, some researchers manage to integrate convolutional layers in different modules of vision Transformers, e.g. token or backbones, to capture both local and global information of the image and accelerate convergence considering that token embeddings can be mapped into fixed positions of the input images. For instance, [177] replaced the patch-based token preprocessing with several small-size convolutional layers directly and found a better, faster, and more robust convergence compared with the original ViT. In addition to this, LeViT [51], CMT [55] and CeiT [192] also used a convolution-based tokenization on top of transformer blocks. Others add convolutional layers in the middle of Transformer backbones, e.g. CvT [175] employed squeezed convolutional projection in replace of linear projection to obtain keys/-queries/values for multi-head attention in transformer blocks, LocalViT [103] added depth-wise convolution layer to the feed-forward module in transformer blocks and CoaT [181] implemented a depth-wise convolution-based position encoding in the multi-head attention layer. Another folder to accelerate convergence is to design pyramid-like structure modelling CNN backbones, including PVT [167], Swin Transformer [108], MViT [45], NesT [208] and LiT [130]. However, these works focus on extracting diverse features based on the low information density images themselves and put feature extraction pressure on the networks. Instead, this work explicitly increases the input information density at the beginning, undertakes some responsibility from the networks, and fills the gap between languages and images.

3.2.2 Learning in non-RGB domain

Although most computer vision algorithms are RGB image-based, there are some efforts to explore learning in other domains. One popular domain is from one stage in the JPEG encoding and decoding process. JPEG format is widely employed to compress images to save the cost of transmitting and storing images. In the encoding stage, it consists of converting RGB images to YCbCr, quantifying block DCT (where information loss happens), and encoding with Huffman. The decoding process is just the inverse. Normally, the widely used RGB image input is the output after decoding JPEG images. [52] proposes to learn from the quantified DCT domain in the decoding stage, i.e. feed the quantified DCT coefficients to the network directly before they are converted back to the RGB domain, accelerating convergence by removing the decompressing process to transform DCT representation back to image pixel and learn similar feature with its first layers. [44] formulates deep learning in JPEG transform domain, representation after quantifying the DCT coefficient in the encoding stage, and shows the equivalence after conversing models learning on the RGB domain to this intermediate JPEG domain. [50] incorporates DCT transformation on the feature generated by the first convolutional layer, speeding up convergence a lot compared with using standard CNN. The most relevant work is [179], which proposes to learn from an earlier stage, block DCT domain before quantization, and then disregard learnable useless frequency channels to save communication costs between CPU and GPU. However, [179] designs some learnable parameters and Gumble sampling to select channels while it turns out to be a square shape-based selection strategy works a little bit better. Differently, this work employs GradCAM heatmap to check functional channels directly, which is simpler, more intuitive and more efficient.

3.3 Method

The proposed framework is shown in figure 3.1. In this section, we will introduce the details of each preprocessing module, including the block DCT, channel-wise heatmap-based channel selection strategy, and the relationship between frequency channel selection and information density.

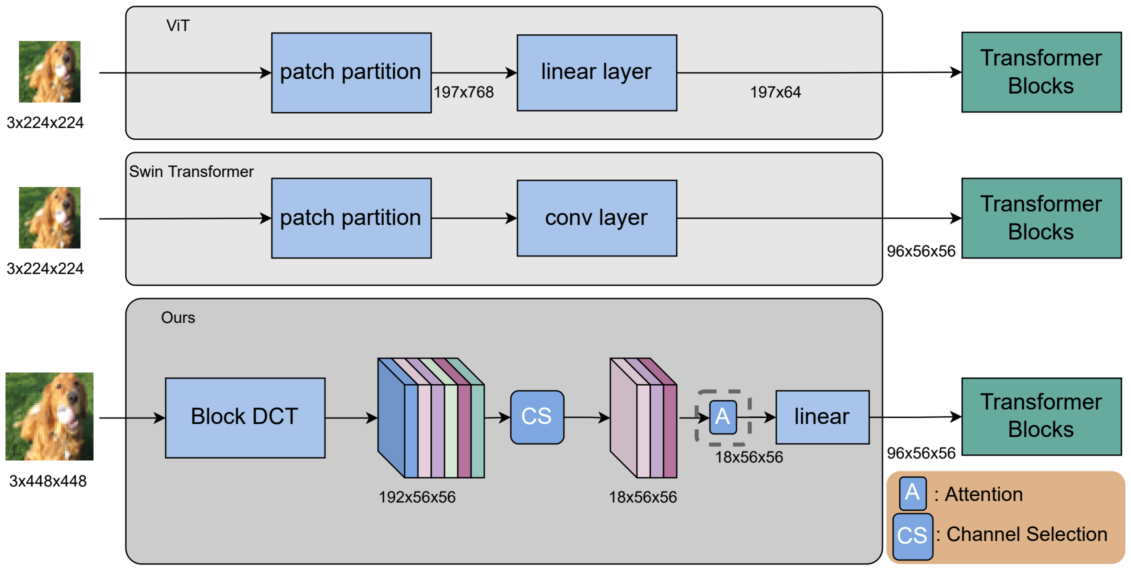


Figure 3.1: Vision Transformer (ViT), Swin Transformer, and our Dense DCT-based vision Transformer. In our dense DCT pre-processing, RGB images are resized to 448×448 so that the output size of DCT remains the same, i.e., 56×56. Then, the frequency representation with all 192 frequency channels goes through their channel selection based on the heatmap results, and a dense frequency representation with only 18 channels is obtained. The attention module is optional for the dense frequency representation to pass through directly before the Transformer block. Finally, to keep the input channel and following Transformer blocks the same, we pass the 18-channel input to a linear layer and get 96 channels as in Swin Transformer.

3.3.1 Block DCT

Block DCT is applied as in [179] and its illustration can be found in Figure 3.2. First, we convert the RGB images into YCbCr representation. Then, apply DCT transformation on each small block (e.g. 8×8) of the YCbCr component. After that, we reorganize DCT coefficients from the same frequency to one channel keeping block positions. Finally, we remove useless frequency channels after applying the channel selection strategy. As the human visual system is more sensitive to luminance component Y than the color components Cb and Cr, we downsample the Cb and Cr components as in JPEG compression in step (b) and then upsample them in step (d).

3.3.2 Channel-Wise Heatmap

To learn better from the DCT frequency domain, [179] utilized a Gumbel softmax-based feature selection algorithm to reduce input channels while they found their shape-based (i.e., top left

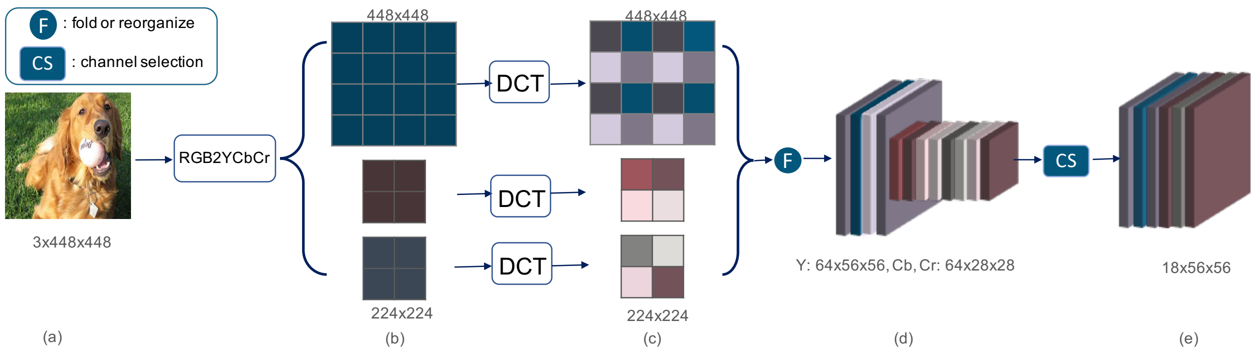


Figure 3.2: Block DCT illustrated by 8×8 DCT transformation. (a) The resized RGB images. (b) Convert RGB images to YCbCr with Cb and Cr channels downsampled. (c) Perform patch-based non-overlapping DCT for all 3 channels. (d) Rearrange the same frequency from all positions to the same channel. (e) Upsample Cr and Cb channels and then select frequency channels.

square) fine selection strategy achieves competing performance compared with the keeping learned frequency channels from feature selection algorithm. However, shape-based selection in the frequency domain lacks interpretability. Instead, we propose a channel-wise heatmap-based channel selection strategy as shown in 3.4.

GradCAM [151] is a method used to find the functional region of an image input in CNN by relating the output high-level features with their input positions. However, for input with multiple meaningful channels, different input channels might function differently as in [179]. To evaluate how each input contributes to the final high-level features, we mask out all channels except the candidate channel and then calculate the GradCAM heatmap, i.e., channel-wise heatmap.

Specifically, frequency representations with full frequency channels, i.e. 192 for 8×8, are fed into a ResNet50 backbone with cosine classifier as shown in Figure 3.3. After that, to obtain the channel-wise heatmap, keep only one candidate frequency channel of the frequency representation and set other channels to 0 as shown in Figure 3.4. Then, we pass the revised frequency representation into the pretrained model and calculate the GradCAM heatmap. Following the same process in Figure 3.4, each image has 192 heatmaps (for 8×8 block DCT), and each heatmap is 56×56 following block position layout.

In this way, we calculate the average channel-wise heatmap of images from the training set and then average across all 56×56 positions for each channel heatmap. The results for 8×8

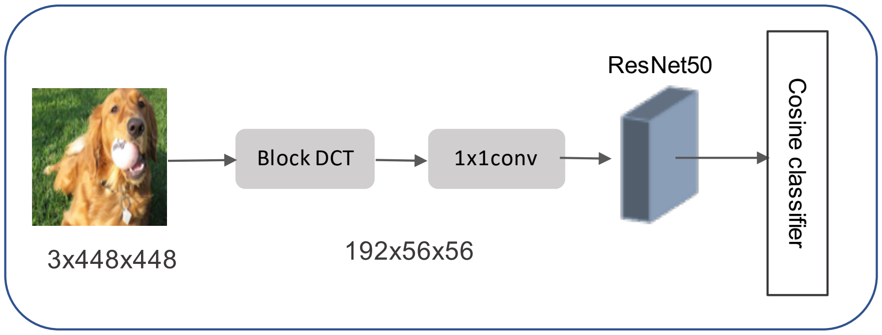


Figure 3.3: Training pipeline to calculate the channel-wise heatmap.

block DCT can be found in Figure 3.5, where each frequency channel has only one value. This result is surprisingly consistent with the one from the channel selection network proposed in [179], while our method requires no extra parameters. This indicates that the channel selection network in [179] is indeed learning functional channels as supposed. Finally, to select kept frequency channels and get dense DCT input, we simply set thresholds based on values in Figure 3.5 and remove the channels that have no contribution to the performance. Frequency channels in red blocks are our 18 kept channels after comparing with the accuracy from different thresholds, and those 24 channels within yellow squares are based on the square channel selection strategy concluded in [179].

3.3.3 Frequency Channel Selection and Information Density

In DCT transformation, top left corner represents lower frequencies and bottom right are the higher ones. To visualize the energy distribution, we go through all frequency channels following the zig-zag route as shown in Figure 3.6 and get Figure 3.7. Figure 3.7 we know, that most energies are pushed to sparse and low-frequency channels (top left on Figure 3.6) and most high-frequency channels have only few energy (which is also the principle for JPEG compression to disregard those zero channels after quantization). According to Figure 3.7, most selected channels (denoted by red dots) also lie on the low-frequency part. This indicates that, after channel selection, the overall information entropy is slightly reduced, while the size of the frequency representation is significantly decreased, i.e. the average information entropy or the information density is increased greatly.

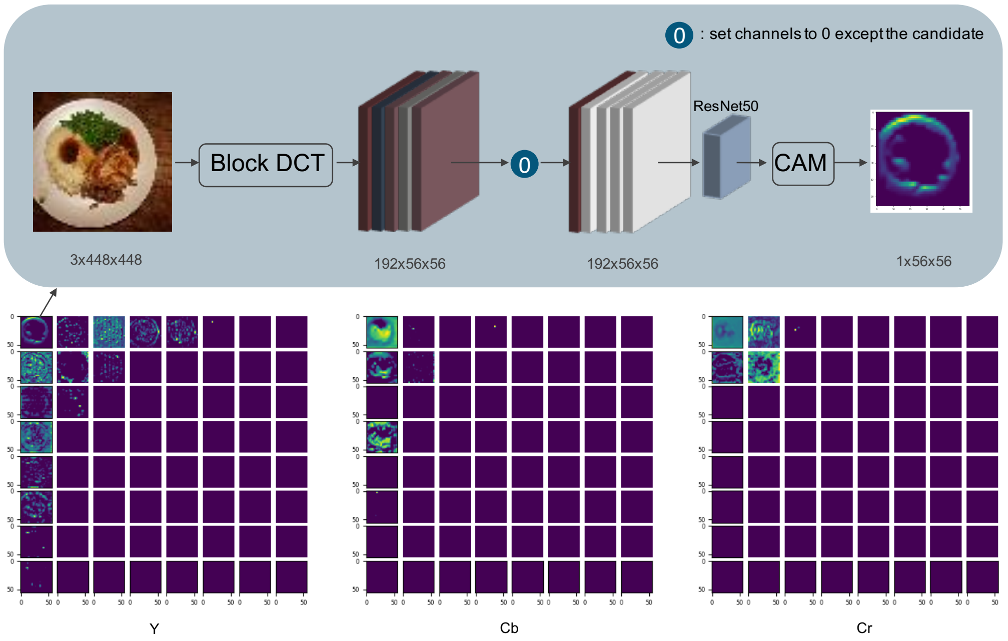


Figure 3.4: Calculate the input channel-wise heatmap for one image based on pretrained model in 3.3. 8×8 DCT preprocessing on the RGB image results in a tensor with full 192 frequency channels. Then all channels except the candidate are set to zeros. The GradCAM heatmap is calculated based on the masked tensor and the model trained with full frequency channels and yields the input channel-wise heatmap for the candidate channel, e.g., channel 0 in this figure.

3.4 Experiments

3.4.1 Setup

Dataset Information 5 small datasets are employed to test the image classification performance of our framework, Tiny ImageNet [93], CIFAR-10/100 [87], Flowers-102 [129], and SVHN [127]. More details of these datasets are shown in Table 3.1.

Training Setup Data augmentation methods we used include CutMix [194] and Mixup [202] as in [106]. The initial learning rate is set to $5e^{-4}$ and 20 epochs warmup. We report the best top-1 accuracy after 100 epochs. The batch size is 128 per GPU and four A100 GPUs are used in total. Following [106], we resize all images to 3×224×224 on all datasets to get our baselines and resize all images to 3×448×448 unless otherwise specified. For a fair comparison, all vision Transformer backbones we selected have a comparable amount of parameters with ResNet50.

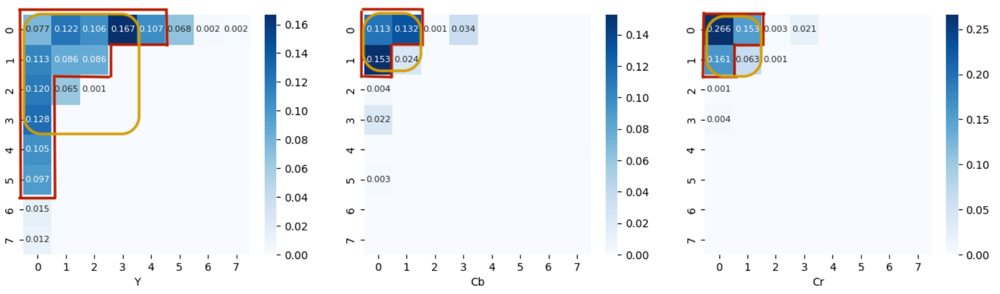


Figure 3.5: Averaged channel-wise GradCAM heatmaps for 8×8 by averaging all 56×56 for 8×8 positions in all training images from the same frequency channel. Channels in red are 18 selected channels using our method and those in yellow are 24 channels based on the method in [179].

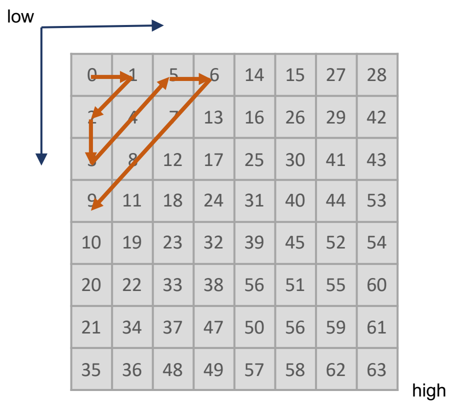


Figure 3.6: Zig-zag route for 8×8 DCT transformation.

3.4.2 Results

Among these 5 datasets, CIFAR-10/100 and SVHN datasets have the smallest image size, 32×32, compared with Tiny ImageNet and Flowers-102, holding less raw information, while the image size for Flowers-102 is greater than 500. We first test our dense DCT token with Swin Transformer [108] and Focal Transformer [184]. From Table 3.2, we can see an obvious performance gap between ResNet50 [62] and vision Transformers, including Deit-S [158], T2T [192], Swin [108] and Focal Transformer [184] on all datasets. With the proposed dense DCT input, the gap is significantly reduced and the performance is close to ResNet. For Swin Transformer, our dense DCT input achieves 4.80% improvement on CIFAR-100 compared with RGB input and up to 17.05% on the Flowers-102 dataset. We have achieved a 17.02% increase on Flowers for Focal-Tiny transformer.

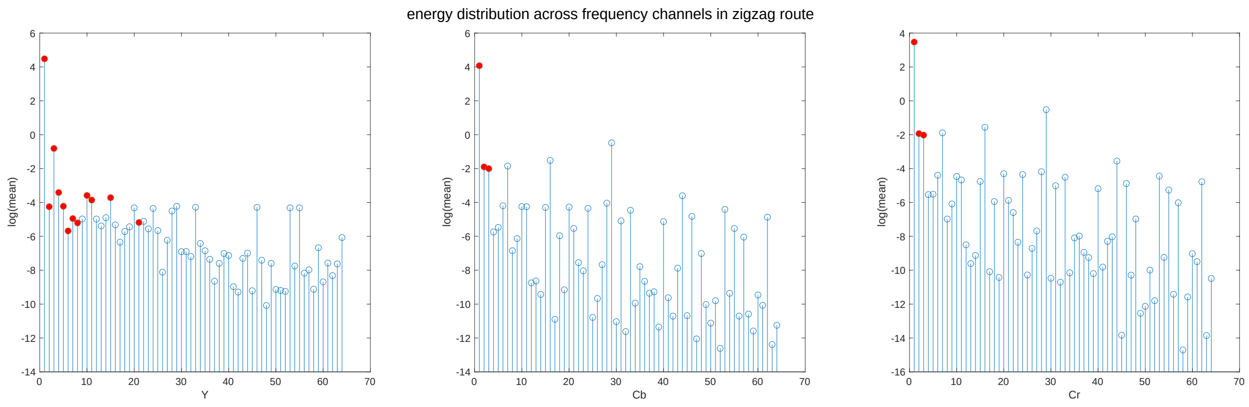


Figure 3.7: Energy distribution across all 192 frequency channels from Y, Cb and Cr components.

Dataset	**Train split**	**Val split**	**Size**	**Class #**
Tiny ImageNet	100,000	10,000	64	200
CIFAR-10	50,000	10,000	32	10
CIFAR-100	50,000	10,000	32	100
SVHN	73,257	26,032	32	200
Flowers-102	2,040	6,149	> 224	102

Table 3.1: Dataset information for classification on small datasets

In addition, we also add an attention module, Coordinate Attention [69], in our experiments as illustrated in 3.1. From Table 3.2, we can still see an increase for some backbones, e.g. Swin Transformer with Dense DCT further increases 0.31% on CIFAR-10 dataset after adding the attention module and the attention module also adds 0.32% to Focal Transformer with Dense DCT input on Tiny-ImageNet. However, it may harm the performance sometimes, e.g. also for the Focal Transformer on SVHN dataset, it decreases the accuracy by 0.90%.

3.4.3 Ablation Study

In this section, we check how each component of the proposed approach affects the performance, including the channel selection strategy, attention module on the input frequency domain, and resizing. We further study how resizing influences accuracy.

Backbone	Param.#	CIFAR-10	CIFAR-100	SVHN	Tiny IMG	Flowers
ResNet50[62]	25M	92.77	73.24	96.78	63.43	41.34
Deit-S[158]	22M	81.79	59.11	90.64	50.09	37.71
T2T-ViT-14[193]	22M	87.58	67.22	96.53	55.97	23.81
Swin-Tiny [108]	28M	81.91	62.30	91.29	56.28	32.04
DenseDCT_Swin-T	28M	84.20	67.10	95.65	57.36	49.09
		(*+2.29*)	(*+4.80*)	(*+4.36*)	(*+1.08*)	(*+17.05*)
DenseDCT_Swin-T$_{attn}$	28M	84.51	67.19	95.25	57.59	48.21
		(*+0.31*)	(*+0.09*)	(*-0.40*)	(*+0.23*)	(*-0.88*)
Focal-Tiny [184]	29M	88.31	71.58	95.97	63.06	32.54
DenseDCT_Focal-T	29M	89.59	73.94	96.13	64.11	49.56
		(*+1.28*)	(*+2.36*)	(*+0.16*)	(*+0.51*)	(*+17.02*)
DenseDCT_Focal-T$_{attn}$	29M	89.62	73.49	95.23	64.43	49.40
		(*+0.03*)	(*-0.45*)	(*-0.90*)	(*+0.32*)	(*-0.16*)

Table 3.2: Top 1 classification accuracy on CIFAR-10/100, SVHN, Tiny ImageNet and Flowers-102 dataset

Input	Size	Patch Size	Input size	Channel #	Resize	Attn	Top-1 ↑ (%)
RGB	224	4	56	—	✓	—	56.28
RGB	448	8	56	—	✓	—	55.59
DCT	64	8	8	192 (all)	—	—	38.50
DCT	448	8	56	192 (all)	✓	—	53.11(*+14.61*)
DCT	448	8	56	24 [179]	✓	—	57.02(*+3.91*)
DCT	448	8	56	18	✓	—	57.36 (*+0.34*)
DCT	448	8	56	18	✓	✓	57.59 (*+0.23*)

Table 3.3: Ablation study on Tiny ImageNet with Swin Transformer backbone. For Tiny ImageNet, the original image size is 64. 'Attn' here means adding Coordinate Attention on the DCT input directly

3.4.3.1 All Components

This experiment is based on the Swin Transformer backbone on the Tiny ImageNet dataset. According to the results by setting the threshold to their heatmaps, we select 18 channels for our dense DCT representation. The attention module we employed here is Coordinate Attention [69].

The results are shown in Table 3.3, where 'size' in the second column represents the resizing in Figure 3.1 at the beginning. 'Input size' and 'Channel #' denote the height (equal to the width) and the number of channels of the input embedding before the Transformer blocks in Figure 3.1 separately. From the results, we can see that the DCT input with all 192 channels and no resizing on the input yields a bad result, 38.50%. However, after resizing, it becomes 53.11%, increased by 14.61%. The channel selection strategy further improves the performance due to the increase in information density. The proposed heatmap-based channel selection achieves 57.36%, 0.34% greater than the shape-based channel selection strategy in [179] with 6 fewer input channels. To exclude the influence of larger image size between RGB input and DCT input, we also resize the RGB version to 448 and use 8 patches in the input embedding layer to get the same input size as DCT input, i.e. 56×56, as shown in row 3. However, resizing to 448 decreases the performance from 56.28% to 55.59%, inferior to the resizing in frequency domain 57.36%. Based on these results, we fix the image size to 224 with patch size 4 when training RGB input versions in Table 3.2. Finally, the attention module also brings a 0.23% increase, indicating the potential to add the attention module to the input images.

Dense DCT holds some interesting properties. 1) No convolution layer or linear layer on the input layer. The combination of DenseDCT input with Transformers or pure MLP networks makes the whole network more explainable, i.e., the whole framework is literately a function of DCT features, instead of any unknown features from convolution layers or the combination of low semantic pixels. 2) Higher semantic information makes it possible to perform attention at the early stage, literally the raw data since DCT transformation is invertible. 3) Taking larger input images with the same backbone as the Swin Transformer. The whole network can benefit a lot from larger input images, either from the original larger image to take more information or from resizing as shown above. However, there is still some space for improvement. For instance, both our heatmap-based method and dynamic/shape-based method in [179] take static input channel, while as shown in Figure 3.4, functioning channels vary from the final selected channels when it comes to every single image, e.g. channel 1 for Cb component in Figure 3.4 is selected considering the overall performance for the whole dataset according to 3.5, while activated channel 24 is removed. This compromise indicates included redundancy and encourages us to design a customized dynamic channel selection strategy to further improve useful input information density.

Threshold	0.08	0.07	0.06	0.03	0.022	0.02	0.01	0.000	all
Channel #	17	18	21	22	24	25	28	41	192
Top-1 (%)	56.34	**57.36**	57.11	57.03	56.86	57.02	56.89	56.94	53.11

Table 3.4: Effect of setting different threshold in input channel-wise heatmap (8×8)

3.4.3.2 Effect of Selected Channels

In this experiment, we show the results by setting different thresholds to the input channel-wise heatmap obtained by training 8×8 DCT full frequency channels, i.e., 192, and making all channels except the candidate one as in Figure 3.4. As shown in Table 3.4, even with channels 18, 21 and 22, which are less than 24 from the shape-based static channel selection strategy in [179], we also achieve higher accuracy, 57.36, 57.11 and 57.03, compared to 57.02 as shown in Table 3.3. Based on the experiment, we choose 18 channels in our implementation of 8×8 DCT transformation, which happen to be the low-frequency channels according to the heatmap. Apart from these, the accuracy for 17 channels is 56.34 while it's 57.36 for 18 channels, increasing 1.02. This vital channel is the DC component for the Y component.

To check what are those deleted frequency channels and how they affect the RGB images, we visualize some images from the Flowers-102 and Tiny-ImageNet as shown in the Appendix. From the visualization, we know that most information is kept even after deleting many frequency channels, from 192 channels to 24 or 18, and only some high-frequency details are lost. This can also be found in columns 3 and 5, where removed information forms contours. Besides this, we also checked the effect of blocks where we calculated the GradCAM, classifier, epochs, and datasets as shown in the Appendix. We found that heatmaps from Flowers-102 tend to focus on different higher frequencies compared with heatmaps from Tiny-ImageNet and CIFAR-10.

3.4.3.3 Effect of Resizing

In this section, we analyze the effectiveness of resizing and show the results in Table 3.5, where the window size is set to 8 to adapt to all resizes in the table. Since the input tensor before the Transformer blocks of the Swin Transformer is associated with the window size in the attention

Resize Ratio	1	2	4	8
Image Size	64	128	256	448
Accuracy	40.49	49.30	54.89	56.44

Table 3.5: Analysis of resizing for DCT input and Swin Transformer.

module inside the Transformer blocks, i.e., should be divisible by the window size, and resizing can affect the shape of the input tensor, we fix the window size to 8 and set resize ratio to 1, 2, 4 and 8 in this experiment to control variables and 18 selected channels from previous results. In Table 3.5, the first row denotes the resizing ratio compared with the original image 64 on Tiny ImageNet (ratio 1 indicates no resizing).

From 3.3 we can see that resizing brings 14.61% improvement, from 38.50% to 53.11%. The same effect is also observed in Table 3.5. The accuracy is increased from 40.49% using original images with size 64 to 56.44% after resizing to 448. Both results indicate that resizing is a significant preprocessing method for training in the DCT frequency domain. The effect of resizing is illustrated in 3.8. Suppose that the size of the original image is 8×8 before resizing, and an 8×8 DCT is performed on the whole image. While if we resize it to 16×16, the same DCT window only covers one-quarter of the original image, which means sparser features embedded in the image are processed by DCT. The resizing preprocess would better separate the useful information from redundant information in the original image and promote further channel selection. Moreover, with smaller raw image blocks after resizing, position information is more accurate and approaches actual positions within each block since block DCT misses position information within blocks.

More Analysis: The appendix provides more analysis of the effect of deleted frequency channels on the RGB images and the influences of channel-wise heatmaps w.r.t. classifiers, blocks, and epochs.

3.4.4 Visualization

To check what are those deleted frequency channels and how they affect the RGB images, this

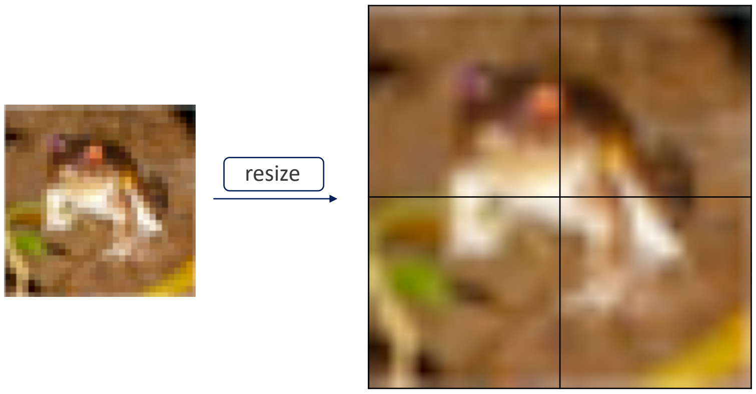


Figure 3.8: The effect of resizing.

section visualizes some images from the Flowers-102 (first 2 rows) and Tiny-ImageNet (since the third row) as shown in Figure 3.9. The first column represents the original images, IDCT_{24} and IDCT_{18} denote the inverse DCT results of the final kept frequency representations in Figure 2 in Section 3.1. From columns 2 and 4 we know, that most information is kept even after deleting many frequency channels, from 192 channels to 24 or 18, and only some high-frequency details are lost. This can also be found in columns 3 and 5, where removed information forms contours.

3.4.5 A Closer Look at Channel-Wise Heatmaps

This section explores what influences the channel-wise heatmaps and how they change them. All experiments are based on 100-epoch training and heatmaps in this section are calculated on 256 images (1 batch) for demonstration.

We first checked the effect of backbones. Following the same setting as in Figure 3 in Section 3.2, ResNet50 and Deit-S are trained. ResNet50 returns a concentrated heatmap as in Figure 5 in Section 3.2 while Deit-S heatmaps are scattered. Based on this, we used ResNet50 for all channel selections. In this sense, knowledge learned from ResNet50 is transferred to vision Transformers in Table 2.

3.4.5.1 Cosine Classifier vs Linear Classifier

In the general classification framework, images are passed into backbones like ResNet50, followed by a linear classifier and softmax function to obtain classification results. As shown in Figure 3 in Section 3.2, to get the pre-trained model for channel-wise heatmaps, frequency representations after block DCT are fed into backbones like ResNet50, and the following is a cosine classifier instead of a linear classifier as often used in few-shot learning algorithms [1]. To examine the difference, the same channel-wise heatmap generating process is conducted based on ResNet50. Resulted heatmaps are as Figure 3.10. As we can see, both heatmaps show similar preferences on frequency channels and most of them lie in the top left corners. As the heatmap from the cosine head is more similar to the one in [27] by sampling, we choose the cosine head to obtain heatmaps everywhere else in this chapter.

3.4.5.2 Heatmaps on Different Blocks

This section checks the heatmaps corresponding to different blocks in ResNet50 on the Tiny-ImageNet dataset. The best model after 100 epochs is used. From Figure 3.11, the first 4 layers focus only on the low-frequency channels for the Y, Cb, and Cr components. After that, higher frequency channels are emphasized lightly like blocks 6, 7, and 12, and heavily like blocks 5 and 8. We simply use the heatmap from block 4 for qualitative tables in previous sections.

3.4.5.3 Heatmaps Across Epochs

This section examines the growth of heatmaps across epochs, including epochs 0, 30, 60, 90 and 98 (when we get the best model within 100 epochs). From Figure 3.12, the network focuses on sparse frequency channels at the beginning, and then it explores more high-frequency channels at epoch 30. Starting from epoch 60, frequency channels are determined focusing on finetuning. It becomes closer and closer to the heatmap from the best model, which is used to calculate the heatmaps throughout this chapter.

