Final Project

Xiaowei Zeng
Department of Management
Fudan University
20307100124@fudan.edu.cn

Yixin Fan
Department of Management
Fudan University
20307100150@fudan.edu.cn

Abstract

This project contains 3 neural network tasks. 1) The first one is to train an implicit network of Functa. We implement our own latent modulated SIREN functa model, and evaluate and display the model performance on two datasets – celeb_a_hq and MNIST. Our model exhibits fairly good performance in both continuity and time-memory saving property, and obtains satisfactory results in our downstream classification experiments compared to using the discretized representation of the original images as the input to a modified ResNet18 model. 2) The second one is to prune the Vision Transformer (ViT) via Sparsity. Three models – P-ViT, AP-ViT, CP-ViT – are implemented in this task, in which CP-ViT performs best at the given 9 levels (0.1 - 0.9) of pruning ratios because it takes layer-aware interdependency into consideration. 3) The third task includes two small topics: one is to analyze the effect of BN on gradient predictiveness and effective β-smoothness; the other is to implement DessiLBI with Base SGD and Adam. Results show that with DessiLBI, Adam is higher in stablility but lower in sparsity especially in convolutional layers compared to SGD.

1 Introduction to Functa

Functa was first proposed in 2022 as a concise term for implicit neural representation (INR) that conceives INRs as data [1]. The INR itself is not a new concept, which refers to a continuous neural function trained to output the appropriate measurement value for any input spatial location, for example, to continuously map 2D pixel coordinates to RGB values for an input image, and is proposed to make up for the deficiencies of common discrete representation in various data modalities. Therefore, functa shares the same goal of optimization with INR:

$$\min_{\theta} \mathcal{L}\left(f_\theta, \{\mathbf{x}_i, \mathbf{f}_i\}_{i\in\mathcal{I}}\right) = \min_{\theta} \sum_{i\in\mathcal{I}} \left\| f_\theta\left(\mathbf{x}_i\right) - \mathbf{f}_i \right\|_2^2 . \tag{1}$$

We first briefly demonstrate the main advantages of functa representation in comparison with the conventional discrete representation, then provide explanations to its applications in several downstream tasks.

1.1 Main Advantages

For discretizable modalities like image, array based discrete representations can scale poorly with resolution, which is highly memory-consuming as a network input, and the array size (network input size) can be varied a lot in reality, which reduces the model generation ability. In contrast, the scalability of functa, which can also be constructed as an array but an array of model parameters, is capable to be well controlled by suitable model design, and is insensitive to input data size.

For modalities that cannot be discretized, functa can still be applied as an encoder (a neural network component that takes an input data sample and transforms it into a lower-dimensional representation)

Final Project of Neural Network and Deep Learning (DATA130011.01, 2023).

or a decoder (the counterpart of the encoder and aims to reconstruct the original input data from the encoded representation in the latent space) due to its continuity property.

The above two advantages enable functa to be capable of handling different modalities as well as enabling easier implementation for downstream applications described in the next part in a both time and memory saving way.

1.2 Applications

Two mainstream of functa's applications are the generative and discriminative tasks.

1.2.1 Generative Modeling

The goal of **generative modeling** is to learn the underlying patterns and structures present in the training data and use this knowledge to generate new, previously unseen data that is similar in nature. The two typical generative model used in [1] are normalizing flows and diffusion.

1.2.1.1 Normalizing Flows

The basic idea behind normalizing flows is to define a sequence of invertible transformations, called flow transformations, that map samples from a simple distribution (e.g., Gaussian) to samples from the target distribution. By chaining multiple flow transformations together, the model can capture complex dependencies and generate samples from a highly flexible distribution.

Each flow transformation consists of two steps:

Forward Transformation: In the forward transformation step, a sample from the simple distribution is transformed into a more complex distribution. This transformation is typically parameterized by a neural network that takes the input sample and outputs a transformed sample. The transformation should be invertible, meaning that it is possible to recover the original sample from the transformed sample.

Inverse Transformation: The inverse transformation step allows the reconstruction of the original sample from the transformed sample. It maps the transformed sample back to the original space. Similar to the forward transformation, the inverse transformation is typically parameterized by a neural network.

1.2.1.2 Diffusion

The main idea behind diffusion models is to simulate the gradual spreading or diffusion of noise throughout the image generation process. The model starts with a random noise vector, and at each step, it applies a diffusion process that updates the noise vector to produce a more realistic image. The diffusion process consists of multiple iterations, where the noise vector is transformed step-by-step to generate increasingly coherent image samples.

During each diffusion step, a carefully designed transformation function is applied to the current noise vector. This transformation function captures the dependencies and patterns in the data distribution. The transformation is typically implemented using neural networks, specifically invertible neural networks, to ensure that the process is reversible. Reversibility is crucial for training and sampling from the diffusion model.

Training diffusion models involves optimizing the model parameters to match the target data distribution. The model is trained by maximizing the likelihood of the observed data samples. This is done by comparing the generated samples at each diffusion step with the corresponding real samples, using techniques such as maximum likelihood estimation or score matching.

1.2.2 Discriminative Task

The goal of **discriminative task** is to learn the relationship between input data and their corresponding labels or classes. The focus is on modeling the decision boundary that separates different classes or categories in the data.

In discriminative modeling, the goal is to build a model that can accurately classify or predict the label or class of new, unseen data points based on their features. The model learns to discriminate

between different classes by capturing the patterns and characteristics that distinguish one class from another.

Common examples of discriminative tasks include: classification (which will be implemented in our downstream experients), regression, object detection and sentiment analysis etc.

2 Related Work of Functa

2.1 Review of SIREN

SIREN was first proposed [2] to solve the deficit of previous INR networks that typically use a ReLU as activation function, which are incapable of modeling signals with fine detail, and fail to represent a signal's spatial and temporal derivatives, despite the fact that these are essential to many physical signals defined implicitly as the solution to partial differential equations.

SIREN aims at obtaining a function ψ which satisfy equations of the form:

$$F\left(\mathbf{x}, \Phi, \nabla_{\mathbf{x}}\Phi, \nabla_{\mathbf{x}}^2\Phi, \ldots\right) = 0, \quad \Phi : \mathbf{x} \mapsto \Phi(\mathbf{x}). \tag{2}$$

SIREN itself is a simple neural network for INR but using periodic **sine** as its activation function, and is therefore endowed with the desirable property of "any derivative of a SIREN is itself a SIREN", which enable this architecture to exhibit good model performance in various tasks.

2.2 Review of Latent Modulated SIREN

The latent modulated SIREN structure is proposed with the purpose to create large datasets of functa using SIREN as its base INRs throughout in a scalable and effective manner.

The two main fundamental techniques referred to by this structure are *modulations* and *meta-learning*. Here is our description of how the latent modulated SIREN structure utilizes these two techniques to achieve its goal of scalability and effectiveness.

