Exploring Functa, Pruning Vision Transformers, and Optimization Strategies for Deep Learning

Mao Keyu
School of Data Science
Fudan University
20307130241

Shen Jianzhi
School of Data Science
Fudan University
20307130030

Abstract

This research project focuses on the implementation and evaluation of various techniques in the field of computer vision. Initially, we introduce Functa, a meta-learning framework designed to efficiently fit the Implicit Neural Representation (INR) of each image within a dataset. The exploration begins with the exploration of SIREN, followed by the development of Functa on MNIST dataset. Additionally, both Functa and Spatial Functa are implemented on the CIFAR10 dataset, and their performance is assessed in downstream classification tasks.

Subsequently, our attention shifts towards pruning vision transformers through the utilization of sparsity. Our baseline model, ViT-16-base, serves as the foundation for this investigation. Three distinct methods of ViT pruning are explored, namely L1 norm pruning, Structural pruning based on dependency graph, and Non-Structural pruning based on token sparsification. The pruned models are then fine-tuned using the CIFAR10 dataset, and the effectiveness of each pruning method, in comparison to the baseline, is evaluated using the test set. Notably, Structural pruning based on dependency graph demonstrates the most favorable results.

Finally, we delve into the impact of batch normalization by examining the gradient predictiveness and β-smoothness. Through the utilization of DessiLBI, we uncover structural sparsity in both the MNIST and CIFAR10 datasets, as well as observe over-parameterization tendencies within modern CNN frameworks.

1 Functa

1.1 Introduction

Within the context of discrete representation of images, computer vision has demonstrated significant achievements in tasks such as classification, detection, segmentation, and generation, employing sophisticated convolutional neural network (CNN) models. However, there has been a recent surge in interest in treating images as continuous spatial signals. This approach mirrors the functioning of the human brain, which creates a continuous perception through association. By considering visual signals as functions of spatial points, and discrete signals as sampled data, a continuous representation of an image can be established. Once the function is learned, the parameters form an implicit neural representation (INR) of the image. Continuous representation offers various advantages over discrete representation, including freedom from resolution restrictions and lower dimensionality.

To further enhance our understanding, we seek a model capable of learning the INR of an image using meta-learning techniques. By establishing a unified INR framework, we can treat the INRs specific to each sample as data points, referred to as "functa," and utilize them in multiple downstream tasks.

2023 Neural Network and Deep Learning Final Project, School of Data Science, Fudan University.

In this section, we will begin by reviewing the evolution of essential concepts and methodologies. Subsequently, we will elucidate the methods we have replicated. Following that, we will delve into the implementation details and present the results of our experiments.

1.2 Related Work

1.2.1 Implicit Neural Representation (INR)

The INR framework utilizes multilayer perceptrons (MLPs) to map spatial points to RGB pixel values. However, traditional MLP approaches tend to generate smoother mappings, as they approximate neural tangent kernel (NTK) regression, causing a rapid reduction in eigenvalues as the frequency increases. Consequently, conventional MLPs struggle to effectively reconstruct edge information. Several approaches have been proposed to address this issue.

To improve the NTK approximation, positional encoding based on Fourier features [20, 15] was introduced. It allows for tuning the bandwidth of the NTK, bringing shift-invariance to the encoding. The authors further reduced the computational cost of constructing positional encoding over large data samples by implementing sampling techniques.

SIREN [19] takes a different approach by utilizing sine activations. Sine-based MLPs can fit data with any order of differentiability. However, since Sine-based MLPs have numerous local optima, the authors proposed an initialization method to stabilize the distribution across the layers.

1.2.2 Modulation

Modulation refers to the learnable feature transformation within each layer of a model. Once a general model is established, modulation can be applied to adapt the model output for specific tasks through few-shot learning.

FiLM [16] was initially proposed to address the task of visual reasoning. By applying adaptive affine transformations to the features in each layer, the model can effectively learn task-specific information.

Pi-GAN [2] combines SIREN and a modified version of FiLM, known as concatenation-based conditioning, in the context of a generative adversarial network (GAN). This integration achieves high performance in 3D generative tasks.

Dual-MLP [14] improves the capability of SIREN on high-resolution signals by modulating it with a rectified linear unit (ReLU) modulator.

1.2.3 Meta-Learning

Meta-learning focuses on learning a mapping that outputs functions for specific tasks, rather than training for specific tasks by finding an input-to-output map. In meta-learning, the unit of the learning target is a task, and within each task, a specific model is fitted to the data points.

The MAML framework [5] formulates meta-learning as a double loop. In the outer loop, a batch of tasks is sampled, and within each task, the inner loop goes through one or more gradient steps based on the task-specific loss. This temporary loss does not affect the model parameters. The true update of the model parameters in the outer loop is based on the task-specific loss with the new parameters.

Meta-SGD [11] takes a step further by learning how to learn quickly. It achieves this by setting a task-specific learnable tensor as the learning rate on each dimension. This tensor is fixed during the inner loop and then updated in the outer loop.

CAVIA [22] aims to prevent meta-overfitting by partitioning the parameters into context parameters and shared parameters. In the inner loops, only the context parameters are updated, while in the outer loops, only the shared parameters are updated.

1.3 Methods

1.3.1 Functa

The task is to train a model which can generate an INR to any image in the dataset in only a few gradient steps. Functa[3] combined all the three techniques mentioned above. It used SIREN as

```
for params in outer_params:  # Inner loop
    params.requires_grad = False
latent_params = torch.zeros((batch_size, *model.latent_dim)).to(device)
latent_params.requires_grad = True
meta_loss = []
for inner_step in range(inner_num):
    modulations = model.l2m(latent_params)
    logits = model(coords, modulations)
    loss = criterion(logits, pixels) / image_dim
    latent_params = meta_sgd.update_weights(loss, latent_params)
    meta_loss.append(loss)
meta_loss = torch.stack(meta_loss).mean()
for params in outer_params:   # Outer update
    params.requires_grad = True
modulations = model.l2m(latent_params)
logits = model(coords, modulations)
loss = criterion(logits, pixels) / image_dim + meta_loss
outer_optim.zero_grad()
loss.backward()
outer_optim.step()
```

Figure 1: Meta SGD

the INR framework, and proposed latent modulation, which uses a linear transformation to map a low-dimensional vector, as known as the latent modulation, to the modulation within each SIREN layer. The authors further formulated the task as meta-learning, where the context parameters are the latent modulation, and the shared parameters are the SIREN parameters and the linear transformation from latent modultations to modulations. After the training is done, we can use the model to quickly build an INR for a similar image, and use the latent modulation as a continuous representation of the image for downstream tasks. We tried classification on the dataset, and explored the performance of INR on super-resolution and denoise.

1.3.2 Spatial Functa

However, without the spatial information, the linear transformation from latent modulation to modulation is hard to train, and the latent representation as uni-dimensional vector is hard to utilize as feature. Spatial Functa[1] fixed this by using a 3-dimensional tensor as latent modulation, and the transformation from latent modulation to modulation is a convolutional layer followed by an interpolation, so that every pixel has customized modulation.

1.4 Implementation Detail

The basic SIREN is an MLP with sine activation, with the last layer as a Vanilla linear transformation. The initialization of the parameters conforms to paper [19], as $U(-\frac{1}{W}, \frac{1}{W})$ for the first layer, and $U(-\sqrt{\frac{6}{W}}/\omega_0, \sqrt{\frac{6}{W}}/\omega_0)$ for the others, where W is the dimension of the input feature, and ω_0 is by default 30. We fit the image for 1000 steps.

We implemented Functa on half sample in MNIST, and tested its generalization capability by letting it fit the out-of-domain sample. We mainly used Spatial Functa on CIFAR10.

Meta-SGD is initialized to uniform 1E-2, and shares the optimizer with outer parameters. The loss of Meta-SGD is the mean loss across the inner loop, namely meta-loss, as shown in Figure 1, where meta_sgd only updates latent_params, and the adaptive learning rates in meta_sgd are put in the outer_optim.

The binary representation trick used in Spatial Functa for interpolation is not implemented due to lack of detail. And we tried Nearest as the interpolation mode.

Number of inner loop is set to 3, and the total outer loop steps is approximately 2E5 for MNIST. CIFAR10 takes more steps for its complexity. Empirically, 5E5 updates suffices. However, due to time limitation, we only trained CIFAR10 for 3E5 updates.

Configuration	Batch size	SIREN width	SIREN depth	Latent dim	Inner update	Outer loop learning rate	ω_0
1	128	256	6	256	SGD 0.01	3E-5	30
2	32	512	16	512	MetaSGD	3E-6	30
3	64	256	6	$8 \times 8 \times 16$	MetaSGD	3E-5	10

Table 1: Different hyperparameter configuration

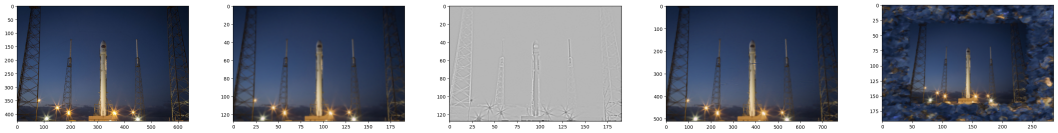
The MNIST and CIFAR10 Functa classification uses an MLP with one hidden layer, 1024 hidden units, Dropout 0.2, SiLU activation, batch size 256, 100 epochs, AdamW optimizer with learning rate 1E-3, weight decay 0.1, Cross Entropy Loss with label smoothing 0.1.

