NN Final Project Report

Wanyi Wang 20307110462 Siyu Wang 19307110429

Abstract—The first task focuses on exploring the concept of Functa. We utilize SIREN model to train an implicit network and evaluate its fitting ability for different model sizes and image sizes by calculating the peak signal-to-noise ratio. The shared model parameters across images are trained using the Modulated SIREN model, and a qualitative and quantitative comparison is provided with single-image Functa. We use CIFAR-10 classification as our functa downstream tasks and we get 43.54% accuracy on test set. The second task addresses the challenge of reducing the parameter count of Transformer models while maintaining performance. We explore sparse regularization techniques for pruning a Vision Transformer (ViT) model trained on CIFAR10. Evaluation involves measuring pruning ratio from 0.2-0.4 with the interval 0.1. The third task revolves around gradient predictiveness, "effective" β-smoothness analysis. We explore the impact of Batch Normalization (BN) on gradient stability and plots the "effective" β-smoothness of standard and BN networks throughout the training process. Furthermore, a new optimizing solver called DessiLBI is introduced and we present experimental results on LeNet for MNIST and extends the analysis to CIFAR-10.

Index Terms—Functa Pruning Transformer DessiLBI

1 TASK1

1.1 Baisc Functa

1.1.1 Model

We use SIREN (Sitzmann et al., 2020b) [8] as the base architecture for INRs since it is known to efficiently represent a wide range of data modalities. The easy approach for representing functa is to take the parameter vector of the SIREN. INRs are functions $f_\theta : \mathcal{X} \to \mathcal{F}$ mapping coordinates $\mathbf{x} \in \mathcal{X}$ (e.g. pixel locations) to features $\mathbf{f} \in \mathcal{F}$ (e.g. RGB values) with parameters θ. Given a data point as a collection of coordinates $\{\mathbf{x}_i\}_{i\in\mathcal{I}}$ and features $\{\mathbf{f}_i\}_{i\in\mathcal{I}}$ (where $\mathcal{I}$ is an index set corresponding to e.g. all pixel locations in an image), INRs are fitted by minimizing mean squared error over all coordinate locations:

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

Hence each f_θ corresponds to e.g. a single image. Typically, f_θ is parameterized by a feedforward neural network (MLP) with positional encodings or sinusoidal activation functions.

1.1.2 Experiments

We reconstructed some images from CIFAR-10 dataset. We tried different training steps and image sizes through resize transformation. We calculated the peak signal-to-noise ratio(PSNR) to evaluate the fitting ability of the trained model. Some results are shown below.

steps	100	200	400	800	2000
psnr	15.68	22.02	29.06	37.99	55.47

TABLE 1: Psnrs at different training steps. Results obtained from image size=(32,32).

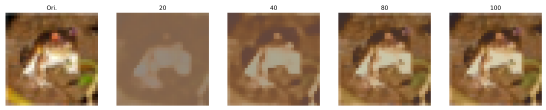

Fig. 1: Reconstructed figures with different training steps(origin, 20, 40, 80, 100)

Fig. 2: Reconstructed figures with different training steps(origin, 200, 400, 800, 2000)

size	32	64	96	128
psnr(s=800)	37.99	38.46	34.38	36.45
psnr(s=2000)	55.47	50.12	44.34	42.96

TABLE 2: Psnrs at different image sizes. Results obtained from training steps=800 and 2000.

1.1.3 Conclusions

For 32×32 images, training 200 steps can reconstruct images well and when training 2000 steps, the reconstructed graph is almost indistinguishable from the original. As the number of training steps increases, the difference between the reconstruction image and the original image becomes smaller, and the PSNR becomes larger. Resizing the same image to a larger size reduces the reconstruction performance.

In conclusion, our experiment successfully implemented the Functa approach for image restoration and downstream tasks using the SIREN model. We verified the fit effect of the model for different sizes of images and model architectures, and visually demonstrated the accuracy of image restoration.

1.2 Share the model parameters across images

1.2.1 Model

Fitting INRs can be slow. Based on SIREN, Dupont et al. proposed the concept of Functa [2] in 2022. They used SIREN to implicitly fit the discretized data and directly used model parameters as data for secondary training. Dupont et al. mainly focused on optimizing Functa itself, using modulation method to share model parameters between data points, and using meta learning method to accelerate large-scale sample fitting.