2.2.1 Modulations

Basically, the modulations can be divided into two parts: shared modulation and individual modulations which are further processed into latent modulations.

2.2.1.1 Shared Modulation

In shared modulation, the idea is to have a base network or model that is shared among multiple data points or instances. The shared base network captures the common underlying structure or patterns shared across these data points. By sharing parameters and weights among the instances, the model can leverage the similarities and commonalities to enhance learning and improve generalization.

Typically, shared modulation is employed in scenarios where the data points have some intrinsic similarity or share common features. This is exactly the application case for training a dataset of functa, where each of the original input data point is similar with each other in the sense of e.g.,pixel distribution attribute etc.

Therefore, instead of separately learning a model for each data point – as it is the case when implementing the basic SIREN model or other INR models – a single base network is designed and trained to extract and model the shared structure; and in this case, the shared modulation is a base SIREN.

2.2.1.2 Individual Modulations

In individual modulations, a specific modulation is individually learned for each input data point. Typically, a modulation is represented as elementwise affine transformations applied to the activations of the neural network.

Here, *elementwise* means that the operations are applied independently to each element of the activations, i.e., the output values generated by each neuron in the network after going through the activation function. *affine transformations* involves both shifting (translation) and scaling (multiplication) of the activation values.

Practically, this can be achieved by incorporating additional trainable parameters based on the shared modulation SIREN. These parameters control the amount of shift and scale applied to each activation, allowing the network to learn how to modulate or adjust the behavior of individual neurons or groups of neurons, enhancing its flexibility and adaptability to different inputs or situations. These parameters altogether form the individual modulation representation.

2.2.1.3 Latent Modulation

To further achieve both the time and memory effectiveness goal, a latent modulation vector is then mapped to the individual modulation composed by affine transformation parameters. This mapping can be implemented by another MLP or a simple linear mapping (the latter one is later adopted in our implementation because of its better performance).

These latent modulations or simply called **modulations** are then viewed as the implementation form of the concept **functa**.

2.2.2 Meta-Learning

Meta-learning, also known as "learning to learn", refers to the process of training a neural network to learn how to quickly adapt and generalize to new tasks or environments. In the context of meta-learning a network, it involves training a network to become proficient at learning from a set of training tasks – in this task, a shared modulation SIREN – such that it can effectively learn new tasks with minimal additional training – in this task, learn an individual modulation for each sample data point.

The success of meta-learning is measured by the network's ability to generalize well to new tasks and achieve good performance with limited adaptation or training on each specific task. The network should be capable of learning from a few examples or data points of the new task and rapidly adapting its internal representations to achieve optimal performance.

The goal of meta-learning in this task is to define a effective structure to initialize the shared modulation SIREN and to "fine-tune" the individual modulations based on this fixed shared initialization.

The proposed meta-learning structure is a **inner-outer loop method of parameter update**. Specifically, in the outer loop, only the parameters of the shared modulation SIREN are updated, while in the inner loop, only the parameters of the sampled batch individual modulations are updated. By repeatedly executing this inner-outer loop from a random initialization, the shared modulation SIREN can be suitably initialized.

This suitably initialized SIREN will then be applied with the affine transformation parameters to fit each individual data point and produce the individual modulations just as the function of the inner loop. This process is significantly simplified to a few gradient update step due to the pre-trained shared modulation, which is far more time-saving than simply train a model on each data point to generate a corresponding modulate.

2.3 Review of Spatial Functa

Spatial Functa is proposed by the latest paper [3] with the purpose of overcome the limitations of functa when scaling up to even moderatly complex datasets such as CIFAR-10, by using spatially arranged latent representations of neural fields which capture local signal information in input space which allows the model to be able to scale up to much larger datasets like ImageNet-1k at 256 x 256 resolution.

3 Experiment Implementation of Functa

We follow the original model structure and algorithms [1] to implement our own latent modulated SIREN functa model, and to evaluate and display the model performance on two datasets – celeb_a_hq and MNIST.

Our detailed model settings for training on MNIST are listed as below:

1. Inner-outer loop settings:
   Number of hidden layers of shared SIREN = 15
   Hidden layer size & modulation (before latent) size = 512
   Only use shift affine transformation weights for its performance equals use both shift and scale and can save store memory
   Inner step = 3
   Outer step = 100000
   Training batch size = 4
2. Modulation (before latent) to latent modulation mapping setting:
   Use linear mapping instead of a deeper MLP as has mentioned above for its better performance in PSNR
   Latent modulation size = 128
3. Optimizer settings:
   Optimization criterion = MSE loss
   Inner optimizer = SGD
   Inner learning rate = 1e-2
   Outer optimizer = Adam
   Outer learning rate = 3e-6, weight decay = 1e-4
4. Evaluation methods:
   Inner MSE loss & PSNR (see details in the next section)

4 Experiment Results of Functa

4.1 Fit Effect of Functa Model of Different Settings

We first test the fit effect of our latent modulated SIREN functa model of different modulation (i.e., latent modulation) sizes on celeb_a_hq. Modulation size of 64, 128, 256, 512, 1024 are tried.

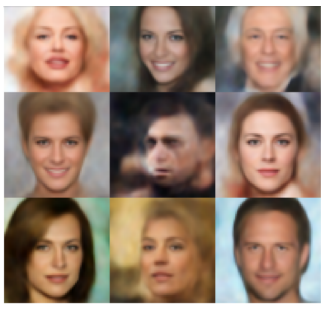


Figure 1: Modu_size=64

Figure 2: Modu_size=128

Figure 3: Modu_size=256

Figure 4: Modu_size=512

Figure 5: Modu_size=1024

It can be shown from the reconstructed images that although lower dimension of modulation size has already been able to capture the RGB pixel distribution characteristics of the original input image to

some degree, the larger the dimension grows, the more details of the original image can be captured. Also, the time needed to reconstruct the modulation increases as the dimension grows; however, this time will not be associated with a square increasing rate for 2D images or a cubic increasing rate for 3D voxel grids as has been explained in previous *Advantage* part.

We then test the fit effect of our latent modulated SIREN functa model of different reconstruction sizes with fixed modulation size (512) on celeb_a_hq. Reconstruction size of 64 and 128 are tried.

Figure 6: Reconstruct_size=64

Figure 7: Reconstruct_size=128

It is clear to show that the larger the reconstruction size, the smoother the image is displayed and much more closer to the real world continuous data representation of images. However, as the continuous mapping calculations grow squarely with the reconstruction size, it can be quite time-consuming when this size is set larger.

Also, although the training process for the inner-outer loop takes a few GPU days on MNIST, it is much more faster then to gain the individual modulations based on this training process – with each image takes only 3 gradient update inner step the PSNR and the corresponding reconstruct performance are already satisfactory, and this process only takes no more than a second for each image with GPU – which is far more fascinating than simply using a SIRN to directly fit to each image individually and store the whole set of model parameters as the modulations, which is far more time and memory consuming.