The CIFAR10 Spatial Functa classification uses two convolutional layers with batch normalization, and then max pools the features into a linear classifier. Training hyper-parameters are the same with the MNIST classfier.

One of our limitations is that, due to restriction in computational power, we do not have time to do enough ablation experiments.

1.5 Experiments

1.5.1 Basic Functa



(a) Original Image (b) Reconstruction (c) Laplacian (d) Scale up (e) Extend

Figure 2: Basic Reconstruction

Figure 2 shows the SIREN reconstruction of a rocket photo. Before fitting, it's height is resized into 128. Subfigure (c) shows that SIREN successfully recover the order-2 differential information. And subfigure (d) shows the reconstruction of SIREN, which was trained on 128×186 image, on a 512×745 grid. The effect is similar to bilinear interpolation. By feeding coordinate values out of [-1, 1], SIREN can produce extended images as (e) shows. Unfortunately, the extension is merely periodic and lack of semantic meanings.

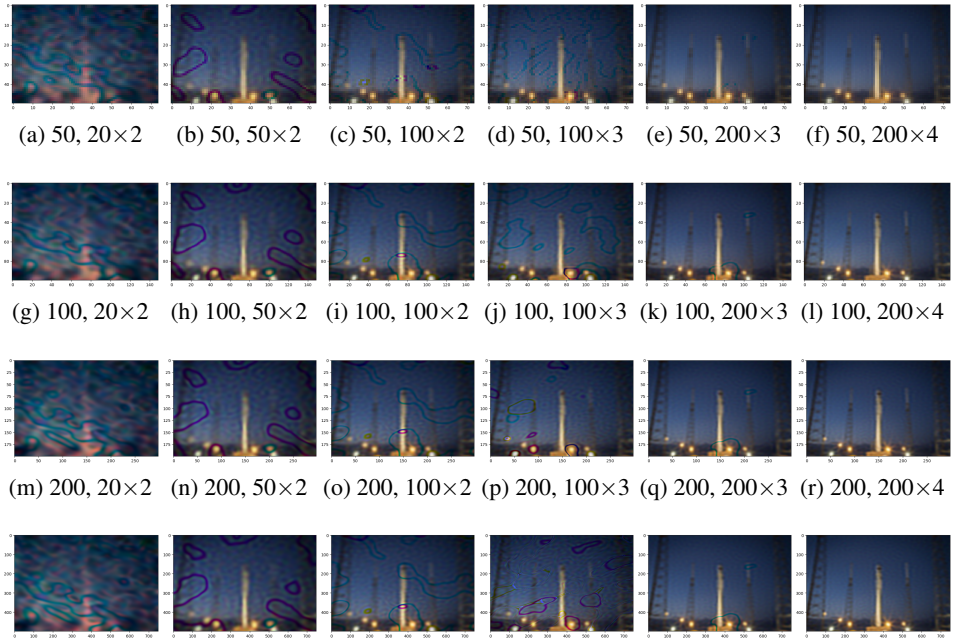
Figure 3 shows reconstruction results of different sized image (length 50, 100, 200, 500) fitted by SIRENs of different complexities (20×2, 50×2, 100×2, 100×3, 200×3, 200×4 hidden units). Generally larger image sizes call for more complex networks to fit. Under-fitting results in periodic noise and water mark-like noise, which is caused by the periodic form of the activation.

1.5.2 Functa

Configuration	Dataset	PSNR
1	MNIST	21.37
2	MNIST	19.36
1	CIFAR10	20.25
3	CIFAR10	22.42

Table 2: Caption

As can be seen in Figure 22 in Appendix, the whole meta-training process mainly undergoes these phases: First it forms an embryo, where the fitted value is a mixture of all the figures. And quickly it develops and learns to tell some basic structural differences. However in that phase it may produce some error, as shown in subfigure (d). Then it becomes more precise and can tell some nuance between images, but the confidence is low, and the overall vision effect is blurred. And last it can fit relatively well to all the images in the dataset. With meta-SGD, the length of Embryo phase is extended, because the model cannot stably fit the image in 3 steps.



(a) 50, 20×2 (b) 50, 50×2 (c) 50, 100×2 (d) 50, 100×3 (e) 50, 200×3 (f) 50, 200×4

(g) 100, 20×2 (h) 100, 50×2 (i) 100, 100×2 (j) 100, 100×3 (k) 100, 200×3 (l) 100, 200×4

(m) 200, 20×2 (n) 200, 50×2 (o) 200, 100×2 (p) 200, 100×3 (q) 200, 200×3 (r) 200, 200×4

(s) 500, 20×2 (t) 500, 50×2 (u) 500, 100×2 (v) 500, 100×3 (w) 500, 200×3 (x) 500, 200×4

Figure 3: Comparison among different sizes and network complexities

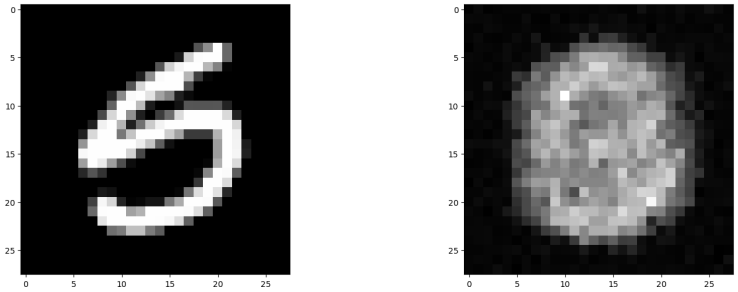


Figure 4: Failure when the training gets unstable

Figure 23 in Appendix shows some out-of-domain fitting results. The average out-of-domain PSNR achieved by configuration 1 is 18.97. Together with more in-domain fitting examples in Figure 24 we can see the modulated SIREN is relatively week in fitting fine curves.

We found that the outer learning rate should be lowered when SIREN gets deeper and the batch size gets unavoidably smaller. Otherwise the convergence will be unstable and the model outputs rough figures like what is shown in Figure 4. The results come from altering the outer loop learning rate in configuration 2 as 3E-5. Due to the smaller learning rate it takes the model more steps to converge to a satisfying position.

During CIFAR10 training process, as shown in Figure 25, our modulated SIREN first generates a chaotic average, and then learns to coarsely distinguish regions with high contrast. After that it gradually learns to produce an impressionism effect, as the contour emerges. Later it becomes more precise during the final 1E5 steps.

Spatial Functa converges sooner than Functa. However, after the PSNR reaches 20, it is harder to proceed further.

Config	Dataset	ACC
1	MNIST	96.94
2	MNIST	96.59
1	CIFAR10	51.56
3	CIFAR10	56.24

Table 3: Classification Result with Different Configurations

1.5.3 Downstream Tasks

Using the latent modulation as feature, we achieved relatively high accuracy rate on MNIST. The reason may be that MNIST is structurally easy, and, as we can see in the training process, the mis-impression phenomenon indicates that the latent modulated SIREN is trying to pre-classify the image, so the latent modulation learned in a few steps has already been capable to capture enough information for classification.

However, the performance is not as well as training a classification directly on the image. That is because some fine curve detail is still not captured by the latent modulation. If the Functa is trained more sufficiently, the result may get better.

The performance on CIFAR10 is relatively unsatisfactory. The convergence of both Functa and Spatial Functa is not sufficient, and loses many important detail of the image. The variations are much more complex, and both Functa may fail to learn some uncommon visual features in 3E5 steps. But we can still see the potential of Spatial Functa here, since by Vanilla convolutional layer it has achieved noticeable improvement from Functa.

2 Network Sparsity: Pruning Transformer via Sparsity

2.1 Introduction

Vision Transformers (ViTs) have emerged as a groundbreaking architecture for computer vision tasks, particularly in image classification and object detection. Unlike traditional convolutional neural networks (CNNs), ViTs rely on self-attention mechanisms to capture global dependencies and capture long-range interactions in images. This approach has proven to be highly effective and has achieved state-of-the-art results on various benchmark datasets.

However, the increased performance of ViTs comes at the cost of significantly higher computational requirements compared to CNNs. ViTs consist of multiple self-attention layers, and each layer involves computations that scale quadratically with the input size. Additionally, the large number of parameters in ViTs contributes to increased memory consumption during training and inference. These factors make ViTs computationally expensive and memory-intensive, hindering their deployment in resource-constrained environments.

Pruning ViTs via sparsity offers a solution to mitigate the computational and memory burdens associated with these models. By removing redundant or less important parameters or removing less important paths in the network, pruning helps reduce the overall model size and computational complexity. Sparsity achieved in pruning process will effectively create a sparse representation that requires fewer computations and consumes less memory.

In this section, we will use different methods to prune the vision transformer via sparsity, and compare the results with different strategies. Before we start, let's have an brief introduction of transformer pruning.

2.2 Related Work

Pruning vision Transformers via sparsity is a task that aims to reduce the computational complexity and memory footprint of vision Transformers, which are widely used deep learning architectures for computer vision tasks. The objective is to identify and eliminate redundant or unnecessary parameters in the model, leading to a more efficient and compact representation without significant performance degradation. Existing pruning approaches can be categorized into two main schemes: structural pruning and non-structural pruning.