1.2.2 Experiments

We trained about 3×10^4 iterations in the outer loop where inner loop gradient steps is 3 on CIFAR-10 training set. Then train 5000 iterations in the inner loop. We get PSNR around 7 in the outer loop training, PSNR around 27 in the inner loop training on training set and PSNR around 27 in the inner loop training on testset . Some results are shown below.

Fig. 3: Last 4 PSNR reconstructed images on training set

Fig. 4: Top 4 PSNR reconstructed images on training set

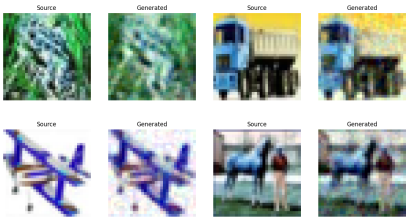

Fig. 5: Last 4 PSNR reconstructed images on test set

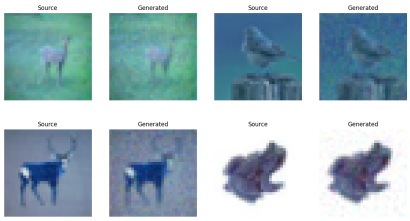

Fig. 6: Top 4 PSNR reconstructed images on test set

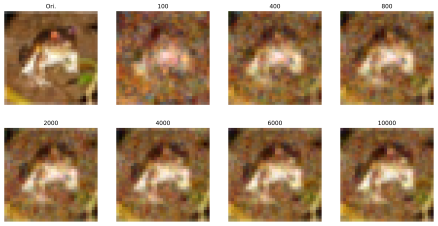

Fig. 7: Reconstructed figures with different inner steps training

1.2.3 Conclusions

Compared with single-image functa, the reconstructed image is worse, which may be that the training of our model's outer loop is not enough. From the perspective of increasing the number of training steps to make the reconstruction image clearer, the noise of the reconstructed image of functa with shared parameters is much more than that of a single image functa. As the number of training steps increases, a single image functa improves PSNR by making the RGB value closer to the original RGB value, and functa with shared parameters improves PSNR by reducing noise.

In conclusion, by sharing model parameters across images using the Modulated SIREN model and utilizing meta-learning for training shared parameters, we completed the task. We get worse performance than the single image functa maybe due to the insufficient training for outer loop. The shared parameters captured essential dataset features, while the sample-specific modulation parameters allowed for fine-tuning and adaptation to individual samples.

1.3 Functa for downstream tasks

1.3.1 Model

After completing the batch training of the sample SIREN model, we get latent vectors for training set and test set. Each latent vector has length 512. We use two MLP(Multilayer Perceptron) model. The specific network structures can be seen in Appendix 2.1.

1.3.2 Experiments

Having around 30000 iterations for outer loop and 200 iterations for inner loop, we get latent vectors for training set and test set. Set learning rate as 1e-3, batch size as 128, optimizer as Adam. Using the MLP1 model, we get best

accuracy 43.77% on test set at epoch 400. Using the MLP2 model, we get best accuracy 43.54% on test set at epoch 300.

epoch	100	200	300	400	500	600
test_acc	0.4109	0.4230	0.4413	0.4377	0.4346	0.4376
train_acc	0.4870	0.5730	0.6171	0.6485	0.6636	0.6828

TABLE 3: Accuracy using MLP1

epoch	100	200	300	400	500	600
test_acc	0.4220	0.4219	0.4354	0.4324	0.4313	0.4307
train_acc	0.4994	0.5600	0.5985	0.6297	0.6503	0.6628

TABLE 4: Accuracy using MLP2

epoch	100	200	300	400	500
test_acc	0.4156	0.4287	0.4251	0.4273	0.4321
train_acc	0.4579	0.5017	0.5331	0.5369	0.5529

TABLE 5: Accuracy using MLP1 with weight decay=1e-4

1.3.3 Conclusions

The performance is worse than the model trained based on original images. The reasons for that can be: 1) Loss of Information: The process of obtaining latent vectors from modulated SIREN involves compressing the original images into a higher-dimensional representation. This compression may result in the loss of important information that is necessary for accurate classification. 2) Insufficient Pre-training: Although the latent vectors have higher dimensionality, the insufficient training for generating the latent vectors may cause information loss. 3) Model selection: The model we used to train on latent vectors is worse than the model we used to train on origin images.