4.2 Evaluation of Our Implemented Latent Modulated SIREN Functa Model

We use two metrics – peak signal-to-noise ratio (PSNR) (per batch) between the original and restored images, and inner loop MSE loss (per batch) – on our model's performances.

Peak Signal-to-Noise Ratio (PSNR) is a metric used to evaluate the quality of a reconstructed or compressed signal by comparing it to the original, reference signal. It measures the ratio between the peak power of the signal and the power of the noise, expressed in decibels (dB).

The formula for calculating PSNR is as follows:

$$\text{PSNR} = 10 \times \log_{10}(\text{MAX}^2/\text{MSE}), \tag{3}$$

where MAX is the maximum possible pixel value of the signal (for example, 255 for an 8-bit grayscale image), and MSE (Mean Squared Error) is the average squared difference between the reference signal and the reconstructed signal over all pixels.

PSNR is commonly used in image and video processing applications, where it provides a quantitative measure of the fidelity of the reconstructed or compressed data. Higher PSNR values indicate higher similarity and better quality between the original and reconstructed signals.

It's important to note that PSNR is based on mean squared error and doesn't always align perfectly with human perception. It is just one of several metrics used for quality assessment, and different applications may require different metrics based on their specific needs.

We displayed the evaluation results on MNIST with modulation size = 512 as an example. The mean inner loop MSE is 0.0454 (Figure 8), and the mean PSNR is 19.5356 (Figure 9).

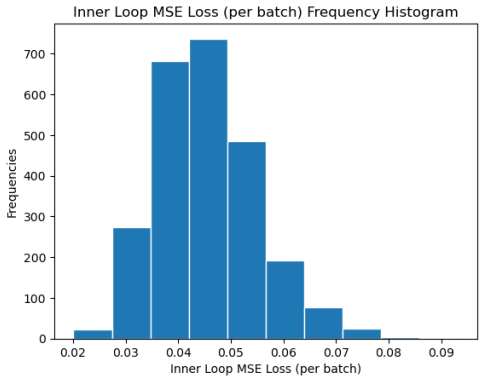


Figure 8: Inner Loop MSE Loss Distribution

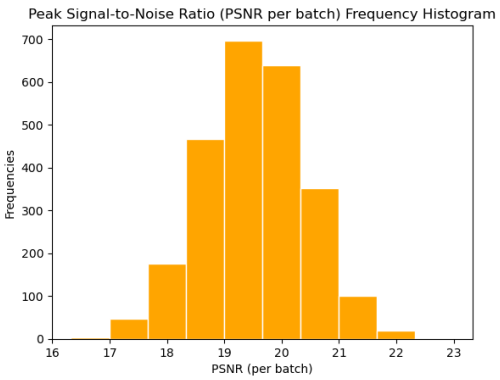


Figure 9: PSNR Distribution

We also display some of the modulation reconstruction results here for better visualization:

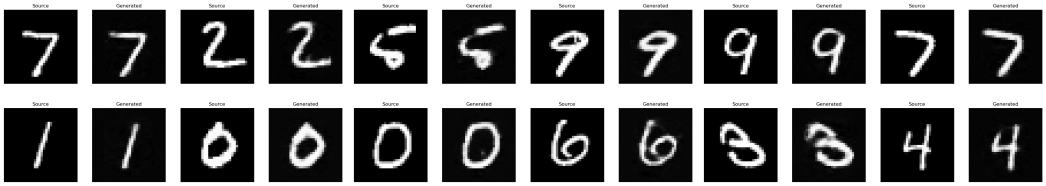


Figure 10: PSNR Distribution

This reconstruction results seem quite satisfactory, especially for those digits with clearer or simpler handwriting.

4.3 Our Implemented Latent Modulated SIREN Functa Model for Downstream Tasks

We use MNIST for image classification and test the classification performance of using latent modulations (functa) as input data compared with using the discretized representation of the original images.

For functa, as has been pointed out in [1], using a shared framework for different data modalities implies that we cannot employ modality specific inductive biases when designing models trained on functa. Indeed, storing functa as modulations removes spatial structure from the data, forcing us to use general MLP architectures instead of for example convolutional networks.

Therefore, we simply implement a three layer MLP with each hidden layer containing 1024 neurons. The input layer size is correspondingly set to 512 as the modulation size, and the output layer size equals to the number of classes 10.

For the discretized representation of original images, we adopt the **ResNet18_CD_PreActivation** model structure as has been implemented and tested in *Project 2*. This model structure is added some refinements based on the basic ResNet18 network, which are listed below as a review:

1. Replace the 7x7 convolution kernel by three 3x3 kernels in the set-up stage, and postpone the down-sampling operation to the 3rd convolution layer (Figure 11).
2. Change the down-sampling channel-expansion identity connection in residual block B shown above into two layers composed by first a down-sampling average pooling layer and second a channel-expansion 1x1 convolution layer without down-sampling.

   This can alleviate the information loss caused by using 1x1 convolution kernel and stride = 2 simultaneously (Figure 12).
3. Implement full pre-activation in the residual block to enhance the expression ability of the activation function (Figure 13).