2.2.1 Structural and Non-structural Pruning

In the domain of structural pruning, researchers have explored various methods. For instance, Fu et al. introduced a novel approach based on differential inclusions of inverse scale spaces, which generates a family of models with different complexities through the coupling of parameters[6]. Their work focused on finding sparse network structures directly by utilizing the coupled structure parameter. Similarly, Liu et al. concentrated on complex structures such as residual connections, group/depthwise convolution, and feature pyramid networks[12]. They presented a general channel pruning approach applicable to various complex structures, emphasizing the importance of simultaneously pruning coupled channels to achieve higher speedup. They employed mask sharing in coupled channels to achieve this goal. Structural pruning involves removing entire structural components or groups of parameters based on predefined criteria, such as their importance or contribution to the overall model performance. This reduction in structural complexity leads to computational savings during training and inference.

In contrast, non-structural pruning focuses on eliminating individual parameters within the vision Transformer without removing entire components or groups. Lee et al. challenged the conventional idea of pruning after training and proposed that pruning can be performed at initialization, based on a saliency criterion known as connection sensitivity[8]. They formally characterized initialization conditions to ensure reliable connection sensitivity measurements and analyzed the signal propagation properties of the pruned networks. They also introduced a data-free method to enhance the trainability of the pruned models. Sanh et al. proposed movement pruning, a simple and deterministic first-order weight pruning method that is more adaptive to fine-tuning pretrained models[18]. Their work focused on movement pruning methods where importance is derived from first-order information. Some convential idea like l1 regularization can be also treated as this kind of pruning method[21, 7]. Non-structural pruning methods result in a sparse representation where a significant portion of the parameters becomes zero, effectively reducing the overall model size and computational requirements.

2.2.2 Recent Advances in Pruning Techniques

In recent years, significant progress has been made in the field of pruning techniques, resulting in improved efficiency and accuracy of pruned models. These advancements have been presented in renowned international conferences, contributing to the advancement of the field.

Fang et al. recognized that existing pruning methods lack generalizability across different models due to the significant variations in parameter-grouping patterns. To address this limitation, they embarked on a challenging and relatively unexplored task of general structural pruning for arbitrary architectures, such as transformers. Their approach, Dependency Graph (DepGraph), introduced a comprehensive and fully automatic method to explicitly model the dependencies between layers and facilitate comprehensive grouping of coupled parameters for pruning. By ensuring that all removed parameters are consistently identified as unimportant, their models successfully mitigate structural issues and minimize performance degradation post pruning[4].

Token pruning has emerged as a popular technique, particularly in the context of Vision Transformers (ViTs). Li et al. proposed a dense/sparse training framework to achieve a unified model that allows weight sharing across various token densities. They specifically introduced adaptive token pruning, enabling optimization of patch token sparsity based on the input image. Additionally, they explored the use of knowledge distillation to enhance the token selection capability in early transformer modules, thereby improving the overall effectiveness of token pruning in ViTs[10].

2.3 L1 Regularization and L1 Pruning

2.3.1 L1 Regularization for Sparsity Induction

L1 regularization is a widely employed technique in machine learning to promote sparsity within models. It involves adding a penalty term to the loss function, which is determined by the L1 norm of the weights:

$$\text{reg} = \lambda |\mathbf{W}|_1 \tag{1}$$

Here, $\mathbf{W}$ represents the weights of the model, and λ serves as the regularization parameter that governs the strength of the regularization.

When applying L1 regularization, the objective is to minimize the absolute value of the weights, as indicated by the penalty term. As a consequence, the optimization process tends to drive certain weight parameters to zero, resulting in a sparse model representation. By introducing a bias towards solutions where a subset of the weights becomes precisely zero, L1 regularization effectively induces sparsity.

This sparsity manifests as a reduction in the number of non-zero parameters within the model, leading to enhanced compactness and computational efficiency. The regularization parameter λ plays a crucial role in controlling the level of sparsity: higher values of λ encourage more parameters to be pushed towards zero.

In our task, we fine-tune the pretrained ViT-Base-16 model, which was initially trained on ImageNet and subsequently fine-tuned on CIFAR10. During this process, we incorporate an L1 regularization term into the loss function. However, it is important to note that the degree of sparsity cannot be directly controlled. Instead, the value of λ serves to weigh the importance of sparsity, with higher values promoting greater parameter pruning.

2.3.2 Pruning Based on L1 Norm

In the context of network pruning, the application of L1 norm as a pruning criterion has been widely explored. L1 norm enables the identification and removal of non-essential parameters in the network, thereby reducing computational complexity. The absolute values of weight parameters are intuitively considered as indicators of importance. By setting a sparsity ratio, we can sort all parameter weights and remove a portion of them based on the specified sparsity ratio.

However, L1 pruning has several limitations that need to be considered:

1. Fragmented Pruning: L1 pruning operates by removing individual parameters without considering structured patterns. As a result, related weights may be pruned independently, potentially leading to fragmented pruning. This fragmented pruning may have adverse effects on the network's overall performance and functionality.
2. Importance Assessment: Relying solely on the absolute value of weight parameters may not accurately represent their importance. Additionally, the global pruning carried out through the L1 norm treats all weight matrices equally, potentially introducing pruning bias.
3. Dependency Relation Disruption: L1 pruning simply sets a proportion of parameters to zero, which can disrupt the dependency relations among layers. This disruption may have a significant impact on the network's behavior and performance.

Despite these limitations, L1 pruning remains valuable in introducing sparsity and improving computing efficiency. In the current task, we vary the sparsity ratio from 0.1 to 0.9 for the model (pretrained on ImageNet and finetuned on CIFAR10) and subsequently fine-tune the model on CIFAR10 for two epochs.

2.4 Structural Pruning: Dependency Graph

2.4.1 Dependency and Dependency Matrix

In this section, we delve into the concept of structural pruning with a focus on the dependency graph. Neural networks exhibit dependencies among their parameters, necessitating careful consideration of these relations when applying structural changes to the network.

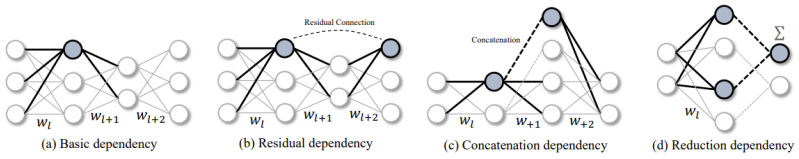


Figure 5: Dependency Relation Examples

Consider the example depicted in Figure 5, where we aim to achieve sparsity by pruning three weight layers w_l, w_{l+1}, and w_{l+2} in a simple network. By examining the network, we can readily identify certain dependencies. For instance, $w_l \overset{d}{\leftrightarrow} w_{l+1}$ represents a dependency relation, denoted as $\overset{d}{\leftrightarrow}$. Consequently, if we intend to remove the k-th neuron that bridges the two layers, we must simultaneously eliminate the k-th row of w_l and the k-th column of w_{l+1}.

To facilitate the pruning of layers with dependency relations, we introduce the notation of a dependency matrix. Let $D \in 0, 1^{N \times N}$ represent the dependency matrix, where N denotes the number of layers targeted for pruning. The elements of the dependency matrix can be defined as follows:

$$D_{ij} = \begin{cases} 0, & \text{if } w_i \overset{d}{\not\leftrightarrow} w_j \\ 1, & \text{if } w_i \overset{d}{\leftrightarrow} w_j \end{cases} \tag{2}$$

Moreover, we define the set of all elements that have a dependency relation with the i-th layer as d_i:

$$d_i = \{k \mid \forall k : D_{ik} = 1\} \tag{3}$$

However, in a neural network, the dependency between two layers is not solely determined by the two layers themselves. Intermediate layers can also influence the dependency relations. Such implicit relations make it challenging to construct the dependency matrix accurately and may result in the failure to detect potential dependencies. To overcome this obstacle, we introduce the Dependency Graph, an equivalent yet more easily estimated method for modeling dependencies.

2.4.2 Dependency Graph

We observe that when detecting dependency relations among layers, there are redundant cases to consider. For example, the following two sets of expressions lead to the same conclusion: $layer_1 \overset{d}{\leftrightarrow} layer_2 \overset{d}{\leftrightarrow} layer_3$.

$$\begin{cases} layer_1 \overset{d}{\leftrightarrow} layer_2 & layer_2 \overset{d}{\leftrightarrow} layer_3 & layer_1 \overset{d}{\leftrightarrow} layer_3 \\ layer_1 \overset{d}{\leftrightarrow} layer_2 & layer_2 \overset{d}{\leftrightarrow} layer_3 \end{cases} \tag{4}$$

To address this redundancy, we can select $layer_1$ as the starting layer and detect all dependency relations related to $layer_1$. For those layers that have a dependency relation with $layer_1$ (in this case, $\{layer_2\}$), we consider them as new starting points. By employing this approach, we can identify all the dependency relations. When two layers are found to have a dependency relation using this method, we say that a path exists between them, denoted by $\overset{p}{\leftrightarrow}$. The Dependency Graph can be represented as:

$$\tilde{D}_{ij} = \begin{cases} 0, & \text{if } w_i \overset{p}{\not\leftrightarrow} w_j \\ 1, & \text{if } w_i \overset{p}{\leftrightarrow} w_j \end{cases} \tag{5}$$

Mathematically, $\tilde{D}$ can be considered as a transitive reduction of the matrix D. $\tilde{D}$ is sparser than D and serves the same purpose of detecting dependency relations among layers.