2 Task2

2.1 Introduction

In recent years, Transformer has shown remarkable results in various applications such as Natural Language Processing (NLP) and computer vision. However, such results rely on millions or even billions of parameters, which limits its deployment in limited storage platforms. This project aims to effectively prune the transformer model via sparse regularization while preserving its performance on the CIFAR10 dataset.

2.1.1 CIFAR10 Dataset

We use CIFAR10 as the dataset. The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. CIFAR-10 consists of 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1,000 randomly selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5,000 images from each class.

2.1.2 Transformer

The transformer is a type of neural network architecture introduced in [10]. The transformer is composed of an encoder and a decoder,each made up of multiple identical layers. The encoder consists of encoding layers that process the input iteratively one layer after another, while the decoder consists of decoding layers that do the same thing to the encoder's output.Each encoder layer is encoding information about the relevance among parts of the inputs. A core concept in the Transformer is the "attention mechanism", which weighs the influence of different input parts in the processing of each output part. In other words, it allows the model to focus on different parts of the input sequence when producing each element of the output sequence.To effectively learn multiple types of relevance, the transformer incorporates multi-head attention, which is defined as:

$$\text{MultiheadedAttention}(Q, K, V) =$$

$$\text{Concat}(\text{Attention}(QW_i^Q, KW_i^K, VW_i^V))W^O$$

Where $\text{Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$ and the matrics W_i^Q, W_i^K, W_i^V are "projection matrices" owned by individual attention head i, and WO is a final projection matrix owned by the whole multi-headed attention head. The decoder has these components as well, but also includes an additional attention mechanism that focuses on the encoder's output.

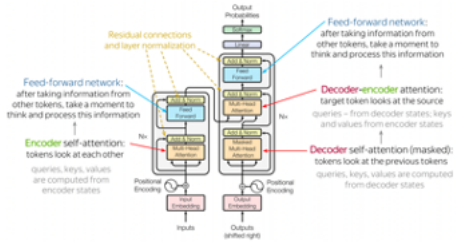

Fig. 8: Transformer architecture

2.1.3 Sparse Regularization

Sparse regularization is a technique that encourages models to use fewer features for prediction. This is achieved by adding a penalty term to the loss function that increases as the number of features used by the model increases. The most common form of sparse regularization is L1 regularization, also known as Lasso regression. In Lasso, the penalty term is the sum of the absolute values of the model parameters. This has the effect of pushing many parameters to exactly zero, effectively excluding the corresponding features from the model. However the Lasso only applies to linear models. [1] introduce LassoNet, a neural network framework with global feature selection.

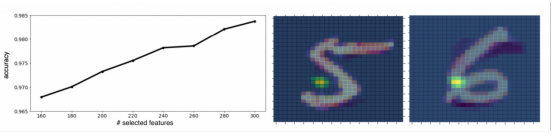

Fig. 9: Demonstrating LassoNet on the MNIST dataset

Sparse Regularization also has been applied to network pruning. For example, [5] quantitatively measured the accuracy-sparsity relationship with different grain sizes.Besides, the [3] considers the structural sparsity. Recently, it has been applied to prune the transformer, by exploiting different candidates for pruning.For example, the [6] introduced a new technique to prune parameters in a Deep neural network and show empirical results from its application on text classification models based on transformer architecture.They applied the A* search algorithm to prune heads (termed as A* Pruning) in downstream models.

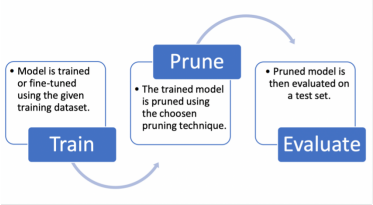

Fig. 10: Pruning pipeline

2.2 ViT Model

2.2.1 Model overview

Our base model for pruning is the Vision Transformer (ViT) model [1]. The key idea of the Vision Transformer is to treat an image as a sequence of patches, in the same way that a text is treated as a sequence of words in the original Transformer model. The image is divided into a grid of small patches (e.g., 16x16 pixels), and each patch is flattened and linearly embedded to create a sequence of vectors. These vectors are then fed into a standard Transformer encoder. Unlike convolutional neural networks (CNNs), which make use of local and translational equivariant features, the Vision Transformer captures global dependencies between patches in an image, as the self-attention mechanism in the Transformer allows each patch to attend to all other patches.