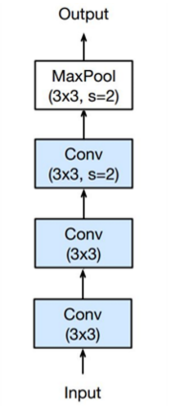


Figure 11: Set-Up Plus

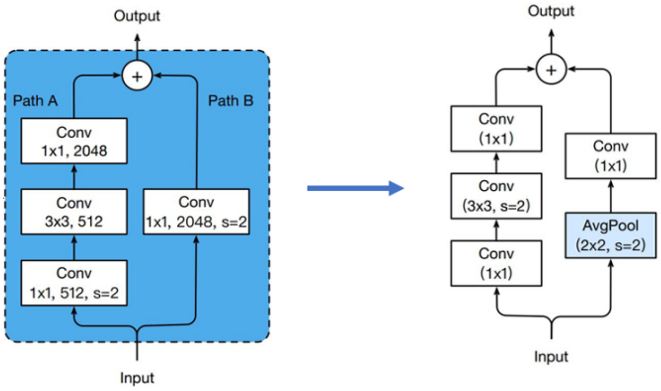


Figure 12: Identity Block Plus

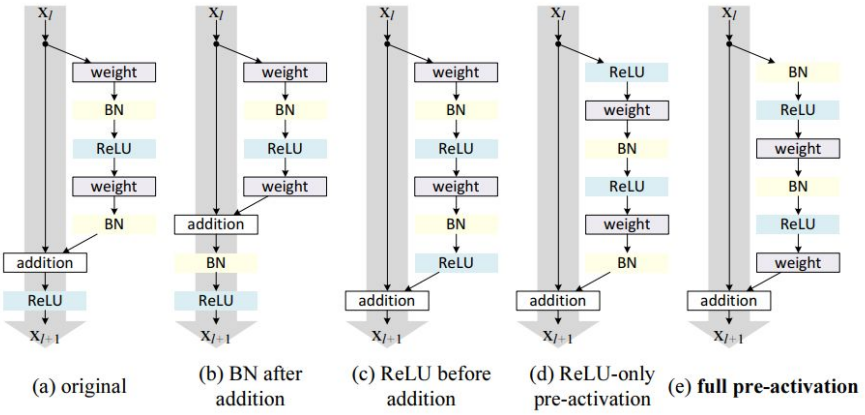
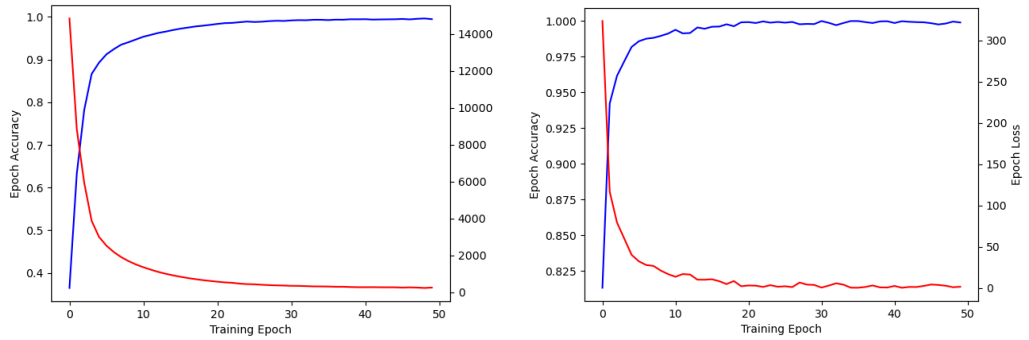


Figure 13: Full Pre-Activation

Note that because we use 90000 MNIST images as training set for our latent modulated SIREN functa model training and use the remaining 10000 images as test set, we therefore use modulations generated by this test set for functa downstream classification task, with a split of 9:1 for training and test functaset. We then accordingly downsample the the discretized representation of the original images also to 10000 with a split of 9:1.

The corresponding experiment results are displayed as below. Using latent modulated SIREN functa as input, the testing loss is 0.951 (Figure 14a). Using discretized representation of the original images as input, the testing loss is 0.977 (Figure 14b).

From the results we can see that although the ACC curve of the functaset input with MLP doesn't rise as steep as that of the discretized representation of the original images as input with ResNet, their later training process and evaluation performances very close. Considering functaset id only be applied to a simple MLP model in our experiment, this result is relatively satisfactory.



(a) MLP (b) ResNet18_CD_PreActivation

Figure 14: Learning Curve

4.4 Conclusion

To conclude, our latent modulated SIREN functa model exhibits fairly good performance in both continuity and time-memory saving property, and obtains satisfactory results in our downstream classification experiments. However, as has been mentioned before, this model may not be satisfactory when scale up to even moderatly complex datasets such as CIFAR-10 and ImageNet. We can further try implementing spatial functa to test the performance improvement to this latent modulated SIREN functa model in future experiments.

5 Pruning Transformer via Sparsity

5.1 Background

Nowadays, Transformer has attracted much attention and shown remarkable results in various applications such as Natural Language Processing (NLP) and computer vision. A transformer is made up of an encoder and a decoder, which were originally designed for sequence to sequence tasks like machine translation, and has recently been applied to image classification, object detection, etc. However, most of the transformers rely on millions or even billions of parameters, whose storage, run-time memory, and computational resource requirements are hindering the deployment to limited storage platforms and edge devices.

A typical Vision Transformer (ViT) architecture consists of Multi-Head Self-Attention (MHSA), Multi-Layer Perceptron (MLP), layer normalization, activation function, and shortcut connection. The widely-used fully-connected layers in the transformer will lead to computation and storage burden. Therefore, in this section, we will try to prune the ViT via sparsity while preserving its performance on the CIFAR-10 dataset. Due to time constraints, we will not fine-tune the models after completing the pruning process.

5.2 Models

5.2.1 P-ViT: ViT Pruning

The article *Vision Transformer Pruning* [4] proposes to prune the vision transformer according to the learnable importance scores. Suppose the features are $X \in \mathbb{R}^{n \times d}$, where n denotes the number of features and d denotes the feature dimensions, the optimal discrete importance scores are $\mathrm{a}^* \in \{0,1\}^d$ and the relaxed importance scores are $\hat{\mathrm{a}} \in \{0,1\}^d$. We can obtain the soft pruned features:

$$\hat{X} = X\mathrm{diag}(\hat{\mathrm{a}}) \tag{4}$$

Next, in order to enforce the sparsity of importance scores, $l1$ sparsity regularization is applied on the importance scores by adding a term $\lambda||\hat{\mathrm{a}}||_1$ to the training objective function.

Finally, a pruning rate τ (or threshold) is decided to obtain the discrete $a^* = \hat{a} > \tau$. Prune the transformer according to a^*. The whole model architechture is shown in Figure 15.

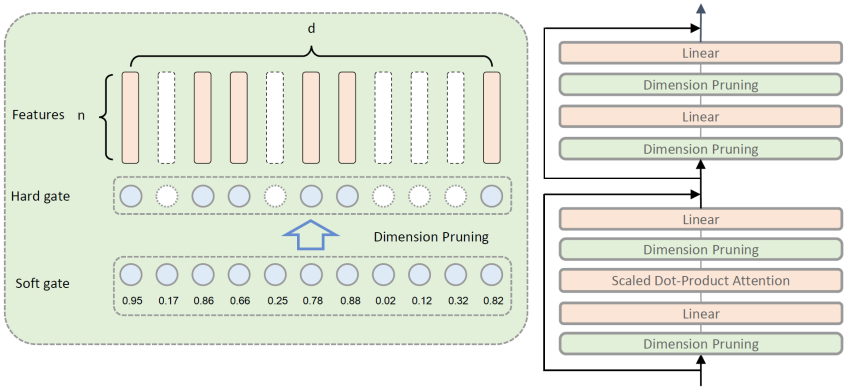


Figure 15: Vision Transformer Pruning

5.2.2 AP-ViT: Automatic ViT Pruning

Automatic Vision Transformer Pruning (AP-ViT) leverages reinforcement learning to construct an agent that can accurately and automatically dynamically prune parameters with little significance [5]. This approach enables the compression of long sequences of information into extremely short sequence information, resulting in a significant reduction in computational burden.

AP-ViT model is trained using a three-channel mode (Figure 16). For the reinforcement learning agent, the key challenge is how to measure the feedback generated by actions. To evaluate the impact of a single action, the following three steps are employed:

1) Make predictions using a regular model (without pruning module);

2) Make predictions using a model with a pruning module (a single pruning action);

3) Compare the two predictions, reward the pruning action if pruning reduces the loss which indicates improvement in fitting real images, otherwise penalize the pruning action.