3.4.5.4 The Effect of Datasets

This section shows the heatmaps of block 3 on different datasets, Tiny-ImageNet, CIFAR-10, and Flowers-102. From Figure 3.13, Tiny-ImageNet and CIFAR-10 focus more on similar frequency channels, while Flowers-102 on a different one. This is because Tiny-ImageNet and CIFAR-10 contain similar categories since CIFAR-10 is a subset of Tiny-ImageNet while images in Flowers-102 are more fine-grained, including only 102 classes of flowers. Moreover, from Table 2, the accuracy on the Flowers-102 dataset is relatively small compared with other datasets, indicating the selected channels are far from reliable convergence.

3.5 Conclusion

This chapter proposes to explicitly increase input information density to fill the gap between languages and images so as to increase the performance of vision Transformers on small-scale datasets. To achieve this, we have designed a simple yet effective pipeline to calculate the channel-wise heatmaps and select essential frequency channels in the DCT frequency domain. Surprisingly, our parameter-free heatmaps are highly consistent with sampling-based heatmaps from previous work and are more efficient and explainable. It is also simple to implement by using the well-developed JPEG compression API. Classification results on small-scale datasets including CIFAR-10/100, Tiny ImageNet, SVHN, and Flowers-102 show the effectiveness of our approach. Moreover, the selection strategy can be directly implemented in other tasks to reduce the communication cost between GPU and CPU while improving overall performance.

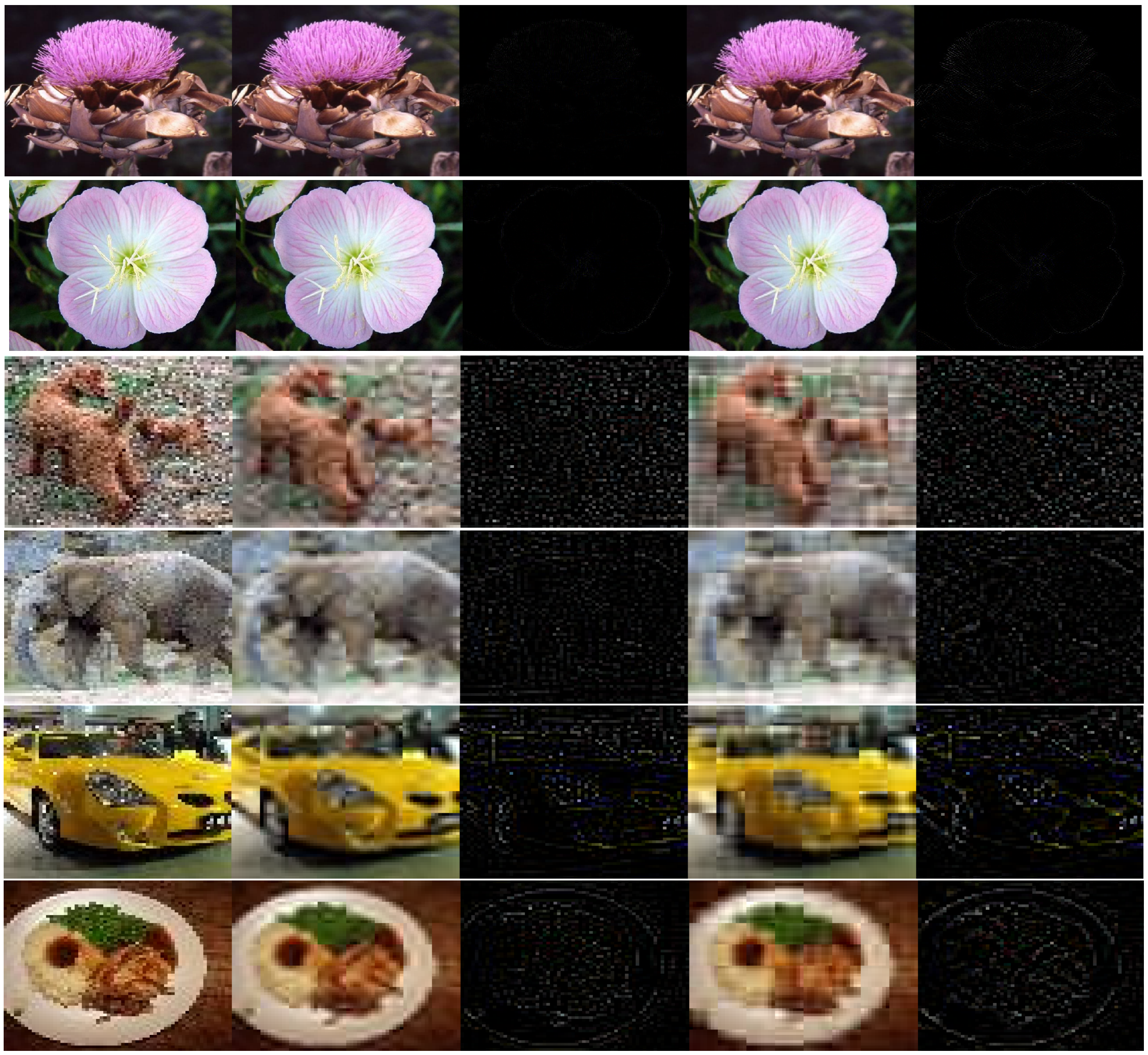


Figure 3.9: Visualization of the effect of removing frequency channels on RGB domain. From left to right column: (1) original image; (2) IDCT_{24}: reconstructed image after removing frequency channels as in [27] and keeping only 24 channels; (3) image-IDCT_{24}: removed information in RGB domain; (4) IDCT_{18}: reconstructed image with our channel selection strategy; (5) image-IDCT_{18}: our removed information in RGB domain. Images from the first 2 rows are from Flowers-102 dataset and the rest rows are from the Tiny-ImageNet dataset.

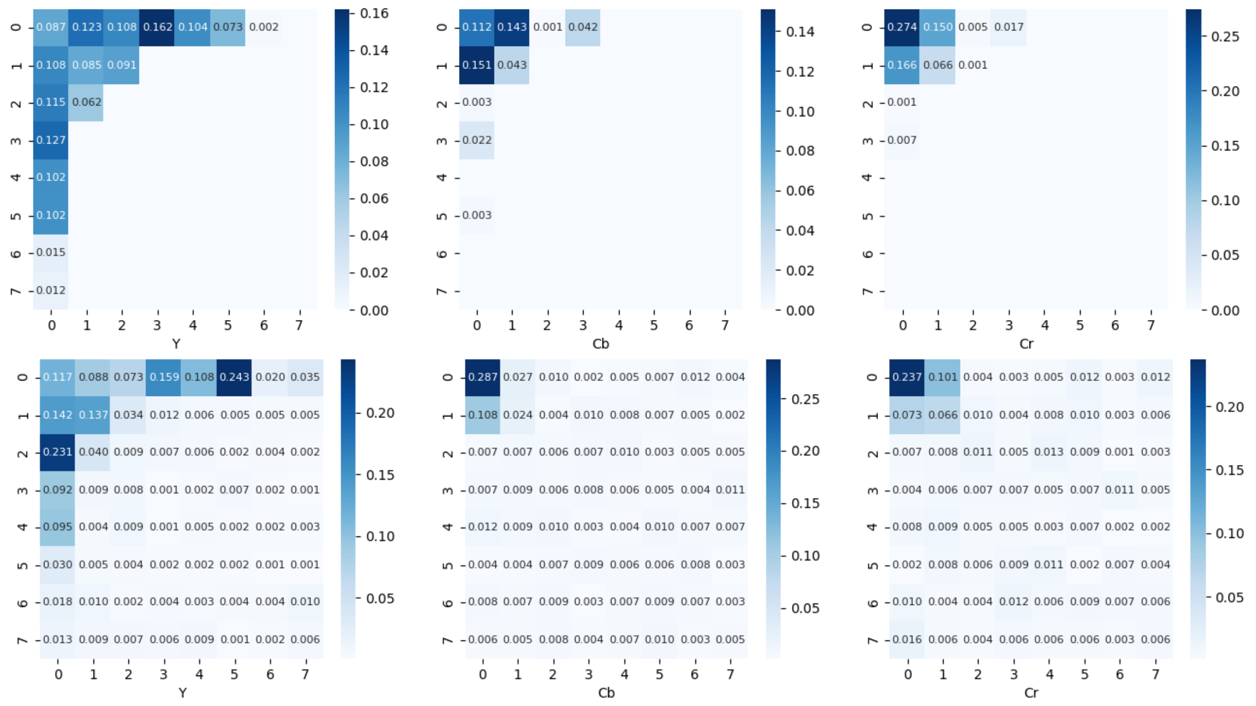


Figure 3.10: Channel-wise heatmaps with different classifier head. Upper: cosine classifier. Lower: linear classifier.

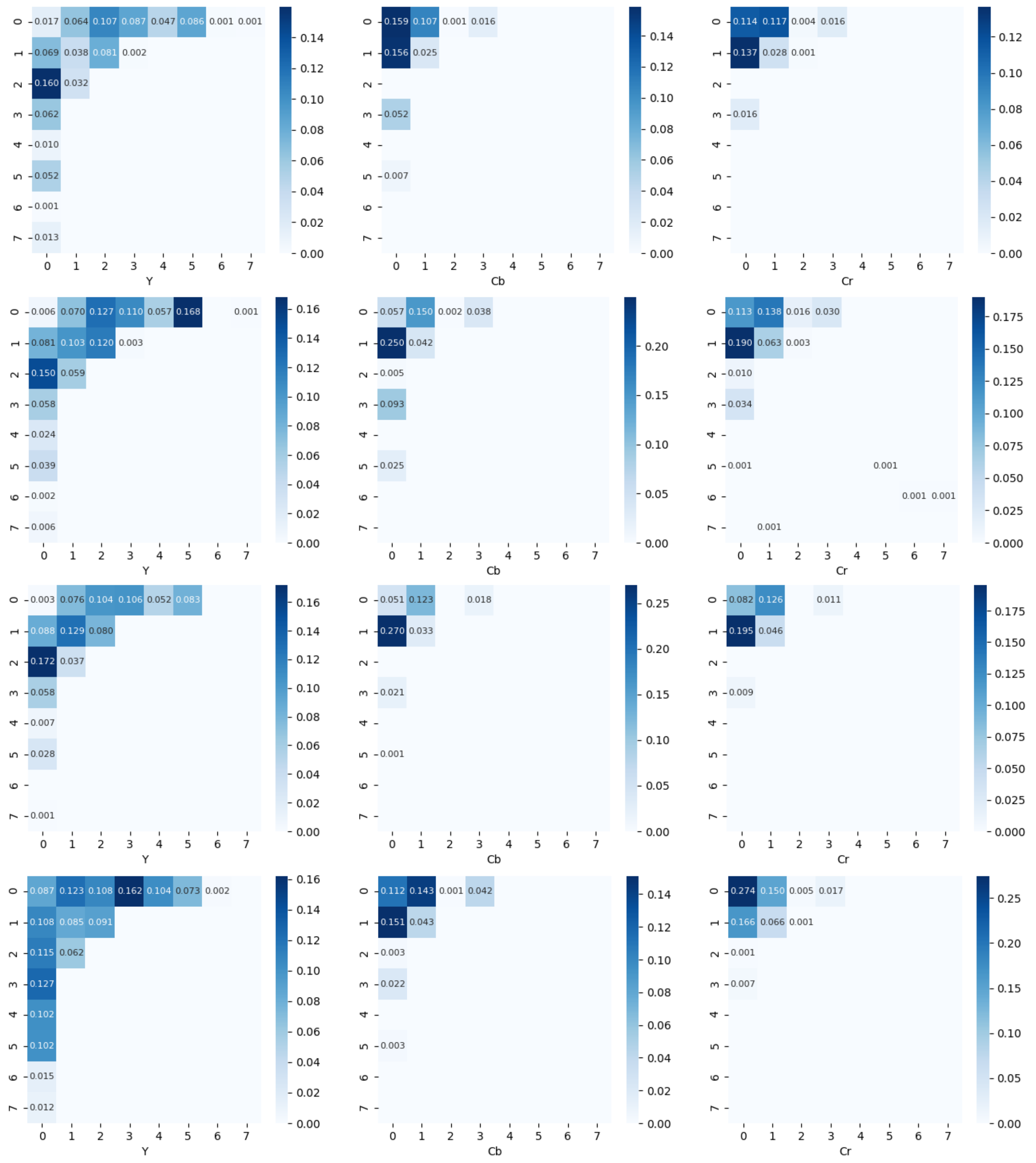


Figure 3.11: Heatmaps corresponding to different blocks in ResNet50. From top to bottom are from first four blocks, block 1 to block 4.

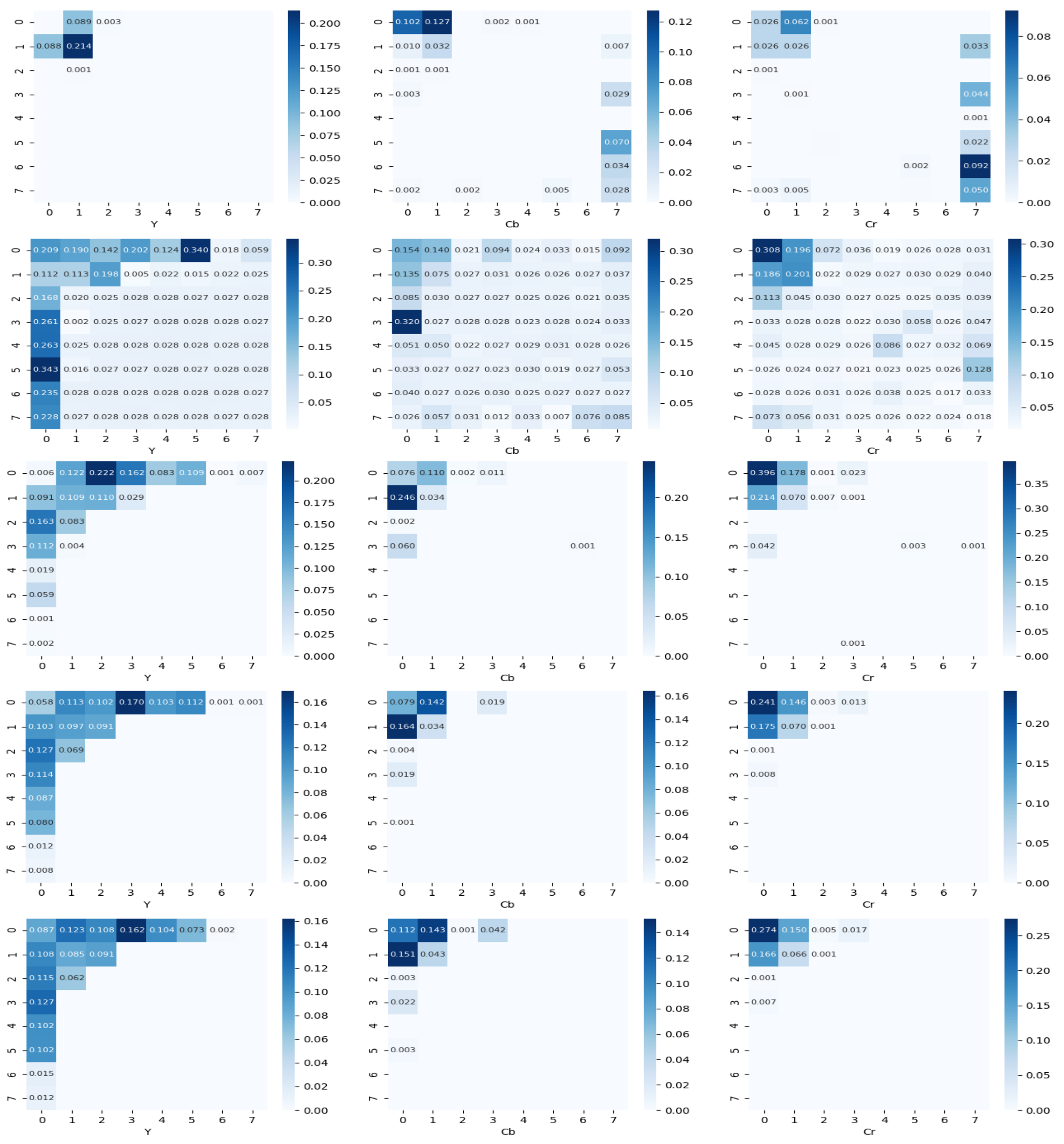


Figure 3.12: Heatmaps corresponding to different epochs in ResNet50. From top to bottom, epoch 0, 30, 60, 90 and best (epoch 98).

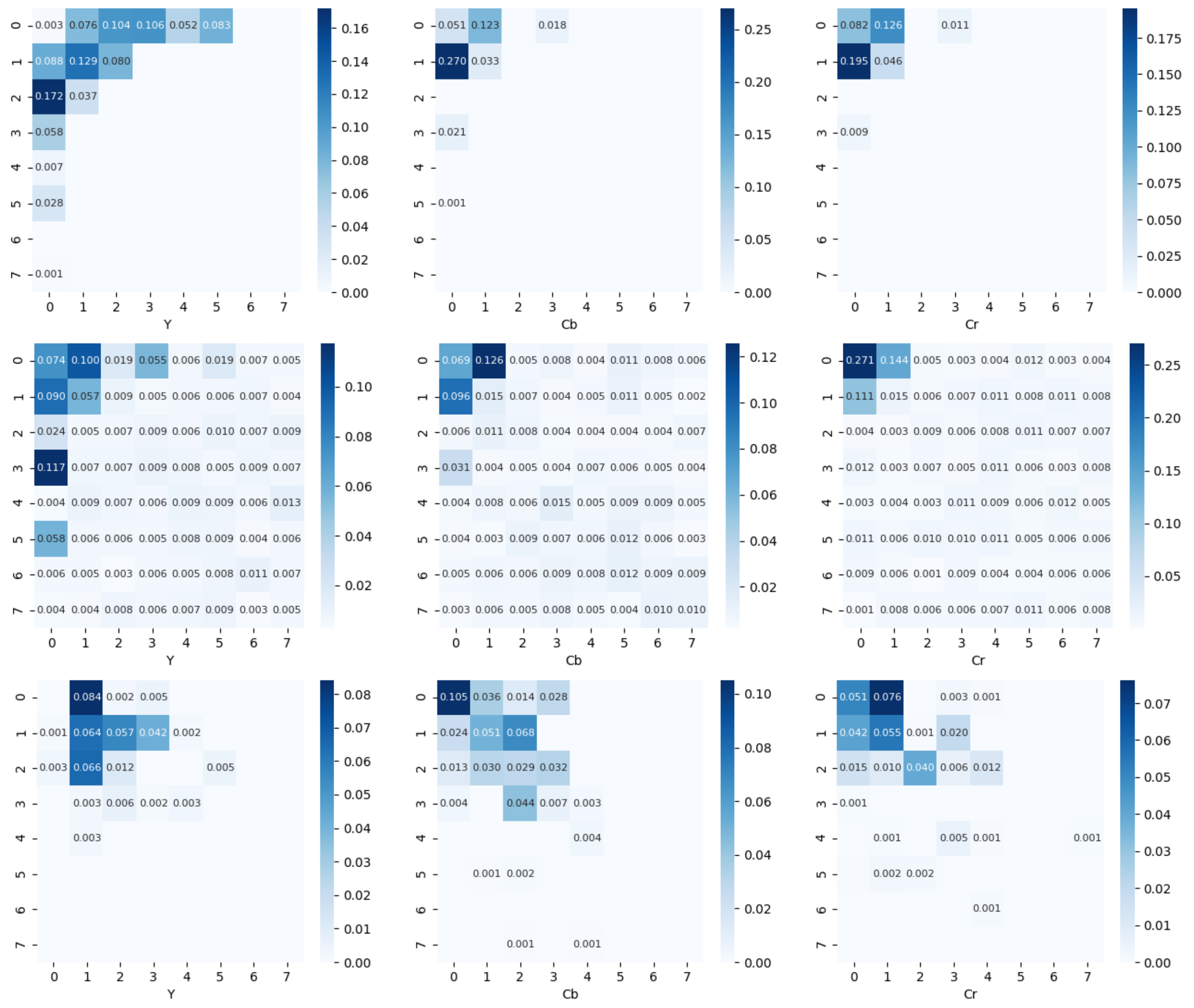


Figure 3.13: Heatmaps correspond to block 3 on different datasets with backbone ResNet50. From top to bottom, Tiny-ImageNet, CIFAR-10 and Flowers-102 respectively.

Chapter 4

Model Efficiency: Accumulated Trivial Attention Matters in Vision Transformers

In addition to data efficiency, model efficiency is another key track in efficient deep learning. Specifically, for computer vision tasks, they are CNN and Vision Transformers. In this chapter, we take a closer look at the vision transformer models and find an inherent weakness in the structure that may affect its model efficiency and hence the performance [21].

4.1 Introduction

Convolutional Neural Networks (CNN) have dominated computer vision tasks for the past decade, especially with the emergence of ResNet [62]. Convolution operation, the core technology in CNN, takes all the pixels from its receptive field as input and outputs one value. When the layers go deep, the stacked locality becomes non-local as the receptive field of each layer is built on the convolution results of the previous layer. The advantage of convolution is its power to extract local features, making it converge fast and a good fit, especially for data-efficient tasks. Different from CNN, Vision Transformer (ViT) [40] and its variants [32, 39, 48, 108, 120, 145] consider the similarities between each image patch embedding and all other patch embeddings. This global attention boosts its potential for feature extraction, however, it requires a large amount of data to feed the model and limits its application to small datasets.

On the one hand, CNNs have demonstrated superior performance to ViT regarding the accuracy, computation and convergence speed on data-efficient tasks, like ResNet-50 for image classification [12, 23, 106, 95], object detection [205] and ResNet-12 for few-shot learning [27]. However,

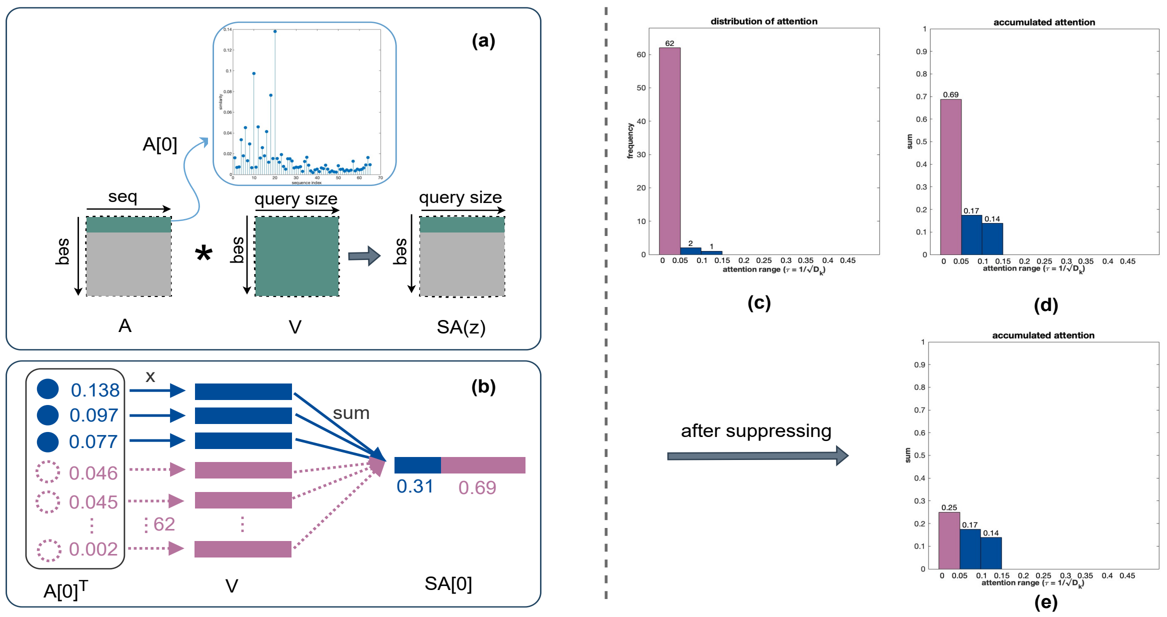


Figure 4.1: Our proposed SATA strategy. (a) The multi-head self-attention module in Vision Transformers. Each row in A represents attention weights corresponding to all sequences in $\mathbf{v}$. (b) A closer look at how to get the first sequence $SA[0]$ after applying attention. We set the threshold to 0.05. The blue part denotes larger attention weights and the purple is for trivial ones. We get up to 62 trivial attention weights and sum up to 0.69 in the entire attention in SA[0] compared with 0.31 from similar sequences. (c) The distribution of attention weights. (d) Accumulated attention within each bin. (e) The result of suppressing trivial weights by our approach. Even if single attention trivial weight contributes little, the accumulated trivial attention is still dominating, which is harmful when the attention contains much noise as in shallow layers of some backbones.

to improve the performance is to find more inductive biases to include, which is tedious. The local attention also sets a lower performance ceiling by eliminating much necessary non-local attention, which is in contrast to Vision Transformers. On the other hand, the stronger feature extraction ability of Vision Transformers can perfectly make up for the lack of data on small datasets. As a result, Vision Transformers show promising direction for those tasks.

To adapt Vision Transformers to data-efficient tasks, some researchers focus on transfer learning [86, 158, 176], semi-supervised learning and unsupervised learning to leverage large datasets. Others dedicate to self-supervised learning or other modalities to dig the inherent structure information of images themselves [25]. For supervised learning, one path is to integrate convolution operations in Vision Transformers to increase their locality. Another approach is to increase efficiency by revising the structure of Vision Transformers themselves [131]. The proposed method belongs to the second category.

The main transformer blocks include a multi-head self-attention (MHSA) module and a multi-layer perceptron (MLP) layer, along with some layer normalization, where the MHSA module is key to enriching each sequence by including long-range dependencies with all other sequences, i.e. attention. Intuitively, this attention module is expected to have larger coefficients for those sequences with higher similarity while smaller values for those less similar as the example $A[0]$ in Figure 4.1(a). In this way, all sequences can be enhanced by other similar sequences. However, this only considers single similarities themselves, but not their accumulation. Taking a closer look at the dot product operation on how each weighted sequence is obtained in Figure 4.1(a), we can find it is from weighting each sequence with attention coefficients and then summing up into one sequence as shown in Figure 4.1(b). This is problematic when the sequence length is large and those less similar sequences are noise. When the similarities are added from all less similar sequences, the accumulated sum can be even greater than the largest similarity as in Figure 4.1(d) caused by the small-value but large-amount trivial attention coefficients. This means the accumulated trivial attention dominates the attention, which brings much noise to the convergence of the Transformer. As a result, the trivial attention would hinder the training of the Transformer on small datasets. To solve this rooted problem in Vision Transformers and make it better deploy on small datasets, we proposed to suppress all trivial attention and hence the accumulated trivial attention to make sequences with higher similarity dominant again.

The contributions of this chapter are summarized below.

1. We found the accumulated trivial attention inherently dominates the MHSA module in vision Transformers and brings many noises on shallow layers. To cure this problem, we propose Suppress Accumulated Trivial Attention (SATA) to separate out trivial attention first and then decrease the selected attention.

2. We propose a *trivial weights suppression transformation (TWIST)* to control accumulated trivial attention. The proposed transformation is proved to suppress the trivial weights to a portion of maximum attention.

3. Extensive experiments on CIFAR-100 and Tiny-ImageNet demonstrated up to 2.3% gain in accuracy by using the proposed method.

4.2 Related Work

Vision Transformer has become a powerful counterpart of CNN in computer vision tasks since its introduction in 2020 [40], benefiting from its power to capture long-term dependencies. This ability is brought by their inherent structures in ViT, including the MHSA attention module which enhances each sequence with all other sequences, and MLP layers to model the relationships across all sequences. Including global attention also has weaknesses, like requiring large datasets to train, unlike the local attention in CNN. However, such large datasets are not easily accessible in many cases considering both time and effort cost in labeling and maintaining, e.g. rare diseases in the medical field. One direct solution is to search for more data, either borrowing data or knowledge from available large datasets and applying it to small datasets like transfer learning [86] and distillation [158, 176] or digging other information like self-supervised learning [99, 11, 61] and other modalities to exploit available labels [137, 145]. Another folder is to adjust the structure of transformers. For instance, integrate convolutional layers to transformers to mitigate its reliance on the amount of data like CvT [175], LeViT [51], CMT [55] and CeiT [192], design efficient attention modules to replace the quadratic computation complexity MHSA as Reformer [85], Swin [108], Swin-v2 [107], Twins [32], HaloNet [163] and Cswin [39], or remove MLP layers [38].

Regarding the MHSA module in vision Transformers, previous works can be divided into two paths according to the components in the attention function, input and the function itself. For the input of MHSA module, i.e. $\mathbf{qk^T}$, Swin [108] and Swin-v2 [107] calculate attention within windows instead of full sequences, CvT [175] uses convolutional layers to replace the linear layers to get $\mathbf{kqv}$. And there are also some works argue that the Softmax function in original vision transformers can be revised (e.g. adding learnable temperature [95]) or even replaced with other functions (e.g. l_1 norm in SimA [86] and Gaussian kernel in SOFT [114]). In this work, we post-process the results after the Softmax function, which can be categorized into the attention function folder.

The attention module is designed to focus on more alike sequences and less on different ones. This is based on the premise that all sequences are clear and include little noise. In this way, attention can enhance each sequence with alike sequences and useful signals get emphasized. However, this does not hold true when the features contain much noise. As shown in Figure 4.1, the example attention for one sequence $A[0]$ looks reasonable, with less similar attention weights but many trivial weights. However, when we check the sum of all trivial weights, it is far greater than the maximum attention. Just imagine all these sequences assigned trivial attention are harmful sequences, the accumulated attention from trivial weights is dominating the whole attention and even covers it. This happens when attention contains too much noise, like some shallow layers as mentioned in [172]. For shallow layer features, the image is a natural signal and has low information density. In other words, the image itself contains much noise, like the background of an object. This noise can even extend to several shallow layers due to the limitation of current feature extraction models, making shallow layers contain much more noise than deeper ones. However, determining the boundary for “shallow" is difficult since it is dependent on the depth of the model, feature extraction of the model, and the noisy degree of datasets. Thus, we designed a learnable suppressing scale s for all layers, avoiding finding this boundary by adding little computation.

4.3 Method

This section first introduces the MHSA module in Vision Transformers and its limitations, followed by the proposed suppressing steps, setting a threshold to separate out trivial attention coefficients and then decreasing their sum as shown in Figure 4.2.

4.3.1 Revisit MHSA

MHSA modules [40] in vision Transformers is the key to enriching sequence embedding in capturing long-range dependencies. To get this, input $\mathbf{z} \in \mathbb{R}^{\mathbf{N}\times\mathbf{D}}$, where N is the length of sequence

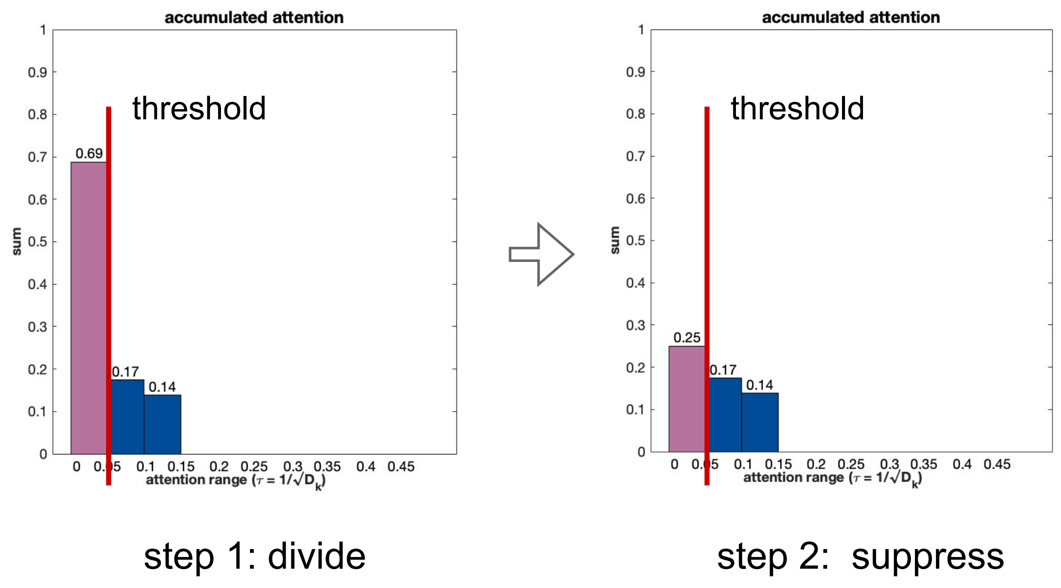


Figure 4.2: Our proposed suppressing method SATA. Step 1: divide trivial and non-trivial weights (0.05 in this example). Step 2: apply TWIST transformation on trivial weights.

and D is the dimension of sequence, is first passed through linear layers to get $\mathbf{q}$, $\mathbf{k}$ and $\mathbf{v}$. Then calculate the attention A to weight each sequence.