2.2.2 ImageNet-21k Dataset

ViT is pre-trained on ImageNet-21k (14 million images, 21,843 classes) at a resolution of 224 $\times$ 224. ImageNet-21k is a large-scale dataset used for training machine learning and artificial intelligence models. It is a subset of the larger ImageNet dataset, which was designed for use in visual object recognition research. The ImageNet-21k dataset is particularly useful for pre-training models that will later be fine-tuned on a smaller dataset. This is because the large number of categories provides a broad base of knowledge that can help the model to better understand and generalize from the smaller dataset it is fine-tuned on.

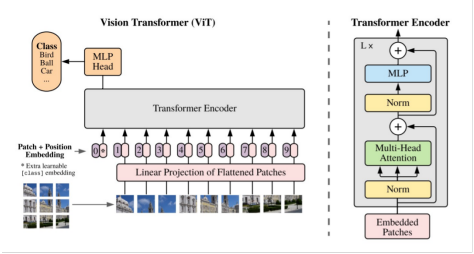

Fig. 11: ViT Model Overview :Split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence.

2.3 Method: CP-ViT

In [9], authors proposed a cascade pruning framework named CP-ViT by predicting sparsity in ViT models progressively and dynamically.

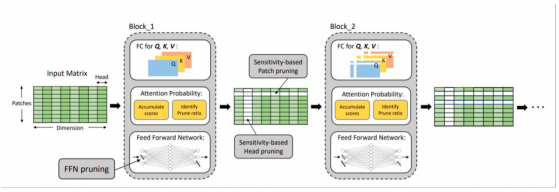

Fig. 12: The overall framework of the proposed CP-ViT

This paper proposed patch- and head-based cascade pruning that can dynamically locate informative patches and heads (collectively called as PH-region), so as to reduce the computation complexity with minimized accuracy loss. Cascade pruning means once an uninformative PH-region is pruned, the involved computations in all following layers will be skipped.And the informative PH-regions appear in previous layers may disappear in the successive layer. This is because different layers extract different features and the later layer may ignore informative features extracted by previous layers.Therefore,directly executing the cascade pruning only considering uninformative PH-regions in the current layer may destroy the key information captured by previous layers and consequently lead to unacceptable accuracy loss.So we define the cumulative score to preserve informative PH-regions across the whole ViT model for better accuracy.

2.3.1 Informative PH-Regions in ViT

In this section, we utilize the attention_probability to prove the existence of sparsity in ViT models. We first calculate the average attention_probability value of each patch and then divide PH-regions into three segments according to their average magnitudes. Segment 1 contains the smallest PH-regions and segment 3 contains the largest PH-regions.We select an input image and the first layer of ViT-B 16/224 model as an example to visualize three segments with different luminance.We then prune each segment one by one and see the Top-1 accuracy results.It is obvious that segment 3 domains the informative PH-regions, which will

lead to significant accuracy loss if we prune the values in it. Alternatively, segments 1 and 2 are less informative, which have a smaller impact on accuracy compared to segment 3. The above observations indicate that PH-regions with different magnitude have different impacts on the final accuracy.Thus PH-regions with different magnitude can be viewed as PH-regions with different informativeness, which implies applying structured pruning to uninformative PH-regions can accelerate ViT training and inference while maintaining high accuracy.

2.3.2 Progressive Sparsity Prediction

In this section,we define the cumulative score serving progressive sparsity prediction to distinguish informative PH-regions. To efficiently calculate the cumulative score, we also use the maximum value in the attention_possibility. The total informativeness of patch p0 in head h can be defined as:

$$\alpha \sum_i A_h[p_0, i] + \beta \sum_j A_h[j, p_0]$$

Where α and β indicate the difference between the impact of $p0$ on other patches and the impact of other patches on $p0$. We can obtain the total informativeness of patch p0 to the whole layer as:

$$\sum_h (\alpha i A_h[p_0, i] + \beta \sum_j A_h[j, p_0])$$

To define the informativeness of head h, the formula is:

$$\sum_i \sum_j A_h[i, j]$$

After computing the average informativeness of each patch in the 4th layer, we obtain its distribution as Figure 7. The distribution shows that the informativeness of different patches varies greatly.

