Project II
CIFAR, BatchNorm, DessiLBI

Shao Yi

2022-05-04

Abstract

First part, we conduct some investigation on network architecture and training process, ultimately our best model achieves accuracy of 96.65%. Next, guided by the BatchNorm paper, we discover its benefits on loss descent, gradient predictiveness and beta smoothness. Finally, we reproduce the results of DessiLBI and combine Adam with it.

1 Train Networks on CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 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 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

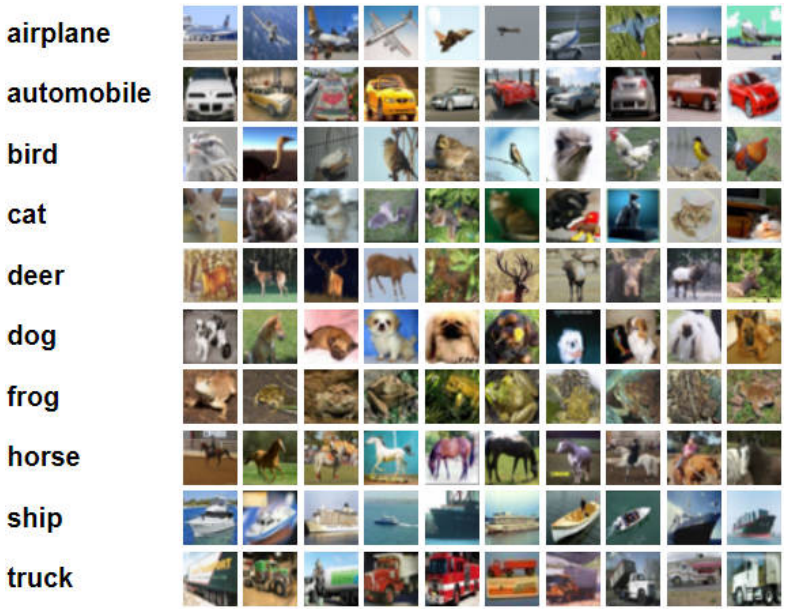

Figure 1: CIFAR-10 Dataset Example

1.1 Network Architecture

This part of the project is to build different neural network architectures and compare their performance on the CIFAR-10 dataset. Networks we tried include: ResNet, PreActResNet, WideResNet, ResNeXt, DenseNet, Dual Path Network, and Deep Layer Aggregation Network. By training these networks with

fixed initial hyper-parameters and same training policy, we will compare the optimal test accuracy of each network. After that, some ablation studies will be conducted to investigate the effect of nets' **depth, width and cardinality**.

1.1.1 Brief Introduction

ResNet is a widely used network architecture for image classification. The **residual block** proposed by He et al. efficiently solves the degradation problem frequently occurred in deep networks, which is an **identity shortcut connection** that skips one or more layers shown in the illustration below. Thanks to this kind of mapping, the network can always backpropagate first-order information from one layer to another no matter how deep it is, so as to address the problem of **vanishing/exploding gradient**.

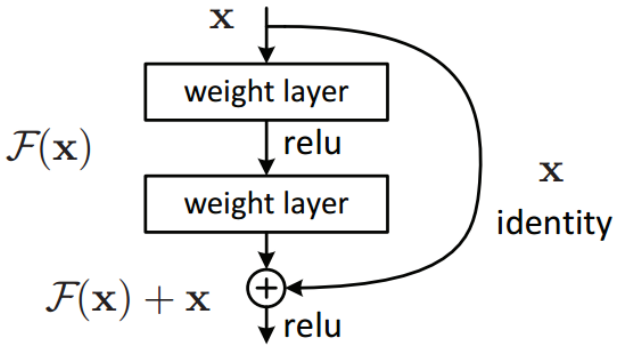

Figure 2: Residual Block

Pre-Activation ResNet is a variant of ResNet that implements **pre-activation** layers, in which case the shortcut connection stays **clean** because ReLU is applied in conjunction with BatchNorm, thus the network no longer violates the identity mapping anymore.