However, building a Dependency Graph encounters several challenges. For instance, when pruning the k-th neuron of layer i, it is unclear whether we should prune the k-th column or the k-th row. Additionally, non-parameterized operations such as residual addition also exist in the network, and we should not simply ignore them. To address these challenges, we consider decomposing the neural network $F(x, w)$ into more comprehensive components.

$$F(x, w) = \{f_1, f_2, \cdots, f_N\} \tag{6}$$

Here, f represents either a parametrized layer, such as convolution, or a non-parameterized operation, such as residual addition (Figure 6 provides an example).

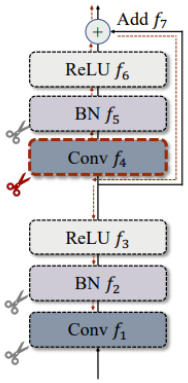


Figure 6: Layer Notation

Moreover, we decompose each f into two parts, f^{in} and f^{out}, to denote the input and output of the layer, respectively. In other words:

$$F(x, w) = \left\{ f_1^{in}, f_1^{out}, f_2^{in}, f_2^{out}, \cdots, f_N^{in}, f_N^{out} \right\} \tag{7}$$

2.4.3 Dependency Modeling

By utilizing the decomposition of the neural network, we can identify two types of dependencies in the task:

$$\begin{cases} \text{Type 1:} & f_i \overset{p}{\leftrightarrow} f_j \quad (i \neq j) \\ \text{Type 2:} & f_i^{in} \overset{d}{\leftrightarrow} f_i^{out} \end{cases} \tag{8}$$

In these types of dependencies, the following conditions should be met:

1. Type 1 dependencies result in connected layers between two weight parameters.
2. Type 2 dependencies exist only when there is the same pruning scheme or plan for the input and output layers: $pl(f_i^{in}) = pl(f_i^{out})$, denoting that the input layer and output layer should be prune simultaneously.

When two layers are connected within a neural network, it signifies that they correspond to the same intermediate features in the network, thereby establishing a dependency relation between them. However, in the case of the input and output of a layer, a dependency exists only if they share the same pruning scheme or plan. It is important to note that layers such as convolutional layers do not adhere to this criterion. To comprehend the criterion and the notation pl in a more precise manner, the following conditions can be considered:

1. If one of the input or output layers is pruned, is it necessary for the other layer to be pruned as well?
2. If the answer to the previous condition is affirmative, do the pruning data of the two layers match exactly?

The equality of the pl (pruning scheme) for the input and output layers is established only when both of the aforementioned conditions are satisfied.

In conclusion, we can define the dependencies as follows:

$$\tilde{D}_{ij} = \max(I(f_i^{in} \overset{p}{\leftrightarrow} f_j^{out}), I(pl(f_i^{in}) = pl(f_j^{out}) | i = j)) \tag{9}$$

This equation take dependency relation within a same layer and between different layers into account.

Algorithm 1: Dependency Graph

Data: Input $F(x, w)$
Result: Output $\tilde{D}$
Initialize $\tilde{D} = 0_{N \times N}$;
Decompose F into two parts;
$f^{in} = \{f_1^{in}, f_2^{in}, \ldots, f_N^{in}\}$;
$f^{out} = \{f_1^{out}, f_2^{out}, \ldots, f_N^{out}\}$;
for i *in 1:N* **do**
  for j *in 1:N* **do**
    $\tilde{D}(f_i^{in}, f_j^{out}) = \max(I(f_i^{in} \overset{p}{\leftrightarrow} f_j^{out}), I(pl(f_i^{in}) = pl(f_j^{out})|i = j))$;
    $\tilde{D}(f_i^{out}, f_j^{in}) = \max(I(f_i^{in} \overset{p}{\leftrightarrow} f_j^{out}), I(pl(f_i^{in}) = pl(f_j^{out})|i = j))$;
  end
end
return $\tilde{D}$;

The algorithm 1 describes the process for us to get dependency graph.

2.4.4 Pruning

Once we obtain our dependency graph, we can proceed to the next phase: pruning. Assessing the importance of grouped parameters presents a significant challenge in pruning as it involves several interconnected layers. In this task, we employ a simple norm-based criterion [9] to establish a practical method.

For a group of parameters $G = (w_1, \cdots, w_n)$, basic methods such as the L2 norm can produce independent scores for each parameter w, denoted as $Imp(w) = |w|2$. An intuitive approach to estimating the importance score is by computing an aggregated score, such as $Imp(G) = \sum w \in GImp(w)$. However, if we estimate importance scores for different layers independently, these scores will not be additive and render this addition meaningless. This non-additivity arises from the fact that the distribution of parameters across different layers differs. To align with this intuitive idea, we employ a sparse training method in which a simple regularization term is applied during the training process:

$$Reg(G, K) = \sum_{k=1}^{K} \gamma_k Imp_{G,k} \tag{10}$$

Here, K denotes the total prunable dimensions for each parameter w, and we define $Imp_{G,k} = \sum_{w \in G} |w[k]|_2^2$. Additionally, γ_k is a hyper-parameter defined with a controllable exponential strategy:

$$\gamma_k = 2^{\alpha(Imp_G^{\max} - Imp_{G,k})/(Imp_G^{\max} - Imp_G^{\min})} \tag{11}$$

After sparse training, a simple importance score can be applied:

$$\hat{Imp}_{G,k} = N \cdot Imp_{G,k} / \sum \{\text{TopN}(Imp_G)\} \tag{12}$$

This score can be used to determine the importance of parameters and directly remove unimportant parameters.

2.5 Non-Structural Pruning: Dynamic Token Sparsification

2.5.1 Idea of Token Pruning

Our investigation reveals that when utilizing ViT for prediction tasks, the most informative portions of the image are predominantly utilized. This observation implies that, for tasks like image classification, the key requirement is to focus on a few specific regions in the image where the objects to be classified are concentrated—referred to as patches according to ViT's terminology[17]. This realization inspires us to reconsider the conventional approach of pruning the network structure and instead concentrate

on pruning the tokens generated by ViT during image processing, retaining only the most informative parts. By adopting this approach, we can enhance the network's computational efficiency and increase its sparsity.

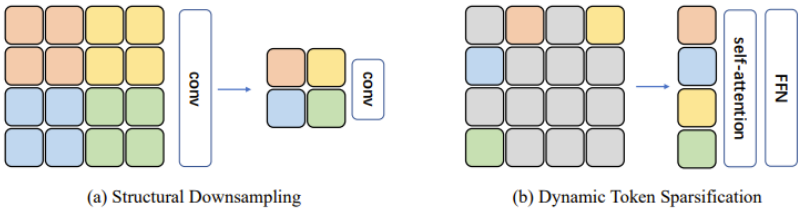


Figure 7: Downsampling Strategy

In contrast to CNN models that typically employ structural downsampling strategies to construct hierarchical architectures, as depicted in Figure 7(a), our approach employs an unstructured and data-dependent downsampling method, as shown in Figure 7(b). This alternative method enables us to better exploit sparsity in the input data.

It is important to note that pruning tokens is equivalent to removing parts of the image. By adjusting the sparsity level, which denotes the amount of information to be removed from the images, we can visualize the effects of token pruning, as depicted in Figure 8 and Figure 9, each representing a different sparsity level. In our experiments, we utilize a pre-trained importance score function and fine-tune the raw ViT base model on CIFAR10. The visualization results are generated using our final model with varying sparsity levels.

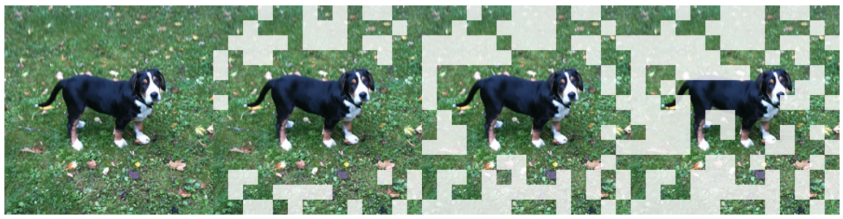


Figure 8: Sparsity:0,0.2, 0.36, 0.49

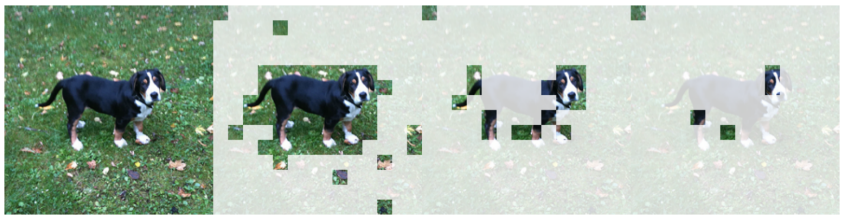


Figure 9: Sparsity:0,0.7, 0.91, 0.97

2.5.2 Core Method: Hierarchical Token Sparsification

The fundamental concept behind dynamic token sparsification is to perform hierarchical pruning, gradually dropping unimportant tokens during the computation process. To implement this idea, a decision mask $M \in \{0,1\}^N$ is employed to determine whether each token should be kept or dropped, where N is the number of patch embeddings. Initially, all elements of this mask are set to 1. The mask M and the tokens $x \in \mathbb{R}^{N \times C}$ serve as inputs to the ViT model. The process of token pruning can be divided into the following steps:

Firstly, an MLP is applied to the input tokens x to derive local features:

$$f^l = \text{MLP}(x) \in \mathbb{R}^{N \times F} \tag{13}$$

Here, F denotes the dimension of the features, which is typically smaller than C. A common choice for F is $C/2$. Similarly, global features are derived by considering all local features:

$$f^g = \mathcal{A}(\text{MLP}(x), M) \in \mathbb{R}^F \tag{14}$$

where $\mathcal{A}$ is an operator used to aggregate all local features. In this task, $\mathcal{A}$ is defined as:

$$\mathcal{A}(w, M) = \frac{\sum_{i=1}^{N} M_i w_i}{\sum_{i=1}^{N} M_i} \tag{15}$$

Local features represent the information for individual tokens, while global features encompass information from the entire image (all tokens). Hence, both types of features can provide valuable information for determining the importance score. Consequently, these two types of features are combined, and another MLP is used to process the combined feature and predict the probability of discarding each token:

$$f_i^c = (f_i^l, f_i^g), \quad 1 \leq i \leq N \tag{16}$$

$$p = \text{softmax}(\text{MLP}(f^c)) \in \mathbb{R}^{N \times 2} \tag{17}$$

where $p_{i,0}$ denotes the probability of discarding the i-th token, and $p_{i,1}$ denotes the probability of keeping the i-th token. Consequently, the current decision mask M_c is generated by sampling from p. The overall decision mask M is then updated using the current decision mask M_c as follows:

$$M \leftarrow M \odot M_c \tag{18}$$

Here, $\odot$ denotes the Hadamard product of matrices. This procedure explains the gradual update of the decision mask.

In this task, we use a score function(Namely, all the components that used to generate decision mask) pretrained on ImageNet, and then finetune our ViT base model with pretrained score function on CIFAR10 for several epochs.

2.6 Experiment and Results

2.6.1 L1 Regularization and L1 Pruning

In this section, we present the experimental setup and results for training models with L1 norm, including simple L1 regularization and L1 pruning. As discussed in previous sections, the magnitude of regularization is used to measure sparsity in this initial experiment. Specifically, by increasing the regularization coefficient, the model encourages more weight parameters to approach zero during the training process.

Table 4: Results for L1 Regularization

Reg	0(baseline)	5e-8	5e-7	5e-6	5e-5	5e-4	5e-3	5e-2
acc	0.9852	0.9781	0.9746	0.9758	0.9766	0.9705	0.9355	0.8678
acc@5	0.9995	0.9997	0.9996	0.9997	0.9993	0.9992	0.9984	0.9921

Table 4 shows the results of experiment. Our experimental results show that when the regularization coefficient is small, the accuracy of the model remains high and comparable to the baseline performance. This indicates that the regularization has a minimal impact on the model's parameter weights, allowing it to maintain its predictive capabilities.

However, as the regularization coefficient increases to a larger value, such as 5e-2, the model's accuracy decreases by approximately 10%. This reduction in accuracy is significant but still acceptable considering the emphasis on achieving sparsity through stronger regularization. In conclusion, L1 regularization can almost maintain the original function of model when applying some sparsity to the model. However, this task is just an exploration since we cannot set the sparsity by this method.

As for the results of L1 pruning, we have pruned our model based on L1 norm with sparsity from 0.1 to 0.9, where sparsity here denotes the ratio of non-zero parameters. After pruning, the model is finetuned on CIFAR10 for extra 2 epochs. The results are shown in following Table 5.

Table 5: Results for L1 Pruning

Sparsity	base	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
acc(%)	98.52	98.31	98.13	98.02	97.72	96.34	94.03	77.55	58.22	42.81
acc@5(%)	99.98	99.98	99.98	99.96	99.96	99.96	99.94	99.03	99.18	90.76

After applying L1 pruning to ViT16, the effect on the test set performance remains relatively stable when the sparsity is low, ranging from 0.1 to 0.6. The accuracy is maintained at around 95%, indicating that the network structure is preserved quite well. However, beyond a sparsity of 0.6, the accuracy starts to decline, and it drops below 0.5 when the sparsity reaches 0.9.

This observation suggests that L1 pruning, which involves eliminating less important weights based on their absolute magnitude, can effectively compress the network without significantly affecting the performance when the sparsity is relatively low. The network retains its ability to generalize and classify accurately.

2.6.2 Results of Structural Pruning with Dependency Graph

In this section, we present the results of structural pruning using a dependency graph. Our approach involves removing unimportant parameters from the network based on their dependencies with other parameters. The sparsity level in this task represents the ratio of removed unimportant parameters, rather than the ratio of non-zero parameters. This definition aligns with the concept of structural pruning. Additionally, we consider the dependency relations between parameters, resulting in a gradual variation in the number of parameters, which will be demonstrated in the results.

To evaluate the effectiveness of our approach, we conducted both local pruning and global pruning with the structural method. For each pruned model, we further fine-tuned it for an additional 20 epochs. The results of local pruning and global pruning are shown in Figure 10 and Figure 11, respectively.

Local pruning involves removing a fixed percentage of units or connections from each layer by comparing within the layer itself. On the other hand, global pruning considers all parameters collectively across layers and selects a global fraction of them for pruning.

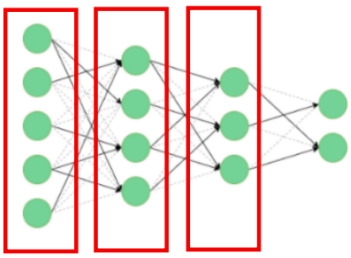


Figure 10: Local Pruning

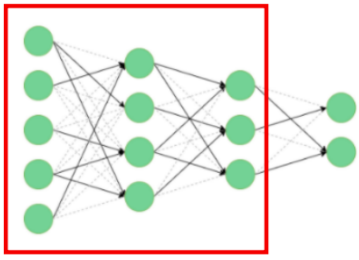


Figure 11: Global Pruning

The results of global pruning are shown in Table 6 and the results of local pruning are shown in Table 7. During our analysis, we observed that local and global structural pruning exhibit different variations in parameters under the same sparsity settings. This disparity arises due to the differences in their parameter consideration strategies. Moreover, considering the interdependence between parameters leads to a larger variation in pruned parameters than the specified pruning ratio. In our results, global pruning outperformed local pruning, consistently maintaining an accuracy above 90% for a sparsity of 0.5. With the exception of the overly sparse 0.9 sparsity, the accuracy remained above 80% in all cases. Overall, global pruning demonstrated better results compared to L1 pruning, indicating that it achieved favorable outcomes by balancing importance scores and dependency relationships during the pruning process. On the other side, local pruning indeed shows a relatively poorer performance compared to global pruning. As the sparsity increases beyond 0.5, the accuracy

drops below 80%. This decline in performance can be attributed to the local pruning strategy, which removes a fixed percentage of units or connections within each layer, without considering the overall impact on the network.

Table 6: Results for Global Structural Pruning

Sparsity	base	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
acc(%)	98.52	98.17	96.69	95.41	93.02	91.04	87.26	79.34	71.64	37.88
acc@5(%)	99.98	99.96	99.94	99.86	99.81	99.6	99.52	98.97	98.17	90.86
Params(M)	85.81	65.6	50.67	42.54	37.36	30.3	24.77	18.88	9.03	0.24

Table 7: Results for Local Structural Pruning

Sparsity	base	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
acc(%)	98.52	97.55	96.59	95.57	91.64	78.99	69.79	62.99	53.6	46.22
acc@5(%)	99.98	99.98	99.94	99.94	99.76	99.19	98.09	96.93	95.26	93.13
Params(M)	85.81	70.35	56.24	42.6	31.8	21.64	14.28	8.36	3.68	1.06

2.6.3 Results of Non-Structural Pruning with Token Sparsification

In this section, we present the results of token pruning using non-structural techniques. Our experiment involves the utilization of a score function pretrained on ImageNet, as described in previous sections, which is subsequently fine-tuned on the CIFAR10 dataset. The visualization effects of token pruning are displayed in Figure 8 and Figure 9. Additionally, our pretrained model, with an unfinetuned classifier, is accessible through the provided Google Drive link.

Following the finetuning of the score function, we freeze the corresponding parameters. These fixed score function parameters are then employed to perform pruning on the ViT model, considering various sparsity levels. Here, sparsity refers to the ratio of removed input tokens or the information pertaining to input figures. Subsequently, we further finetune the classifier of the pruned ViT16 base model for 20 epochs. The detailed results are presented in Table 8.

Table 8: Results for Token Pruning

Sparsity	base	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
acc(%)	98.52	86.57	85.92	85.12	83.51	79.42	71.3	52.13	10.21	10.14
acc@5(%)	99.98	99.5	99.43	99.43	99.2	98.65	96.64	89.86	50.88	50.94

Following token pruning on the images, the achieved results on the test set are not satisfactory. When the sparsity is below 0.5, a significant amount of information is retained in the images, which is sufficient to describe the main content and aid the model in completing the classification task. As a result, the accuracy on the test set remains consistently above 80% for this range of sparsity. However, as the sparsity increases further, particularly at 0.8 and 0.9, the model completely loses its ability to classify the images, resembling a random choice. This phenomenon is illustrated by the last two images in Figure 9, where the excessive sparsity causes crucial information to be lost, rendering classification impossible. This distinction from model pruning is significant since pruning the model preserves the network structure, enabling it to leverage that structure for image classification, regardless of the pruning extent.