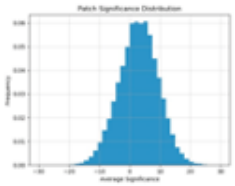

Fig. 13: Patch informativeness distribution of ViT-B 16/224

This sum-based criteria is inefficient and hinders ViT acceleration.It can be simplified by using the maximum value in attention_probability. We obtain the informativeness of the patch by comparing attention_probability and then choosing the maximum one. Besides, head informativeness is the sum of patch informativeness in this head. To minimize accuracy loss, we further define the cumulative scores to represent the informativeness based on the attention_probability of multiple layers rather than a single layer.For each layer, we will accumulate the attention_probability of the current layer and the layers before it so as to obtain the cumulative scores. We sort the cumulative scores and select the smallest $L \times r_{l,p}$ and $H \times r_{l,h}$ scores representing the uninformative patches and heads, where $r_{l,p}$ and $r_{l,h}$ are the pruning ratios of patches and heads respectively.

2.3.3 Layer-Aware Cascade Pruning

According to [11], we can regard the attention range as a guiding role, and leverage it to adjust the pruning ratios for different layers. Consequently, we can precisely control the number of pruned PH-regions in each layer to reduces computations while ensuring accuracy.

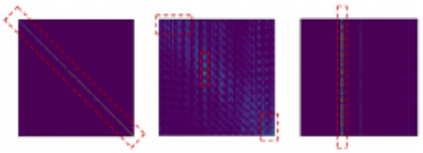

Fig. 14: Visualization of attention_probability

Figure 14 shows three typical distribution of attention probability, brighter area means stronger interdependency between corresponding patches: 1) When the bright area is distributed near the diagonal, it indicates that the attention range of this layer is very short, and only the interdependency of patches that are very close to the current patch is captured by this layer. At this time,we cannot identify the informativeness of each patch, so that the pruning ratio should be reduced; 2) When the bright area is distributed on the vertical line, it indicates that the attention range of this layer is very long, and the interdependency of patches that are far from the current patch can also be captured by this layer. We can easily identify the informative patches by locating those bright vertical lines, and we increase the pruning ratio.

2.4 Experiments

We use ViT-B_16/224 model.
When directly apply CP-ViT to pre-trained models without finetuning:

Pruning ratio	Baseline	0.2	0.3	0.4
Accuracy(%)	98.1	97.8	97.4	96.2

TABLE 6: Apply CP-ViT to pre-trained models without finetuning.

When we finetune the model:

Pruning ratio	Baseline	0.2	0.3	0.4
Accuracy(%)	98.1	98.2	98.0	97.8

TABLE 7: Apply CP-ViT to pre-trained models with finetuning.

3 TASK3

3.1 Gradient Predictiveness

Calculate the gradient of the loss function for both the i-th and (i-1)-th steps, and try multiple experiments with different learning rates. In the implementation process, I recorded the gradients of the last layer, trained the model with learning rates of [1e-3, 2e-3, 1e-4, 5e-4] respectively. Similar to the implementation of losslandscape, I selected the maximum and minimum gradient distances from two

models and recorded them, visualizing these two curves and the interval. The final result is as follows:

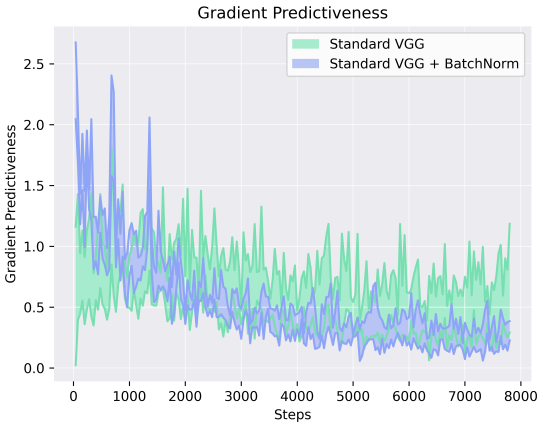

Fig. 15: Gradient Predictiveness

It can be seen that after adding the BN layer, not only the gradient distance becomes smaller, but also the variance of the gradient distance under different learning rates becomes smaller. The smaller norm can show that the predictability of the model for the next step is improved, the convergence speed of the model is greatly improved, and the volatility is reduced. The smaller variance can also show that the convergence of the model is more stable, and after adding the BN layer, the model's dependence on the learning rate value is greatly reduced.

In addition, it can be observed that when the model is first trained, the loss gradient of the model with the BN layer is higher than that of the model without the BN layer, which shows that the model with the BN layer uses a larger The pace of progress is conducive to faster convergence, and in the later stage, it is cautiously updated, which is conducive to the convergence of model parameters and reduces volatility.