$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U}_{\mathbf{qkv}} \tag{4.1}$$

$$A = softmax(\mathbf{q}\mathbf{k}^{\mathbf{T}} / \sqrt{\mathbf{D_h}}) \tag{4.2}$$

Finally, compute a weighted sum across all sequences to get the final enhanced sequences, $SA(z)$ in Figure 4.1(a).

$$SA(\mathbf{z}) = \mathbf{A}\mathbf{v} \tag{4.3}$$

Taking a closer look at the detailed operations in Equation (4.3) as in Figure 4.1(b), to get the first row $SA[0]$, V is first weighted by all elements of the first row of A, and the weighted sequences are summed up to one sequence. One example of the attention weights $A[0]$ on the CIFAR-100 dataset can be found in Figure 4.1(a), where several weighs are larger, indicating the corresponding sequence is more similar to the current query sequence and hence with higher attention, while most of them are small indicating less similarity and importance.

We further explore the statistics of $A[0]$, the distribution of attention as in Figure 4.1 (c), where each bar denotes the number of attention weights in each similarity range. As in the example, up to 62 attention weights are below 0.05 indicating less similar sequences, while only 3 weights are greater than it. Surprisingly, after calculating the sum of attention weights in each bar and getting the graph in Figure 4.1 (d), the sum of trivial weights (below 0.05) is far larger than the sum of important weights, i.e. SA[0] is composed of more information from trivial sequences than non-trivial sequences. In other words, trivial attention is dominating the MHSA altogether. This also means the MHSA modules bring more noise than information, making it converge slowly, especially on small datasets. To handle this, we need to first set a threshold to separate trivial/non-trivial attention weights and then suppress trivial ones.

4.3.2 Divide

The first step is to get all trivial attention weights before suppressing them. Here we use a suppression threshold to divide the attention weights into trivial and non-trivial ones. Those attention weights below the suppression threshold are regarded as trivial weights, otherwise as non-trivial weights. However, there are also some choices to set the threshold.

Relative or absolute? A relative threshold is a portion t of a value, e.g. the maximum attention weight x_m in a row. i.e.

$$threshold = t * x_m \tag{4.4}$$

Compared to this, an absolute threshold is a given value t. i.e.

$$threshold = t \tag{4.5}$$

Notice that the absolute attention weights depend on the length of the sequence, the function to get it (e.g. temperature in Softmax function), datasets and so on. We set it a relative threshold as in Equation (4.4), where t is the relative scale. This dividing process can be achieved by multiplying the original attention with a mask M with '1' for trivial positions and '0' for nontrivial ones as below:

$$M = \begin{cases} 1, & attention \leq t * x_m \\ 0, & elsewhere \end{cases} \tag{4.6}$$

Then the final attention after normalization can be obtained by multiplying the original attention with the mask M as below.

$$A' = M \odot A \tag{4.7}$$

where $\odot$ denotes element-wise product.

4.3.3 Suppress

To suppress the sum of trivial attention, we transform each trivial attention weight from x_j to x'_j and call this transformation as TWIST.

Lemma 1. *Given n positive attention weights* $x_1 + x_2 ... + x_k + x_{k+1} + ... + x_{n-1} + x_m = 1$*, where* $x_1, ..., x_k$ *are trivial weights, less than a threshold* T*,* x_m *is the maximum weight and* $x_{k+1}, ..., x_{n-1}$ *are the rest of weights. If*

$$x_j{}' = \frac{x_j^2}{\sum_{i=1}^{k} x_i}, j = 1...k \tag{4.8}$$

Then,

$$\sum_{i=1}^{k} x_i{}' \leq x_m \tag{4.9}$$

and

$$x'_j \leq x_j, j = 1..k \tag{4.10}$$

Proof. Since for all x_j, $0 < x_j \leq x_m \leq 1$ where $j = 1...k$, we have $x_j^2 \leq x_j * x_m$. Then

$$\sum_{j=1}^{k} x_j^2 \leq (x_1 + x_2 + ... + x_j) * x_m \tag{4.11}$$

Dividing both sides by the sum yields

$$\sum_{j=1}^{k} x_j^2/(x_1+x_2+...+x_j) \leq x_m \tag{4.12}$$

or

$$\frac{\sum_{j=1}^{k} x_j^2}{\sum_{i=1}^{k} x_i} \leq x_m \tag{4.13}$$

Rewrite the numerator,

$$\frac{x_1^2+x_2^2+...+x_k^2}{\sum_{i=1}^{k} x_i} \leq x_m \tag{4.14}$$

which is exactly Equation (4.9) after substituting Equation (4.8).

To prove Equation (4.10), we only need to prove:

$$\frac{x_j^2}{\sum_{i=1}^{k} x_i} \leq x_j, j=1...k \tag{4.15}$$

Dividing both sides by positive value x_j yields

$$\frac{x_j}{\sum_{i=1}^{k} x_i} \leq 1, j=1...k \tag{4.16}$$

As x_j is one of the items in the denominator, Equation (4.16) holds true. $\square$

Lemma 1 means if we want to make the sum of trivial weights less than the maximum weight, we can simply transform x_j to ${x_j}'$ based on Equation (4.8), and attention weights after transformation are always no more than original weights. We can also add a scale s on both side of Equation (4.14) to make the sum smaller than a portion s ($s \geq 0$) of the maximum. As a result, our final transformation on $x_1,...x_k$ to suppress the accumulated trivial attention weights is

$${x_j}' = s * \frac{x_j^2}{\sum_{i=1}^{k} x_i}, j=1...k \tag{4.17}$$

where the suppressing scale s is learnable. We name this transformation in Equation 4.17 as *trivial*

	lr1 (model)	lr2 (CIFAR-100)	lr2 (T-ImageNet)
ViT	0.003	$7e^{-5}$	0.001
PiT	0.001	0.001	$3e^{-4}$

Table 4.1: Learning rates for model (lr1) and suppressing scale s (lr2) on CIFAR-100 and Tiny-ImageNet.

weights transformation (TWIST). This transformation can guarantee

$$\sum_{i=1}^{k} x_i' \leq s * x_m \tag{4.18}$$

4.4 Experiments

This section presents experiment settings, results and discussions after implementing the suppressing of Vision Transformers.

4.4.1 Settings

We perform image classification on small datasets, including CIFAR-100 [87] and Tiny-ImageNet [93]. CIFAR-100 includes 60,000 images with size 32×32, 50,000 for train split and 10,000 for validation split. Tiny-ImageNet has 100,000 and 10,000 64×64 images for train and validation split respectively. Following settings in [95], we perform data augmentations including CutMix [194], Mixup [203], Auto Augment [35], Repeated Augment [36], regularization including random erasing [209], label smoothing [157] and stochastic depth [75]. Optimizer is also AdamW [112]. The batch size is 128 and all models are trained for 100 epochs on one A100 GPU. The learning rate of model is set to 0.003 for ViT [40] and 0.001 for PiT [65]. The suppressing scale s is initialized to 0.5 and its learning rate can be found in Table 4.1 *lr2*. The threshold coefficient t is fixed to 0.05 for PiT on CIFAR-100 and 0.1 for all other experiments.

Model	Param (M)	CIFAR-100	T-ImageNet
ResNet56*	0.9	76.36	58.77
ResNet110*	1.7	79.86	62.96
EfficientNet B0*	3.7	76.04	66.79
ViT	2.8	73.70	56.45
SATA-ViT (ours)	2.8	**74.93**(*+1.23*)	**58.77**(*+2.32*)
PiT	7.1	72.31	57.87
SATA-PiT (ours)	7.1	**73.52**(*+1.21*)	**58.15**(*+0.28*)

Table 4.2: Classification results on CIFAR-100 and Tiny-ImageNet dataset. Top 1 accuracy (%) is reported.

4.4.2 Integrating with Vision Transformers

To evaluate the effect of our proposed suppressing method, we integrate it with both the original ViT [40] and PiT [65] following the scale for small datasets in [95], where the patch size is set to 4 for CIFAR-100 and 8 for Tiny-ImageNet, resulting in the same number of tokens 64 and 1 class token. From the results in Table 4.2 we see that the accuracy for ViT is increased by 1.23% on CIFAR-100 and up to 2.32% on Tiny-ImageNet by integrating our trivial attention suppressing module. It also boosts PiT on both datasets with up to 1.21% on CIFAR-100. These improvements demonstrate that taking care of trivial attention weights explicitly is necessary and suppressing them can improve performance. Besides this, examining the effect of our method on a different scale of tokens may be interesting future work.

4.4.3 Ablation Study

To verify how each module works, we decompose each component in the proposed suppressing module using ViT on Tiny-ImageNet. Specifically, to understand the necessity of suppressing, we let s be a hyperparameter as t and perform a grid search on both hyperparameters. The best accuracy and its search result are listed in Table 4.3. Comparing row 2 with row 0 in Table 4.3, we observe that suppressing brings 1.24% more accuracy to ViT after grid search. In addition, we also set suppressing scale s to 0 and find that accuracies for most threshold t are near 56.45% when no suppressing exists as shown in Table 4.4. This indicates that those trivial attention weights are still

index	suppress	threshold	Top 1
0	-	-	56.45
1	0	0.075	57.47
2	0.75	0.1	57.69
3	learnable	0.1	**58.77**

Table 4.3: Ablation study. In this table, we compare suppressing with grid search *s*, suppressing to 0, suppressing with learnable *s* and no suppressing.

t \ s	1	0.75	0.5	0.25	0.1	0
0.1	56.43	**57.69**	56.22	57.30	56.46	56.51
0.075	56.79	57.06	56.74	56.29	56.99	57.47
0.05	56.11	56.72	56.47	56.87	57.24	56.47
0.025	56.06	56.85	56.71	56.65	56.95	56.31
0.01	57.23	56.31	56.63	56.45	56.44	56.55
0	56.45					

Table 4.4: Grid search on *s* and *t* for ViT on Tiny-ImageNet dataset. The baseline without suppressing is the last row when $t = 0$ and the last column denotes removing the attention directly.

helpful.

Grid Search. For grid search, we select *s* from $[1, 0.75, 0.5, 0.25, 0.1, 0]$ and *t* from $[0.1, 0.05, 0.025, 0.01, 0]$. The learning rate is 0.003, the same as ViT on CIFAR-100 and Tiny-ImageNet when no suppressing exists. In Table 4.4, we can find the best result is from $s = 0.75$ and $t = 0.1$ on Tiny-ImageNet, while it is $s = 0$ and $t = 0.075$ for ViT on CIFAR-100 according to 4.5, which means we get a better performance when removing those attention directly. We also select the initialization values for learnable *s* from their average performance. From both Table 4.4 and Table 4.5, we can see that the accuracies are increased in most cases with suppressing compared with the baseline when no suppressing happens, i.e. when the last row $t = 0$ in both tables. The last column in Table 4.4 shows that deleting the attention will hurt the performance most of the time on Tiny-ImageNet while it helps all the time on CIFAR-100. We also observe that the relationship between *s* and *t* is complicated, neither linear nor inverse. This is reasonable since both parameters are highly correlated.

Learnable *s*. Learned *s* of ViT on both CIFAR-100 and Tiny-ImageNet can be found in Figure

$t \backslash s$	1	0.75	0.5	0.25	0.1	0
0.1	74.61	74.51	74.46	74.07	74.42	74.50
0.075	74.81	74.01	73.89	74.65	73.97	**74.75**
0.05	74.71	74.18	74.82	74.46	74.32	74.96
0.025	74.34	73.39	73.93	74.34	74.21	73.86
0.01	74.15	74.75	74.60	74.15	74.49	73.71
0	73.70					

Table 4.5: Grid search on s and t for ViT on CIFAR-100 dataset. The baseline without suppressing is the last row when $t = 0$ and the last column denotes removing the attention directly.

4.3. Note that s denotes the suppressing scale to the maximum attention for each sequence. In Table 4.3 we can see, for the first 2 layers on Tiny-ImageNet and the first 4 layers on CIFAR-100 dataset, the sum of trivial weights is ensured to be below the maximum. While for deep layers, the sum of trivial attention can be scaled up to several times the maximum. This is reasonable since for deeper layers, features contain less noise, and hence suppressing trivial attention weights is no longer necessary. Instead, another function of our SATA works is to adjust the distribution of attention by increasing trivial attention weights to be more comparable with the maximum and hence influence the distribution.

4.4.4 Comparison with different normalization

Softmax with temperature. The original Softmax can be denoted by

$$A = softmax(\tau \mathbf{q}\mathbf{k}^{\mathbf{T}}) \tag{4.19}$$

where τ is the temperature to control the scale of the Softmax function. In the original ViT [40], the normalization of attention modules is

$$A = softmax(\frac{\mathbf{q}\mathbf{k}^{\mathbf{T}}}{\sqrt{D_h}}) \tag{4.20}$$

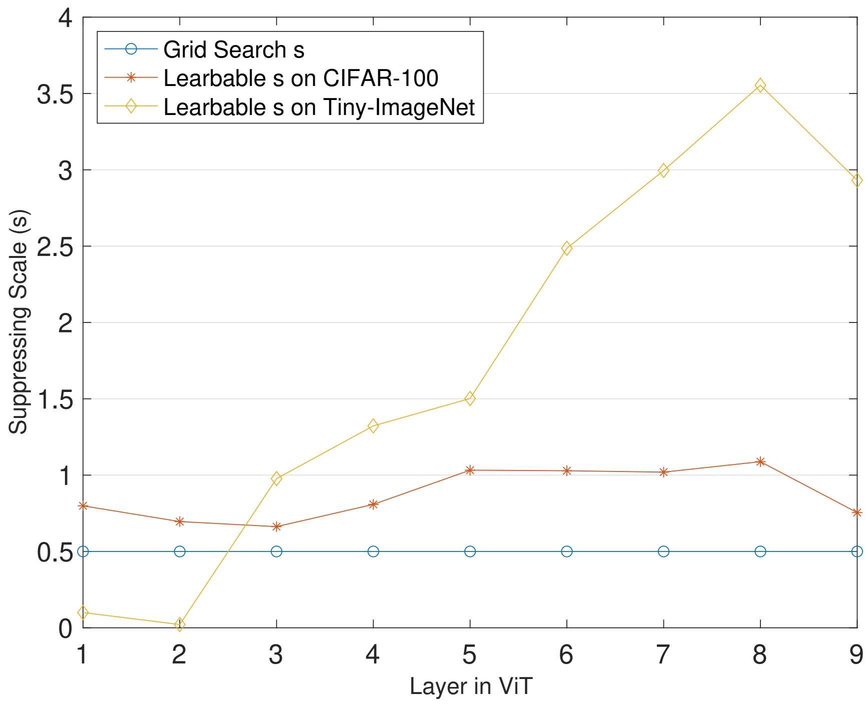


Figure 4.3: Learnable s vs. fixed s from grid search for ViT.

where they use $1/\sqrt{D_h}$, the dimension of the head, as the temperature τ of the Softmax function to adjust the distribution of attention weights. The higher the temperature τ, the sharper the Softmax function as shown in Figure 4.4. Figure 4.4 shows that small values become even smaller and large ones are even larger when increasing the temperature of Softmax function from $\tau = 1$ to $\tau = 2$. This can also enlarge the ratio of larger values to smaller values, which is similar to the effect of our proposed suppressing trivial weights. However, increasing the temperature of Softmax cannot solve this. More specifically, it also brings side effects along with the suppression of trivial attention weights.

To check this, we first conduct experiments by setting different temperatures. The results can be found in Table 4.6. Table 4.6 shows that, compared with the Softmax with default temperature $\tau = 1/\sqrt{D_h}$ in original ViT, increasing the temperature to $2\times$, $3\times$ and $4\times$ all improves the performance on Tiny-ImageNet dataset. And the best result is from $\tau = 4/\sqrt{D_h}$ Tiny-ImageNet.

In summary, Softmax with higher temperature can mitigate the dominating accumulated trivial

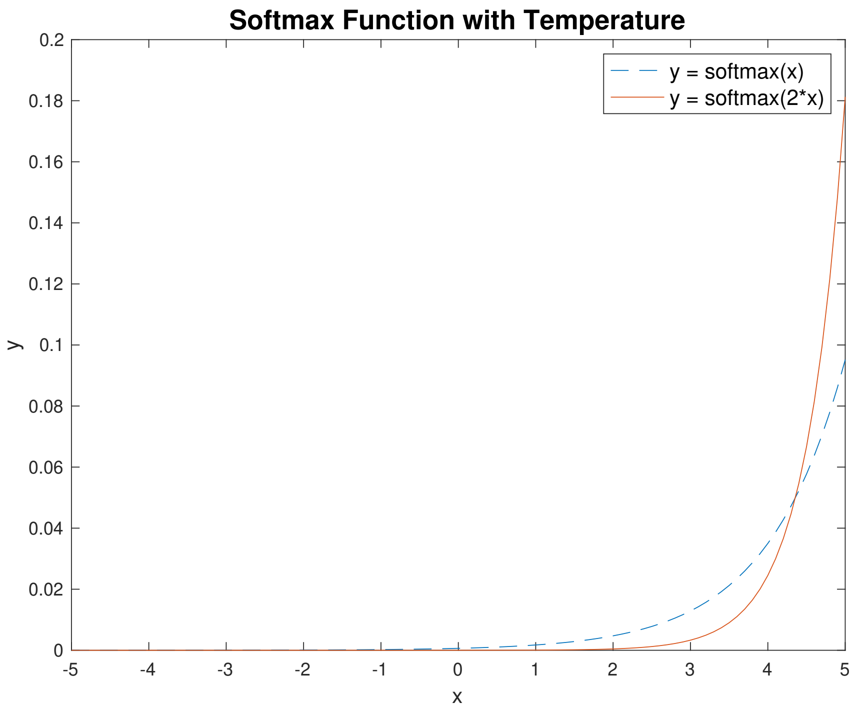


Figure 4.4: Softmax function with temperature. With the increase of temperature, the difference between large values and smaller values gets enlarged by decreasing smaller values and increasing greater ones.

attention effect partly at the cost of changing the distribution of sensitive larger attention weights. While our proposed suppressing method decouples trivial and non-trivial attention weights to solve the dominating effect, making us available to take advantage of both adjusting trivial attention weights according to noise level and adjusting the distribution of non-trivial attention weights with higher temperature freely. The results to build our SATA on softmax with temperature are shown in Table 4.6. According to Table 4.6, the performance for Softmax with temperature is improved with the increase of τ, while it decreases after applying our proposed module. In most cases e.g. $\tau = 1\times, 2\times, 3\times$ of default temperature, SATA module further boosts the accuracy. This indicates that the performance increase of Softmax with temperature is from the enlargement of the gap between larger and smaller values, which is good for shallow layers while harming deeper layers. Our module can adjust trivial attention weights in both situations, noisy or noiseless, and especially when both happen in the same model while requiring different handling.

model	$\tau' = 1$	$\tau' = 2$	$\tau' = 3$	$\tau' = 4$
ViT + τ	56.45	57.20	57.21	**57.81**
ViT + τ + SATA	**58.77**	58.12	57.72	57.58

Table 4.6: Combining Softmax with different $\tau = \tau' \times \frac{1}{\sqrt{D_h}}$ on Tiny-ImageNet. We adjust the learning rate for s after adding τ. The best results are reported.

model	CIFAR-100	Tiny-ImageNet
ViT	73.70	56.45
ViT + LSA	75.40	57.82
ViT + LSA + SATA	75.47	58.28

Table 4.7: Integrating with LSA module.

Diagonal suppressing. This module LSA is proposed in [95] considering that the attention in diagonal is from self-attention in the MHSA module for Vision Transformers, which is not necessary since the skip connection in the MHSA module will add the attention itself with a larger ratio compared with the self-attention in the attention branch. However, this self-attention usually is larger than other attention, leaving less room for other attention to get large values. To this end, they propose to manually set the diagonal to an extremely small value, making the attention after Softmax small. As the goals of this module and our SATA module are different, we can combine both methods together to yield better attention. The results are shown in Table 4.7. According to this table, the performance is increased on both CIFAR-100 and Tiny-ImageNet after implementing the LSA module and our module further adds up to 0.46% on Tiny-ImageNet.

4.5 Conclusion

In this chapter, we have examined the MHSA modules in Vision Transformers and discovered that attention from the trivial sequence is dominating the final attention after accumulation, affecting its performance by including more noise than information on shallow layers. This issue is not handled by the attention function e.g. Softmax. To solve this challenge, we propose to handle trivial weights explicitly by first separating out trivial attention weights with a relative threshold to the maximum attention and then adjusting them to a portion of the maximum attention weight. Experiments show up to 2.3% increase in accuracy, indicating this process is necessary to make the attention function work.

Chapter 5

Training Efficiency: SuperLoRA–Parameter-Efficient Unified Adaptation of Multi-Layer Attention Modules

Training efficiency aims to use fewer training iterations and/or memory usage to achieve comparable or even better performance, which saves resource consumption at training time for each downstream task. A commonly used method is to leverage pretrained large models like ViT and diffusion models, and then fine-tune on downstream tasks with only fewer parameters. In this way, both training time and memory required for downstream tasks are reduced greatly. This chapter introduces a unified adaptor, SuperLoRA, which reduced required trainable parameters to by 3 to 10 fold [22].

5.1 Introduction

Large neural network models are dominating machine learning recently with the emergence of exceptional models, such as Vision Transformer (ViT) [40], ConvNeXt [110] and Stable Diffusion [67] for vision tasks, and large language models (LLMs) including GPT [1], PALM2 [5], Gemini [4] and LLaMA2 [159] for natural language processing (NLP). However, the increased resource consumption and data requirement along with model size limits its generalization on downstream tasks. To solve this, Parameter-Efficient Fine-Tuning (PEFT) has been widely explored to fine-tune less parameters while retaining high performance. Among this, adapter-based technique like LoRA (Low-Rank Adaptation) [72] demonstrates superiority in its convenience of plug-and-play nature.

LoRA [72] approximates the weight updates of the base model by approximating the change

ΔW of each weight matrix as the product of two low-rank matrices. This decreases the required parameters from d^2 to $2rd$ when $r \ll d$, where d and r are weight size and the rank, respectively. Most LoRA variants work on solving the inherent *low-rank constraint* of matrix factorization, including LoHA (Low-rank Hadamard) [190], LoKr (Low-rank Kronecker) [190], and LoTR (Low Tensor Rank) [7]. However, we find these variants can be unified within our framework—SuperLoRA—with different hyper-parameters as shown in 5.1. Our proposed SuperLoRA framework is depicted in 5.1, which also yields to some new variants: LoNKr (**N**-split version of LoKr) and LoRTA (**T**ensor version LoRA). Additionally, we introduce two extended options, 1) reshaping ΔW to any arbitrary multi-dimensional tensor arrays before applying LoRA variants, and 2) splitting all ΔW into an arbitrary number of groups, which breaks the boundaries for ΔW across different weights. Moreover, to further compress the number of trainable parameters, a projection layer $\mathscr{F}$ with fixed parameters is inserted to map ΔW_{lora} to the actual ΔW. Accordingly, SuperLoRA provides more flexibility and extended functionality, controlled by a set of hyper-parameters as shown in 5.2. The contributions of this chapter are summarized as follows:

- We propose a new PEFT framework SuperLoRA which gracefully unifies and extends most LoRA variants.
- With projected tensor rank decomposition, SuperLoRA can adapt all weights across layers jointly with a wide range of adjustable parameter amount.
- We investigate the effect of tensor reshaping, grouping, random projection, and shuffling.
- We demonstrate high parameter efficiency for large ViT and diffusion models in two transfer learning tasks: image classification and image generation.
- Significantly reduce parameters by 3 to 10 folds.

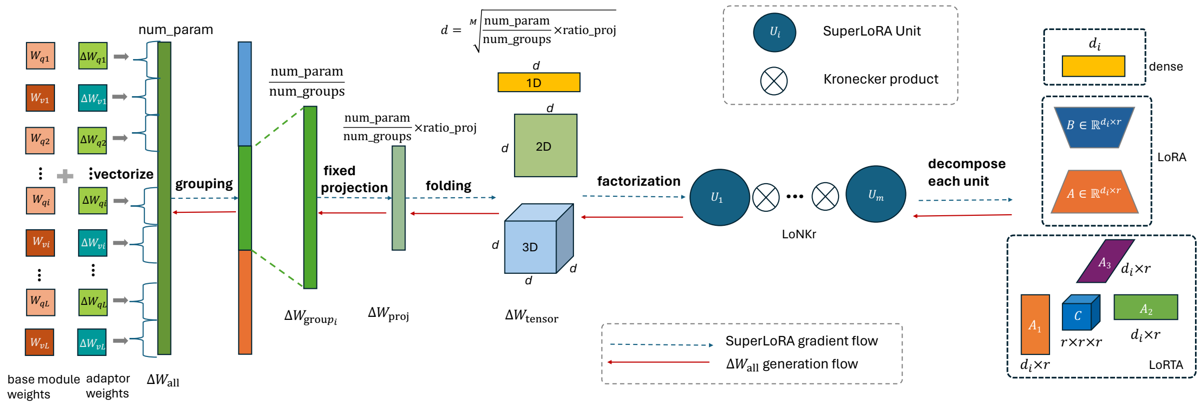


Figure 5.1: Schematic of SuperLoRA to fine-tune multi-layer attention modules at once with vectorizing, grouping, projection, folding, and factorization.

hyper-parameters settings	method
$\mathscr{F} = I$, weight-wise, $K = 1$, $C_{g1} = I$, $M = 1$, $A_{g11} \in \mathbb{R}^{d_{\text{in}}d_{\text{out}} \times 1}$	dense FT
$\mathscr{F} = I$, weight-wise, $K = 1$, $C_{g1} = I$, $M = 2$, $A_{g1m} \in \mathbb{R}^{d_m \times r}$	LoRA [72]
$\mathscr{F} = I$, weight-wise, $K = 2$, $C_{gk} = I$, $M = 2$, $A_{gkm} \in \mathbb{R}^{d_m \times r}$	LoKr [190]
$\mathscr{F} = I$, group-wise, $G = 1$, $M > 2$	LoTR [7]
$\mathscr{F} = I$, group-wise, $K > 2$, $C_{gk} = I$, $M = 2$, $A_{gkm} \in \mathbb{R}^{d_m \times r}$	LoNKr
$\mathscr{F} = I$, group-wise, $K = 1$, $M > 2$, $A_{gkm} \in \mathbb{R}^{d_m \times r}$	LoRTA

Table 5.1: Hyper-parameter settings in SuperLoRA and the resultant LoRA variant

5.2 Related Work

PEFT algorithms are widely explored in transfer learning tasks in both computer vision [79, 64, 78] and NLP fields [102, 96, 53, 60, 81] as it not only saves memory and time at fine-tuning, but also requires much less data to fine-tune, making it feasible to borrow the capacity from large models in few-data tasks. Adapter-based methods [17, 58, 70, 135], that freeze the base model weights and fine-tune only the additional adapter parameters, stand out since their plug-and-play nature enables many downstream tasks to share the same large model, leaving the adapter to hold only the task-specific information. The widely used method LoRA [72] and its extension [59, 211] assume that the weight correction term can be estimated by low-rank decomposition under the low-dimensional manifold hypothesis.

Addressing the inherent *low-rank constraint* of matrix factorization in LoRA, LoHA [190]

notation	description
r	rank of factorization
$\mathscr{F}$	mapping function
ρ	compression ratio
G	number of groups
M	order of tensor modes
K	number of splits

Table 5.2: Hyperparameters and notation.

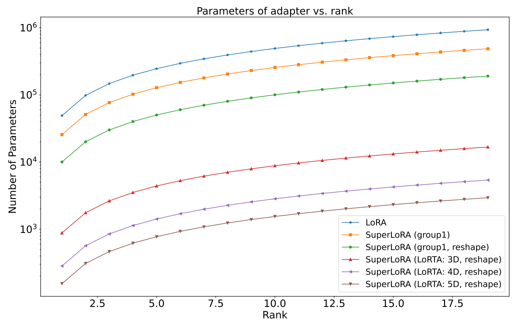


Figure 5.2: Required number of parameters.

divides ΔW into two splits and combines them with Hadamard product, and KronA [43] combines the two splits with a Kronecker product to enlarge the overall rank. LoKr [190] further extended KronA to convolutional layers. LoDA (Low-Dimensional Adaptation) [104] extended LoRA by introducing nonlinearity. Our SuperLoRA can nicely generalize and extend such variants.

Instead of approximating weight-wise updates, LoTR [7] jointly approximates all ΔW across the model with a careful handling to preserve the geometric meaning of each weight. Differently, SuperLoRA relaxes the geometrically meaningful boundaries by caring the total number of parameters and splitting it to any number of groups. For high-order tensor decomposition, LoTR employs more stringent Tensor Train Decomposition to deal with the core tensor explosion, while SuperLoRA coupled Tucker Decomposition with a fixed projection layer. Besides, their proposed methods are restricted to context when ΔW is the same high-order tensor, while with reshaping, SuperLoRA (LoRTA) can be applied to any weight shape.

Most recent work [18] decomposes each convolution kernel into a learnable filter atom and its non-learnable counterparts. The concept of filter atom is similar to the projection layer of SuperLoRA. However, it works on each convolutional kernels separately, resulting in a waste of parameters, while SuperLoRA considers the entire model jointly. Besides, the atom coefficients are obtained from matrix factorization, while SuperLoRA uses a fastfood projection [92], which is faster, simpler and more theoretically justifiable to exploit intrinsic dimensionality [2]. In addition, SuperLoRA can control the size of atoms directly while atoms in their method are restricted in factorization.

Local LoRA [82] aims to reduce memory consumption at fine-tuning by splitting large model into groups and then fine-tune group-by-group sequentially, but no adjustment on the LoRA structure was proposed. Instead, SuperLoRA focuses on how to split and assign LoRA for each group, which is a viable extension of Local LoRA.

5.3 Methodology

5.3.1 Low-Rank Adaptation (LoRA)

LoRA [72] assumes the update ΔW of each weight matrix W for fine-tuning can be approximated by low-rank mapping as $\Delta W = AB^\top$ ($[\cdot]^\top$ denotes matrix transpose), which is added to the frozen weight matrix as shown in 5.3a:

$$W' = W + \Delta W = W + AB^\top, \tag{5.1}$$

where $A \in \mathbb{R}^{d_{\text{in}} \times r}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$, and the rank r. With a smaller r compared with the matrix dimensions, it only requires $(d_{\text{in}} + d_{\text{out}})r$ parameters for each weight matrix, while full fine-tuning (FT) for dense $\Delta W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ results in $d_{\text{in}} d_{\text{out}}$ parameters. LoRA has been widely used in fine-tuning large models as much less trainable parameters save memory usage at training while retaining performance, making it easily adapted to downstream tasks with limited resources.