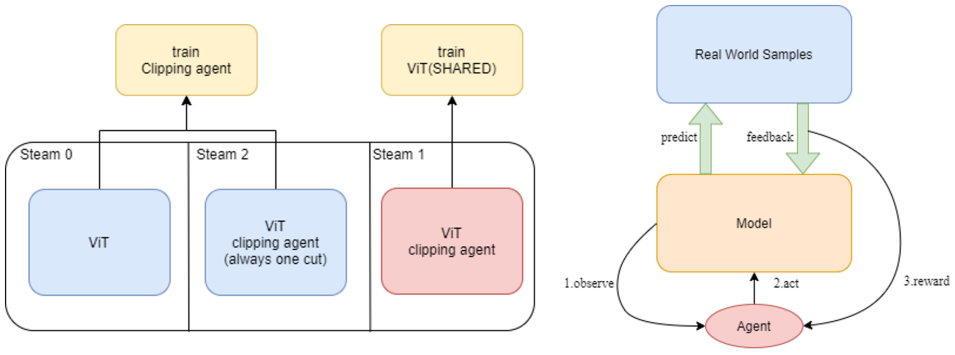


Figure 16: Automatic Vision Transformer Pruning

Several clipping layers' id are selected to clip and drop some channels of the convolutional layers in AP-ViT. Besides, the pruning ratio cannot be directedly determined but should be derived from the softmax probability threshold.

5.2.3 CP-ViT: Cascade ViT Pruning

The paper [6] proposes patch-wise and head-wise Cascade Vision Transformer Pruning (CP-ViT) that can dynamically locate informative patches and heads (collectively called as PH-region), so as to reduce the computation complexity with minimized accuracy loss. The overall framework of CP-ViT is shown in Figure 17.

The highlight of this article mainly lies in dynamic layer-aware cascade pruning. This strategy is implemented to deal with the accuracy degradation problem caused by uniform pruning ratio. Cascade pruning is based on *attention_probability* ($H \times L \times L$) proposed by the author. The element *attention_probability*$[h, i, j]$ involves inner product between the ith row in Q and jth column in K^T, in which $Q, K, V \in \mathbb{R}^{h \times L \times D}$. It can also be interpreted as the interdependency between ith patch and jth patch in head h. By locating the bright vertical lines (in visulization), we can identify the informative patches and increase the pruning ratio.

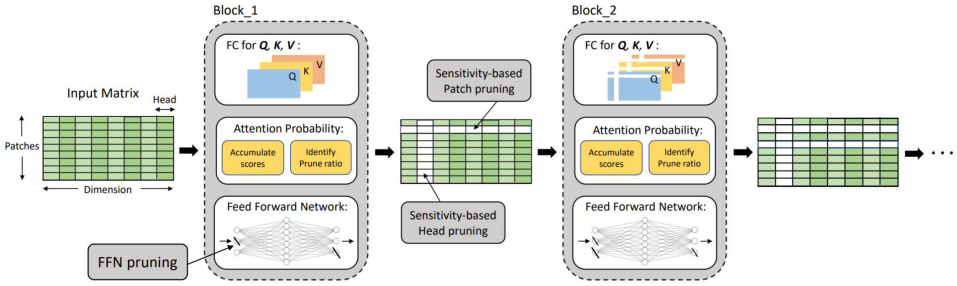


Figure 17: Cascade Vision Transformer Pruning

5.3 Experiments and Results

The code implemented in this section is adapted from these repositories [7, 8, 9, 5, 10], modified to fit the computing environment, and make it convenient to adjust sparsity and to report both top 1 and top 5 accuracies.

The base model for pruning is the Vision Transformer (ViT-B_16/224) model, which is pretrained ImageNet-21k (14 million images, 21,843 classes) at a resolution of 224×224 [11]. Before starting training the model, we first finetune the pretrained model on CIFAR-10 for 5000 steps (batch size = 32). The learning curve is shown in Figure 18.

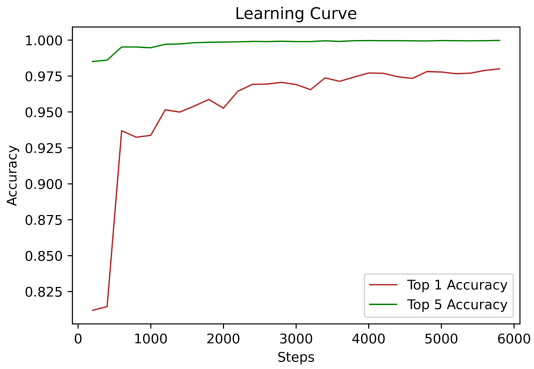


Figure 18: Learning curve

We use the fine-tuned model to prune the ViT. The accuracies of different models with different pruning ratios are listed in Table 1.

P-ViT, AP-ViT and CP-ViT conduct pruning respectively on importance scores, softmax probability of selected clipping layers (Table 2) and attention probability in PH-region.

CP-ViT has the best performance as expected because it possesses the two following unique advantages that the other two models do not have:

1) CP-ViT can provide superior accuracy with or without finetuning, while the other two models might be greatly dependent on finetuning after pruning. This greatly enhances CP-ViTs general applicability for practical deployment.

2) P-ViT prunes statically on importance scores, which cannot reserve the informative values in the input and only notice the sparsity in one layer, making it hard to gain better accuracy. Both

Table 1: Pruning results of models in 5.3

Accuracy	Model	0-Base	0.1	0.2	0.3	0.4
Top 1	P-ViT	98.00%	97.40%	96.98%	94.15%	92.28%
	AP-ViT		97.98%	**97.75%**	96.97%	95.21%
	CP-ViT		**97.99%**	97.73%	**97.36%**	**96.12%**
Top 5	P-ViT	99.98%	99.98%	99.96%	99.92%	99.89%
	AP-ViT		99.98%	99.98%	99.97%	99.94%
	CP-ViT		**99.98%**	**99.98%**	**99.98%**	**99.96%**
Accuracy	Model	0.5	0.6	0.7	0.8	0.9
Top 1	P-ViT	89.44%	85.02%	74.33%	57.90%	28.66%
	AP-ViT	92.89%	88.57%	79.85%	61.28%	33.50%
	CP-ViT	**94.47%**	**90.88%**	**80.34%**	**63.16%**	**36.20%**
Top 5	P-ViT	99.75%	99.08%	98.10%	93.64%	79.52%
	AP-ViT	99.87%	99.65%	98.83%	95.42%	84.40%
	CP-ViT	**99.94%**	**99.79%**	**99.71%**	**96.59%**	**89.43%**

Table 2: Map of pruning ratio vs threshold in AP-ViT

Pruning Ratio	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Threshold	0.525	0.490	0.467	0.451	0.439	0.431	0.419	0.405	0.385

AP-ViT and CP-ViT determine the pruning ratio for each layer dynamically, but their ideas are different. CP-ViT uses the layer-aware pruning ration adjustment technique, and AP-ViT uses the reward-penalization mechanism to encourage great pruning. The former pays more attention to the layer-wise interdependency in one model while the latter focuses more on model-wise information. From the results of the experiements, we might conclude that the layer-wise interdependency is more important to well-pruning instead of model-wise information.