3 Optimizations

3.1 Gradient Predictiveness

We employed three learning rate compositions: 1E-3, 5E-4, and 3E-4. Figure 12 illustrates the behavior of the gradient distance for the non-batchnorm VGG model. Initially, the gradient distance increases, followed by a fluctuating and unstable decay. In contrast, the gradient distance of the batchnorm VGG model steadily decreases and remains relatively stable. This implies that the batchnorm VGG model exhibits better gradient prediction capabilities during the later stages. The

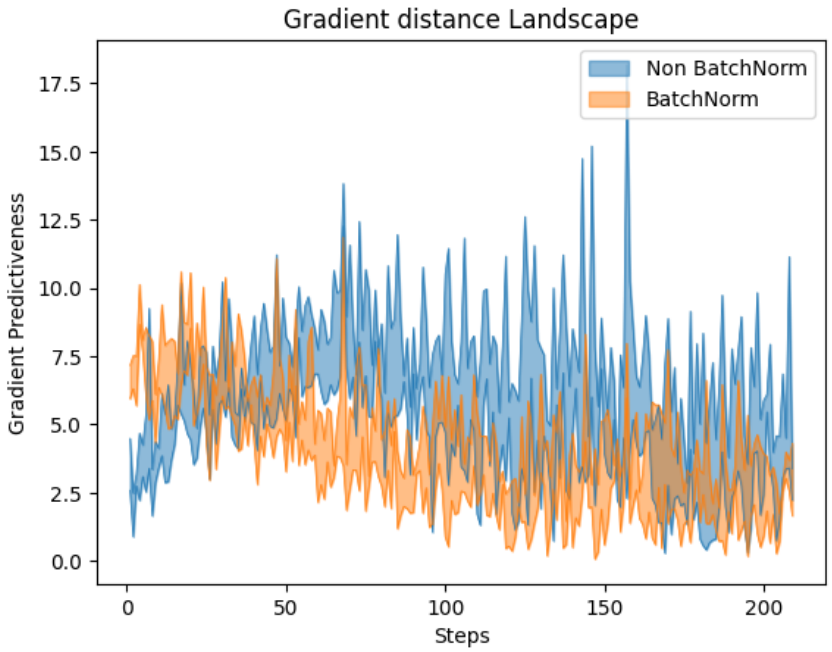


Figure 12: Comparison of Gradient Distance throughout training

relatively large early gradient distance may be attributed to the changes in features across different layers, as batchnorm itself is a learnable layer.

3.2 "Effective" β-Smoothness

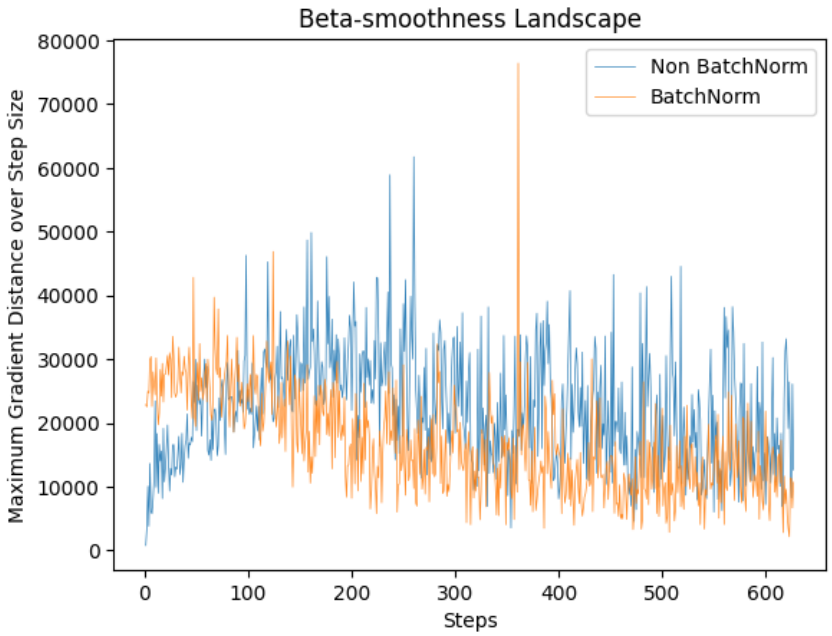


Figure 13: Gradient Distance over Step Size

In the context of "effective" β-smoothness, we investigate the maximum gradient distance over the step size. A function is considered β-smooth if its gradient is β-Lipschitz, meaning that the distance

between two points is multiplied by β when compared to the distance between their respective gradients. By examining the maximum gradient distance over the step size, we can approximate the value of β for the loss function. A smaller β value indicates a smoother function. Figure 13 provides evidence that batch normalization contributes to the smoothness of the loss function.

3.3 New Optimizing Solver

3.3.1 LeNet, MNIST and Vanilla Optims

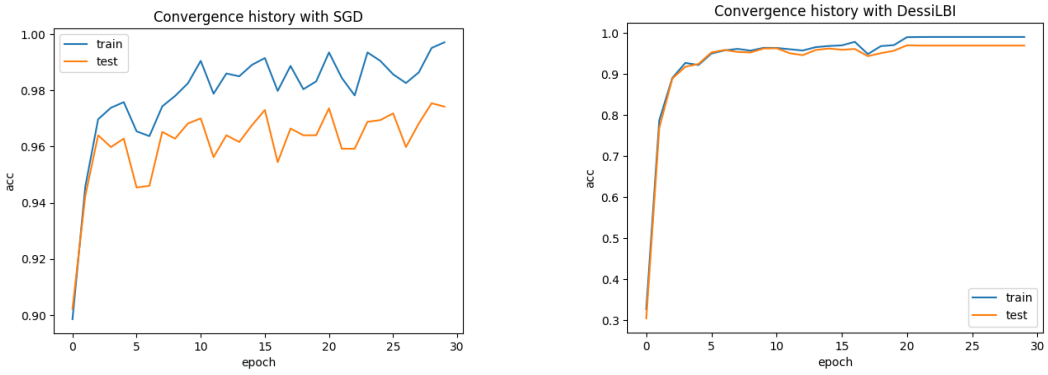
epoch	batch size	learning rate	kappa	mu
30	128	0.1**(epoch//20)	1	20

Table 9: Hyper-parameter

prune rate	0	20	40	60	70	80
conv3	96.88	96.88	96.88	96.10	94.90	85.84
conv3 + fc1	96.88	96.88	96.88	96.10	94.90	85.84

Table 10: Accuracy after pruning

As can be seen in Table 10, the over-parameterization of LeNet on MNIST is evident, since the accuracy rate only begins to drop when the prune rate is over 60%. With the help of DessiLBI optimizer, the sparsity structure is uncovered, as shown in Figure 15, and the overfitting is largely prevented as Figure 14 shows. As the feature map in 16 shows, while SGD assigns every kernel some works, under DessiLBI most kernel does not actually work, and the feature extraction is centralized to some kernels, which makes the fitting compact.



(a) SGD (b) DessiLBI

Figure 14: Comparison over convergence history

3.3.2 ResNet50/VGG19, CIFAR10 and AdamW Optims

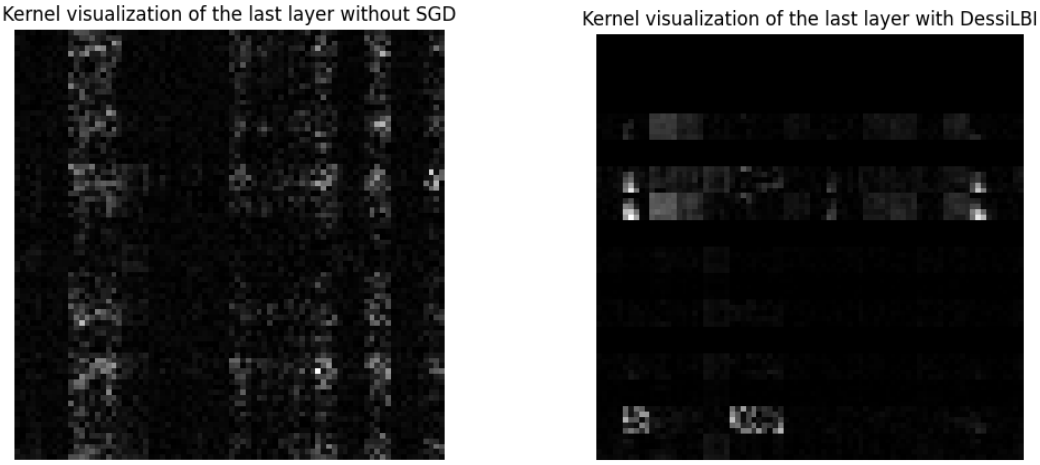
We implemented AdamW[13] optimizer, which simply seperates gradient descent and weight decay in Adam.

epoch	batch size	kappa	mu
30	128	1	20

Table 11: Hyper-parameter

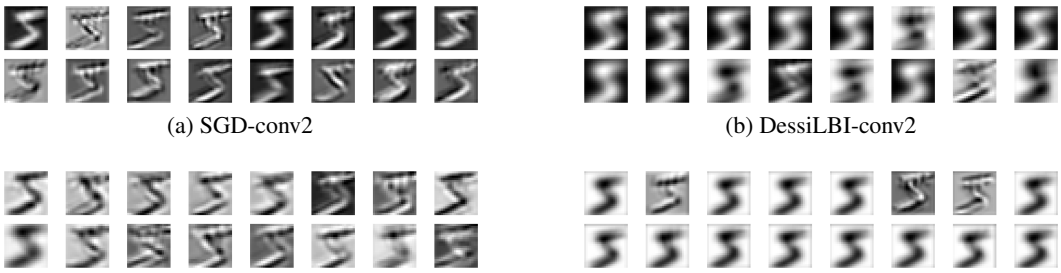
Learning rate is 3E-4 for ResNet50, and 1E-5 for VGG19.