5.3.2 SuperLoRA

5.1 shows the overview of SuperLoRA, which is a generalization of LoRA variants to allow high flexibility in the weight update ΔW. SuperLoRA can be formulated as below:

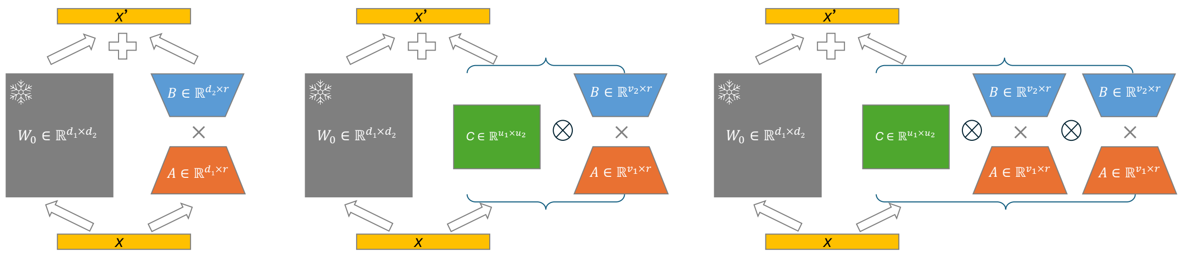
$$\Delta W_{\text{group}_g} = \mathscr{F}(\Delta W_{\text{lora}_g}) = \mathscr{F}\left(\bigotimes_{k=1}^{K}\left(C_{gk}\prod_{m=1}^{M} \times_m A_{gkm}\right)\right), \tag{5.2}$$

where $\mathscr{F}(\cdot)$ is a simple projection function applied on the results of SuperLoRA modules. We denote $\prod_{m=1}^{M} \times_m$ as tensor products from mode-1 to mode-M. Here, M represents the order of the reshaped ΔW_{lora_g} tensor modes, and high-order Tucker decomposition is employed to formulate this high-order tensor. SuperLoRA units in 5.1 are combined with Kronecker product across K splits in a proper shape. Depending on reshaping, each split has multiple choices including a combination of dense FT (1D), LoRA (2D), and high-order Tucker decomposition [161], where C_{gk} (core tensor) and all A_{gkm} (plane factors) are combined with mode-m tensor product.

For SuperLoRA, we first concatenate all $\Delta W \in \mathbb{R}^{d_i \times d_i}$ across multiple layers to get the total correction of $\Delta W_{\text{all}} \in \mathbb{R}^{\sum_i d_i^2}$. Then, ΔW_{all} is divided into g groups: $\{\Delta W_{\text{group}_g}\}$ for $g \in \{1, 2, \ldots, G\}$. Each LoRA module will then produce $\Delta W_{\text{group}_g}$. Finally, stretch $\Delta W_{\text{group}_g}$ to one dimension, fetch corresponding size of ΔW from those $\Delta W_{\text{group}_g}$ and add it to candidate weight matrix, e.g., query and value projection weights for attention modules across layers.

SuperLoRA and LoTR: While LoRA estimates ΔW in a weight-wise independent way, SuperLoRA considers the whole weights ΔW_{all} jointly. It can relax the restriction of the weight shape and geometric meaning of weight axis unlike LoTR. Here, the number of groups can be adjusted to balance between parameter amount and fine-tuning performance. When the number of groups is the number of weights and the group boundary matches the weight boundary, it corresponds to weight-wise LoRA. When the number of groups is $G = 1$, SuperLoRA corresponds to LoTR [7], but with an additional projection mapping $\mathscr{F}$.

Reshaping to regular tensor: Grouping multiple layers together by concatenating ΔW along one axis results in skew $\Delta W_{\text{group}_g}$, limiting the choice of ranks in LoRA modules and leading to worse approximation. For example, stacking query and value weight updates as $[\Delta W_{\text{q}}, \Delta W_{\text{v}}]$ will be of size $d_{\text{in}} \times 2d_{\text{out}}$, which is less efficient for LoRA as A and B matrices have unbalanced sizes. To solve this, we propose to reshape $\Delta W_{\text{group}_g}$ to a regular tensor: i.e., square-like 2D matrix, cubic-like 3D tensor, or high-order hyper-cubic tensors having same dimension size across all axes. This reshaping can reduce the dimension per axis in the order of $\mathscr{O}[N^{1/M}]$ for N being the number of stacking weights, that in return can allow higher rank size per plane factors. Several examples of


x'
$W_0 \in \mathbb{R}^{d_1 \times d_2}$
$B \in \mathbb{R}^{d_2 \times r}$
$A \in \mathbb{R}^{d_1 \times r}$
x
x'
$W_0 \in \mathbb{R}^{d_1 \times d_2}$
$C \in \mathbb{R}^{u_1 \times u_2}$
$B \in \mathbb{R}^{v_2 \times r}$
$A \in \mathbb{R}^{v_1 \times r}$
x
x'
$W_0 \in \mathbb{R}^{d_1 \times d_2}$
$C \in \mathbb{R}^{u_1 \times u_2}$
$B \in \mathbb{R}^{v_2 \times r}$
$A \in \mathbb{R}^{v_1 \times r}$
$B \in \mathbb{R}^{v_2 \times r}$
$A \in \mathbb{R}^{v_1 \times r}$
x

(a) LoRA (b) LoKr (c) LoNKr (weight-wise version, ours)

Figure 5.3: Overview of (a) LoRA; (b) LoKr; (c) LoNKr (weight-wise version, ours).

grouping and reshaping are discussed in 5.6.3, and its geometric analysis in 5.6.4.

SuperLoRA and LoKr/LoNKr: LoKr is depicted in 5.3b, which can be extended as shown in 5.3c. We call it LoNKr, which combines K splits composed of sub LoRA units through Kronecker products: i.e., $K > 2$ in 5.2. When $K = 2$, it reduces to LoKr but with an additional flexibility. For example, LoNKr can still adapt multiple attention modules at once with an adjustable group size G, unlike weight-wise adaptation of LoKr.

LoRTA: Folding a matrix $\Delta W_{\mathrm{group}_g}$ into high-order tensor (e.g., 3D, 4D, 5D) can decrease parameters with tensor rank decomposition, like Tucker decomposition, where $\Delta W_{\mathrm{group}_g}$ is represented by M 2D plane factors and one MD core tensor. We refer to this variant of SuperLoRA using Tucker decomposition as LoRTA. For example, when $M = 4$ and $K = 1$, we have 4D tensor rank decomposition as follows:

$$\Delta W_{\mathrm{group}_g} = C_{gK} \times_1 A_{gK1} \times_2 A_{gK2} \times_3 A_{gK3} \times_4 A_{gK4} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}, \tag{5.3}$$

where $C_{gK} \in \mathbb{R}^{r_1 \times r_2 \times r_3 \times r_4}$ is a reshaped core tensor, $A_{gKm} \in \mathbb{R}^{d_m \times r}$ is a mode-m 2D plane factor, and $\times_m$ denotes mode-m tensor product. For simplicity, we set a rank $r = r_m$ for any mode $m \in \{1, 2, \ldots, M\}$.

The core tensor may cause the explosion of parameters with larger rank as the number of parameters is exponential as r^M. It may be resolved by restricting the core tensor to be strongly diagonal or identity. For instance, $M = 2$ with identity core tensor $C_{gK} = I$ corresponds to the original LoRA, and $M = r = 1$ identity core tensor corresponds to the dense FT. When using diagonal core tensor, it reduces to canonical polyadic (CP) decomposition. 5.2 shows the number of required parameters with CP decomposition. One can see that higher-order tensor decomposition can significantly reduce the total number of trainable parameters at a certain rank. We provide another solution without limiting the core tensor by coupling with the projection layer $\mathscr{F}$ below.

Projection: Most LoRA variants assume the resultant $\Delta W_{\mathrm{lora}_g}$ from LoRA modules is the final ΔW added to W directly. However, we can further modify the $\Delta W_{\mathrm{lora}_g}$ through a simple mapping: e.g., we can project much smaller $\Delta W_{\mathrm{lora}_g}$ into larger final $\Delta W_{\mathrm{group}_g}$ to improve the parameter efficiency. The compression ratio is $\rho = |\Delta W_{\mathrm{lora}_g}|/|\Delta W_{\mathrm{group}_g}|$, where $|\cdot|$ denotes the total number of elements of the tensor. We consider a random projection layer based on the fastfood projection [92] to map $\Delta W_{\mathrm{lora}_g}$ to $\Delta W_{\mathrm{group}_g}$.

Specifically, the fastfood projection is performed as follows:

$$\Delta W_{\mathrm{group}_g} = \mathscr{F}(\Delta W_{\mathrm{lora}_g}) = \mathsf{vec}[\Delta W_{\mathrm{lora}_g}]\,\mathscr{H}'\,\mathsf{diag}[\mathscr{G}]\,\Pi\,\mathscr{H}\,\mathsf{diag}[\mathscr{B}], \tag{5.4}$$

where $\mathsf{vec}[\cdot]$ is a vectorization operator, $\mathsf{diag}[\cdot]$ denotes a diagonalization operator, $\mathscr{H}$ is Walsh–Hadamard matrix, $\mathscr{H}'$ is its truncated version, $\mathscr{G}$ is a random vector drawn from normal distribution, Π is a random permutation matrix for shuffling, and $\mathscr{B}$ is a random vector drawn from Rademacher distribution. It is a fast Johnson–Lindenstrauss transform with log-linear complexity due to the fast Walsh–Hadamard transform, and no additional parameters are required when the random seed is predetermined. Further, a nonlinear function such as tanhshrink can be added to make this layer nonlinear. To avoid introducing extra parameters for the projection layer, weights of this projection layer is reproduced on the fly with a known random seed and fixed during training and inference.

Shuffling: Another simple projection is to use a shuffling function without compression. It can be achieved by simplifying the fastfood projection without $\mathscr{H}$, $\mathscr{H}'$, $\mathscr{G}$, and $\mathscr{B}$ but with the random permutation Π and projection ratio $\rho = 1$. As SuperLoRA updates all weights at once, we have a flexibility in a way to distribute $\Delta W_{\text{group}_g}$ towards which element of W. To understand how the weight assignment method impacts, we consider a random shuffling case for the projection function $\mathscr{F}$. Several projection variants including shuffling are discussed in 5.6.6.

5.4 Empirical Experiments

We evaluate SuperLoRA on two different transfer learning tasks: image classification and image generation. Below, we describe the experiment settings and results for these tasks as well as visualization of the generated images.

5.4.1 Classification transfer task

5.4.1.1 Settings:

Transfer learning for image classification is conducted between ImageNet21k [37] and CIFAR100 [87] based on a ViT-base [40] model. Specifically, a ViT model[1] pretrained on ImageNet21k is loaded with a new classifier head[2] for classifing CIFAR100 dataset. More details of the ViT model are described in 5.6.1. The query and value projection layers in the attention modules are then fine-tuned with SuperLoRA. All layers of the pretrained ViT model are frozen except the SuperLoRA parameters and the new classifier head. The model is trained for 5,000 steps with the stochastic gradient descent (SGD) optimizer, with a batch size of 128 and a learning rate of 0.05. The OneCycleLR [153] scheduler is used.

We evaluated SuperLoRA with grouping with/without reshaping to square-like for 2D $\Delta W_{\text{group}_g}$, reshaping version for higher-order $\Delta W_{\text{group}_g}$ including 3D, 4D and 5D. The fixed projection layers are inserted to SuperLoRA with reshaping (2D version) and also dense. Original weight-wise LoRA is also examined for comparison by setting the number of groups to the number of query and value weights (24 for 12-layer ViT-base) as all projection weights for ViT-base are equal

[1]`https://huggingface.co/google/vit-base-patch16-224-in21k`
[2]`https://github.com/bwconrad/vit-finetune`

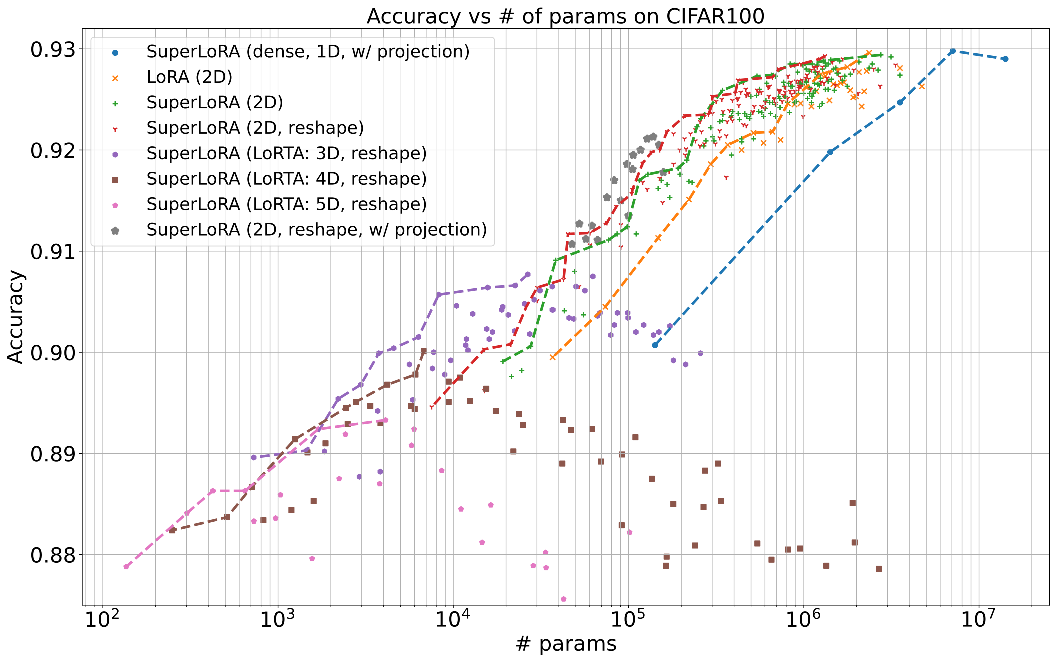


Figure 5.4: Classification on CIFAR100 dataset with SuperLoRA.

size. Each correction weight is of size 768×768 as the projection weight for query/value, resulting in 14M parameters. The number of groups are selected from $G \in \{1, 4, 8, 12, 24\}$, every two ranks are evaluated from 1 to 32 for most methods and LoRA α is set to be the same as rank. Except for most cases, more ranks are needed to span the parameter axis well, including larger ranks from 34 to 128 and smaller ranks below 8 for LoRTA. Projection compression ratio is from $\rho \in \{0.5, 0.25, 0.1, 0.01\}$, and the fixed projection parameters are shared across all groups in our experiments.

5.4.1.2 Results:

Classification results versus the number of parameters are shown in 5.4 with Pareto frontier lines. Comparing group-wise SuperLoRA (2D with/without reshape) with weight-wise LoRA, we can find that SuperLoRA versions show better performance in terms of the trade-off between classification accuracy and the number of parameters. Noticeably, we observe three to four times advantage in terms of parameter efficiency for the same accuracy. As the largest number of groups is set to 24 (i.e. LoRA), it indicates smaller number of groups are superior. This may be because ViT model is excessively large for the CIFAR100 dataset, with much more redundant weights. Grouping weights and layers together can reduce noise brought by the redundancy. With reshaping $\Delta W_{\text{group}_g}$ to a square matrix, classification accuracy further increases in the lower parameter regime and the range of parameters the model can cover becomes wider as higher rank can be used while maintaining a smaller number of parameters.

To discuss the effect of higher-order tensor folding, the order M is set to be 3, 4 and 5 for SuperLoRA (i.e. LoRTA) as well as 2. For $M = 2$ cases with 2D tensor, we use identity core tensor like typical LoRA. With the increase of order from 2 to 5, higher order takes place lower-order at fewer-parameter regimes. Moreover, data points for high-order LoRTA show a hill-like trend with the increase of parameters. This may be caused by the inefficient core tensor, which increases parameters rapidly without benefiting the accuracy. When comparing the lowest rank LoRA (which achieves around 0.9 accuracy with about 4×10^4 parameters), our LoRTA (3D) significantly improves the accuracy by about 1% at the comparable number of parameters, and more significantly reduces the number of parameters by 10 folds to keep the comparable accuracy of 0.9.

Finally, we address the impact of the random but fixed projection layer $\mathscr{F}$. Fixed fastfoood projection is applied on SuperLoRA (1D, dense) and SuperLoRA (2D, reshape). For 1D dense with projection, the plot for a projection ratio of $\{1, 0.5, 0.25, 0.1, 0.01\}$ is placed from right to left in 5.4. The classification accuracy dropped less than 1% from projection ratio 1 to 0.1 (i.e. 90% less parameters), but it is worse than LoRA. For SuperLoRA (2D, reshape, w/projection), we evaluated a projection ratio of $\{0.5, 0.25, 0.1\}$. To get some results of projection for the number of parameters around 10^4 and 10^5, we select a few settings for SuperLoRA (2D, reshape) with $G = 1$ as shown in the figure with a marker of dark stars. Most projection results demonstrate better accuracy compared with other SuperLoRA settings without projection in the same number of parameters level. This result shows a smaller adapter with fixed projection layer is a strong functionality to improve the parameter efficiency of SuperLoRA.

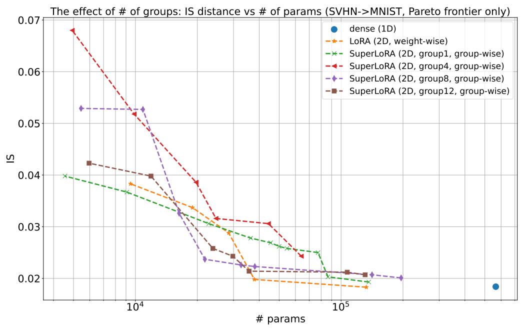
Another experiment of transfer learning from ImageNet1k to CIFAR10 classification shows similar results, achieving 3 to 10-fold improved parameter efficiency, as discussed in 5.6.7.

5.4.2 Image generation transfer task

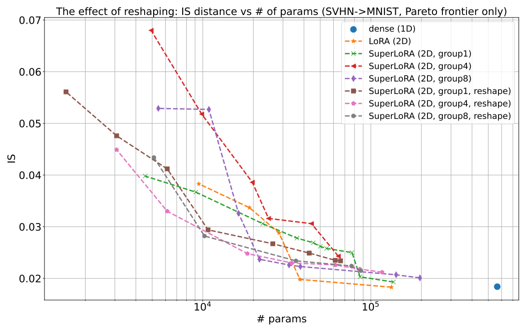
5.4.2.1 Settings:

For the image generation task, SuperLoRA is evaluated by transfer learning between SVHN [127] and MNIST datasets [94]. Both datasets have 10 classes corresponding to images of the digits 0 to 9, where the SVHN images have a more complicated color background, while the MNIST images are nearly black-and-white with a plane black background. We mainly work on the transfer learning from SVHN to MNIST. The reverse transfer learning from MNIST to SVHN is discussed in 5.6.9.

The model we worked on is a classifier-free diffusion model [68] and the correction weights from LoRA variants are added to query and value projection matrices in the attention modules of U-Net backbone [141]. Note that the size of projection weights differs across layers for this U-Net structure, which allows us to examine the performance of SuperLoRA after breaking the boundaries of different weight matrices. More details of the diffusion model are described in 5.6.2. For comparison, the original weight-wise LoRA and dense FT are also evaluated. For SuperLoRA variant, LoRA, LoNKr and LoRTA consider three versions: weight-wise, group-wise and group-reshaped. The scaling factor α of LoRA is fixed to 2.0 for all variants unless specified. 40 epochs with a batch size of 32 are carried out and results plotted are mainly from epoch 20 noticing convergence becomes stable around epoch 20. The maximum rank is set to 32 by default and a constraint $r < \min(d_{\text{in}}, d_{\text{out}})$ is imposed. To evaluate the quality of images generated by the fine-tuned diffusion model, we consider several metrics including Inception Score (IS) [147], Fréchet Inception Distance (FID) [66], Multi-Scale Intrinsic Distance (MSID) [160], Kernel Inception Distance (KID) [10], Recall and Precision [89]. Except for the recall and precision metrics, all metrics should be lower for higher-quality image generations. As we found ℓ_1-distance based IS is more consistent to the perceptual visual quality, we mainly focus on IS metric results in the main



(a) weight-wise v.s. group-wise



(b) reshaping v.s. non-reshaping

Figure 5.5: Comparison between weight-wise LoRA and group-wise SuperLoRA: (a) sweeping rank and group; (b) with/without reshaping to square-like.

content, while the results for other metrics can be found in 5.6.8. For following figures, Pareto frontier lines/dots are mainly shown to provide the limit of each method, while Appendix provides more complete figures with all data points.

5.4.2.2 Grouping effect:

First, we evaluated how splitting all ΔW_{all} into multiple groups affects the performance. 5.5a shows the results of dense, original weight-wise LoRA and group-wise SuperLoRA with different number of groups. Sweeping the rank and the number of groups, we plot the image quality metrics in y-axis and the required number of trainable parameters in x-axis. Pareto frontier lines/data points are also shown in the figure.

5.5a shows that the dense FT for ΔW presents the best IS, while requiring most parameters. Original weight-wise LoRA is closest to dense, in terms of both IS and parameter amount. However, in low-parameter regimes, SuperLoRA (2D, group1) shows the best results compared with other grouping. While in the middle of parameter amount axis, other splittings including groups $G = 8$ and 12 show slightly better IS compared with LoRA. Besides, splitting ΔW_{all} shows much more data points compared with both LoRA and dense, providing us higher flexibility to adjust the trade-off between quality and parameter efficiency especially when the memory resource is limited.

5.4.2.3 Reshaping effect:

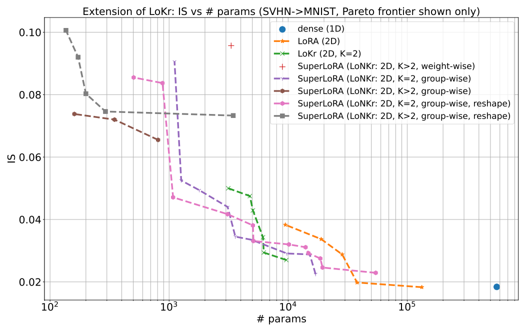
To evaluate the importance of reshaping, we compare group-wise SuperLoRA with and without reshaping in 5.5b. For weight-wise LoRA, most weight matrices corrected are square already. For all splitting with groups $G = 1$, 4 and 8, we confirmed that reshaping shows smaller number of parameters and better IS compared with their corresponding non-reshaping counterparts. This indicates that reshaping ΔW to regular tensor array (square, cube, and hyper-cube) is vital for SuperLoRA fine-tuning to prevent unbalanced skew tensors when adapting multiple weights at once.

5.4.2.4 LoKr vs. LoNKr:

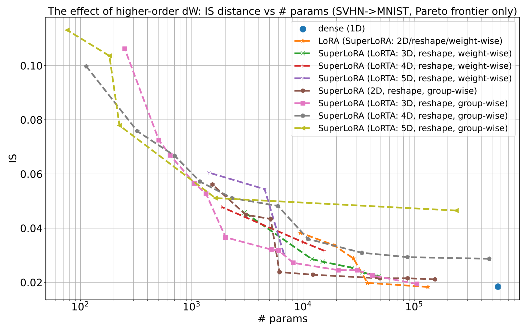
In 2D ΔW, we also compared LoKr with our proposed extension LoNKr, a variant of SuperLoRA. We evaluated LoNKr when the number of splits is $K \in \{2, 3, 4\}$, where $K = 2$ corresponds to the original LoKr. For the dense factor on the left in LoNKr/LoKr as shown in 5.3c, dimension is fixed to 6, 8 or 10. 5.6a shows that more splits provide us more choices in low-parameter regimes, especially for group-wise LoNKr. LoNKr shows much more data points and better IS when the number of parameters is less than 5,000. And the least parameter for LoKr and LoNKr dropped greatly from 500 to 150.

5.4.2.5 LoRTA:

LoRTA reshapes ΔW_{all} to high-order tensor. We evaluated 3D, 4D and 5D, as data points become much less when the dimension is too small for all planes when order is larger than 5D. From 5.6b, the higher the order of tensor folding, the less data points we have. In both weight-wise and group-wise version, 5D LoRTA reduces the least parameter it requires. Especially for group-wise LoRTA, 5D LoRTA requires less than 80 parameters to produce a result compared with beyond 1000 for 2D LoRTA and beyond 200 for 3D LoRTA, while original LoRA needs about 10^4 parameters, about 120-fold more parameters. To achieve a comparable IS of LoRA having 10^4 parameters, LoRTA (3D) just needs 2×10^3 parameters, i.e. 5-fold reduction.



(a) SuperLoRA (LoNKr)



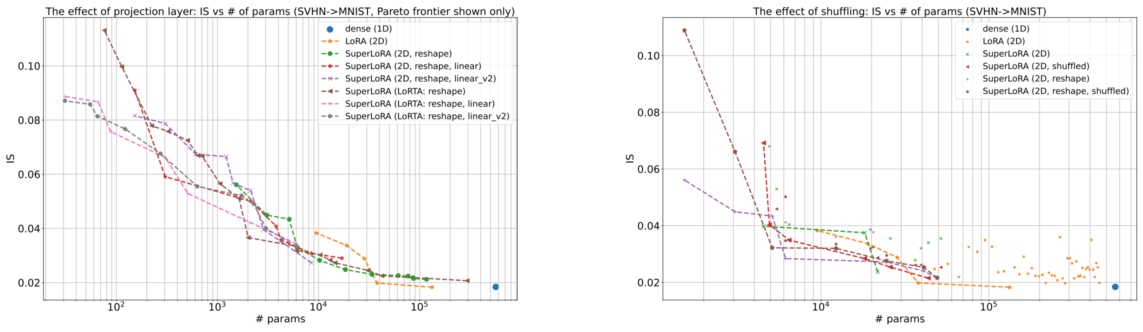
(b) SuperLoRA (LoRTA)

Figure 5.6: Performance of SuperLoRA variants: (a) LoNKr as shown in 5.3c; (b) LoRTA when folding to high-order tensor.

5.4.2.6 Projection effect:

SuperLoRA can use a projection layer $\mathscr{F}$ which is randomly initialized but fixed at both fine-tuning and inference. Linear fastfood projection and nonlinear projection with tanshrink applied after the linear projection matrix are evaluated. Besides, a modified version of fastfood projection with random Gaussian instead of random binary $\mathscr{B}$ is also tested for both linear and nonlinear versions, denoted as $\text{linear}_{\text{v2}}$ and $\text{nonlinear}_{\text{v2}}$ respectively. The projection matrix is shared across all groups. We evaluated number of groups $G \in \{1,4\}$, rank $r \in \{1,4,8\}$ and projection ratio $\rho \in \{0.01, 0.1, 0.5\}$ on SuperLoRA (2D, reshape) and SuperLoRA (LoRTA, reshape) for 3D, 4D and 5D tensor.

5.7a demonstrates with smaller projection ratio, required parameters for both SuperLoRA (2D, reshape) and SuperLoRA (LoRTA, group-wise) are pushed to extremely low-parameter regimes. The least parameter required becomes only about 30, compared with 10,000 for original LoRA. Surprisingly, linear version for both methods shows better performance than nonlinear version which are attached in 5.6.6. Besides, in extremely low-parameter regimes, higher rank with projection layer for SuperLoRA (LoRTA, group-wise) works better than small ranks itself, showing promising direction to explore projection layer in extremely low-parameter regime. In terms of linear v.s. $\text{linear}_{\text{v2}}$, $\text{linear}_{\text{v2}}$ shows better performance in higher-parameter area while linear works better in lower-parameter area, even better than SuperLoRA (LoRTA) without projection.



(a) fixed random projection within group

(b) fixed random shuffling within group

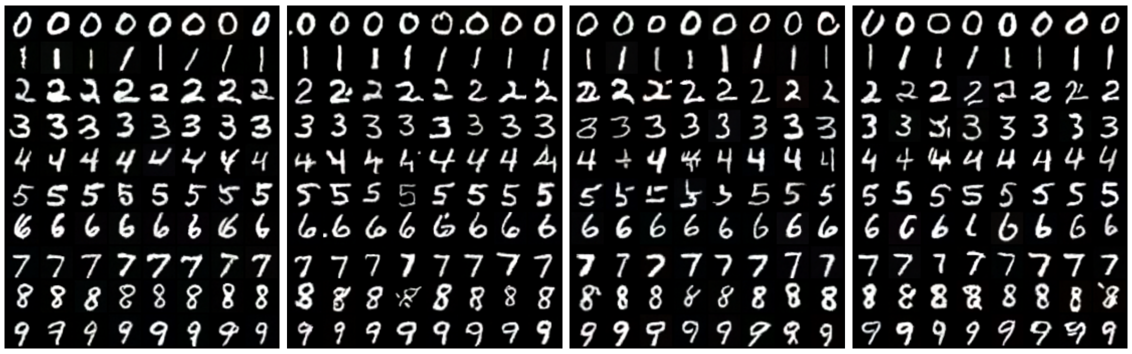
Figure 5.7: SuperLoRA under different projection modes: linear, $\text{linear}_{\text{v2}}$ and shuffling.

5.4.2.7 Shuffling effect:

As another simple projection, we studied a random shuffling to distribute ΔW_{group} before adding it to corresponding W. We evaluated SuperLoRA (2D) and SuperLoRA (2D, reshape) with/without shuffling for groups $G \in \{1, 4, 8, 16]\}$ and ranks $r \in \{1, 4, 8\}$, where the shuffled indexes are shared across all groups. The shuffling corresponds to one of fastfood projection modes by setting projection ratio to $\rho = 1$ with only permutation matrix Π. As shown in 5.7b, shuffling inside groups had no harm on IS. It even improved IS for SuperLoRA (2D) in most cases.

5.4.3 Visualization

To better understand the superiority of SuperLoRA, especially in low-parameter regimes, we visualize a set of generated images from SuperLoRA, as well as dense FT and LoRA, from a range of parameter setting: high-parameter ($> 70{,}000$), middle-parameter (from 5,000 to 10,000), low-parameter (around 1,000) and extremely-low parameter (< 100) regimes. We selected one image with the best IS for each hyper-parameter setting we have tested under same level of parameter amount. 5.8 shows that all generated images by the transfer learning model from SVHN to MNIST are close to images from MNIST dataset itself with black-white background, removing most domain information of color SVHN. SuperLoRA (2D, group8, rank13) in 5.8c shows competitive results with LoRA (rank8) using 5,000 less parameters.


(a) dense
#param 565,248
IS 0.0184

(b) LoRA $r = 8$
#param 75,776
IS 0.03025

(c) SuperLoRA
#param 70,720
IS 0.0305

(d) SuperLoRA
#param 73,728
IS 0.0263

Figure 5.8: Visualization of generated images under high-parameter level ($>$ 70,000).

For the middle-parameter regimes, 5.9 shows visualization of LoNKr, SuperLoRA (2D, reshape), LoRTA (3D, reshape), LoRTA (4D, reshape) and SuperLoRA (2D, reshape, projection). More domain information with colorful digits and background occur occasionally. There are also some missing digits presented in middle-parameter area.

When the number of parameters is as low as 1,000, even though only few choices left like LoNKr and LoRTA, one can always stretch hyper-parameter settings from middle-parameter level coupled with fixed linear projection layer to compress the tensor size. In this way, the strength of middle-parameter level gets extended to low-parameter area. As shown in 5.10, compared with the visualization from middle-parameter results, more missing digits and more colorful backgrounds are presented.

Finally, we also visualized a few images from extremely-low parameter level less than 100 in 5.10. Surprisingly, domain transfer in those images is somewhat realized from SVHN to MNIST even with such an extremely few parameter case such as 31, which is more than four orders of magnitude smaller than dense FT.

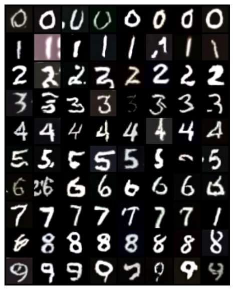
5.5 Conclusion

We proposed a new unified framework called SuperLoRA, which generalizes and extends LoRA variants including LoKr and LoTR. SuperLoRA provides some extended variants, which we refer

(a) LoNKr
#param 5,112
IS 0.036

(b) SuperLoRA
#param 10,752
IS 0.0294

(c) LoRTA
#param 8,160
IS 0.0272


(d) LoRTA
#param 11,100
IS 0.036

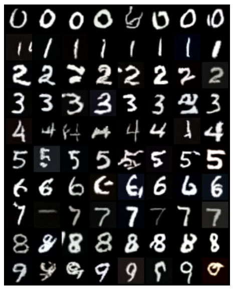
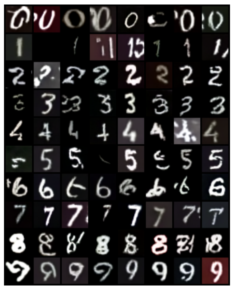
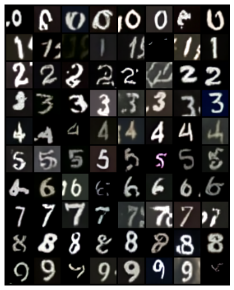

(e) SuperLoRA w/ p
#param 8,512
IS 0.0273

Figure 5.9: Visualization of generated images under middle-parameter level ($[5{,}000, 20{,}000]$).