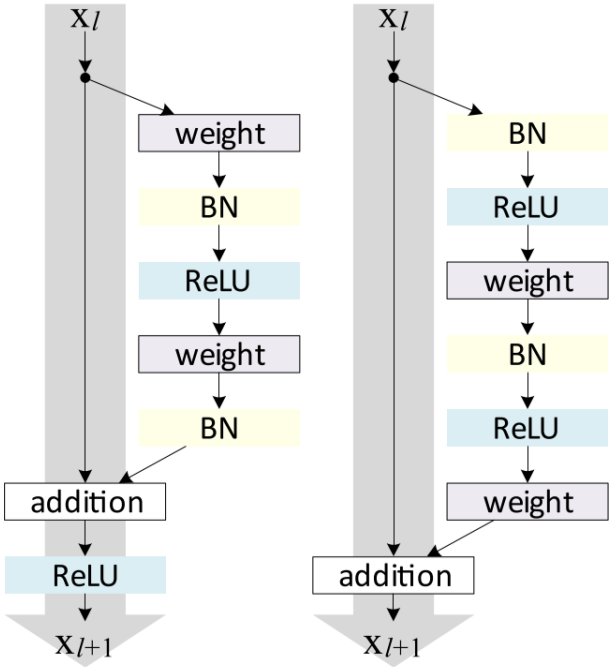

Figure 3: **Left:** Residual Block. **Right:** Pre-Activation Block.

Wide Residual Network proposes that as the gradient flows through the network there is nothing to

force it to go through the residual block weights and thus it can avoid learning during training, so it is possible that there is either only a few blocks that learn useful representations, namely many blocks share very little information with small contribution to the final goal. This problem is formulated as **diminishing feature reuse**. And as widening the residual blocks, **dropout** should be inserted between convolutional layers to **regularize training and prevent overfitting**.

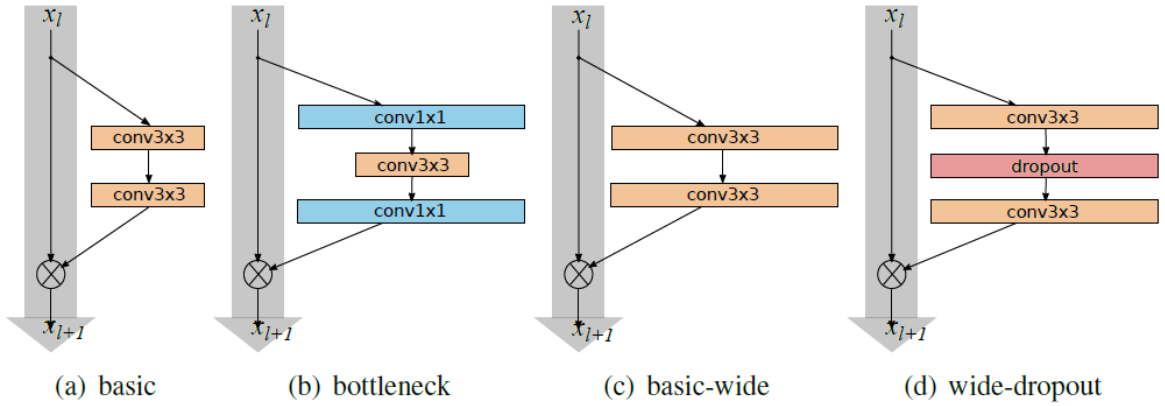

Figure 4: Wide Residual Block

ResNeXt repeats a building block that aggregates a set of transformations with the same topology, indicating that **cardinality** (the size of the set of transformations) is a concrete, measurable dimension that is of central importance, in addition to the dimensions of width and depth.

Experiments have demonstrated that **increasing cardinality** is a more effective way of gaining accuracy than going deeper or wider, especially when depth and width starts to give diminishing returns for existing models. With roughly the same complexity, ResNeXt can achieve higher accuracy than ResNet.