As the result shows, both networks, although complex, do not show much over-parameterization on CIFAR10. It is mainly due to the structural complexity of CIFAR10, since many data are spatially



(a) SGD (b) DessiLBI

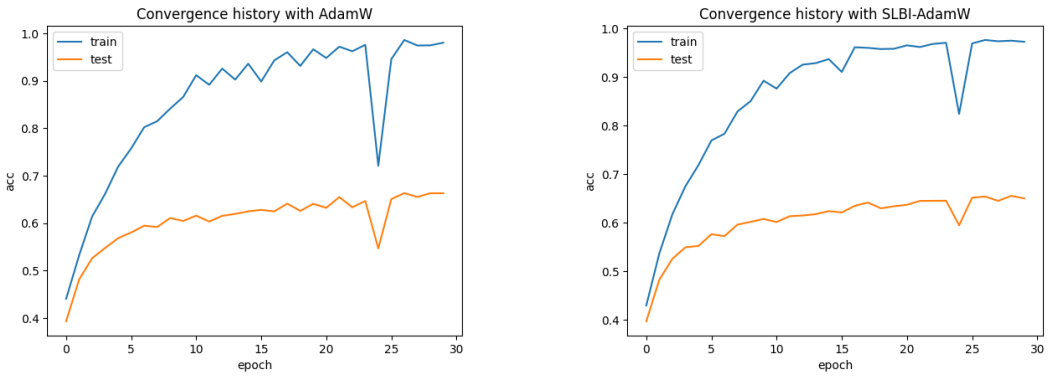
Figure 15: Comparison over kernel sparsity



(a) SGD-conv2 (b) DessiLBI-conv2

(c) SGD-conv3 (d) DessiLBI-conv3

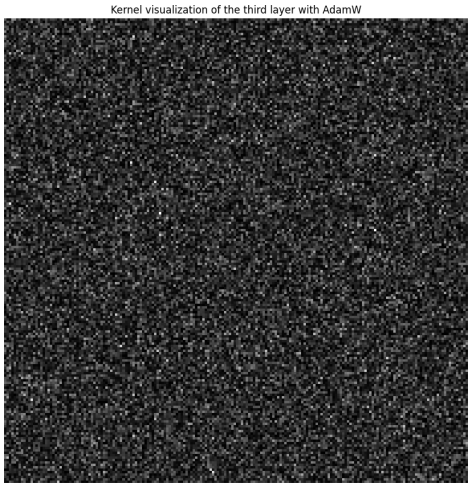
Figure 16: Comparison over feature map



(a) AdamW (b) DessiLBI-AdamW

Figure 17: Comparison over convergence history on ResNet50

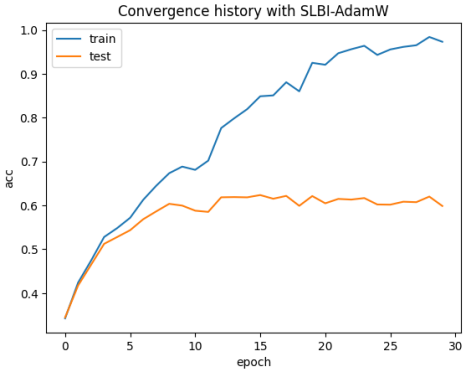
different and have different colors, even though they are from the same class. However, DessiLBI can fix the overfitting problem of VGG on CIFAR10 to some degree.



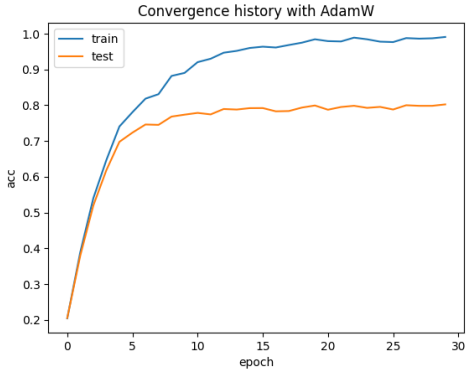
(a) AdamW

(b) DessiLBI-AdamW

Figure 18: Comparison over kernel sparsity on ResNet50



(a) AdamW



(b) DessiLBI-AdamW

Figure 19: Comparison over convergence history on VGG

References

[1] M. Bauer, E. Dupont, A. Brock, D. Rosenbaum, J. R. Schwarz, and H. Kim. Spatial functa: Scaling functa to imagenet classification and generation, 2023.

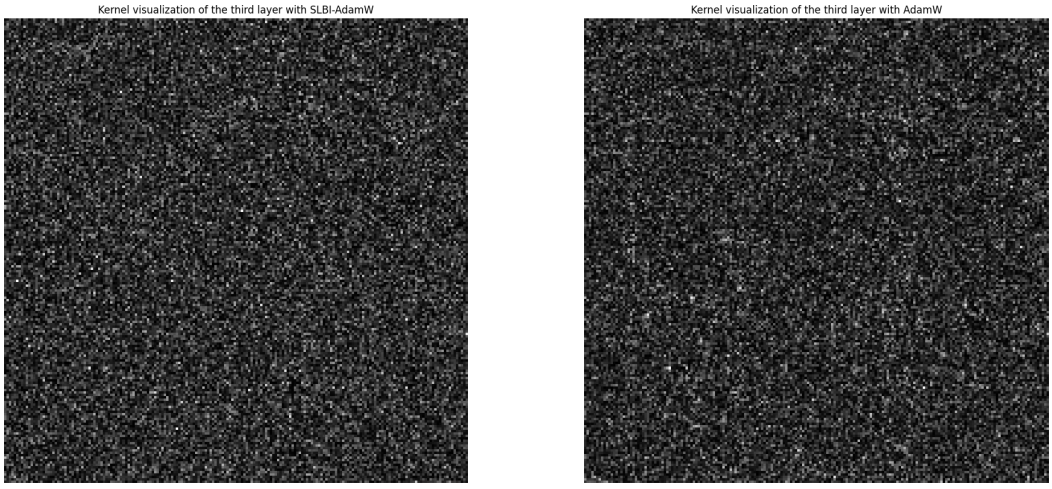
[2] E. R. Chan, M. Monteiro, P. Kellnhofer, J. Wu, and G. Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis, 2021.

[3] E. Dupont, H. Kim, S. M. A. Eslami, D. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one, 2022.

[4] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16091–16101, 2023.

[5] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[6] Y. Fu, C. Liu, D. Li, Z. Zhong, X. Sun, J. Zeng, and Y. Yao. Exploring structural sparsity of deep networks via inverse scale spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1749–1765, 2022.



(a) AdamW (b) DessiLBI-AdamW

Figure 20: Comparison over kernel sparsity on VGG

(a) AdamW (b) DessiLBI-AdamW

Figure 21: Comparison over feature map on VGG

[7] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[8] N. Lee, T. Ajanthan, S. Gould, and P. H. Torr. A signal propagation perspective for pruning neural networks at initialization. *arXiv preprint arXiv:1906.06307*, 2019.

[9] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[10] L. Li, D. Thorsley, and J. Hassoun. Sait: Sparse vision transformers through adaptive token pruning. *arXiv preprint arXiv:2210.05832*, 2022.

[11] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few-shot learning, 2017.

[12] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021.

[13] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019.

[14] I. Mehta, M. Gharbi, C. Barnes, E. Shechtman, R. Ramamoorthi, and M. Chandraker. Modulated periodic activations for generalizable local functional representations, 2021.

[15] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

[16] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer, 2017.

[17] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

[18] V. Sanh, T. Wolf, and A. Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

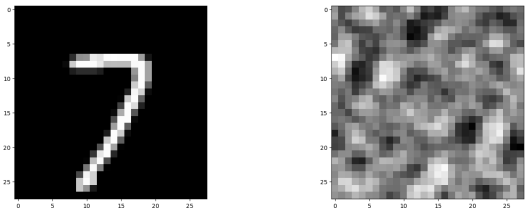
[19] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions, 2020.

[20] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.

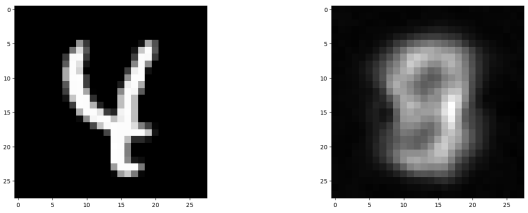
[21] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[22] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning, 2019.