(a) LoNKr
#param 1,080
IS 0.0471

(b) LoRTA
#param 1,060
IS 0.0565

(c) SuperLoRA w/ p
#param 832
IS 0.0607


(d) LoRTA
#param 76
IS 0.1131

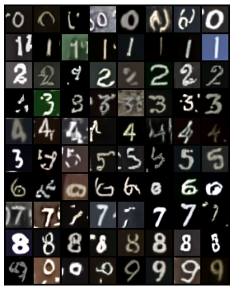

(e) LoRTA w/ p
#param 31
IS 0.0871

Figure 5.10: Visualization of generated images under low-parameter level (1,000) and extremely-low level (< 100).

to as LoNKr and LoRTA. It offers a rich and flexible set of hyper-parameters, including the rank of factorization, the choice of projection function, projection ratio, the number of groups, the order of tensor, and the number of Kronecker splits. Through two types of transfer learning experiments, we demonstrated that SuperLoRA achieves promising results in parameter efficiency for fine-tuning at low-parameter and extremely-low-parameter regimes. We could reduce the required number of parameters by 3 to 10 folds compared to LoRA. Future work includes to study the projection functions to further improve the efficiency in extremely-low-parameter regimes, and applications to various transfer learning tasks along with different large models such as LLMs.

5.6 Supplementary Materials

5.6.1 Illustration of ViT model in detail

The ViT model that we used for the classification task is adapted from a public codebase[3]. The detailed structure of the ViT is depicted in 5.11, where we only fine-tune the projection layers for query and value in the Self-Attention modules. In ViT-base, depth (L in 5.11) is set to be 12 and dimension is 768. The total number of parameters of the ViT base model is 86.6M.

5.6.2 Illustration of diffusion model in detail

The classifier-free diffusion model [68] that we used for image generation is adapted from a public codebase[4]. Its U-Net structure is illustrated in 5.12, which contains 21 attention modules, where the number of input/output channels of the attention modules is either 64 or 128. We only fine-tune the query and value projection layers of those attention modules. The total number of parameters of the U-Net base model is 10.42M, including 300 parameters for the class embedding.

5.6.3 Illustration of grouping mechanism

5.13 illustrates several different cases of the grouping mechanism. 5.13(a) is the conventional weight-wise grouping, used for typical LoRA. Each weight correction, i.e. $\Delta W_{\mathrm{v}\ell}$ and $\Delta W_{\mathrm{q}\ell}$ for value and query projections at layer ℓ, is individually represented by a rank-r decomposition: $A_g B_g^\top$ for group g. 5.13(b) shows layer-wise grouping, where the LoRA unit in each group jointly adapts both value and query projections in each layer. When we stack multiple weight matrices in a naïve way, the 2D array will have unbalanced fan-in/fan-out shape, leading to inefficient low-rank decomposition. 5.13(c) can solve this issue by reshaping the 2D array into a regular square shape before low-rank decomposition. As the reshaping is already breaking the geometric meaning of the original 2D weights, the grouping need not necessarily aligned with the weight boundary as shown

[3]`https://github.com/bwconrad/vit-finetune`
[4]`https://github.com/coderpiaobozhe/classifier-free-diffusion-guidance-Pytorch`

in a general grouping case of 5.13(d). Further, applying a projection function $\mathscr{F}(\cdot)$ as shown in 5.13(e), the element distribution can be shuffled and mixed-up to relax the geometric restriction of original LoRA. LoRTA can further generalize the reshaping by folding the 2D array to any arbitrary M-dimensional tensor array by using the Tucker decomposition as shown in 5.13(f). Relaxing the geometric constraint can improve the parameter efficiency as shown in this chapter. We further make a geometric analysis of our grouping methods below.

5.6.4 Geometric analysis

SuperLoRA adapts multiple attention modules at once, and relaxes the underlying geometric restrictions inherent to the 2D weights for each attention module, by employing grouping, reshaping, and projection (including shuffling). To better understand how SuperLoRA works differently from LoRA, geometric analysis is conducted for the classification task. Specifically, we pick 4 different methods with a comparable number of parameters around 100,000:

- LoRA (2D): #param 147,456, accuracy 0.9113;
- SuperLoRA (2D): #param 115,200, accuracy 0.9170;
- SuperLoRA (2D, reshape): #param 165,572, accuracy 0.9218;
- SuperLoRA (2D, reshape, w/ projection): #param 138,372, accuracy 0.9213.

The weight correction term ΔW is compared to the full dense FT case, which involves 14M parameters achieving an accuracy of 0.9290. We analyze three different geometric measures with respective to the FT weight ΔW_{dense}: i) left-singular similarity; ii) right-singular similarity; and iii) Euclidean distance. Letting U and V denote the left- and right-singular vectors of ΔW for each variant listed above, these metrics are defined as follows:

$$d_{\mathrm{L}} = \frac{1}{\sqrt{k}} \|U_{\text{dense}}[:, :k]^{\top} U_{\text{variant}}[:, :k]\|_2, \tag{5.5}$$

$$d_{\mathrm{R}} = \frac{1}{\sqrt{k}} \|V_{\text{dense}}[:k, :] V_{\text{variant}}[:k, :]^{\top}\|_2, \tag{5.6}$$

$$d_{\mathrm{E}} = \frac{\|\Delta W_{\text{dense}} - \Delta W_{\text{variant}}\|_2}{\|\Delta W_{\text{dense}}\|_2}. \tag{5.7}$$

Note that d_{E} approaches to 0 when ΔW converges to the dense FT case, while d_{L} and d_{R} converge to 1.

The top $k = 5$ principal singular vectors are analyzed as shown in 5.14. The ViT model has 12 attention modules, and we plot the total of 24 points for the query and value projection weights. The first row shows the query weights for d_{L} v.s. d_{R}, d_{E} v.s. d_{R}, and d_{E} v.s. d_{L} from left to right across the columns. The second and third rows are for the value weights, and both query and value weights, respectively.

We see that the Euclidean distance d_{E} is significantly decreased for SuperLoRA, especially with reshaping applied. It explains the improved accuracy with reshaping. Although grouping, reshaping, and projection can break the geometric meaning of the original 2D weights, the subspace similarity is not completely lost. Especially for query weights, SuperLoRA shows higher right-singular similarity than LoRA. As the embedding vector passes through right-hand side of the weight, principal right-singular vectors perform as a low-rank subspace mapping of the input vector while the left-singular vectors work as mapping the subspace towards the output vector. While SuperLoRA with reshaping tends to preserve higher right-singular similarity, LoRA tends to preserve higher left-singular similarity. Further, it is found that the corrections for query and value weights behave differently with reshaping, i.e., right-singular similarities for the value weights are much larger than for query weights.

5.6.5 Grouping effect on SuperLoRA (1D, dense, with projection)

As the fixed projection matrix is shared across all groups, the number of groups will affect the size of the projection matrix directly. To explore this influence, dense FT with projection is tested for different splitting, from 1 to 12 groups. According to 5.15, using 1 group achieves the best overall accuracy and using 4 or 8 groups are comparable to a smaller projection ratio. When the projection matrix is too small, e.g., with 12 groups, accuracy drops greatly. This confirms that jointly updating multiple attention modules is beneficial.

5.6.6 Linear v.s. nonlinear projection

Besides linear projection, nonlinear projection is also examined. We use tanhshrink after the fixed linear projection, resulting in 'nonlinear' and 'nonlinear$_{\mathrm{v2}}$'. Note that the 'v2' projection uses a Gaussian random vector rather than a binary random vector $\mathscr{B}$ for the fastfood projection as shown in 5.16. More specifically, we consider six variants for the projection function $\mathscr{F}(\cdot)$ in this chapter:

- identity (no projection): $\mathscr{F}(x) = x$;
- shuffling: $\mathscr{F}(x) = x\mathit{\Pi}$;
- linear: $\mathscr{F}(x) = x\mathscr{H}'\,\mathsf{diag}[\mathscr{G}]\,\mathit{\Pi}\,\mathscr{H}\,\mathsf{diag}[\mathscr{B}]$;
- linear$_{\mathrm{v2}}$: $\mathscr{F}(x) = x\mathscr{H}'\,\mathsf{diag}[\mathscr{G}]\,\mathit{\Pi}\,\mathscr{H}\,\mathsf{diag}[\mathscr{G}']$;
- nonlinear: $\mathscr{F}(x) = \mathsf{tanhshrink}\big[x\mathscr{H}'\,\mathsf{diag}[\mathscr{G}]\,\mathit{\Pi}\,\mathscr{H}\,\mathsf{diag}[\mathscr{B}]\big]$;
- nonlinear$_{\mathrm{v2}}$: $\mathscr{F}(x) = \mathsf{tanhshrink}\big[x\mathscr{H}'\,\mathsf{diag}[\mathscr{G}]\,\mathit{\Pi}\,\mathscr{H}\,\mathsf{diag}[\mathscr{G}']\big]$.

Here, $\mathit{\Pi}$ performs a random permutation of a vector. The Walsh–Hadamard matrices $\mathscr{H}' \in \mathbb{R}^{N_{\mathrm{in}} \times 2^N}$ and $\mathscr{H} \in \mathbb{R}^{2^N \times N_{\mathrm{out}}}$ are left- and right-truncated versions of a regular Walsh–Hadamard matrix $\mathscr{H}_2^{\otimes N} \in \mathbb{R}^{2^N \times 2^N}$, where $[\cdot]^{\otimes N}$ denotes N-fold Kronecker power and $\mathscr{H}_2 = \frac{1}{\sqrt{2}}\left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$. Letting N_{in} and N_{out} be the number of elements for the input and output of the projection function $\mathscr{F}(\cdot)$ with a compression ratio of $\rho = N_{\mathrm{in}}/N_{\mathrm{out}}$, the exponent N is chosen as $N = \mathrm{ceil}[\log_2(\max(N_{\mathrm{in}}, N_{\mathrm{out}}))]$.

In practice, the left-truncated Walsh–Hadamard matrix is realized by input zero-padding before fast Walsh–Hadamard transform. The random vector $\mathscr{G}$ is hence of size 2^N, and drawn from the normal distribution. Here, $\mathscr{G}' \in \mathbb{R}^{N_\text{out}}$ is another random vector drawn from the normal distribution while $\mathscr{B} \in \{\pm 1\}^{N_\text{out}}$ is a random vector drawn from the Rademacher distribution.

5.17 shows the comparison of several projection variants. Surprisingly, with the same number of parameters, the linear version outperforms the nonlinear versions in most cases.

5.6.7 Transfer learning from ImageNet1k to CIFAR10

For the classification task, transfer learning from ImageNet1k to CIFAR10 is also examined. Most settings are same as 5.15 for transfer learning from ImageNet21k to CIFAR100. The ViT base model[5] is pretrained for ImageNet1k. It fine-tunes with 3000 steps at most and the best accuracy is reported. Detailed ranks tested are as follows:

- LoRA (2D): ranks: 1, 2, 4, 6, 8, . . . , 64, 128
- SuperLoRA (2D): groups: 1, 4, 8, 12; ranks: 1, 2, 4, 6, 8, . . . , 64, 128
- SuperLoRA (2D, reshape):
  - groups: 1, 4, 12, ranks: 1, 2, 4, 6, 8, . . . , 64, 128;
  - group 8, ranks: 1, 2, 4, 6, 8, . . . , 24, 28, 32, 36, . . . , 64
- SuperLoRA (LoRTA: 3D, reshape): groups: 1, 4, 8, 12; ranks: 1–6, 8, 10, 12, . . . , 24
- SuperLoRA (LoRTA: 4D, reshape):
  - group 1; ranks: 1–6, 8, 10, 12, . . . , 22
  - group 4; ranks: 1–6, 8, 10, 12, . . . , 16
  - group 8; ranks: 1–6, 8, 10, 12, . . . , 18
  - group 12; ranks: 1–6, 8, 10, 12

[5]`https://huggingface.co/google/vit-base-patch16-224`

- SuperLoRA (LoRTA: 5D, reshape):
  - groups 1, 4, 8; ranks: 1–6, 8
  - group 12; ranks: 1–6

The classifier head is frozen after selecting most relevant labels in ImageNet1k, i.e. [404, 436, 94, 284, 345, 32, 340, 510, 867], corresponding to [airliner, humming bird, siamese cat, ox, golden retriever, tailed frog, zebra, container ship, trailer truck]. Classification results can be found in 5.18. Even though only attention modules are adapted, overall transfer learning is excellent, reaching an accuracy close to 0.99. Besides, SuperLoRA significantly outperforms original LoRA in terms of both classification accuracy and the parameter range it covers as the transfer learning. SuperLoRA (2D, reshape) shows at least 3-fold reduction in the required number of parameters compared to LoRA. Noticeably, when comparing the lowest-rank LoRA with around 0.97 accuracy, SuperLoRA (2D, reshape, w/ projection) improves the accuracy by about 1%, and moreover the required number of parameters can be greatly reduced by 10 folds with SuperLoRA (LoRTA: 3D, reshape) to maintain the comparable accuracy.

5.6.8 Transfer learning from SVHN to MNIST

5.6.8.1 Grouping effect (complete results)

Scatter plots of all metrics (FID, IS, KID, MSID, Improved Precision and Improved Recall) are given in 5.19. Except IS, all metrics show many examples performing better than dense FT, while worse according to the visualization results, indicating IS is a more reasonable quantitative metric in this case.

5.6.8.2 Reshaping effect (complete results)

Complete results for reshaping, with scatter plots for all metrics, are shown in 5.20.

5.6.8.3 SuperLoRA (LoNKr, complete results)

Complete results for SuperLoRA (LoNKr), with scatter plots for all metrics, are shown in 5.21.

5.6.8.4 SuperLoRA (LoRTA, complete results)

Complete results for SuperLoRA (LoRTA), with scatter plots for all metrics, are shown in 5.22.

5.6.9 Transfer learning from MNIST to SVHN

5.6.9.1 Grouping effect

Transfer learning from MNIST to SVHN is also tested. 5.23 shows that some metrics cannot function when transferred from a simpler dataset to a more complicated one, e.g. FID, IS, KID and Improved Precision, where some ill-posed cases appear. Besides this, we can still find from the Pareto frontiers that SuperLoRA extends LoRA to low-parameter regime and works better occasionally in terms of IS, MSID, Improved Precision and Improved Recall.

5.6.9.2 Reshaping effect

5.24 shows that SuperLoRA with reshaping works better than non-reshaping in most cases in transfer learning from MNIST to SVHN, consistent with the results in transfer learning from SVHN to MNIST.

5.6.9.3 SuperLoRA (LoNKr)

5.25 demonstrates the results of SuperLoRA (LoNKr). From MSID figure, we can see that, LoNKr extends LoKr to low-parameter regime, and achieves a better MSID.

5.6.9.4 SuperLoRA (LoRTA)

From FID and KID in 5.26, LoRTA pushes required parameters from 10^4 to 10^2 compared with LoRA, providing more flexibility when the memory is limited.

5.6.10 Effect of groups in LoNKr and LoRTA

From 5.27 and 5.28, LoNKr and LoRTA behave differently in terms of the number of groups: for LoNKr, fewer groups are better (than more groups) in low-parameter regime, while they are comparable in high-parameter regime. However, LoRTA prefers less groups.

5.6.11 Effect of split K in LoNKr

As shown in 5.29, larger K works better than smaller ones in the low-parameter regime.

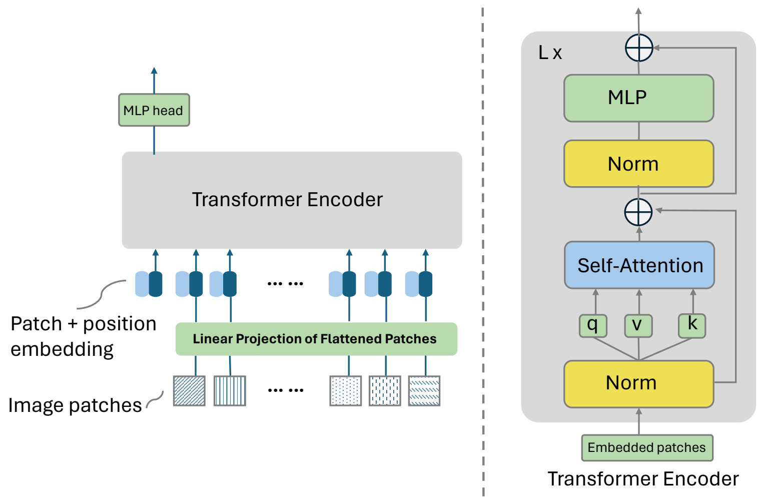


Figure 5.11: ViT model structure.

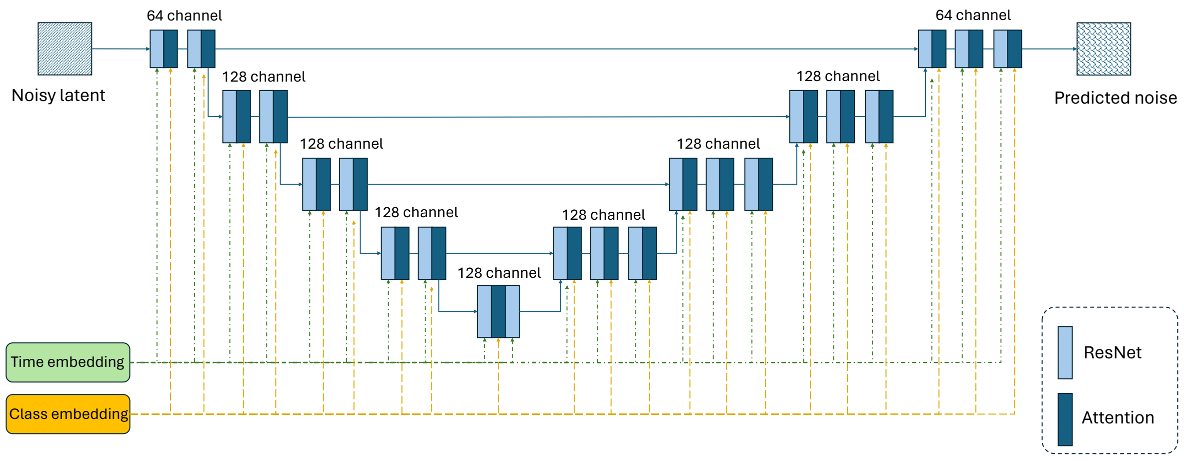


Figure 5.12: Classifier-free diffusion model structure.

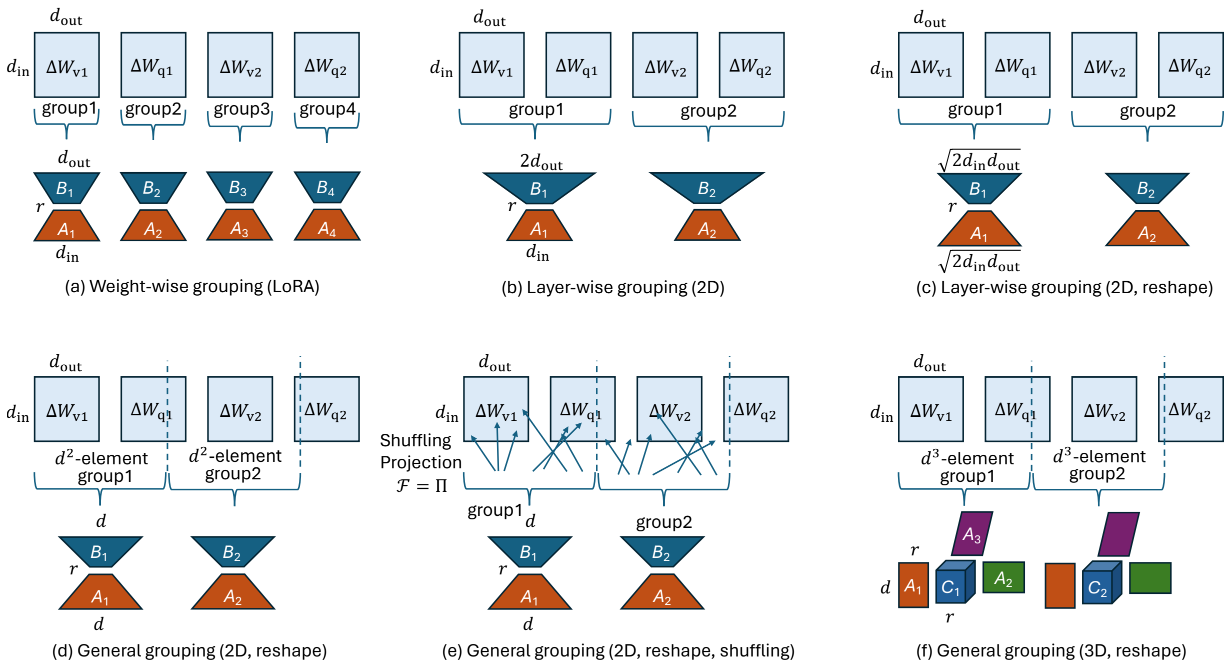

(a) Weight-wise grouping (LoRA)
(b) Layer-wise grouping (2D)
(c) Layer-wise grouping (2D, reshape)
(d) General grouping (2D, reshape)
(e) General grouping (2D, reshape, shuffling)
(f) General grouping (3D, reshape)
Shuffling Projection $\mathcal{F} = \Pi$
d^2-element group1
d^2-element group2
d^3-element group1
d^3-element group2

Figure 5.13: Examples of grouping mechanism.

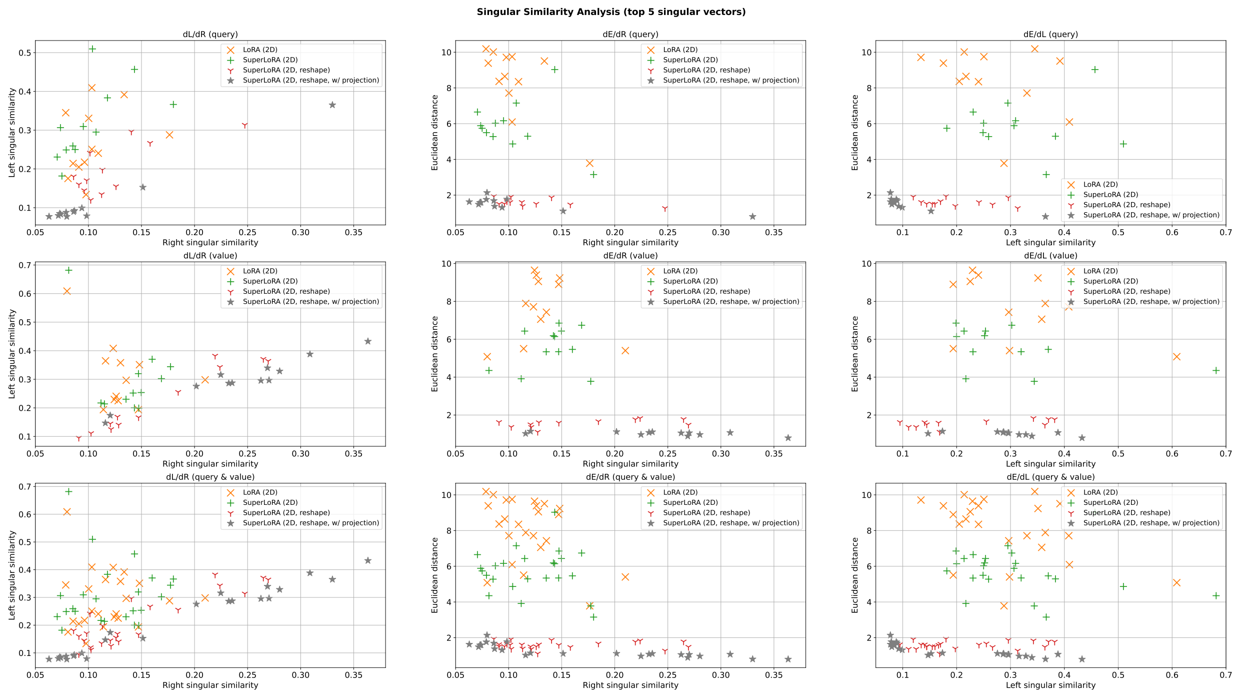


Figure 5.14: Geometric similarity analysis (top 5 principal singular vectors).

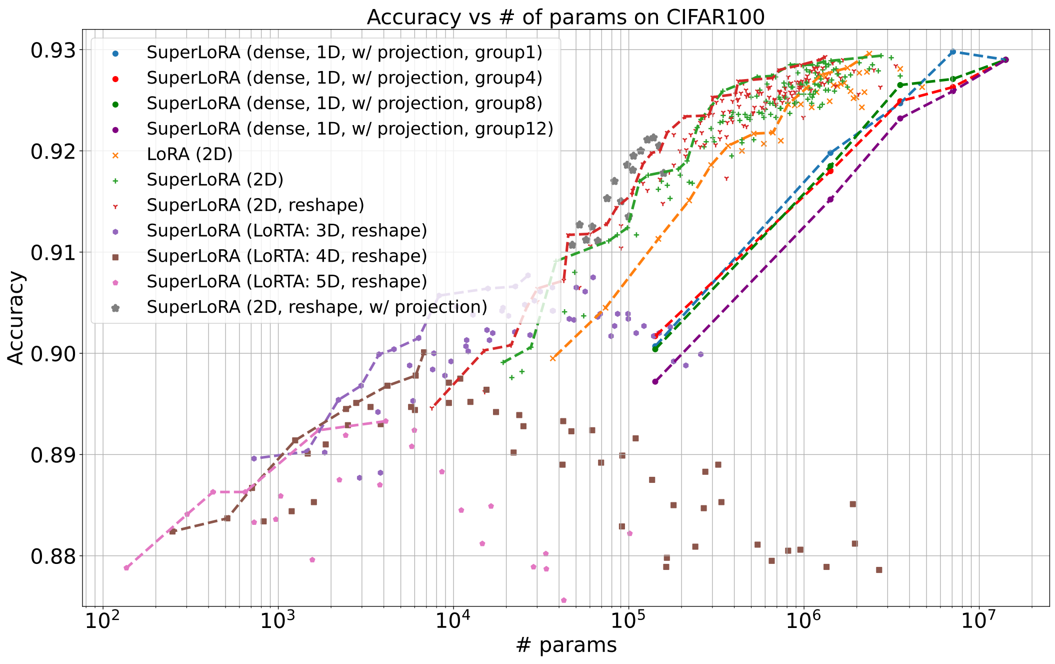


Figure 5.15: More groups (i.e. less fixed projection parameters) on SuperLoRA (1D, dense, w/ projection).

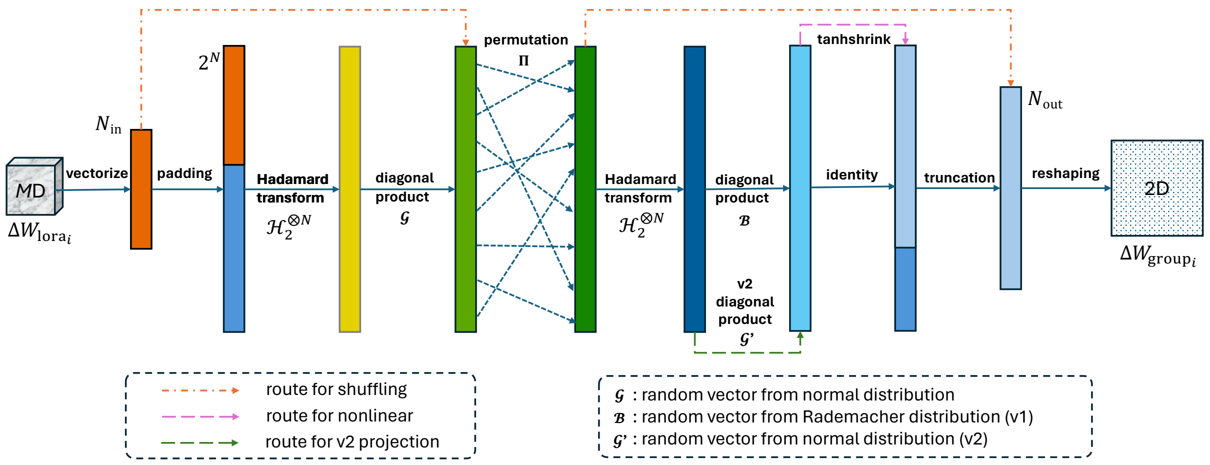
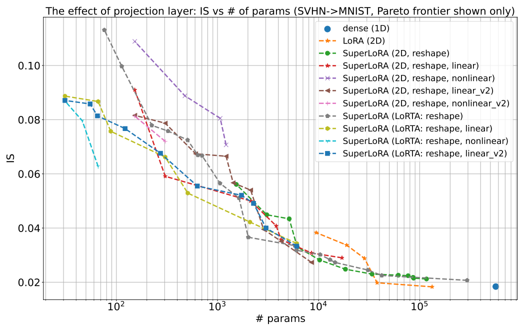

MD
ΔW_{lora_i}
vectorize
N_{in}
padding
2^N
Hadamard transform $\mathcal{H}_2^{\otimes N}$
diagonal product $\mathcal{G}$
permutation Π
Hadamard transform $\mathcal{H}_2^{\otimes N}$
diagonal product $\mathcal{B}$
v2 diagonal product $\mathcal{G}'$
tanhshrink
identity
truncation
N_{out}
reshaping
2D
$\Delta W_{\text{group}_i}$
route for shuffling
route for nonlinear
route for v2 projection
$\mathcal{G}$: random vector from normal distribution
$\mathcal{B}$: random vector from Rademacher distribution (v1)
$\mathcal{G}'$: random vector from normal distribution (v2)

Figure 5.16: Illustration of fastfood projection and its variants.


(a) linear v.s. nonlinear (IS, Pareto only)

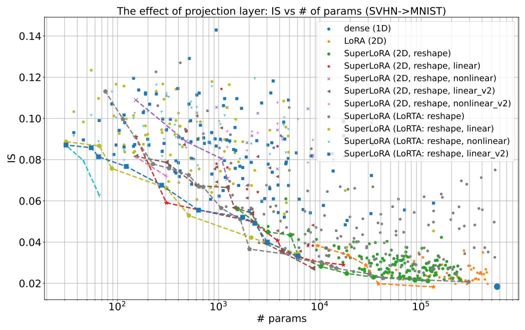

(b) linear v.s. nonlinear (IS, all points)

Figure 5.17: Comparison between Linear/Linear$_{v2}$/Nonlinear/Nonlinear$_{v2}$ projections.