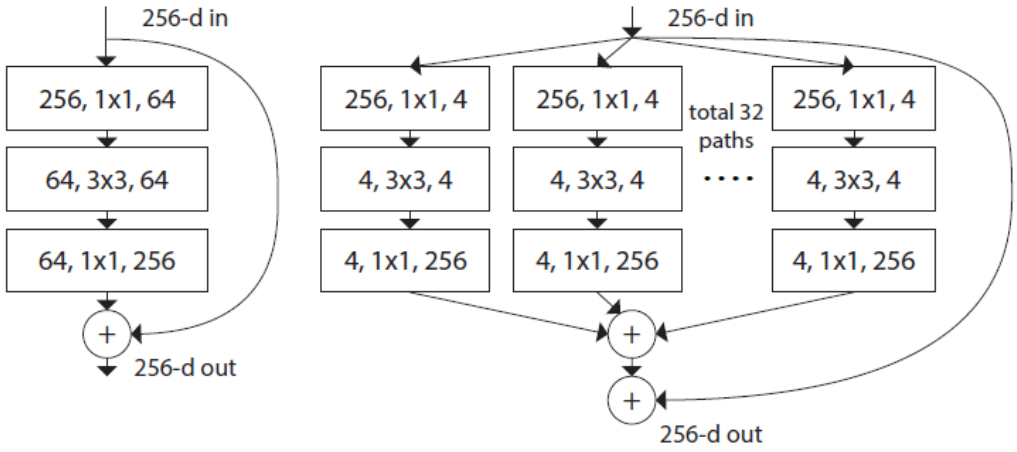

Figure 5: **Left:** Residual Block. **Right:** ResNeXt Block with cardinality = 32.

In **DenseNet**, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. **Concatenation** is used. Since each layer receives a **"collective knowledge"** from all preceding layers, network can be **thin and compact**.

In this design, error signal can be easily propagated to earlier layers more which is is a kind of **implicit deep supervision**, and with **smaller parameter size** comes **more diversified features**.

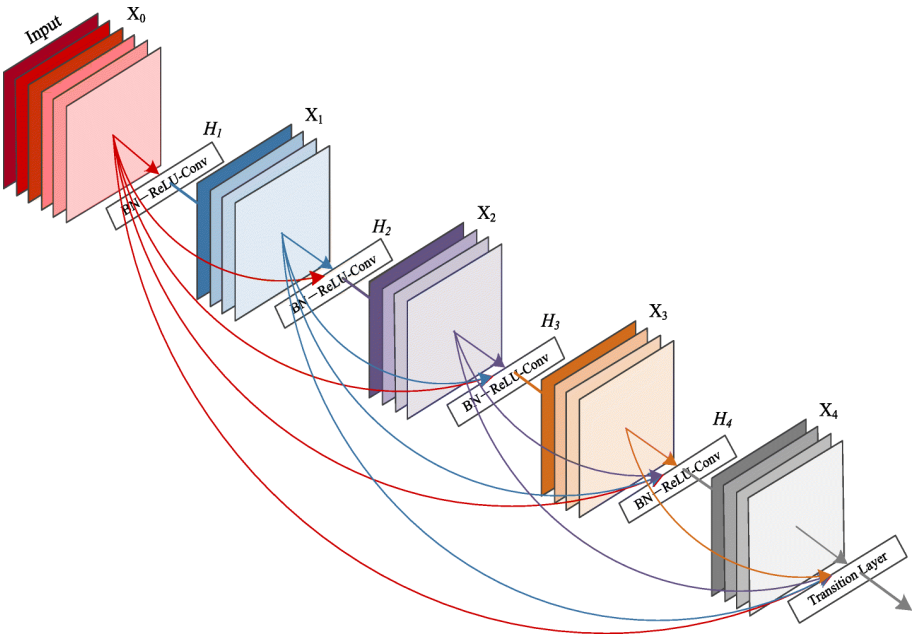

Figure 6: DenseNet Illustration

Dual Path Network is a convolutional neural network which presents a new topology of connection paths internally. The intuition is that ResNets enables **feature re-usage** while DenseNet enables **new feature exploration**, and both are important for learning good representations. To enjoy the benefits from both path topologies, Dual Path Networks **share common features** while maintaining the flexibility to **explore new features** through dual path architectures.