3.2 "Effective" β-smoothness

The calculation of β-smoothness utilizes the maximum gradient distance Euclidean norm divided by the displacement distance:

$$\beta = \frac{\|\Delta l(w_1) - \Delta l(w_2)\|_2}{\|w_1 - w_2\|_2}$$

The above equation signifies the impact of a small change in weight on the gradient of loss, which measures the smoothness of the first-order gradient.

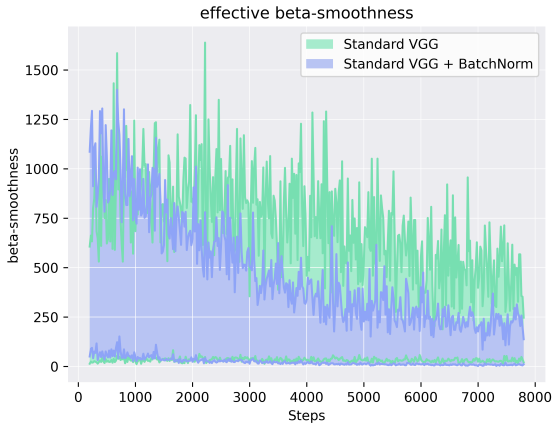

Fig. 16: "Effective" β-smoothness

It can be clearly seen that the β-smoothness of the model with BN is smaller than that of the model without BN, indicating that its first-order gradient is much smoother than that of the model without BN, and it is less sensitive to the learning rate, improving the adaptability of the model.

3.3 DessiLBI

For MNIST and Cifar-10, the default hyper-parameters of DessiLBI are $\kappa = 1$, $\mu = 10$, α_k which is initially set as 0.01, decreased by 1/10 every 20 epochs. On MNIST and Cifar-10, batch size is 128 and epoch is 20. The models for MNIST and Cifar-10 are respectively Lenet [4] and VGG-A with BN [7]. The specific network structures can be seen in Appendix 3.1.

3.3.1 *Combine DessiLBI with Adam*

Optimizer	Adam	DessiLBI	DessiLBI+Adam
Accuracy	98.77%	99.02%	98.92%

TABLE 8: Test accuracy at different optimizers on MNIST.

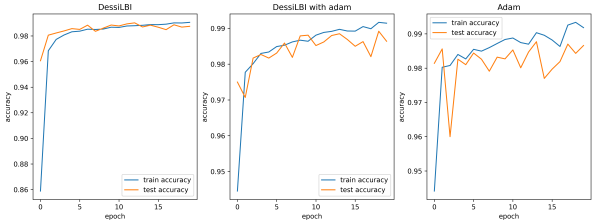

Fig. 17: The training and test accuracy using the three optimizers on MNIST.

Optimizer	Adam	DessiLBI	DessiLBI+Adam
Accuracy	80.30%	80.57%	80.66%

TABLE 9: Test accuracy at different optimizers on CIFAR-10.

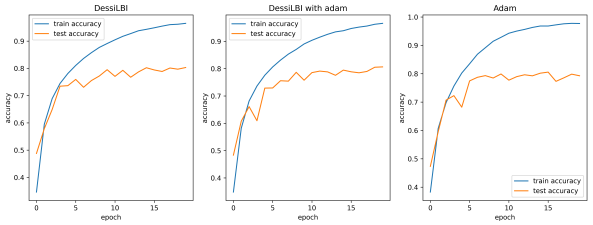

Fig. 18: The training and test accuracy using the three optimizers on CIFAR-10.

The training accuracy cruves for the three optimizers on MNIST using Lenet are distinct while they on CIFAR-10 using VGG-A with BN are similar. For both models, the accuracy achieved using the three optimizers is close.

3.3.2 *Prune the parameters*

In the section, we try to find how sparse the layers in models are by pruning parameters. All experiments use DessiLBI optimizer.

Prune	0%	20%	40%	60%	80%
Accuracy	98.7%	98.68%	98.53%	97.72%	93.42%

TABLE 10: Accuracy at different degrees of pruning. Results obtained from pruning convolutional layer *conv3*.

Prune	*conv1*	*conv2*	*conv3*	*fc1*	*fc2*
Accuracy	89.42%	85.59%	93.42%	98.67%	98.05%

TABLE 11: Accuracy at pruning different layers. Results obtained from pruning 80%.