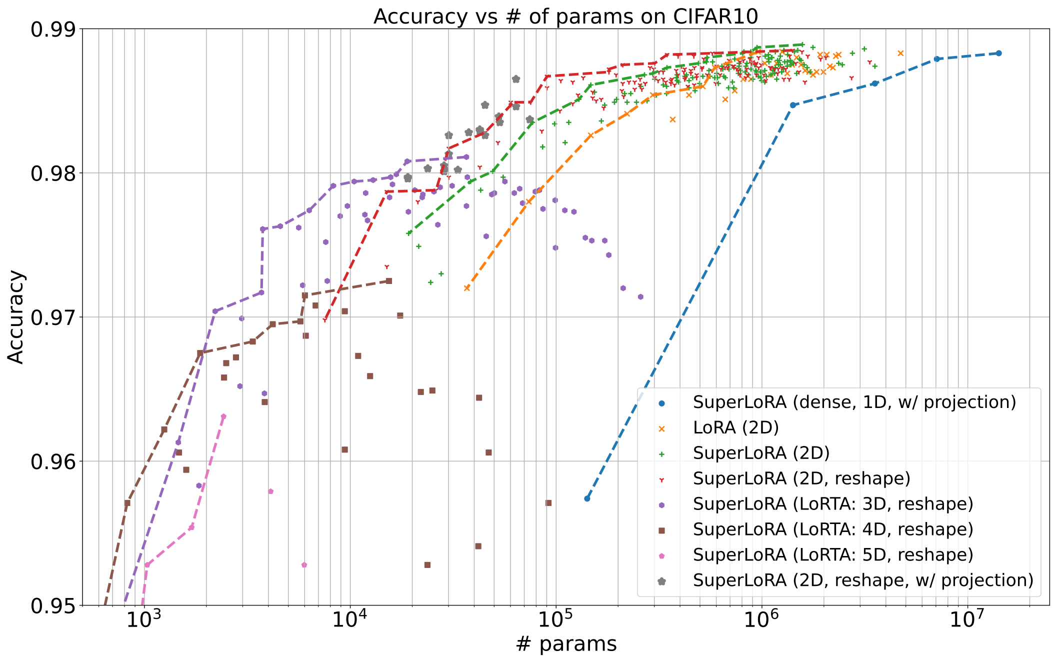
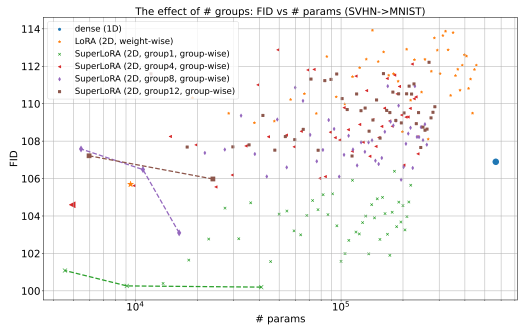
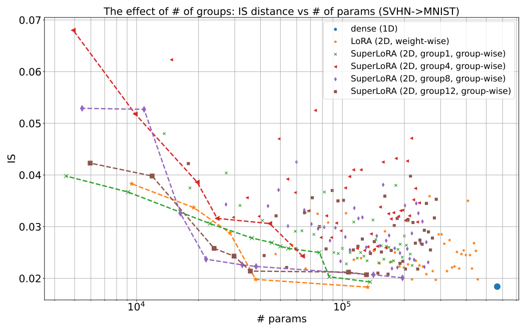


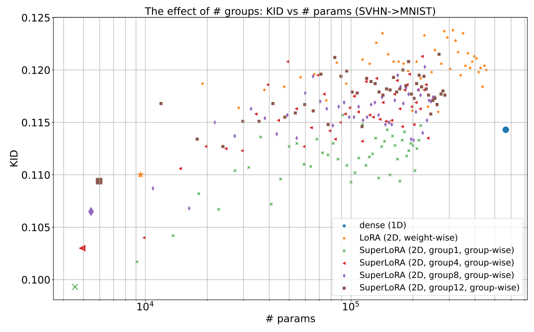
Figure 5.18: Classification accuracy for transfer learning from ImageNet1K to CIFAR10 with SuperLoRA.



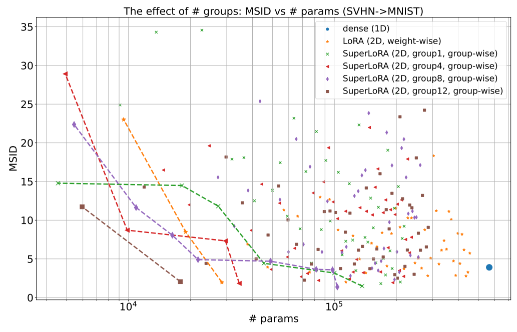
(a) weight-wise v.s. group-wise (FID)



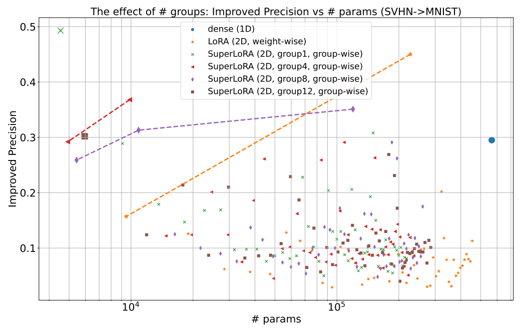
(b) weight-wise v.s. group-wise (IS)



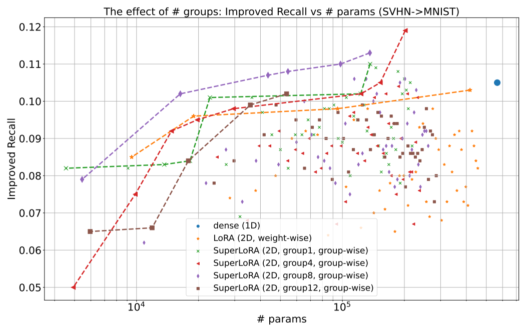
(c) weight-wise v.s. group-wise (KID)



(d) weight-wise v.s. group-wise (MSID)

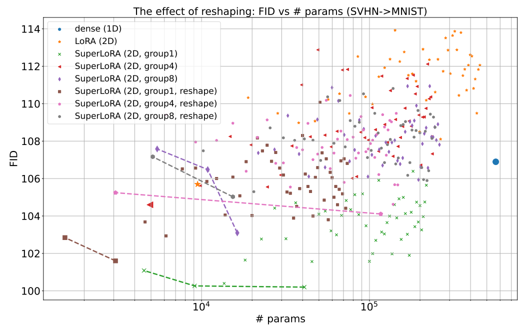


(e) weight-wise v.s. group-wise (Improved Precision)

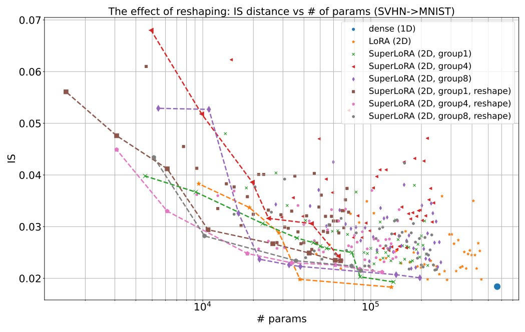


(f) weight-wise v.s. group-wise (Improved Recall)

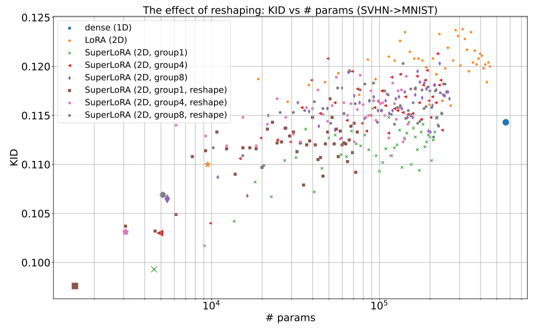
Figure 5.19: Complete comparison between weight-wise LoRA and group-wise SuperLoRA.



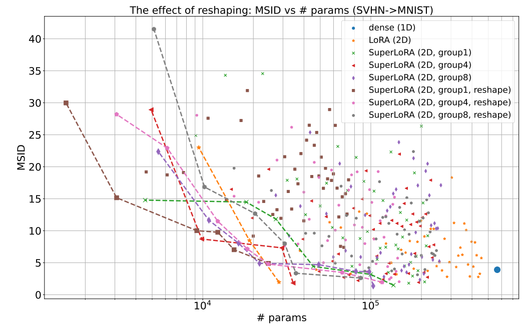
(a) reshaping v.s. non-reshaping (FID)



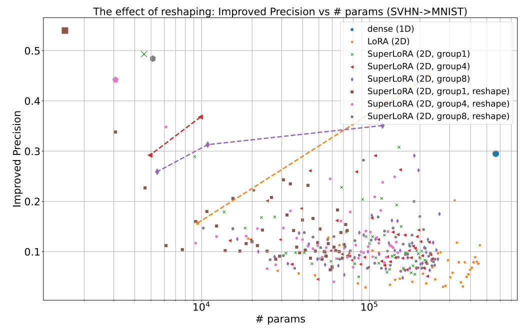
(b) reshaping v.s. non-reshaping (IS)



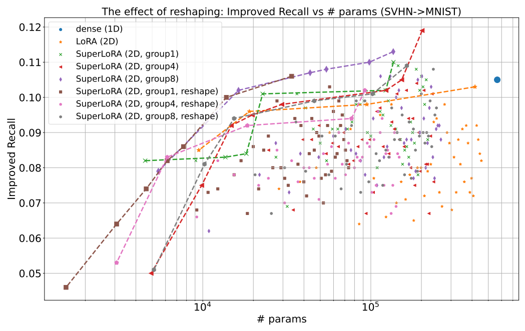
(c) reshaping v.s. non-reshaping (KID)



(d) reshaping v.s. non-reshaping (MSID)

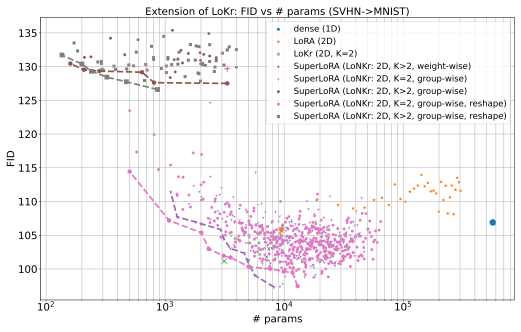


(e) reshaping v.s. non-reshaping (Improved Precision)

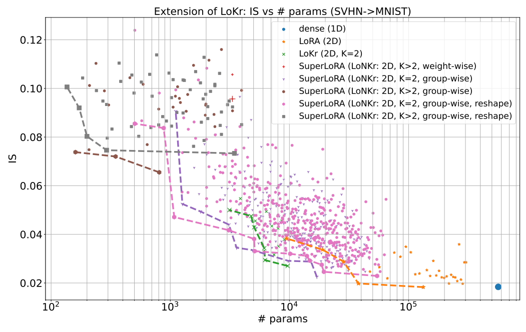


(f) reshaping v.s. non-reshaping (Improved Recall)

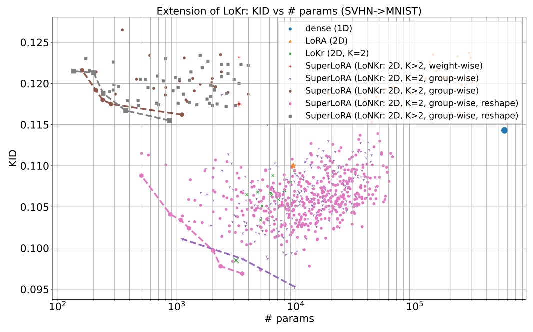
Figure 5.20: Complete comparison between reshaping and non-reshaping SuperLoRA.



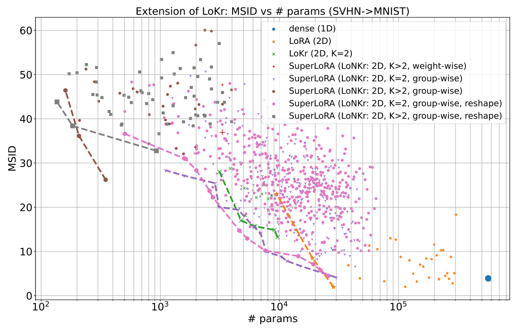
(a) SuperLoRA (LoNKr, FID)



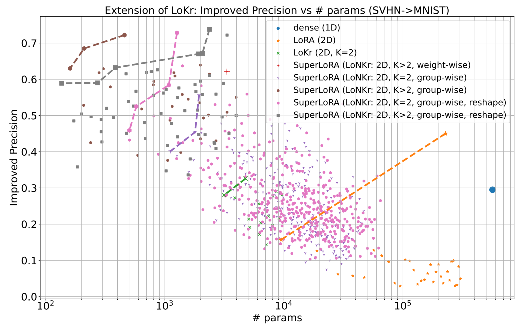
(b) SuperLoRA (LoNKr, IS)



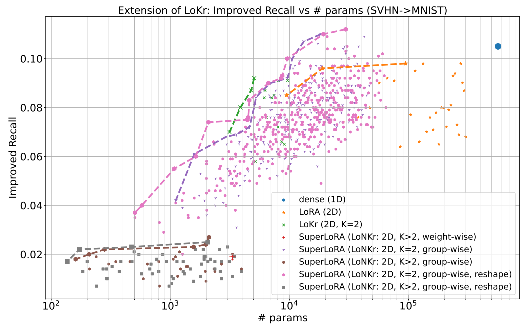
(c) SuperLoRA (LoNKr, KID)



(d) SuperLoRA (LoNKr, MSID)

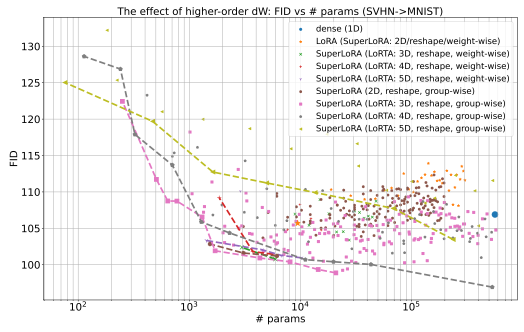


(e) SuperLoRA (LoNKr, Improved Precision)

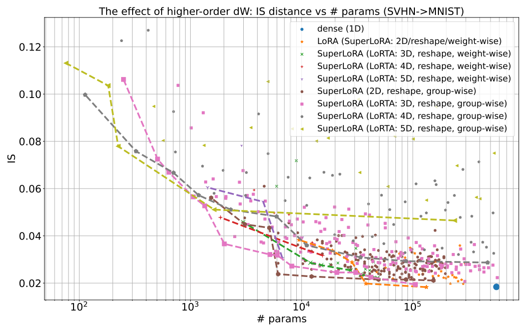


(f) SuperLoRA (LoNKr, Improved Recall)

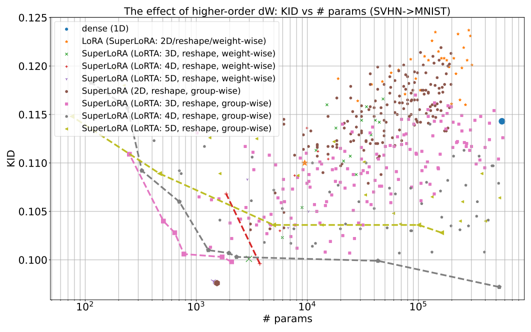
Figure 5.21: Complete results for LoNKr.



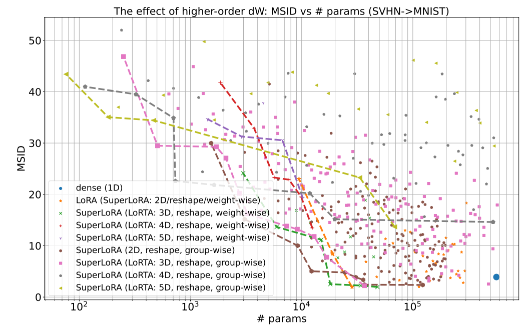
(a) SuperLoRA (LoRTA, FID)



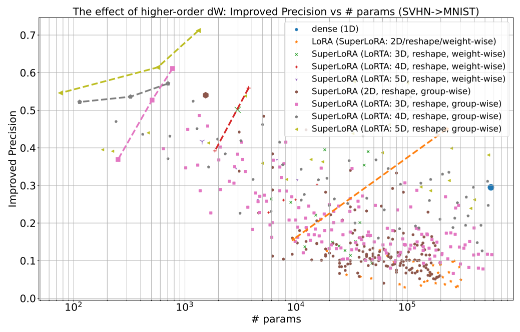
(b) SuperLoRA (LoRTA, IS)



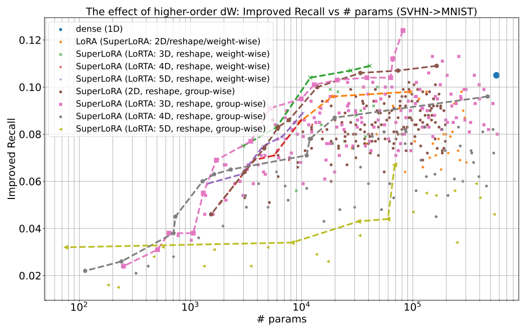
(c) SuperLoRA (LoRTA, KID)



(d) SuperLoRA (LoRTA, MSID)

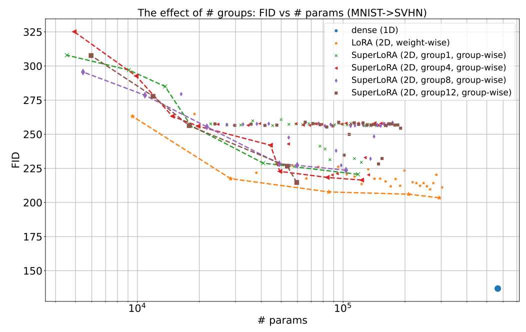


(e) SuperLoRA (LoRTA, Improved Precision)

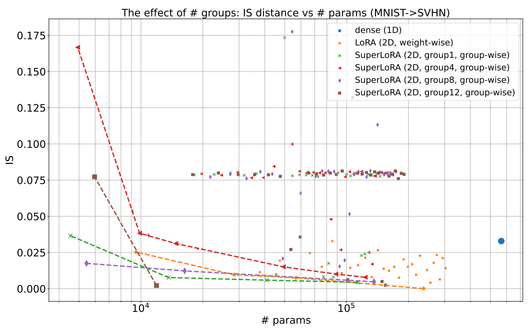


(f) SuperLoRA (LoRTA, Improved Recall)

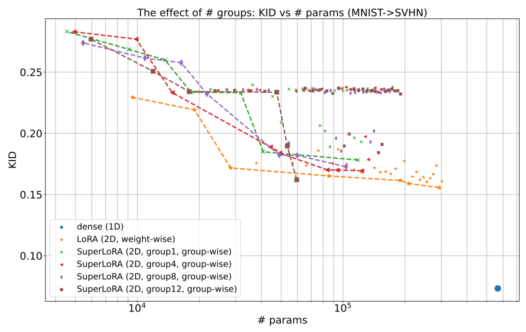
Figure 5.22: Complete results for LoRTA.



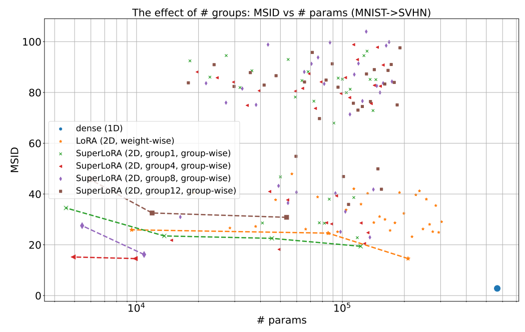
(a) weight-wise v.s. group-wise (FID)



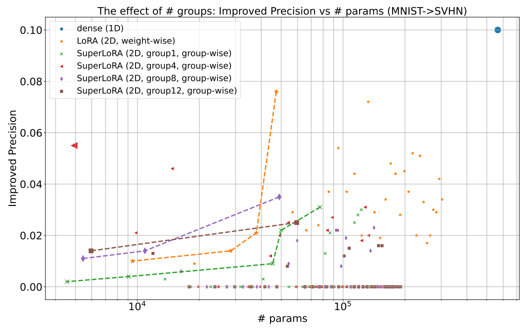
(b) weight-wise v.s. group-wise (IS)



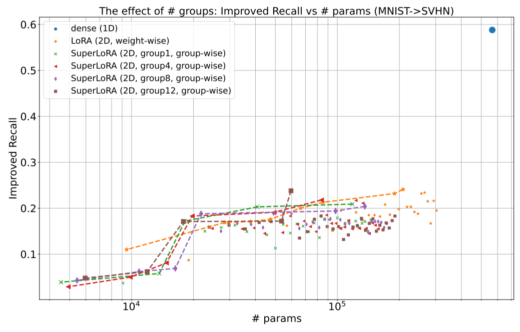
(c) weight-wise v.s. group-wise (KID)



(d) weight-wise v.s. group-wise (MSID)

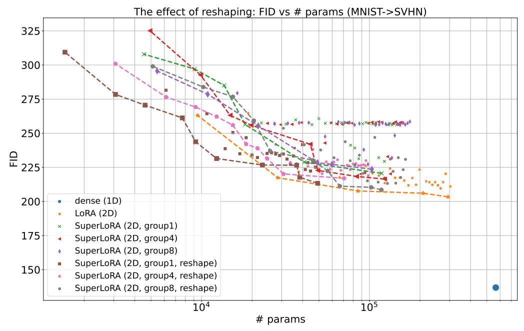


(e) weight-wise v.s. group-wise (Improved Precision)

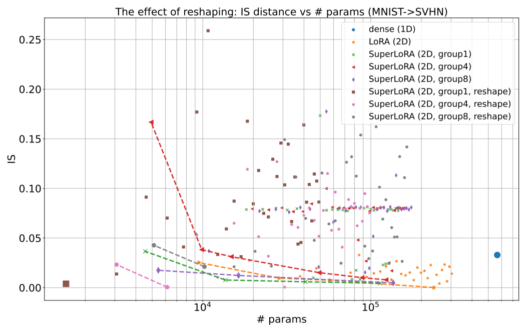


(f) weight-wise v.s. group-wise (Improved Recall)

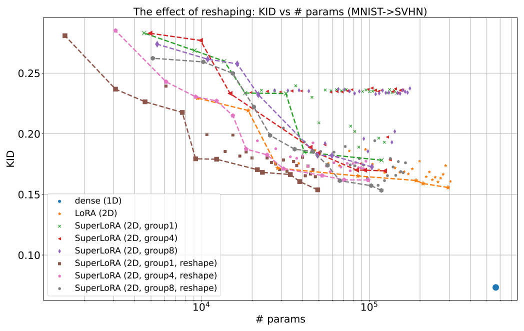
Figure 5.23: Complete comparison between weight-wise LoRA and group-wise SuperLoRA for transfer learning from MNIST to SVHN.



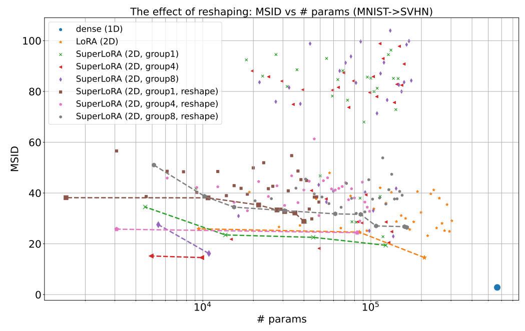
(a) reshaping v.s. non-reshaping (FID)



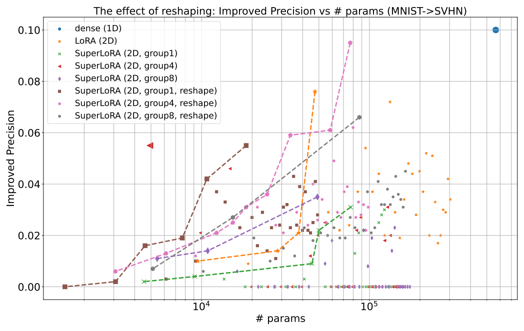
(b) reshaping v.s. non-reshaping (IS)



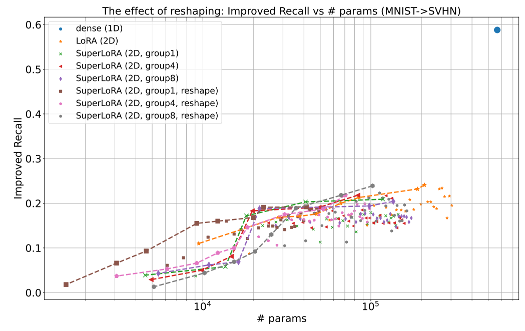
(c) reshaping v.s. non-reshaping (KID)



(d) reshaping v.s. non-reshaping (MSID)

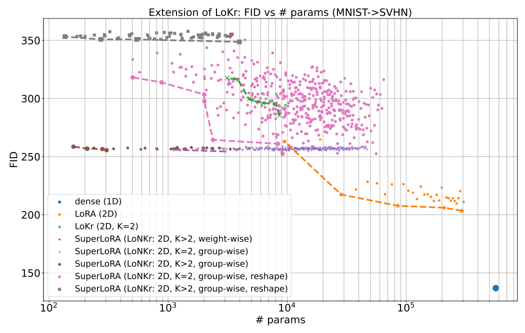


(e) reshaping v.s. non-reshaping (Improved Precision)

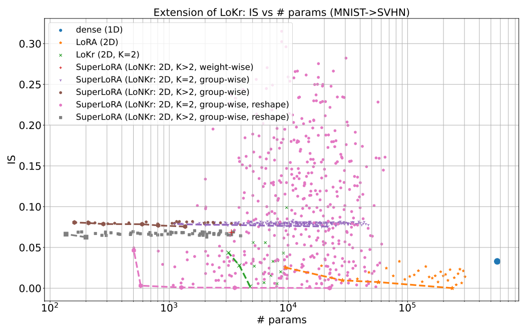


(f) reshaping v.s. non-reshaping (Improved Recall)

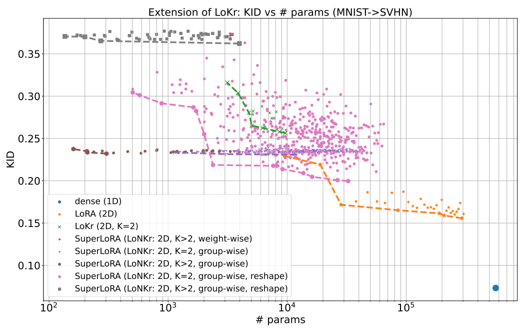
Figure 5.24: Complete comparison between reshaping and non-reshaping SuperLoRA for transfer learning from MNIST to SVHN.



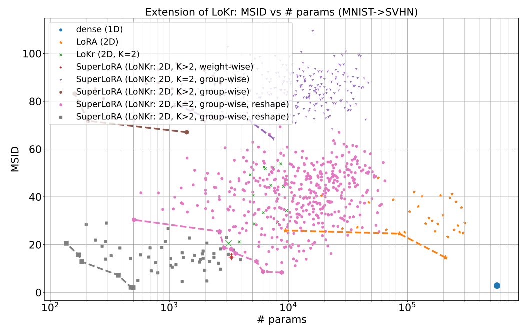
(a) SuperLoRA (LoNKr, FID)



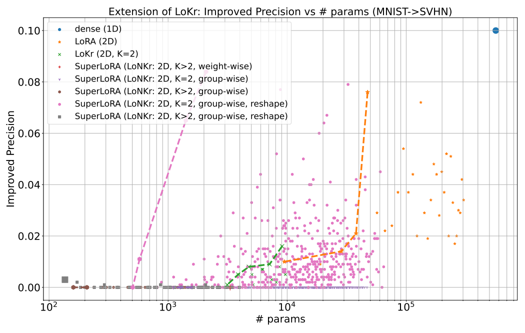
(b) SuperLoRA (LoNKr, IS)



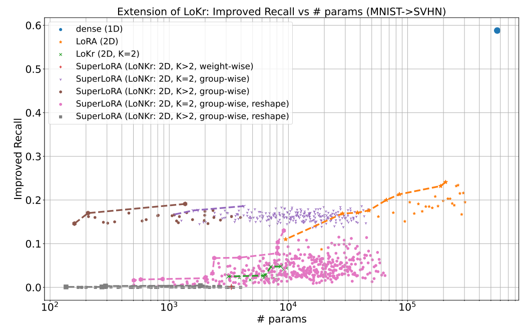
(c) SuperLoRA (LoNKr, KID)



(d) SuperLoRA (LoNKr, MSID)

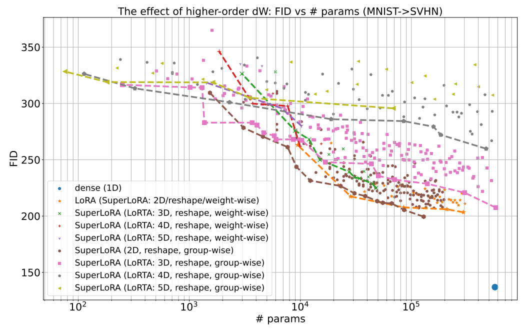


(e) SuperLoRA (LoNKr, Improved Precision)

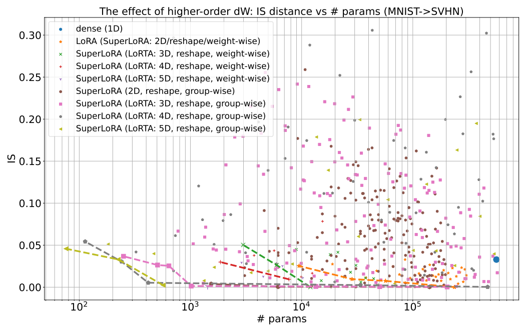


(f) SuperLoRA (LoNKr, Improved Recall)

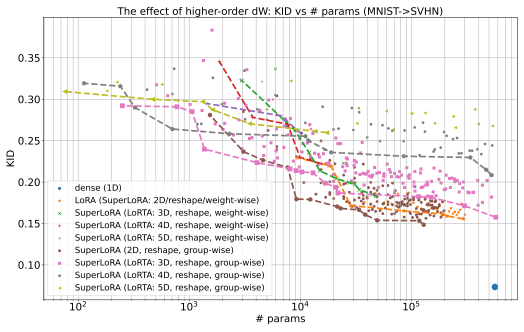
Figure 5.25: Complete results of SuperLoRA (LoNKr) for transfer learning from MNIST to SVHN.



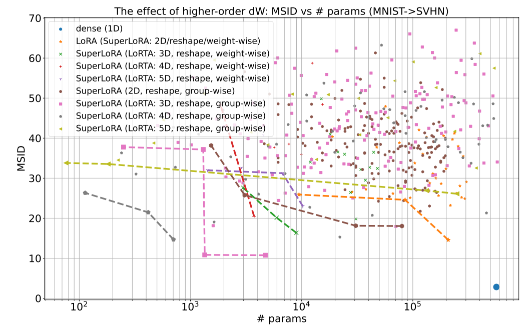
(a) SuperLoRA (LoRTA, FID)



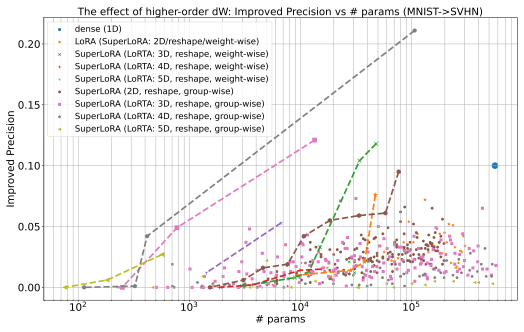
(b) SuperLoRA (LoRTA, IS)



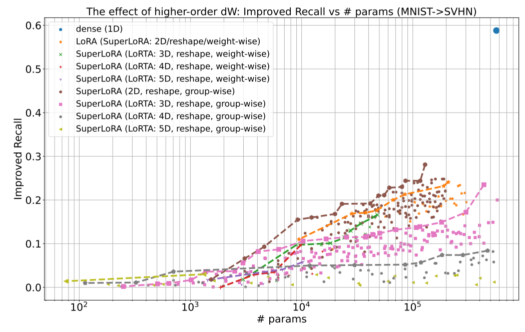
(c) SuperLoRA (LoRTA, KID)



(d) SuperLoRA (LoRTA, MSID)

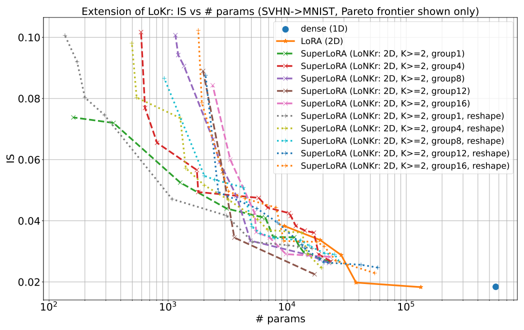


(e) SuperLoRA (LoRTA, Improved Precision)

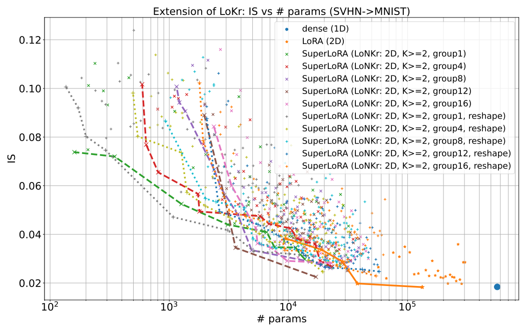


(f) SuperLoRA (LoRTA, Improved Recall)

Figure 5.26: Complete results of LoRTA for transfer learning from MNIST to SVHN.