6 How does BN help optimization?

6.1 Background

Batch Normalization (BatchNorm, BN) controls the mean and variance of layer inputs in a neural network by adding additional layers, ensuring that the input distribution of each layer remains consistent. In practical applications, BatchNorm accelerates model convergence, reduces sensitivity to learning rates, and significantly improves model training effectiveness. The researcher who first proposed BN believed that its advantages lie in addressing the Interval Covariate Shift (ICS) problem. However, in practice, there is no direct evidence to suggest a direct correlation between the outstanding performance of BN and ICS.

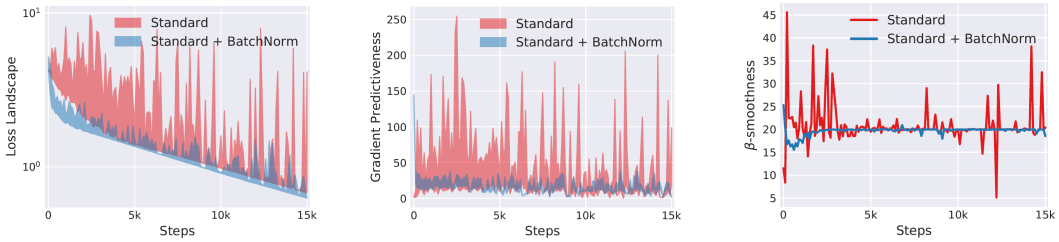
In the article *How Does Batch Normalization Help Optimization?*[12], the author suggests that the role of BN in alleviating the ICS problem is minimal. Its true impact lies in making the landscape of the optimization process smoother, resulting in more stable gradient updates and accelerated model convergence. The addition of BN layers improves the Lipschitz property of the gradients,

$$|f(x_1) - f(x_2)| \leq L||x_1 - x_2||, \quad \forall\, x_1 \text{ and } x_2, \tag{5}$$

which indicates that the variation in model output is consistently constrained within a constant coefficient range, enhancing the stability and predictability of the gradients, thereby improving the effectiveness of deep learning training based on gradient descent.

In Project 2, we have already computed the gradient of the loss at the training step and measure how the loss changes, which refers to the first experiment done in the article (Figure 19a). Therefore, to further investigate the effects of introducing BN to model training, we will replicate the two other experiments on the optimization landscape as conducted in the article using the VGG-A model (Figure 19b, 19c). Since the purpose of the experiments is solely to compare the stability of the

model before and after adding BN layers, we do not include other strategies such as learning rate decay, L2 regularization, or Dropout in the model.



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 19: Experiments done in the article

6.2 Impact of BN on Accuracy

Before replicating the experiments, we first examine the differences in training speed and effectiveness between the standard VGG-A model with or without BatchNorm. To control the experimental conditions and save computation time, we use the basic SGD optimizer and trained the models for 20 epochs. The parameters used in the model are listed in Table 3.

Table 3: Parameters used in 6.2

Optimizer	Loss	Epoch	Batch Size	Learning Rate
SGD	CrossEntropy	20	128	0.1

The result (Figure 20) indicates that after adding BN layers, the model exhibits faster improvement in accuracy on both the training and testing sets. The final performance is better as well (standard VGG accuracy: 79.5%, VGG+BN accuracy: 82.8%), suggesting that BN can accelerate model convergence and outperform the VGG network without BN in terms of generalization.

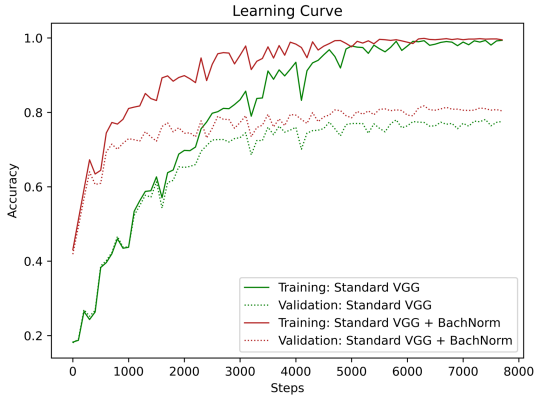


Figure 20: Standard VGG vs VGG+BN

In the following replication of the experiments described in the paper, we carefully selected a set of learning rates: [0.05, 0.075, 0.1, 0.125, 0.15]. These learning rates are moderate in magnitude, ensuring that the model will not converge too slowly (too small learning rates) and gradient explosion will not occur (too large learning rates). The parameters used in the model are listed in Table 4.

Gap equivalent to 30 indicates that the landscape is computed every 30 steps in order to save computation time and to beautify the visualization of the result images.

Table 4: Parameters used in 6.3, 6.4

Optimizer	Loss	Epoch	Batch Size	Learning Rate	Gap
SGD	CrossEntropy	20	128	0.05, 0.075, 0.1, 0.125, 0.15	30

6.3 Gradient Predictiveness

We use the l_2-norm of the difference between gradients before and after the gradient update as the gradient distance landscape. This measure reflects the predictiveness of the gradients.

$$\text{grad\_dist}^{(t+1)} = ||\text{grad}^{(t+1)} - \text{grad}^{(t)}||_2, \quad t = 1, 2, ... \tag{6}$$

The result (Figure 21) shows that the VGG network without BN exhibits a larger fluctuation range in gradient prediction errors. However, there are some differences between Figure 2 in the article and the results obtained. In the VGG+BN model, the mean gradient prediction error is larger than that of the standard VGG model during the initial training stage, and the fluctuation range is also larger. However, in the later stages of training, the gradient prediction error of the VGG+BN model becomes significantly more stable, with a lower mean error compared to the standard VGG model.

This observation partly illustrates the mechanism by which BN accelerates model convergence. During the initial training stage, a larger fluctuation range in gradients is advantageous for the model to escape local optima and explore paths leading to global optima. Therefore, BN has less constraint on the stability of gradients during this stage. As the training progresses and the model gradually converges, the focus of the model shifts towards the stability of gradient updates. Consequently, the smoothing effect introduced by BN becomes more pronounced.