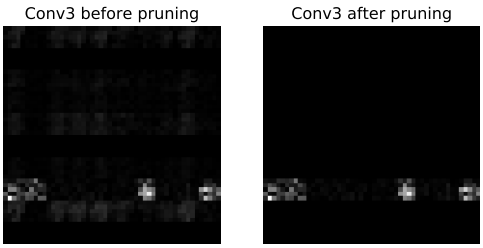

Fig. 19: The third convolution of Lenet on MNIST before and after pruning with 80 percent .

Prune	0%	20%	40%	60%	80%
Accuracy	80.30%	67.52%	34.54%	14.67%	10.36%

TABLE 12: Accuracy at different degrees of pruning. Results obtained from pruning convolutional layer *conv3*.

Prune	*conv1*	*conv2*	*conv3*	*conv4*
Accuracy	10.05%	11.52%	10.36%	58.49%
Prune	*conv5*	*classifier.0*	*classifier.3*	*classifier.6*
Accuracy	72.38%	72.84%	80.34%	80.32%

TABLE 13: Accuracy at pruning different layers. Results obtained from pruning 80%.

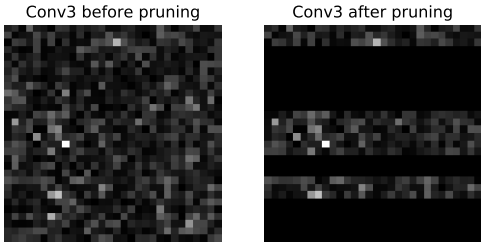

Fig. 20: The third convolution of VGG_BN on CIFAR-10 before and after pruning with 80 percent.

3.4 Conclusions

Lenet is seriously over-parameterized on MNIST, while VGG with BN has not been over-parameterized on CIFAR10. The sparsity is more significant in fully connected (FC) layers than convolutional layers in Lenet.

4 Acknowledgement

References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, and T. Unterthiner. Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[2] E. Dupont, H. Kim, S. Eslami, D. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

[3] 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.

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

[5] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the granularity of sparsity in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 13–20, 2017.

[6] A. Parnami, R. Singh, and T. Joshi. Pruning attention heads of transformer models using a* search: A novel approach to compress big nlp architectures. *arXiv preprint arXiv:2110.15225*, 2021.

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

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

[9] Z. Song, Y. Xu, Z. He, L. Jiang, N. Jing, and X. Liang. Cp-vit: Cascade vision transformer pruning via progressive sparsity prediction. *arXiv preprint arXiv:2203.04570*, 2022.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
[11] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020.

APPENDIX

.1 Basic Functa

.1.1 More experiment details

Dataset CIFAR-10 training dataset.
Hyperparameters Set hidden features as 256, hidden layers as 3, optimizer as Adam with learning rate 1e-4.

.1.2 More experiment results

Fig. 21: Origin size to (32,32)

Fig. 22: Resize to (64,64)

Fig. 23: Resize to (96,96)

Fig. 24: Resize to (128,128)

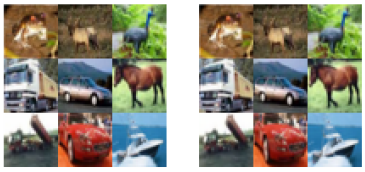

Fig. 25: Combine 9 images of (32,32)

.2 Functa for downstream tasks

.2.1 More model details

One MLP model has one hidden layer.

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(512, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.relu(out)
        out = self.fc3(out)
        out = self.softmax(out)
        return out
```

Fig. 26: MLP Structure 1

Aother MLP model has two hidden layers.

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(512, 256)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(256, 128)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(128, 64)
        self.relu3 = nn.ReLU()
        self.fc4 = nn.Linear(64, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        x = self.relu3(x)
        x = self.fc4(x)
        x = self.softmax(x)
        return x
```

Fig. 27: MLP Structure 2

.3 DessiLBI

.3.1 More model details

1) **Lenet**
The Lenet model has three convolutional layers and two fully connected (FC) layers.

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, padding=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.fc1 = nn.Linear(120, 84)
        self.fc2 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = F.relu(self.conv3(x))
        x = x.view(-1, 120)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Fig. 28: Lenet Structure

2) **VGG-A with BN**
We used VGG-A with BN model.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig. 29: VGG Structure [7]