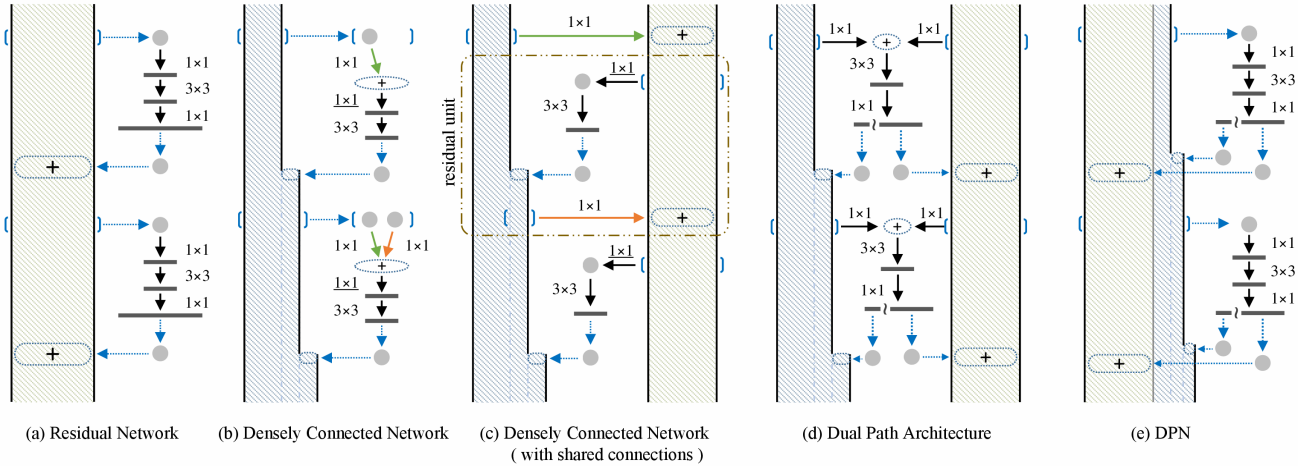

Figure 7: Dual Path Network

Deep Layer Aggregation learns to better extract the full spectrum of semantic and spatial information from a network. **Iterative connections** join neighboring stages to progressively deepen and spatially refine the representation. **Hierarchical connections** cross stages with trees that span the spectrum of layers to better propagate features and gradients. In nutshell, DLA makes no requirements of the internal structure of the blocks and stages. DLA connects across stages with IDA, and within and across stages by HDA.

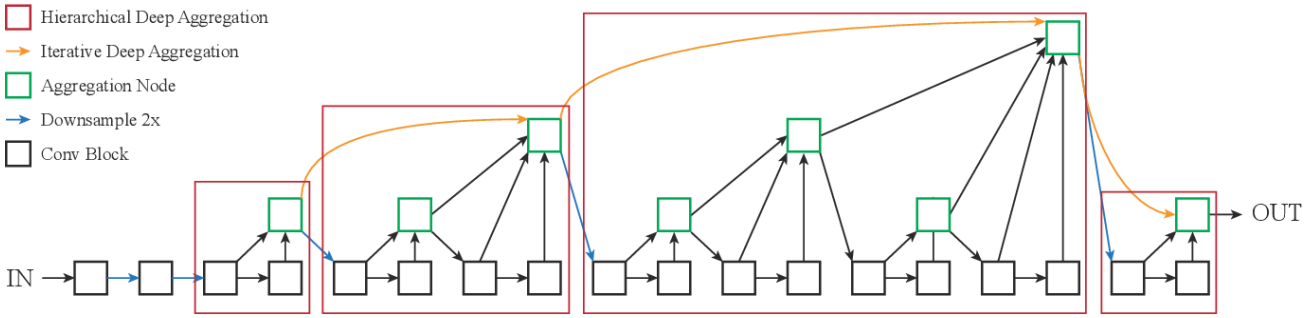

Figure 8: Deep Layer Aggregation

1.1.2 Training Details

We train the models on the 50k training set and evaluate them on the 10k test set. The input images are 32x32 **randomly cropped** from the **zero-padded** 40x40 images or their **horizontally flipped versions**, and **normalization** is performed in the preprocessing step as well. **No other data augmentation is applied**.