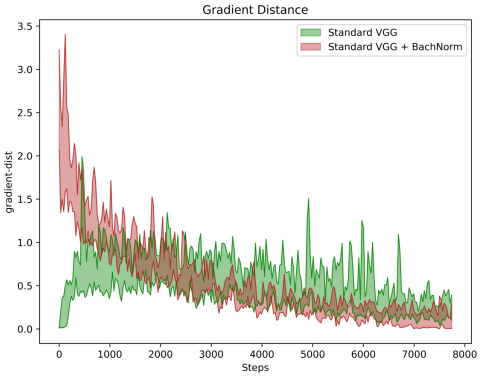


Figure 21: Gradient distance landscape

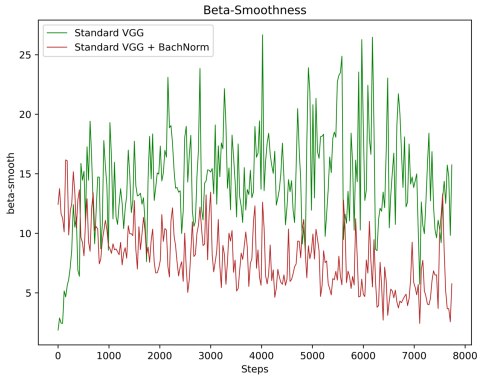


Figure 22: Effective β-smoothness

6.4 Effective β-Smoothness

Here we compare the effective β-smoothness of the VGG-A model with and without the addition of BN layers, where a smaller β value indicates a smoother landscape.

Recall that f is β-smooth if its gradient is β-Lipschitz, which is

$$\|\nabla f(x_1) - \nabla f(x_2)\| \leq \beta \|x_1 - x_2\|. \tag{7}$$

In the SGD optimization process of neural networks, the update of the weights w from time step t to $t+1$ is given by

$$w_{t+1} = w_t - lr * \nabla f(w_t). \tag{8}$$

By substituting the above equation into the β-Lipschitz formula, we have

$$\|\nabla f(w_{t+1}) - \nabla f(w_t)\| \leq \beta \|w_{t+1} - w_t\| = lr * \beta \|\nabla f(w_t)\|. \tag{9}$$

Thus we can derive β as

$$\beta_t = \max \frac{\|\nabla f(w_{t+1}) - \nabla f(w_t)\|}{\|w_{t+1} - w_t\|} = \max \frac{\|\nabla f(w_{t+1}) - \nabla f(w_t)\|}{lr * \|\nabla f(w_t)\|}. \tag{10}$$

The result (Figure 22) shows that during the initial training stage, the model with BN layers has a higher β value compared to the standard VGG model. However, for the training process thereafter, the model with BN layers exhibits a smaller β value, confirming the smoothing effect of BN. This difference in smoothness between the two stages aligns with the hypothesis stated in section 6.3 regarding the mechanism of BN accelerating model convergence.

7 DessiLBI

7.1 Background

Nowadays the great success of deep learning depends on the over-parameterization, which smooths the optimization landscape and helps escape local optimal at nearly no cost of generalization ability. However, the over-parameterization still brings difficulties to training deep neural networks because of a large quantity of parameters.

As mentioned in our lecture, techniques such as pruning and distillation are developed to try to deal with this problem, but they are expensive in fully training a dense network. Therefore, instead of using these backward selection methods, we can use Deep structurally splitting Linearized Bregman Iteration (DessiLBI) [13] to conduct a forward selection that can get the trained over-parameterized model and sparse structure simultaneously.

The idea of DessiLBI is relatively intuitive: performing gradient updates in the gradient space, and then mapping back to the original parameter space for parameter updates (mirrored gradient strategy). Based on the results obtained from Γ, truncation can be applied to search for Winning Ticket subnetworks, leading to a reduction in network parameters.

The paper has already implemented updates under weight decay and momentum methods. In this section, we will modify the update mode by combining DessiLBI with other optimization algorithm such as Adam to find the sparse structure. Additionally, we will also introduce an update path for Γ, since Γ indicates the important sparse structures for DessiLBI.

The key to the update progress lies in calculating the gradients of Γ and W separately.

$$\begin{aligned} \dot{W}_t/\kappa &= -\left(\nabla_W L\left(W_t\right) + \left(W_t - \Gamma_t\right)/\mu\right) \\ \dot{V}_t &= -\left(\Gamma_t - W_t\right)/\mu \\ V_t &\in \partial\bar{\Omega}\left(\Gamma_{t+1}\right) \end{aligned} \tag{11}$$

In which $L(W_t)$ is the loss function without the term Γ_t used in training, e.g. cross entropy loss.

7.2 LeNet on MNIST

In this section, we use SGD optimizer to train LeNet on MNIST dataset and compare the results of LeNet with and without DessiLBI. The parameters used in the model are listed in Table 5.

Table 5: Parameters used in 7.2

Optimizer	Loss	Epoch	Batch Size	Learning Rate
SGD	NLL	32	128	1e-2

The test accuracy of LeNet without DessiLBI is 98.79% and that of LeNet with DessiLBI is 98.61%, which is not a big difference. But the weights of third convolutional layer of the two models greatly differ from each other.

As Figure 23a and Figure 23b showed, most regions in standard SGD model are non-black, indicating non-zero weights, while most regions in SGD + DessiLBI model are black, indicating zero weight values. Interestingly, the test accuracies of both models are nearly the same, which suggests that the problem of over-parameterization does exist and DessiLBI can make the model more sparse.

Then we prune the parameters on the third convolutional layer at different pruning ratios. The test accuracy changes are listed in Table 6. After pruning, the test accuracy decreases slightly, and thus the pruning process is successsful and safe. It turns out that the original model might be very sparse and the degree of over-parameterization might be about 60%.

(a) Standard SGD (b) SGD + DessiLBI (c) Adam + DessiLBI

Figure 23: Conv3 weights visulization

Table 6: Pruning results of DessiLBI

Pruning Ratio	0	0.1	0.2	0.4	0.6	0.8
Top 1 Accuracy	98.61%	98.61%	98.61%	98.58%	98.55%	98.30%
Top 5 Accuracy	99.98%	99.98%	99.98%	99.98%	99.97%	99.92%

7.3 Combination of DessiLBI and Adam

In this section, we train two models on MNIST: SGD + DessiLBI and Adam + DessiLBI. The parameters used in the model are listed in Table 7.

Table 7: Parameters used in 7.3

Optimizer	Loss	Epoch	Batch Size	Learning Rate
SGD, Adam	CrossEntropy	32	128	1e-2

The test accuracies of SGD and Adam are respectively 98.61% and 98.73%. As Figure 24 and Figure 25 shows, the convergence rate of Adam is faster than that of SGD, though the difference is not very significant. The main difference between the two models is the magnitude of coefficients. Specifically, Adam accelerates the penalization rate of the solution and applies a much stronger penalty to the weights compared to SGD. This is likely because in the later stages of training, most solutions become sparse, which means most feature magnitudes become very small. Adaptive gradient strategy can adjust the learning rates for different solutions, and thus results in better optimization performance for sparse problems.