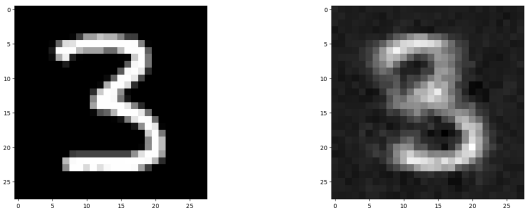
A Appendix



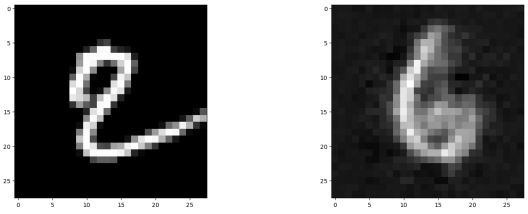
(a) Start



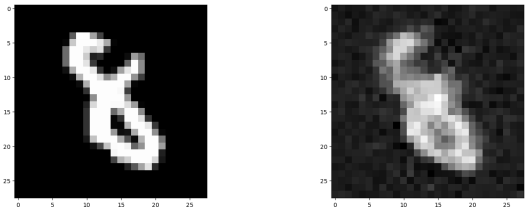
(b) Embryo



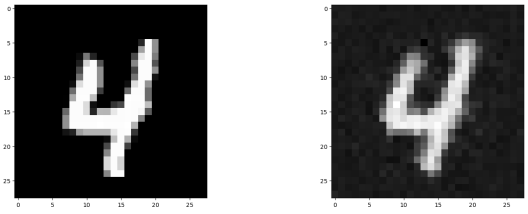
(c) Developing



(d) Mis-impression

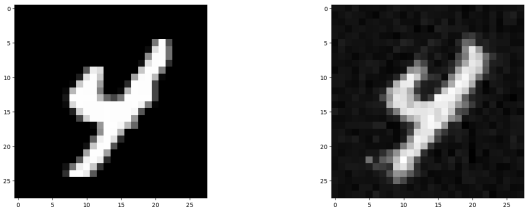


(e) More Precise

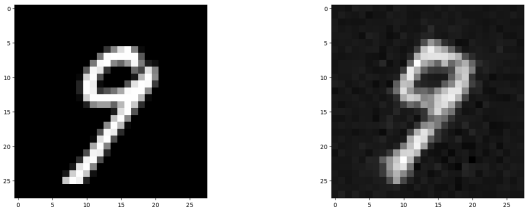


(f) Last

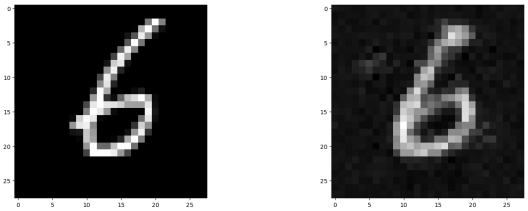
Figure 22: Different phases during training process on MNIST



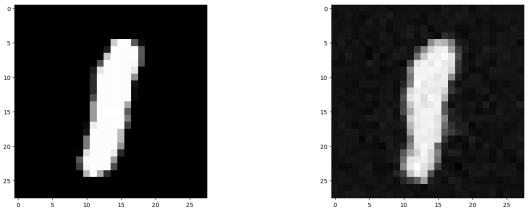
(a)



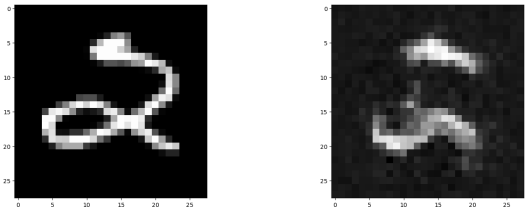
(b)



(c)

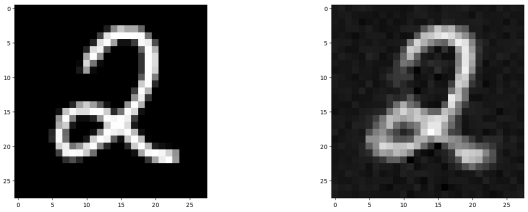


(d)

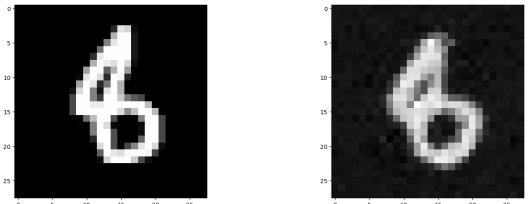


(e)

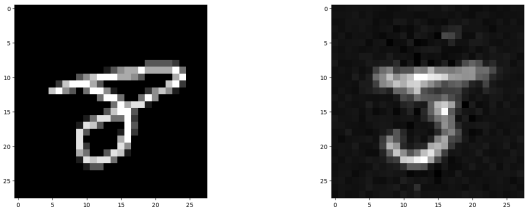
Figure 23: Out-of-domain Fitting



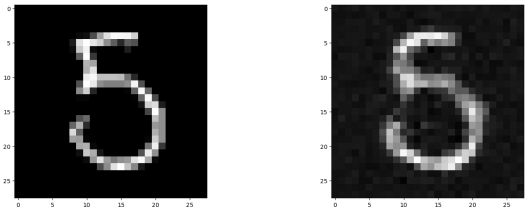
(a)



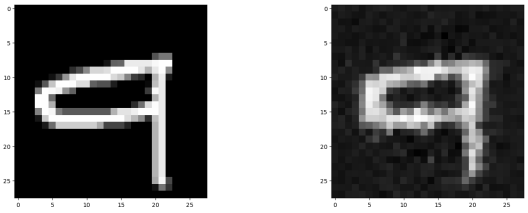
(b)



(c)

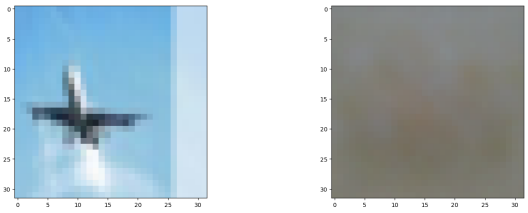


(d)

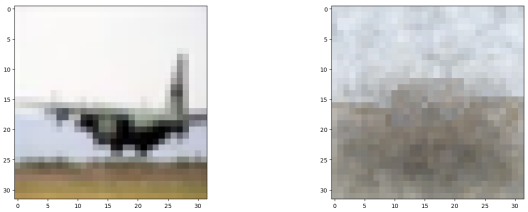


(e)

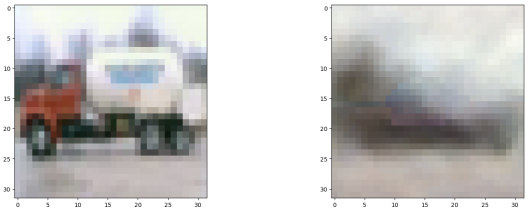
Figure 24: More MNIST in-domain fitting examples



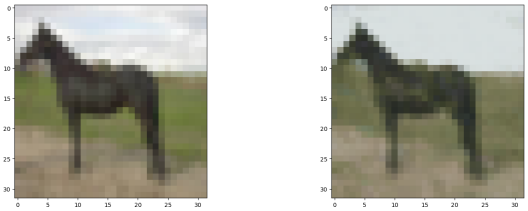
(a) Start



(b) Coarse

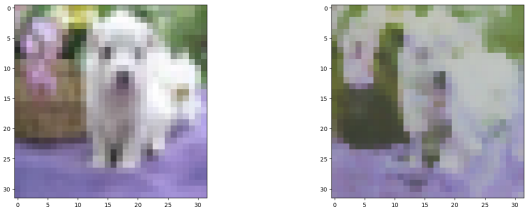


(c) Impressionism

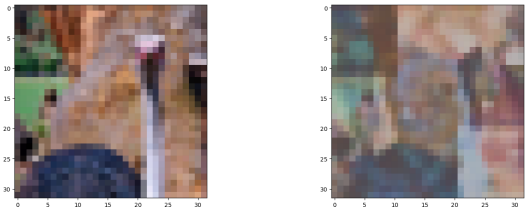


(d) More Precise

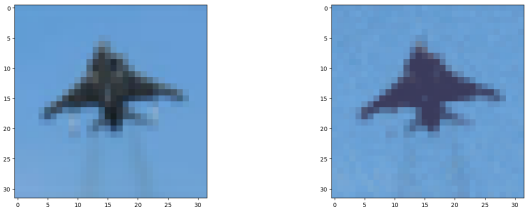
Figure 25: Training process on CIFAR10



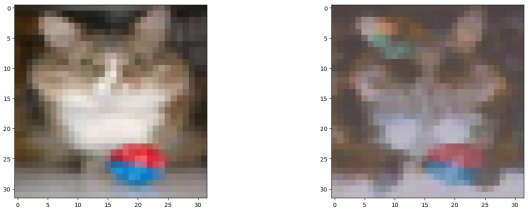
(a)



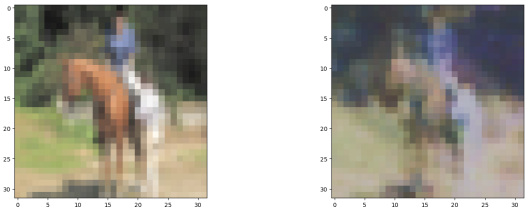
(b)



(c)



(d)



(e)

Figure 26: More CIFAR10 fitting examples