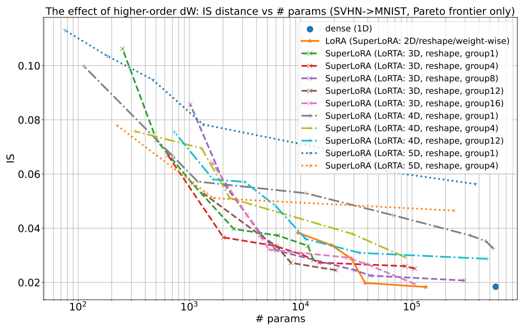


(a) SuperLoRA (LoNKr, Pareto frontier only)

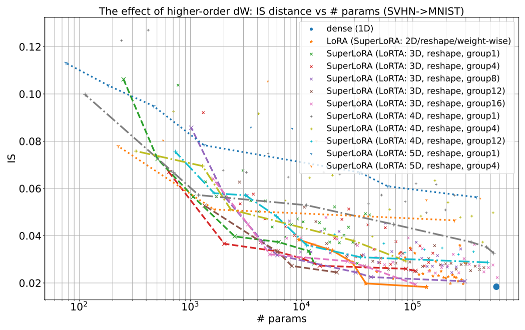


(b) SuperLoRA (LoNKr)

Figure 5.27: Effect of groups in LoNKr.

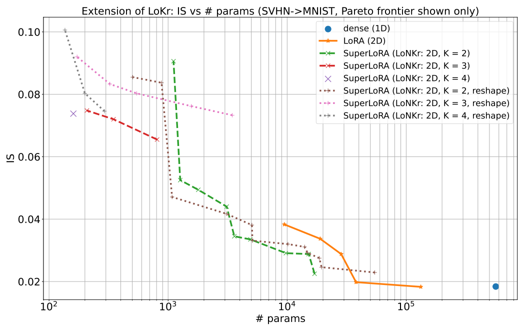


(a) SuperLoRA (LoRTA, Pareto frontier only)

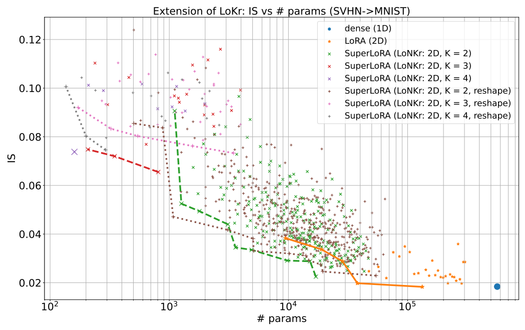


(b) SuperLoRA (LoRTA)

Figure 5.28: Effect of groups in LoRTA.



(a) SuperLoRA (LoNKr, Pareto frontier only)



(b) SuperLoRA (LoNKr)

Figure 5.29: Effect of K in LoNKr.

Chapter 6

Inference Efficiency: A Model Simplification Roadmap for Image Restoration on Mobile Devices

Inference efficiency aims to compress the model targeting on specific devices to reduce inference time at deployment. This chapter proposes a model simplification roadmap to reduce the model size and increase inference speed when deployed on mobile devices [26].

6.1 Introduction

During the past decade, deep learning has been the dominant approach for various computer vision tasks, including image classification [119, 188], object detection [101, 182], segmentation [138, 133], and image restoration [171, 16, 197, 196, 41, 33], owing to its remarkable feature extraction capabilities derived from a substantial number of parameters. However, these large models often demand significant computational resources and result in longer inference latency, posing challenges for direct deployment on mobile devices. As a consequence, their application in mobile photography has been restricted.

To address this challenge, researchers have been exploring different approaches to simplify deep learning models by reducing the size of models directly, like channel dimensions and the number of layers [15, 91, 170]. Further, some works design efficient modules to replace inefficient modules [198, 207, 117, 71, 148, 166, 56], e.g. replacing the original convolution with depthwise convolution [15, 170]. However, most model compression methods focus on reducing the FLOPs or runtime only, at the cost of performance loss.

Partial Convolution (PConv) [14] is a recently proposed efficient module that simplifies original convolutions by performing calculations only on a portion of the channel dimension. In comparison to depthwise convolution, PConv demonstrates faster runtime during deployment on mobile devices, even though it requires more parameters. Building on this observation, we can leverage the idea of adding more parameters during model compression to compensate for any potential performance loss, particularly in terms of PSNR (Peak Signal-to-Noise Ratio). By doing so, the model can still achieve faster runtime through model compression algorithms, while also benefiting from the larger capacity enabled by the additional parameters. As a result, the overall performance of the model is either maintained or even improved, considering the enhanced capacity brought about by the increased parameters.

Motivated by this concept, we propose a model simplification roadmap, which encompasses a set of tricks and techniques designed to optimize image restoration models for deployment on mobile devices. Through the application of these techniques, we aim to strike the right balance between model efficiency and performance, enabling the deployment of high-quality image restoration models on mobile devices without compromising on speed or output quality. The contributions of this chapter are summarized below.

- We propose a model simplification roadmap, consisting of tricks to add parameters to expand model capacity first and then reduce runtime, i.e. ***MO****re parameters and* ***FA****ster (MOFA)*. Given a small model, the model can be further accelerated before deployment with improved performance following the roadmap.
- Building upon the concept of partial convolution, we propose Partial Depthwise Convolution (PDWConv) to further enhance acceleration. PDWConv involves performing depthwise convolution across only a portion of the channel dimension for input features.
- We tested our roadmap on widely used small image restoration models, PMRID and NAFNet, on image restoration tasks including image denoising, image deblurring, and image deraining. Extensive experiments show that using our tricks, runtime for PMRID and NAFNet is decreased by up to 13% and parameters are reduced by 23% with better PSNR and SSIM.

6.2 Related work

This section presents related works on image restoration models on mobile devices as well as those modules designed to increase efficiency.

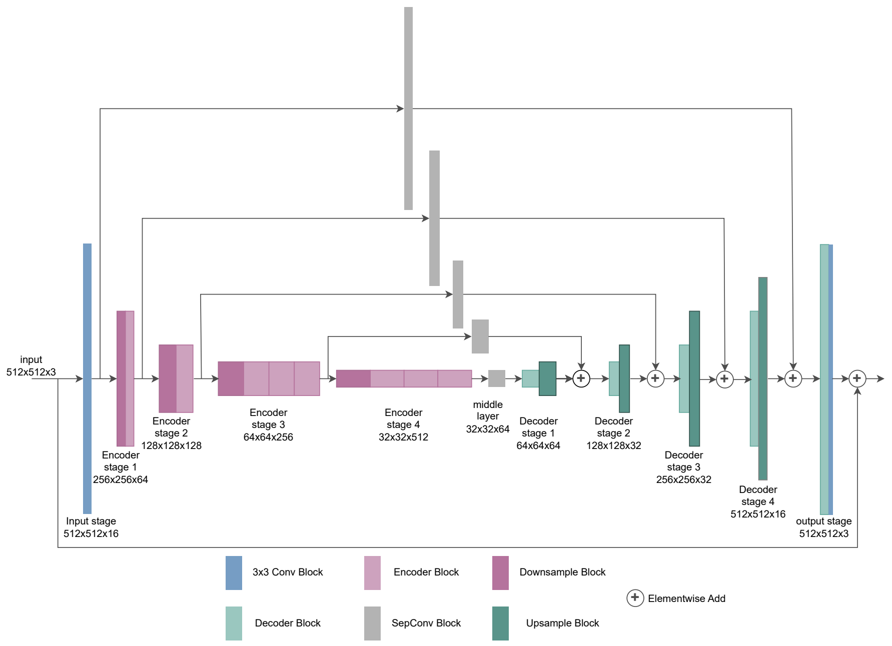


Figure 6.1: The overall network structure of PMRID

6.2.1 Image restoration on mobile devices

Image restoration aims to reverse degraded images into high-quality images. Tasks include image denoising, image deblurring, image draining, etc. These tasks are more commonly used in mobile devices like mobile photography. This encourages researchers to work on designing efficient deep learning models considering both runtime and image quality (IQ) [90, 57, 180, 76, 29, 210, 189, 80, 30, 13, 206]. PMRID [170] is one of the earliest efficient models. It carefully designs a U-Net-like model and increases efficiency by putting most parameters and calculations on the encoder stages. MicroISP [77] proposes an attention-based multi-branch model so that it can be adapted to the different computational powers of edge devices flexibly. MFDNet [111] designs a mobile-friendly attention module coupled with a reparameterization module to increase the denoising efficiency

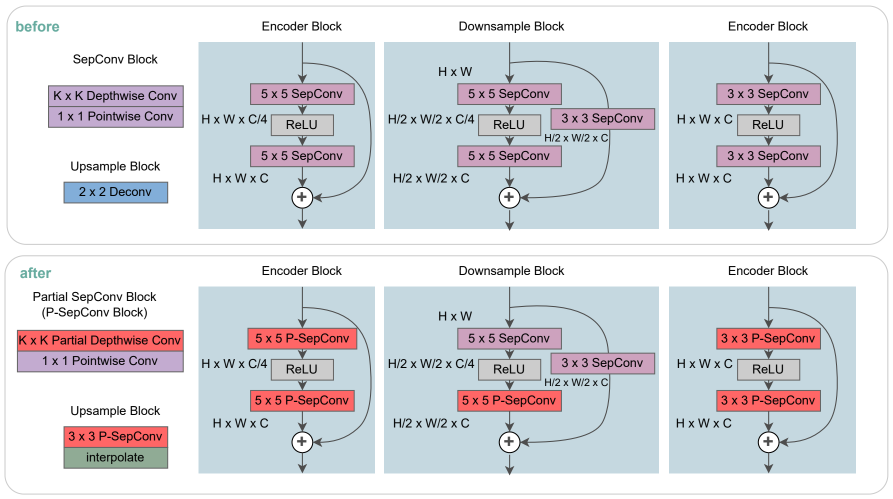


Figure 6.2: Detailed Blocks. We replace all Separable Convolutions (stride = 1) with one of the following convolutions: partial depth-wise convolution and partial convolution (PConv), with the portion 1/4 or 1/2.

on iPhone 11. FusionNet [91] designs a U-Net-like model to take two images from dual cameras as input and output the deblurred images. Beyond image restoration, MobileOne [162] proposes re-parameterizable structures to decrease the runtime during inference for general tasks like image classification. Orthogonally, we propose a roadmap to further accelerate U-Net-like backbones.

6.2.2 Efficient modules on computer vision

The first set of efficient modules is related to the convolution operation. Vanilla convolution performs well but it visits each pixel the number of output channel times, which takes much computation and is inefficient for mobile applications. To improve this, Xception [31] proposes to use separable convolution, a depthwise convolution followed by a pointwise convolution, which requires visiting each pixel only once by depthwise convolution. This reduces computation greatly even counting the following pointwise convolution. PConv [14] proposes to calculate partial convolution, i.e. a portion like 1/4 of the input feature across channel dimension as they observe most kernels are learning similar features.

In addition, some works propose a model simplification process by reducing unnecessary modules or replacing them with more efficient modules, e.g. NAFNet [15], ConvneXt [110] and Efficient Unet [144]. Other efficient modules include dynamic convolution [54], fast nearest convolution [115], IRA block [34], and self-calibrated module [116].

However, all these consider only the efficiency of a single module and attempt to replace them with more efficient ones. Instead, we consider the efficiency of the entire network and manipulate inefficient layers according to the FLOPs distribution. PConv module provides us the flexibility to add parameters freely by setting the portion differently for different layers.

6.3 Roadmap for simplification

This section presents our model simplification roadmap, under the guideline of using more parameters wherever possible to compensate for the performance loss brought by rapid runtime decrease. We employ the widely used PMRID model from image denoising as an example for illustration. Following our roadmap, either the runtime gets reduced drastically or more parameters are added to the model in each step. At the end of optimization, the runtime is decreased by 11%, the number of parameters is reduced by 6% and the PSNR is increased by 0.02dB. Noticeably, this roadmap can be applied to any U-Net-like model as a model compression method before deployment on mobile devices.

Settings We train PMRID on SIDD dataset. Runtime is tested on one single v100 GPU with input size $3 \times 1024 \times 1024$, averaging 600 rounds. Each trick mentioned below from subsection 6.3.2 is built on top of the previous ones.

6.3.1 Starting point - PMRID

PMRID [170] is a U-Net-like model proposed for raw image denoising on mobile devices as in Figure 6.1 and 6.2. It is worth noting that PMRID holds many more parameters than normal U-Net-like image restoration models on mobile devices like DnCNN [204] and NAFNet, but comparable runtime. This is because PMRID puts most of its parameters on its encoder only like half-UNet [113], while most U-Net-like models distribute equally on the encoder and decoder. This asymmetrical design gives more freedom to put even more parameters on high-level encoders, where

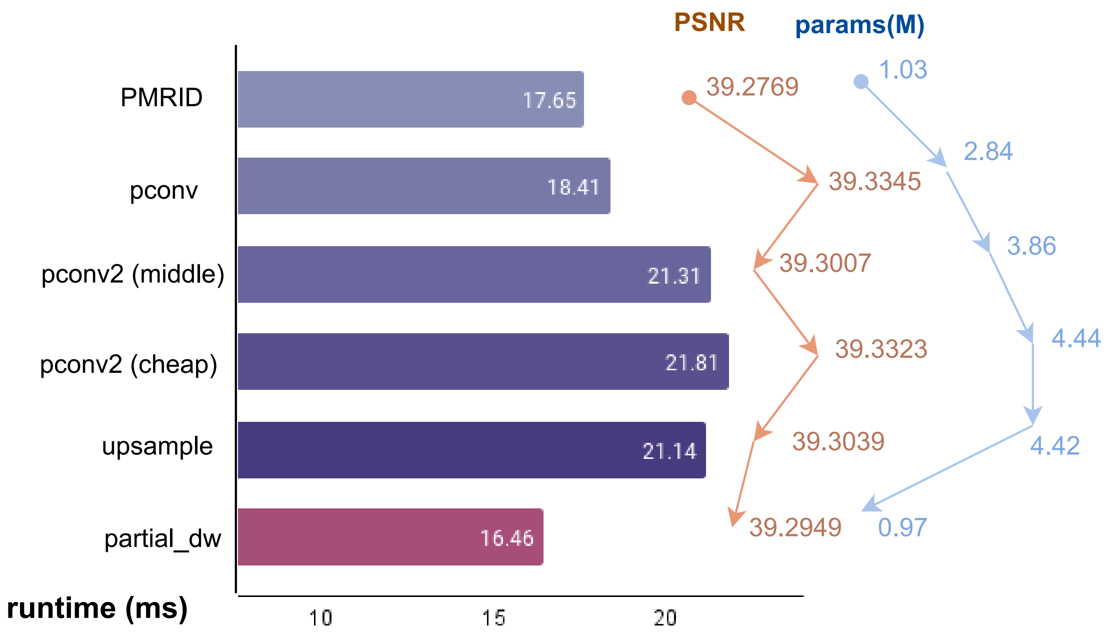


Figure 6.3: Roadmap. Starting from the baseline PMRID, we add each trick on top of all previous tricks, and it ends with partial_dw.

the input feature size is the smallest and hence the FLOPs will not increase drastically along with the increase of parameter amounts. For all convolutional layers, most layers use efficient depthwise convolution followed by point-wise convolution, excluding the input and output layer, where vanilla convolutions are used as the channel dimension is relatively small hence it will not add too many calculations/FLOPs into the whole structure.

6.3.2 Depthwise separable convolution to PConv

Partial convolution (PConv) [14] is a newly proposed efficient replacement for convolution modules on mobile devices, aiming to reduce both the number of floating-point operations (FLOPs), related to the number of multiply-adds, and the floating-point operations per second (FLOPS), considering the hardware deployment efficiency like memory access time. Unlike depthwise convolution which requires much memory access as convolution among different depths does not share parameters, partial convolution calculates only part of the convolution in terms of channel dimension, e.g. $1/4$ by default. In this way, the computation gets reduced as depthwise convolution compared with the original convolution while it also enjoys the deployment time reduction due to parameter sharing as the original convolution, reducing memory access time drastically compared with depthwise convolution. Besides this, partial convolutions provide us more flexibility to manipulate later, e.g. the portion. Hence, we first replace all depth-wise separable convolutions with partial convolutions. If there are vanilla convolutions exist (except the input and output layer), we also replace all those convolutions with partial convolutions directly. After this replacement, PSNR and SSIM increased to 39.3345 and 0.9559 respectively and runtime also increased by 0.76ms. This may be caused by the increase in parameters and FLOPs after the replacement. From now on, we use this partial convolution version as our model.

6.3.3 More parameters - middle layers

More parameters mean higher network capacity and better performance at most times. This motivates us to add more parameters to compensate for the PSNR and SSIM loss brought by the conversion from convolution to partial convolution. To achieve this, we can manipulate how large the “part" is in partial convolution, e.g. increasing it from default $1/4$ to $1/2$. Another problem is, where can we put those extra parameters to increase feature extraction ability without hurting runtime too much?

As for image restoration tasks, they target reconstructing pixels, not the semantic understanding of the whole images as high-level vision tasks, e.g. image classification and detection. Thus we claim that more parameters should be added in shallow layers to increase low-level feature extraction power. However, more parameters in shallow layers usually mean much larger FLOPs and runtime as the feature size for shallow layers is usually larger than deeper layers. Considering both the performance and runtime, we increase the portion from $1/4$ to $1/2$ in partial convolution to add parameters on middle layers for both encoder and decoder, e.g., encoder 3 and decoder 2 for the PMRID model. As a result, PNSR and SSIM decreased a little bit, and runtime increased only 2.9ms even if the number of parameters increased from 2.84M to 3.86M and FLOPs from 2.12G to 3.25G. This increases model capacity.

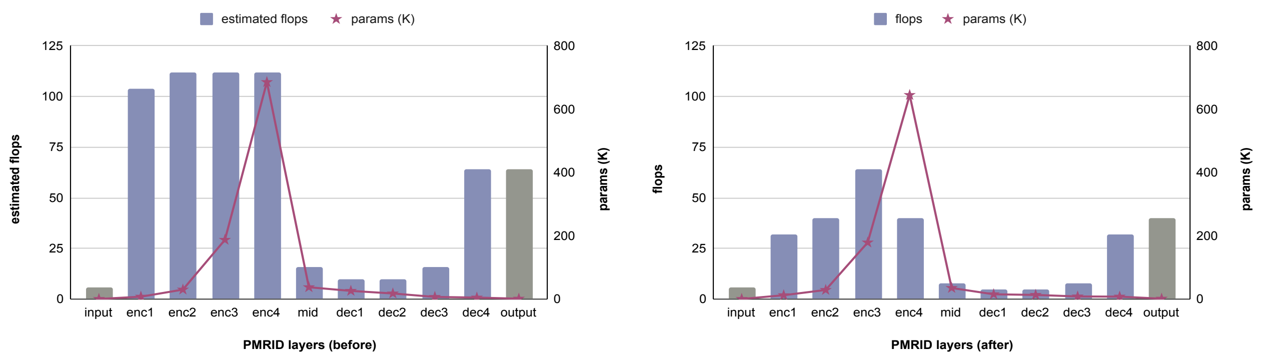


Figure 6.4: Estimated FLOPs distribution across layers before/after applying our roadmap tricks (dimension threshold $d * p$ is 0 here). Note that in PMRID structure, some layers (e.g. input and output layers) are vanilla convolutions and separable convolutions elsewhere. To get a clear structure-related FLOPs distribution (only related to the channel dimensions and feature size changes), we estimate the FLOPs by assuming all layers are vanilla convolutions and excluding FLOPs in skip connections.

6.3.4 More parameters - cheap layers

To add more parameters without adding too many FLOPs and runtime, we first examine the flop distribution for each layer. There are two ways to achieve this. The most intuitive way is to calculate the FLOPs layer by layer using Python tools. However, this will include some unnecessary FLOPs we are not interested in here, e.g. calculation in skip connections. Another way is to use a theoretical formula to estimate the FLOPs of convolution layers only. Here, we use the second method. The estimated FLOPs distribution can be found in Figure 6.4. It shows that all the decoders along with the middle layer take up relatively fewer FLOPs compared with encoders, hence we call them “cheap layers". As a result, we increase the portion from $1/4$ to $1/2$ for partial convolutions in decoder layers. After adding this trick, PSNR, and SIMM increased to 39.3323 and 0.9560 respectively while runtime only increased 0.5ms. The amount of parameters now is 4.44M, which is $4\times$ compared with the original PMRID model. After adding parameters to increase the model capacity using previous tricks, the following subsections focus on reducing the runtime.

6.3.5 Faster - upsampling/downsampling

Downsampling and upsampling modules are vital in U-Net-like networks. There are some choices for them, e.g. PMRID used depthwise separable convolution with stride 2 for downsampling and deconvolution for upsampling while NAFNet employed convolution coupled with pixel shuffle for upsampling. Instead, paper [144] argued that performing downsampling before the convolution and upsampling after the convolution can improve the efficiency, different from the normal order for downsampling and upsampling. Hence, we may first decouple downsampling from convolution with stride 2 to one average pooling layer followed by a convolution layer with stride 1. Upsampling deconvolutions are replaced with an interpolation layer following a stride-1 convolution layer. We select the most efficient downsampling/upsampling from the choices mentioned above and the original downsampling/upsampling strategies for different structures. For PMRID, we replace only the upsampling layers with the decoupled ones. After this change, the runtime gets reduced by 0.67 ms, PSNR dropped a little bit while SSIM remains the same.

6.3.6 Faster - PConv to PDWConv

To further accelerate it, we propose to replace partial convolution (PConv) with partial depthwise convolution (PDWConv). As illustrated in Figure 6.5, Partial depthwise convolution calculates only a portion across the channel dimension of the input features as in PConv and the rest untouched part uses identity mapping. Unlike PConv, we perform depthwise convolution for the selected portion while PConv uses original convolution.

Besides this, noticing that after taking the portion across the channel dimension, the input feature dimension will be decreased drastically. For features with small channel dimensions, e.g. 64 or 32, only 16 or 8 channels are used to learn the parameters, which may hurt the performance worse than larger features. Thus, we replace only those PConv with PDWConv for larger channel dimensions while keeping PConv for smaller ones. For the PMRID model, we set the dimension threshold $d * p$ (d is the input channel dimension for the current layer and p is the portion) to 32 according to the experimental results.

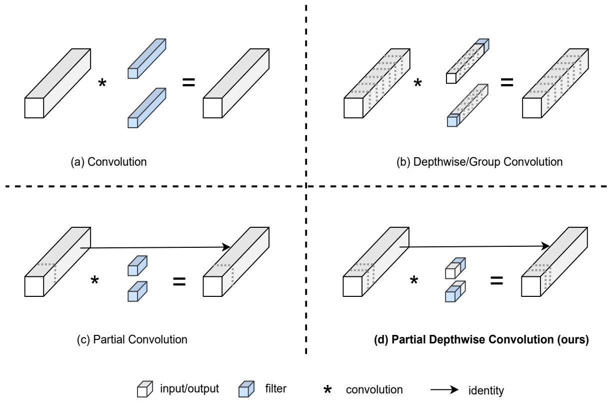


Figure 6.5: The structure of vanilla convolution, depthwise convolution, partial convolution, and partial depthwise convolution.

Finally, the runtime gets decreased to 16.46ms while PSNR and SSIM become 39.2769 and 0.9556 respectively, still better than the original PMRID backbone.

6.4 Experiments

This section presents a detailed ablation study and analysis of some tricks in enhancing PMRID model on SIDD denoising dataset and both quantitative and qualitative results when we accelerate other models on RGB image denoising, image deblurring, raw image denoising, image deblurring with JPEG artifacts and image deraining tasks.

6.4.1 Setting

To verify the effectiveness of our pipeline, we apply it on two different models, NAFNet-tiny [15] which has only 7 blocks, and the backbone from PMRID [170]. For both models, our settings are mainly based on NAFNet [15] as PMRID lacks the settings and reported results on most image restoration benchmarks. Adam optimizer [84] is used with $\beta_1 = 0.9$, $\beta_2 = 0.9$ and weight decay 0. Total iteration is $200K$ during training. In addition, we report the FLOPs (input size $3 \times 256 \times 256$) and the number of parameters for comparison. For the performance, we use PSNR and SSIM for quantitative comparison. Qualitative visualization can be found in Section 6.4.5. We tested runtime using two ways. The first one is GPU-based as mentioned above. We tested on a single v100 with $3 \times 1024 \times 1024$ as the input size, averaging from 600 rounds. Besides this, we also test our models on Android phone Tecno Camon 19 Pro based on Pytorch Mobile framework, using image size $3 \times 256 \times 256$, averaging runtime from 3 forward passes. We use a larger input size on GPU runtime testing because when both our model and input are too small, the variance of runtime would be comparable to the actual runtime on v100 GPU, leading to biased averaged runtime.

	PConv	PConv2 (middle)	PConv2 (cheap)	down/up	PDWConv	FLOPs (G)	Param (M)	PSNR(↑)	SSIM(↑)	runtime (ms) (↓) GPU	Mobile
baseline						1.11	1.03	39.2769	0.9556	17.65	220
	✓					2.12	2.84	**39.3345**	**0.9559**	18.41	**212**
	✓	✓				3.25	3.86	**39.3007**	**0.9557**	21.31	250
	✓		✓			2.69	3.46	**39.3211**	**0.9559**	19.19	218
	✓	✓	✓			3.69	4.44	**39.3323**	**0.9560**	21.81	255
				✓		0.95	1.01	39.2741	0.9556	**16.99**	**153**
	✓				✓	1.03	0.95	39.2104	0.9552	**14.46**	**162**
	✓	✓	✓	✓	✓	1.11	0.97	**39.2949**	**0.9558**	**16.46**	**196**

Table 6.1: Ablation study based on PMRID model for image denoising on SIDD dataset.

6.4.2 Ablation study

In this subsection, we examine how each trick works in this acceleration process. Results can be found in Table 6.1. We can find the first 3 tricks mainly increased the PSNR and SSIM by adding more parameters efficiently without adding too much runtime. Specifically, replacing with partial convolution increased PSNR and SSIM to 39.3345 and 0.9559 respectively compared with the baseline (PSNR: 39.2769, SSIM: 0.9556). Adding more parameters for middle layers also boosts PSNR and SSIM to 39.3007 and 0.9557 separately. Adding more parameters on cheap layers coupled with middle layers together further increased PSNR to 39.3323 and SSIM to 0.9560. All these tricks bring much more parameters to the model to increase the model capacity without introducing too many FLOPs and runtime, making space for the runtime reduction tricks to keep PSNR and SSIM.

For the upsampling/downsampling modules, PMRID uses deconvolution layers to perform upsampling. When we decouple it into a convolution layer and an upsampling layer, the runtime gets reduced by 0.65ms while PSNR and SSIM are still comparable.

Finally, we replace all PConv with PDWConv, i.e. we use depth-wise convolution to replace the original convolution in PConv and it results in partial depth-wise convolution as illustrated in Figure 6.5. When we apply this directly on PConv skipping all steps in the middle, performance dropped from (PSNR: 39.3345, SSIM: 0.9559) to (PSNR: 39.2104, SSIM: 0.9552), even if runtime dropped significantly. This also demonstrates the necessity of expanding the model capacity to increase the model performance in the middle steps.

6.4.3 Analysis of different components

This subsection presents experimental results of how to determine alternatives for each trick.

layers	FLOPs (G)	Param(M)	PSNR	SSIM
baseline	1.11	1.03	39.2769	0.9556
+PConv	2.12	2.84	39.3345	0.9559
enc3, dec2	3.25	3.86	39.3007	0.9557
enc2, dec3	2.58	2.93	39.2508	0.9556
enc1, dec4	2.54	2.86	39.2770	0.9556

Table 6.2: Adding more parameters to middle layers on PMRID.

Analysis of middle layers As mentioned above, we aim to add more parameters in middle-level layers to balance between performance and runtime, as we believe more parameters in shallower layers would be more appealing for low-level vision tasks as they focus more on extracting low-level features. However, more parameters in shallow layers also mean many more FLOPs as they take the largest feature size in the entire U-Net-like network. As a result, we decide to add parameters on the middle layers by increasing the portion of PConv from 1/4 to 1/2. The experiments can be found in Table 6.2.

As adding more parameters on PConv layers in encoder stage 3 and decoder stage 2 increased PSNR from 39.2769 to 39.3007 compared with the PMRID backbone baseline while other encoder-decoder pairs even dropped the PSNR and SSIM, we use encoder stage 3 and decoder stage 2 as our final middle layers to add parameter in PMRID. Besides this, this pair also introduces much more parameters than others. This is because encoder stage 3 in this pair has 4 encoder blocks while other encoder stages have only 2.

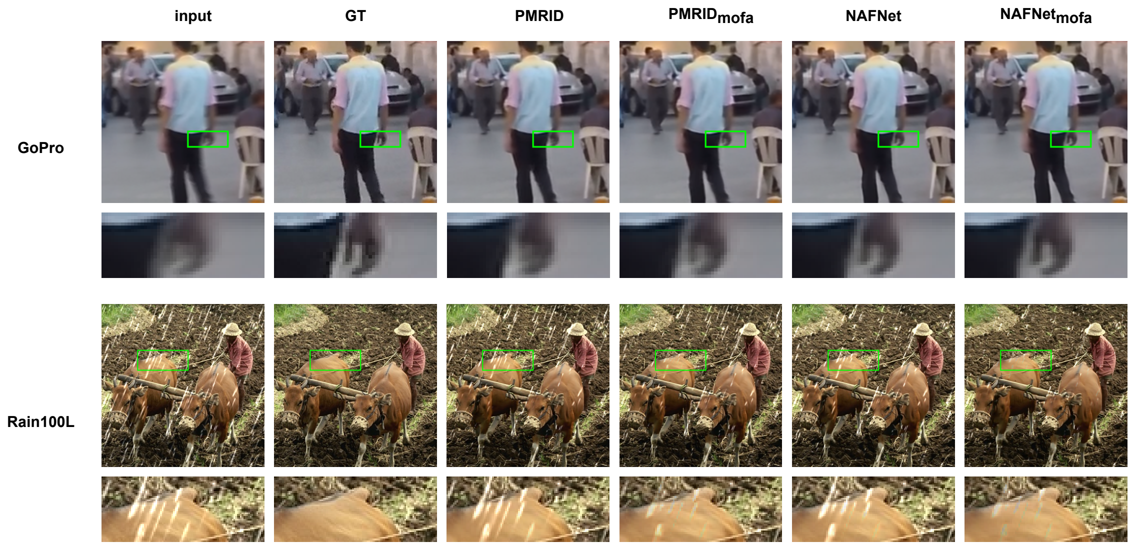


Figure 6.6: Visualization. Image restoration results from PMRID, PMRID$_{mofa}$, NAFNet and NAFNet$_{mofa}$ on different tasks.

replacement	Param(M)	runtime	PSNR	SSIM
no	4.44	255	39.3323	0.9560
upsampling	4.42	268	39.3039	0.9561
downsampling	4.61	460	39.3334	0.9559

Table 6.3: Replacement of upsampling/downsampling in PMRID.

Analysis of upsampling/downsampling layers For upsampling, we replace deconvolution layers with convolution and an upsampling layer in PMRID. This order can decrease the FLOPs compared with the reverse. For downsampling, the downsampling layer coupled with the convolution layer (stride 1) replaced the stride 2 convolution layers. Results can be found in Table 6.3. Here, the runtime is based on Pytorch Mobile tested on Android phones. We can find adding upsampling can reduce GPU runtime a little bit as in Table 6.1 while increasing inference time on the device as in Table 6.3. This may be caused by different acceleration implementations for GPU

Method	runtime (ms) Mobile	# params	FLOPs (G)	denoising (SIDD) PSNR	denoising (SIDD) SSIM	debluring (GoPro) PSNR	debluring (GoPro) SSIM	debluring (REDS) PSNR	debluring (REDS) SSIM
PMRID	220*	1.03M	1.15	39.2769	0.9556	29.0840	0.9153	27.6501	0.8348
PMRID$_{mofa}$	196(*-11%*)	0.97M(*-6%*)	1.11	**39.2949**	**0.9558**	**29.2274**	**0.9176**	27.6493	0.8347
NAFNet	69*	286.6K	1.03	39.2042	0.9555	29.4337	0.9237	27.6747	0.8381
NAFNet$_{mofa}$	60(*-13%*)	222.1K(*-23%*)	0.94	**39.3098**	**0.9560**	**29.4810**	**0.9242**	**27.7450**	**0.8398**

Table 6.4: Image denoising and deblurring results on different datasets. * Represents we also added *split_cat* operations for fair comparison.