As Figure 26 and Figure 27 shows, Adam is much more stable during training compared to SGD, possibly due to its inherent advantage in handling sparse problems. Interestingly, in the convolutional layers, SGD + DessiLBI is sparse, while Adam + DessLBI selects almost all parameters. In the fully connected layers, the Adam model is sparse compared to convolutional layers, but it still selects more parameters than the SGD model.

Visualizing the third convolutional layer of these two models (Figure 23b, Figure 23c), we observe that SGD + DessiLBI is much sparser than Adam + DessiLBI. Furthermore, even in the experiments with other models in Section 7.4, Adam + DessiLBI completely loses the ability to select parameters in the convolutional layers. This indicates that increasing the penalty strength for parameters does not necessarily result in a sparser solution. The higher test accuracy of Adam + DessiLBI compared to SGD + DessiLBI might also be attributed to more non-zero weight convolutional kernels.

7.4 DessiLBI on VGG-A and ResNet-56

We also train the VGG-A (Figure 28) and ResNet-56 (Figure 29) with Adam and DessiLBI on CIFAR-10 dataset. Though the convergence rates of both models are fast, the training processes are very fluctuated and the test accuracies are not satisfying. The sparsity of Adam is consistent to the conclusion in the section above.

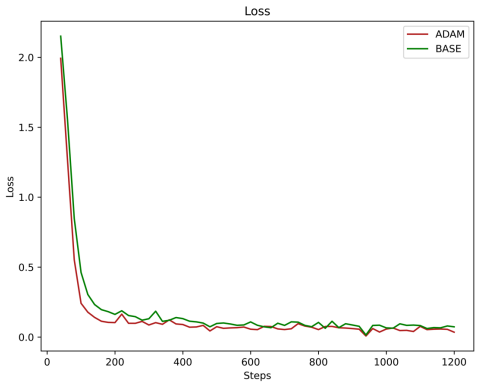


Figure 24: Loss curve

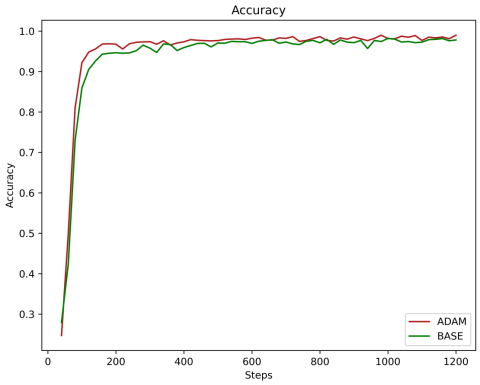


Figure 25: Accuracy curve

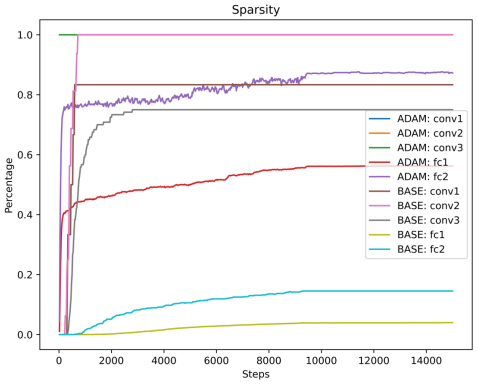


Figure 26: Sparsity curve

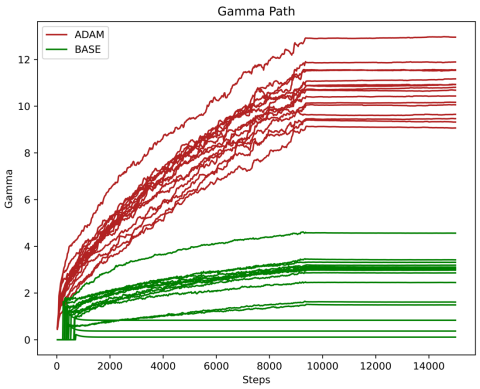


Figure 27: Γ path

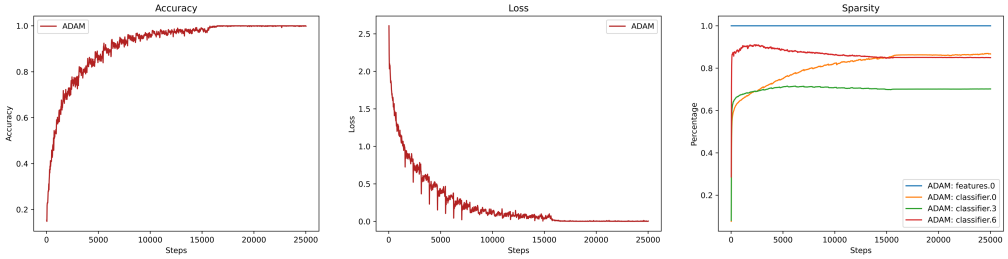


Figure 28: VGG-A + Adam + DessiLBI

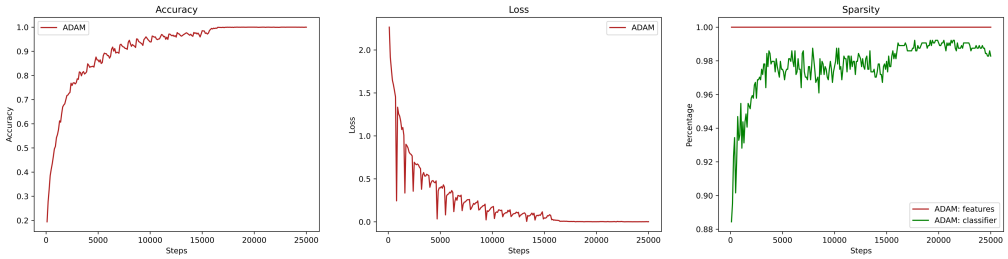


Figure 29: ResNet-56 + Adam + DessiLBI

References

[1] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one, 2022.

[2] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020.

[3] Matthias Bauer, Emilien Dupont, Andy Brock, Dan Rosenbaum, Jonathan Richard Schwarz, and Hyunjik Kim. Spatial functa: Scaling functa to imagenet classification and generation, 2023.

[4] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning, 2021.

[5] `https://github.com/Lucky666123/cv-automatic-pruning-transformer`.

[6] Zhuoran Song, Yihong Xu, Zhezhi He, Li Jiang, Naifeng Jing, and Xiaoyao Liang. Cp-vit: Cascade vision transformer pruning via progressive sparsity prediction, 2022.

[7] `https://storage.cloud.google.com/vit_models/imagenet21k/ViT-B_16.npz`.

[8] `https://github.com/HzcIrving/DLRL-PlayGround/tree/main`.

[9] `https://github.com/Cydia2018/ViT-cifar10-pruning`.

[10] `https://github.com/ok858ok/CP-ViT`.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[12] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2018.

[13] Yanwei Fu, Chen Liu, Donghao Li, Xinwei Sun, Jinshan Zeng, and Yuan Yao. Dessilbi: Exploring structural sparsity of deep networks via differential inclusion paths, 2020.