Method	Test100 PSNR	Test100 SSIM	Rain100H PSNR	Rain100H SSIM	Rain100L PSNR	Rain100L SSIM	Test2800 PSNR	Test2800 SSIM	Test1200 PSNR	Test1200 SSIM
PMRID	28.3400	0.8690	27.9492	0.8343	32.5887	0.9316	32.6028	0.9260	31.4316	0.9001
PMRID$_{mofa}$	28.2158	**0.8739**	**28.3177**	**0.8453**	**32.7912**	**0.9330**	**32.6106**	**0.9272**	**31.4936**	**0.9034**
NAFNet	28.0719	0.8775	27.8863	0.8414	33.6056	0.9467	32.8942	0.9298	32.6602	0.9181
NAFNet$_{mofa}$	**28.9186**	**0.8888**	**28.5133**	**0.8531**	**33.9560**	**0.9500**	**32.9450**	**0.9303**	32.4535	0.9150

Table 6.5: Image deraining results on different test datasets.

and Pytorch Mobile on devices. Besides this, downsampling increased runtime on Pytorch Mobile a lot for PMRID, from 255ms to 460ms. Hence, we replace upsampling only for PMRID.

thres $d*p$	runtime(ms)	Param(M)	PSNR	SSIM
0	14.10	0.94	39.2513	0.9554
16	15.8	0.95	39.2408	0.9557
32	16.46	0.97	39.2949	0.9558
64	16.56	1.21	39.2920	0.9559
128	16.71	1.25	39.2973	0.9561

Table 6.6: The effect of threshold $d*p$ to replace PConv with PDWConv.

Analysis of PDWConv threshold Noticing that PDWConv calculates only a portion of the entire depthwise convolution across channel dimension, and when the dimension is relatively small, e.g. 32, it may hurt performance drastically compared with the convolution in PConv which hold much more parameters. As a result, we replace only PConv with PDWConv with a larger channel dimension and set a threshold to determine. If the channel dimension of input features d divided by the portion p, i.e. $d*p, p=[1/2,1/4]$, the actual channel dimension to be calculated, is larger than the threshold, replace PConv with PDWConv, otherwise, we keep them PConv. Considering both runtime and PSNR/SSIM, we choose 32 as the threshold for PMRID. It may differ for different backbones.

6.4.4 Applications

We apply our roadmap with two different backbones, PMRID [170] and NAFNet [15], and get their accelerated versions PMRID$_{mofa}$ and NAFNet$_{mofa}$. As layer normalization is not deployable on mobile devices, we remove this layer directly to test runtime. We tested these models on various image restoration tasks, including denoising, deblurring, and deraining.

RGB Image Denoising. We test denoising performance on SIDD dataset. Quantitative results are shown in Table 6.4. This table shows for both PMRID and NAFNet, our accelerated versions decreased runtime on mobile devices by 11% and 13%, and PSNR and SSIM are also increased at the same time, from 39.2769dB to 39.2949dB for PMRID and from 39.2319dB to 39.3098dB for NAFNet. Note that runtime reduction is applied to all the following tasks as we use the same models for comparison.

Image Deblurring. We tested image deblurring performance on GoPro [125] dataset. As shown in Table 6.4, we have increased PSNR by 0.14dB for PMRID and 0.05dB for NAFNet while reducing the runtime by more than 10% for both backbones.

Image Deblurring with JPEG artifacts. As in NAFNet [170], we also tested the image deblurring effect with JPEG artifacts. Following settings in NAFNet [170], we use REDS [126] dataset and evaluate on REDS-val-300. Table 6.4 shows NAFNet$_{mofa}$ increased PSNR by 0.07dB compared with NAFNet while the runtime gets decreased and PMRID$_{mofa}$ is comparable to PMRID.

Image deraining. Following settings in HINet [16], we also train our models on image deraining task with Rain13k dataset and test on Test100 [200], Rain100H [187], Rain100L [187], Test2800 [47] and Test1200 [199] datasets. Quantitative results are shown in Table 6.5. From this table, our accelerated versions for both backbones have improved the performance for most deraining datasets while reducing the runtime. Specifically, PMRID$_{mofa}$ defeats PMRID on Rain100H, Rain100L, Test2800 and Test1200 datasets, and NAFNet$_{mofa}$ improves over NAFNet on Test100, Rain100H, Rain100L and Test2800 datasets.

6.4.5 Visualization

To understand the qualitative results, we visualize image restoration results on different datasets, SIDD dataset for image denoising, GoPro dataset for image deblurring, REDS dataset for image deblurring with JPEG artifacts, and Rain100L dataset for deraining as in Figure 6.6.

For image denoising, we can find all these models, PMRID, PMRID$_{mofa}$, NAFNet and NAFNet$_{mofa}$ have good denoising ability. Among them, we observe NAFNet$_{mofa}$ is the best.

In GoPro dataset, the quality of deblurring results increases from PMRID to PMRID$_{mofa}$ where the shape of the hand in the figure changes from blurred for PMRID to clear for NAFNet$_{mofa}$.

For deraining results on Rain100L dataset, we can see both PMRID$_{mofa}$ and NAFNet$_{mofa}$ show better deraining results than their baselines.

6.5 Conclusion and limitation

This chapter has presented a comprehensive roadmap to accelerate image restoration models before deploying them on mobile devices. The key idea is to enhance the model's capacity without significantly increasing runtime by strategically adding more parameters to FLOP-insensitive layers and middle layers. Additionally, we propose the decoupling of upsampling/downsampling layers and apply partial depthwise convolution to further accelerate the models. The proposed roadmap is effectively applied to PMRID and NAFNet for various image restoration tasks. Extensive experimental results demonstrate the effectiveness of our strategy. Following our approach, we achieve a runtime reduction of up to 13% and reduce the number of parameters by 23% while simultaneously improving the image restoration performance.

Our approach offers a powerful solution for deploying image restoration models on mobile devices, striking a balance between increased model capacity and faster inference without compromising restoration quality. For deployment, the decrease of parameters is suitable for parameters-sensitive implementation like sensor programming. The introduction of $split\_cat$ operation in PDWConv and PConv might be less efficient in deployment based on Pytorch Mobile, ONNX and Caffe and need to be further accelerated.

Chapter 7

Conclusion and Future Work

In this proposal, we eye on efficient deep learning and propose methods to increase efficiency from data, model, training and inference. Extensive experiments have demonstrate their effectiveness. There are also some future works we are investigating, including exploring new image representations to increase data efficiency of vision tasks, exploring better quantization algorithms, reducing training time for large vision models from scratch and designing new structure.

As we know, images are represented in terms of pixel density, which enables them to be stored and reproduced on digital devices without significant loss of quality. In contrast, text or languages used in natural language processing are crafted by humans primarily for communication purposes. Text typically demonstrates superior domain generalization and performs better in models like transformers because it possesses higher information density. However, it's important to note that neither representation was originally designed with machine learning models in mind; they were created for human use. Inspired by this observation, new representations for images, and potentially for all modalities, can be developed. An ideal representation would prioritize high information density for machine learning models, rather than for human interpretation.

Current quantization algorithms focus on adaptively/non-adaptively quantize models into lower-precision, even as low as 1-bit. This brings a compression ratio to 1/32 at most from FP32 to 1-bit. However, further quantization can be explored, like 0-bit. In this way, quantization becomes pruning and further compression can be made with encouraging more 0-bit quantization, i.e., pruning.

Training large models with extensive datasets is inherently time-consuming, which, while understandable, is often impractical. To address this, it is possible to explore new algorithms designed to directly reduce training times. Such advancements could eliminate the need for domain matching in transfer learning, as reliance on transfer techniques would no longer be necessary.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report, 2023.

[2] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[3] Amit Alfassy, Leonid Karlinsky, Amit Aides, Joseph Shtok, Sivan Harary, Rogerio Feris, Raja Giryes, and Alex M Bronstein. Laso: Label-set operations networks for multi-label few-shot learning. In *CVPR*, pages 6548–6557, 2019.

[4] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[5] Rohan Anil, Andrew M. Dai, and Orhan Firat et al. PaLM 2 technical report, 2023.

[6] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9368–9377, 2018.

[7] Daniel Bershatsky, Daria Cherniuk, Talgat Daulbaev, and Ivan Oseledets. LoTR: Low tensor rank weight adaptation. *arXiv preprint arXiv:2402.01376*, 2024.

[8] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[9] Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10925–10934, 2022.

[10] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. *arXiv preprint arXiv:1801.01401*, 2018.

[11] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9650–9660, 2021.

[12] Feng Cen, Xiaoyu Zhao, Wuzhuang Li, and Guanghui Wang. Deep feature augmentation for occluded image classification. *Pattern Recognition*, 111:107737, 2021.

[13] Hanting Chen, Yunhe Wang, Jianyuan Guo, and Dacheng Tao. Vanillanet: the power of minimalism in deep learning. *arXiv preprint arXiv:2305.12972*, 2023.

[14] Jierun Chen, Shiu-hong Kao, Hao He, Weipeng Zhuo, Song Wen, Chul-Ho Lee, and S-H Gary Chan. Run, don't walk: Chasing higher flops for faster neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12021–12031, 2023.

[15] Liangyu Chen, Xiaojie Chu, Xiangyu Zhang, and Jian Sun. Simple baselines for image restoration. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VII*, pages 17–33. Springer, 2022.

[16] Liangyu Chen, Xin Lu, Jie Zhang, Xiaojie Chu, and Chengpeng Chen. Hinet: Half instance normalization network for image restoration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 182–192, 2021.

[17] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. AdaptFormer: Adapting vision transformers for scalable visual recognition. *Advances in Neural Information Processing Systems*, 35:16664–16678, 2022.

[18] Wei Chen, Zichen Miao, and Qiang Qiu. Parameter-efficient tuning of large convolutional models. *arXiv preprint arXiv:2403.00269*, 2024.

[19] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[20] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing convolutional neural networks in the frequency domain. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1475–1484, 2016.

[21] Xiangyu Chen, Qinghao Hu, Kaidong Li, Cuncong Zhong, and Guanghui Wang. Accumulated trivial attention matters in vision transformers on small datasets. *arXiv preprint arXiv:2210.12333*, 2022.

[22] Xiangyu Chen, Jing Liu, Ye Wang, Matthew Brand, Guanghui Wang, Toshiaki Koike-Akino, et al. Superlora: Parameter-efficient unified adaptation of multi-layer attention modules. 2024.

[23] Xiangyu Chen, Ying Qin, Wenju Xu, Andrés M Bur, Concong Zhong, and Guanghui Wang. Increasing input information density for vision transformers on small datasets. *Available at SSRN 4179882*.

[24] Xiangyu Chen, Ying Qin, Wenju Xu, Andrés M Bur, Cuncong Zhong, and Guanghui Wang. Improving vision transformers on small datasets by increasing input information density in frequency domain. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2022.

[25] Xiangyu Chen and Guanghui Wang. Few-shot learning by integrating spatial and frequency representation. In *2021 18th Conference on Robots and Vision (CRV)*, pages 49–56. IEEE, 2021.

[26] Xiangyu Chen, Ruiwen Zhen, Shuai Li, Xiaotian Li, and Guanghui Wang. Mofa: A model simplification roadmap for image restoration on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1322–1332, 2023.

[27] Yinbo Chen, Zhuang Liu, Huijuan Xu, Trevor Darrell, and Xiaolong Wang. Meta-baseline: Exploring simple meta-learning for few-shot learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9062–9071, 2021.

[28] Zitian Chen, Yanwei Fu, Yu-Xiong Wang, Lin Ma, Wei Liu, and Martial Hebert. Image deformation meta-networks for one-shot learning. In *CVPR*, pages 8680–8689, 2019.

[29] Haram Choi, Cheolwoong Na, Jinseop Kim, and Jihoon Yang. Exploration of lightweight single image denoising with transformers and truly fair training. In *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval*, pages 452–461, 2023.

[30] Haram Choi, Cheolwoong Na, Jihyeon Oh, Seungjae Lee, Jinseop Kim, Subeen Choe, Jeongmin Lee, Taehoon Kim, and Jihoon Yang. Ramit: Reciprocal attention mixing transformer for lightweight image restoration. *arXiv preprint arXiv:2305.11474*, 2023.

[31] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[32] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *Advances in Neural Information Processing Systems*, 34:9355–9366, 2021.

[33] Xiaojie Chu, Liangyu Chen, Chengpeng Chen, and Xin Lu. Improving image restoration by revisiting global information aggregation. In *European Conference on Computer Vision*, pages 53–71. Springer, 2022.

[34] Marcos V Conde, Florin Vasluianu, Javier Vazquez-Corral, and Radu Timofte. Perceptual image enhancement for smartphone real-time applications. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1848–1858, 2023.

[35] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pages 113–123, 2019.

[36] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[38] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers. *arXiv preprint arXiv:2204.03645*, 2022.

[39] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12124–12134, 2022.

[40] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[41] Akshay Dudhane, Syed Waqas Zamir, Salman Khan, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Burst image restoration and enhancement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5759–5768, 2022.

[42] Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron Elmore, and Michael Franklin. Band-limited training and inference for convolutional neural networks. *arXiv preprint arXiv:1911.09287*, 2019.

[43] Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter efficient tuning with Kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022.

[44] Max Ehrlich and Larry S Davis. Deep residual learning in the jpeg transform domain. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3484–3493, 2019.

[45] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6824–6835, 2021.

[46] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[47] Xueyang Fu, Jiabin Huang, Delu Zeng, Yue Huang, Xinghao Ding, and John Paisley. Removing rain from single images via a deep detail network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3855–3863, 2017.

[48] Kamala Gajurel, Cuncong Zhong, and Guanghui Wang. A fine-grained visual attention approach for fingerspelling recognition in the wild. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3266–3271. IEEE, 2021.

[49] Hanan Gani, Muzammal Naseer, and Mohammad Yaqub. How to train vision transformer on small-scale datasets? *arXiv preprint arXiv:2210.07240*, 2022.

[50] Arthita Ghosh and Rama Chellappa. Deep feature extraction in the dct domain. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3536–3541. IEEE, 2016.

[51] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12259–12269, 2021.

[52] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. In *Advances in Neural Information Processing Systems*, pages 3933–3944, 2018.

[53] Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896, 2021.

[54] Jiaming Guo, Xueyi Zou, Yuyi Chen, Yi Liu, Jianzhuang Liu, Youliang Yan, and Jia Hao. Asconvsr: Fast and lightweight super-resolution network with assembled convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1582–1592, 2023.

[55] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Cmt: Convolutional neural networks meet vision transformers. *arXiv preprint arXiv:2107.06263*, 2021.

[56] Meng-Hao Guo, Cheng-Ze Lu, Zheng-Ning Liu, Ming-Ming Cheng, and Shi-Min Hu. Visual attention network. *Computational Visual Media*, pages 1–20, 2023.

[57] Yu Guo, Axel Davy, Gabriele Facciolo, Jean-Michel Morel, and Qiyu Jin. Fast, nonlocal and neural: a lightweight high quality solution to image denoising. *IEEE Signal Processing Letters*, 28:1515–1519, 2021.

[58] Tianxiang Hao, Hui Chen, Yuchen Guo, and Guiguang Ding. Consolidator: Mergable adapter with group connections for visual adaptation. In *The Eleventh International Conference on Learning Representations*, 2022.

[59] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

[60] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2021.

[61] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2022.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[63] Lei He, Jiwen Lu, Guanghui Wang, Shiyu Song, and Jie Zhou. Sosd-net: Joint semantic object segmentation and depth estimation from monocular images. *Neurocomputing*, 440:251–263, 2021.

[64] Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. Parameter-efficient model adaptation for vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 817–825, 2023.

[65] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11936–11945, 2021.

[66] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[67] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[68] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[69] Qibin Hou, Daquan Zhou, and Jiashi Feng. Coordinate attention for efficient mobile network design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13713–13722, 2021.

[70] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

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

[72] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[73] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter retraining. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12084–12092, 2020.

[74] Yuqing Hu, Vincent Gripon, and Stéphane Pateux. Leveraging the feature distribution in transfer-based few-shot learning. *arXiv preprint arXiv:2006.03806*, 2020.

[75] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[76] Andrey Ignatov, Grigory Malivenko, Radu Timofte, Yu Tseng, Yu-Syuan Xu, Po-Hsiang Yu, Cheng-Ming Chiang, Hsien-Kai Kuo, Min-Hung Chen, Chia-Ming Cheng, et al. Pynet-v2 mobile: efficient on-device photo processing with neural networks. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 677–684. IEEE, 2022.

[77] Andrey Ignatov, Anastasia Sycheva, Radu Timofte, Yu Tseng, Yu-Syuan Xu, Po-Hsiang Yu, Cheng-Ming Chiang, Hsien-Kai Kuo, Min-Hung Chen, Chia-Ming Cheng, et al. Microisp: processing 32mp photos on mobile devices with deep learning. In *European Conference on Computer Vision*, pages 729–746. Springer, 2022.

[78] Shibo Jie and Zhi-Hong Deng. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 1060–1068, 2023.

[79] Shibo Jie, Haoqing Wang, and Zhi-Hong Deng. Revisiting the parameter efficiency of adapters from the perspective of precision redundancy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17217–17226, 2023.

[80] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting

the computational cost of deep learning models. In *2018 IEEE international conference on big data (Big Data)*, pages 3873–3882. IEEE, 2018.

[81] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

[82] Oscar Key, Jean Kaddour, and Pasquale Minervini. Local LoRA: Memory-efficient fine-tuning of large language models. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*, 2023.

[83] Jongyoo Kim and Sanghoon Lee. Deep learning of human visual sensitivity in image quality assessment framework. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1676–1684, 2017.

[84] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[85] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. 2020.

[86] Soroush Abbasi Koohpayegani and Hamed Pirsiavash. Sima: Simple softmax-free attention for vision transformers, 2022.

[87] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[88] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[89] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in Neural Information Processing Systems*, 32, 2019.

[90] Avisek Lahiri, Sourav Bairagya, Sutanu Bera, Siddhant Haldar, and Prabir Kumar Biswas. Lightweight modules for efficient deep learning based image restoration. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(4):1395–1410, 2020.

[91] Wei-Sheng Lai, Yichang Shih, Lun-Cheng Chu, Xiaotong Wu, Sung-Fang Tsai, Michael Krainin, Deqing Sun, and Chia-Kai Liang. Face deblurring using dual camera fusion on mobile phones. *ACM Transactions on Graphics (TOG)*, 41(4):1–16, 2022.

[92] Quoc Le, Tamás Sarlós, Alex Smola, et al. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the international conference on machine learning*, volume 85, page 8, 2013.

[93] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

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

[95] Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets. *arXiv preprint arXiv:2112.13492*, 2021.

[96] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, 2021.

[97] Aoxue Li, Tiange Luo, Zhiwu Lu, Tao Xiang, and Liwei Wang. Large-scale few-shot learning: Knowledge transfer with class hierarchy. In *CVPR*, pages 7212–7220, 2019.

[98] Aoxue Li, Tiange Luo, Tao Xiang, Weiran Huang, and Liwei Wang. Few-shot learning with global class representations. In *ICCV*, pages 9715–9724, 2019.

[99] Chunyuan Li, Jianwei Yang, Pengchuan Zhang, Mei Gao, Bin Xiao, Xiyang Dai, Lu Yuan, and Jianfeng Gao. Efficient self-supervised vision transformers for representation learning. 2022.

[100] Kai Li, Yulun Zhang, Kunpeng Li, and Yun Fu. Adversarial feature hallucination networks for few-shot learning. In *CVPR*, pages 13470–13479, 2020.

[101] Kaidong Li, Nina Y Wang, Yiju Yang, and Guanghui Wang. Sgnet: A super-class guided network for image classification and object detection. In *2021 18th Conference on Robots and Vision (CRV)*, pages 127–134. IEEE, 2021.

[102] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

[103] Yawei Li, Kai Zhang, Jiezhang Cao, Radu Timofte, and Luc Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021.

[104] Jing Liu, Toshiaki Koike-Akino, Pu Wang, Matthew Brand, Ye Wang, and Kieran Parsons. LoDA: Low-dimensional adaptation of large language models. *NeurIPS'23 Workshop on on Efficient Natural Language and Speech Processing*, 2023.

[105] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. In *ICCV*, pages 10551–10560, 2019.

[106] Yahui Liu, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco Nadai. Efficient training of visual transformers with small datasets. *Advances in Neural Information Processing Systems*, 34, 2021.

[107] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12009–12019, 2022.

[108] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[109] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, 34:28092–28103, 2021.

[110] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[111] Zhuoqun Liu, Meiguang Jin, Ying Chen, Huaida Liu, Canqian Yang, and Hongkai Xiong. Mfdnet: Towards real-time image denoising on mobile devices. *arXiv preprint arXiv:2211.04687*, 2022.

[112] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019.

[113] Haoran Lu, Yifei She, Jun Tie, and Shengzhou Xu. Half-unet: A simplified u-net architecture for medical image segmentation. *Frontiers in Neuroinformatics*, 16:911679, 2022.

[114] Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing Xu, Tao Xiang, and Li Zhang. Soft: softmax-free transformer with linear complexity. *Advances in Neural Information Processing Systems*, 34:21297–21309, 2021.

[115] Ziwei Luo, Youwei Li, Lei Yu, Qi Wu, Zhihong Wen, Haoqiang Fan, and Shuaicheng Liu. Fast nearest convolution for real-time efficient image super-resolution. In *European Conference on Computer Vision*, pages 561–572. Springer, 2022.

[116] Long Ma, Tengyu Ma, Risheng Liu, Xin Fan, and Zhongxuan Luo. Toward fast, flexible, and robust low-light image enhancement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5637–5646, 2022.

[117] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[118] Wenchi Ma, Kaidong Li, and Guanghui Wang. Location-aware box reasoning for anchor-based single-shot object detection. *Ieee Access*, 8:129300–129309, 2020.

[119] Wenchi Ma, Xuemin Tu, Bo Luo, and Guanghui Wang. Semantic clustering based deduction learning for image recognition and classification. *Pattern Recognition*, 124:108440, 2022.

[120] Wenchi Ma, Tianxiao Zhang, and Guanghui Wang. Miti-detr: Object detection based on transformers with mitigatory self-attention convergence. *arXiv preprint arXiv:2112.13310*, 2021.

[121] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[122] Puneet Mangla, Nupur Kumari, Abhishek Sinha, Mayank Singh, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Charting the right manifold: Manifold mixup for few-shot learning. In *WACV*, pages 2218–2227, 2020.

[123] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14297–14306, 2023.

[124] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

[125] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3883–3891, 2017.

[126] Seungjun Nah, Sanghyun Son, Suyoung Lee, Radu Timofte, and Kyoung Mu Lee. Ntire 2021 challenge on image deblurring. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 149–165, 2021.

[127] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, number 5, page 7. Granada, Spain, 2011.

[128] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[129] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[130] Zizheng Pan, Bohan Zhuang, Haoyu He, Jing Liu, and Jianfei Cai. Less is more: Pay less attention in vision transformers. In *AAAI Conference on Artificial Intelligence (AAAI), 2022*.

[131] Krushi Patel, Andres M Bur, Fengjun Li, and Guanghui Wang. Aggregating global features into local vision transformer. *arXiv preprint arXiv:2201.12903*, 2022.

[132] Krushi Patel and Guanghui Wang. A discriminative channel diversification network for image classification. *Pattern Recognition Letters*, 153:176–182, 2022.

[133] Krushi Bharatbhai Patel, Fengjun Li, and Guanghui Wang. Fuzzynet: A fuzzy attention module for polyp segmentation. In *NeurIPS'22 Workshop on All Things Attention: Bridging Different Perspectives on Attention*, 2022.

[134] Zhimao Peng, Zechao Li, Junge Zhang, Yan Li, Guo-Jun Qi, and Jinhui Tang. Few-shot image recognition with knowledge transfer. In *ICCV*, pages 441–449, 2019.

[135] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *16th Conference of the European Chapter of the Associationfor Computational Linguistics, EACL 2021*, pages 487–503. Association for Computational Linguistics (ACL), 2021.

[136] Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *CVPR*, pages 5822–5830, 2018.

[137] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[138] Raiyan Rahman, Christopher Indris, Tianxiao Zhang, Kaidong Li, Brian McCornack, Daniel Flippo, Ajay Sharda, and Guanghui Wang. On the real-time semantic segmentation of aphid clusters in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6298–6305, 2023.

[139] Hiranmayi Ranganathan, Hemanth Venkateswara, Shayok Chakraborty, and Sethuraman Panchanathan. Deep active learning for image classification. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3934–3938. IEEE, 2017.

[140] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[141] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[142] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[143] Noveen Sachdeva and Julian McAuley. Data distillation: A survey. *arXiv preprint arXiv:2301.04272*, 2023.

[144] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

[145] Usman Sajid, Xiangyu Chen, Hasan Sajid, Taejoon Kim, and Guanghui Wang. Audio-visual transformer based crowd counting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2249–2259, 2021.

[146] Usman Sajid, Wenchi Ma, and Guanghui Wang. Multi-resolution fusion and multi-scale input priors based crowd counting. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5790–5797. IEEE, 2021.

[147] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *Advances in neural information processing systems*, 29, 2016.

[148] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[149] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016.

[150] Lauren A Schmidt. *Meaning and compositionality as statistical induction of categories and constraints*. PhD thesis, Massachusetts Institute of Technology, 2009.

[151] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[152] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[153] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pages 369–386. SPIE, 2019.

[154] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[155] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[156] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, pages 1199–1208, 2018.

[157] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[158] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[159] Hugo Touvron, Louis Martin, and Kevin Stone et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

[160] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Alex Bronstein, Ivan Oseledets, and Emmanuel Mueller. The shape of data: Intrinsic distance for data distributions. In *International Conference on Learning Representations*, 2019.

[161] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[162] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Mobileone: An improved one millisecond mobile backbone. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7907–7917, 2023.

[163] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12894–12904, 2021.

[164] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[165] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11943–11951, 2020.

[166] Wenhai Wang, Jifeng Dai, Zhe Chen, Zhenhang Huang, Zhiqi Li, Xizhou Zhu, Xiaowei Hu, Tong Lu, Lewei Lu, Hongsheng Li, et al. Internimage: Exploring large-scale vision foundation models with deformable convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14408–14419, 2023.

[167] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021.

[168] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *CVPR*, pages 7278–7286, 2018.

[169] Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. Packing convolutional neural networks in the frequency domain. *TPAMI*, 41(10):2495–2510, 2018.

[170] Yuzhi Wang, Haibin Huang, Qin Xu, Jiaming Liu, Yiqun Liu, and Jue Wang. Practical deep raw image denoising on mobile devices. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI*, pages 1–16. Springer, 2020.

[171] Zhendong Wang, Xiaodong Cun, Jianmin Bao, Wengang Zhou, Jianzhuang Liu, and Houqiang Li. Uformer: A general u-shaped transformer for image restoration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17683–17693, 2022.

[172] Jun Wei, Qin Wang, Zhen Li, Sheng Wang, S Kevin Zhou, and Shuguang Cui. Shallow feature matters for weakly supervised object localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5993–6001, 2021.

[173] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[174] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10734–10742, 2019.

[175] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021.

[176] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers. 2022.

[177] Tete Xiao, Piotr Dollar, Mannat Singh, Eric Mintun, Trevor Darrell, and Ross Girshick. Early convolutions help transformers see better. *Advances in Neural Information Processing Systems*, 34, 2021.

[178] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the Frequency Domain. In *CVPR*, 2020.

[179] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1740–1749, 2020.

[180] Lu Xu, Jiawei Zhang, Xuanye Cheng, Feng Zhang, Xing Wei, and Jimmy Ren. Efficient

deep image denoising via class specific convolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3039–3046, 2021.

[181] Weijian Xu, Yifan Xu, Tyler Chang, and Zhuowen Tu. Co-scale conv-attentional image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9981–9990, 2021.

[182] Wenju Xu, Yuanwei Wu, Wenchi Ma, and Guanghui Wang. Adaptively denoising proposal collection for weakly supervised object localization. *Neural Processing Letters*, 51:993–1006, 2020.

[183] Yuhao Xu and Hideki Nakayama. Shifted spatial-spectral convolution for deep neural networks. In *Proceedings of the ACM Multimedia Asia*, pages 1–6. 2019.

[184] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. *arXiv preprint arXiv:2107.00641*, 2021.

[185] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7308–7316, 2019.

[186] Ling Yang, Liangliang Li, Zilun Zhang, Xinyu Zhou, Erjin Zhou, and Yu Liu. Dpgn: Distribution propagation graph network for few-shot learning. In *CVPR*, pages 13390–13399, 2020.

[187] Wenhan Yang, Robby T Tan, Jiashi Feng, Jiaying Liu, Zongming Guo, and Shuicheng Yan. Deep joint rain detection and removal from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1357–1366, 2017.

[188] Yiju Yang, Tianxiao Zhang, Guanyu Li, Taejoon Kim, and Guanghui Wang. An unsuper-

vised domain adaptation model based on dual-module adversarial training. *Neurocomputing*, 475:102–111, 2022.

[189] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 278–291, 2018.

[190] Shin-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard B W Yang, Giyeong Oh, and Yanmin Gong. Navigating text-to-image customization: From lyCORIS fine-tuning to model evaluation. In *The Twelfth International Conference on Learning Representations*, 2024.

[191] Seungjoo Yoo, Hyojin Bahng, Sunghyo Chung, Junsoo Lee, Jaehyuk Chang, and Jaegul Choo. Coloring with limited data: Few-shot colorization via memory augmented networks. In *CVPR*, pages 11283–11292, 2019.

[192] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 579–588, 2021.

[193] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 558–567, 2021.

[194] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[195] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[196] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5728–5739, 2022.

[197] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Multi-stage progressive image restoration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14821–14831, 2021.

[198] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Learning enriched features for fast image restoration and enhancement. *IEEE transactions on pattern analysis and machine intelligence*, 45(2):1934–1948, 2022.

[199] He Zhang and Vishal M Patel. Density-aware single image de-raining using a multi-stream dense network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 695–704, 2018.

[200] He Zhang, Vishwanath Sindagi, and Vishal M Patel. Image de-raining using a conditional generative adversarial network. *IEEE transactions on circuits and systems for video technology*, 30(11):3943–3956, 2019.

[201] Hongguang Zhang, Jing Zhang, and Piotr Koniusz. Few-shot learning via saliency-guided hallucination of samples. In *CVPR*, pages 2770–2779, 2019.

[202] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[203] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[204] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing*, 26(7):3142–3155, 2017.

[205] Tianxiao Zhang, Bo Luo, Ajay Sharda, and Guanghui Wang. Dynamic label assignment for object detection by combining predicted ious and anchor ious. *Journal of Imaging*, 8(7):193, 2022.

[206] Wei Zhang, Wanshu Fan, Xin Yang, Qiang Zhang, and Dongsheng Zhou. Lightweight single-image super-resolution via multi-scale feature fusion cnn and multiple attention block. *The Visual Computer*, pages 1–13, 2023.

[207] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, pages 6848–6856, 2018.

[208] Zizhao Zhang, Han Zhang, Long Zhao, Ting Chen, Sercan Arik, and Tomas Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In *AAAI Conference on Artificial Intelligence (AAAI), 2022*.

[209] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020.

[210] Yifeng Zhou, Xing Xu, Shuaicheng Liu, Guoqing Wang, Huimin Lu, and Heng Tao Shen. Thunder: Thumbnail based fast lightweight image denoising network. *arXiv preprint arXiv:2205.11823*, 2022.

[211] Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez de Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. *arXiv preprint arXiv:2402.16842*, 2024.