As for the optimizer, we simply use **SGD** with a mini-batch size of **128**, the momentum and weight decay are set to **0.9** and **5e-4** respectively. Furthermore, the learning rate of which is starting at **0.1** and decreasing by a factor of **0.5** when the test error plateaus, and the models are trained for **200** epochs.

For more code details, please refer to the attached **main.py** file.

1.1.3 General Results

We conduct experiments on CIFAR-10 using the models mentioned above, and results are shown in the following table (ps. number of parameters may be different from original paper due to the manual implementation for CIFAR-10 dataset).

architecture	#params	error
ResNet-18	11.17M	4.89
PreActResNet-18	11.17M	5.12
WRN-28-10	36.49M	**4.27**
ResNeXt-29, 32×4d	4.77M	4.85
DenseNet-121	6.96M	4.78
DPN-26	11.57M	5.21
DLA	16.29M	5.02

Table 1: Top-1 error(%) and model size on CIFAR-10

Take ResNet-18 as the benchmark, we can observe that WRN-28-10, ResNeXt-29 and DenseNet-121 achieve lower top-1 error while which is to our surprise, PreActResNet-18, DPN-26 and DLA go even worse.

The reason of performance degradation on the latter two might be that their model complexity is far greater than the CIFAR-10 classification task requires, therefore, 200 epochs is not enough for convergence.

For instance, the training process of ResNet-18 is visualized in the figure below.

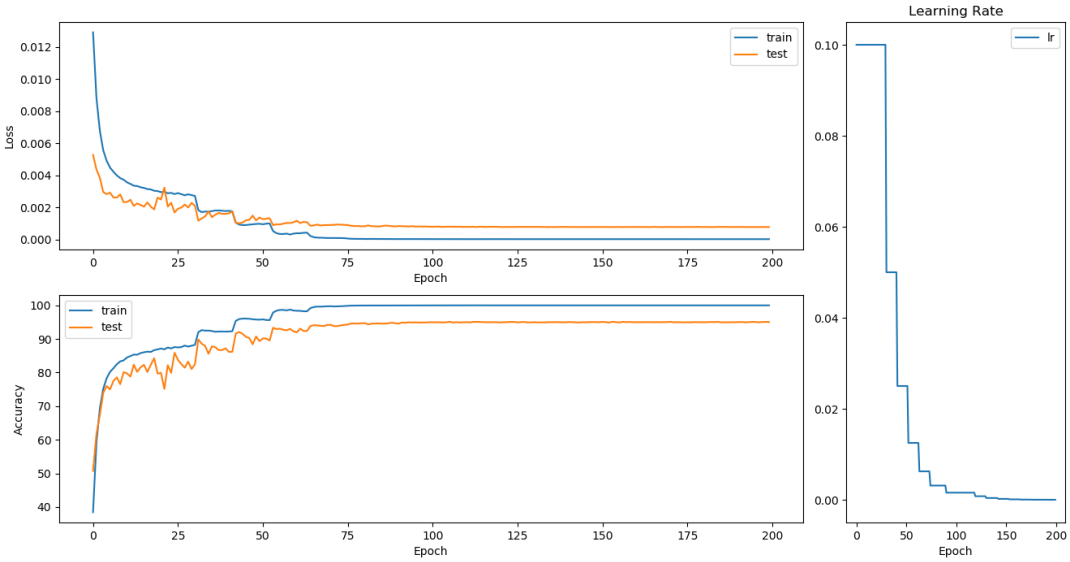

Figure 9: Training process of ResNet-18

1.1.4 Ablation Experiments

We conduct ablation experiments on CIFAR-10 classification task in order to investigate the effect of network hyper-parameters on performance, like **depth, width and cardinality**. The results are shown in the following table.

architecture	setting	#params	error
ResNet-18	1x64d	11.17M	**4.89**
ResNet-50	1x64d	23.52M	5.37
ResNeXt-29	2x32d	2.30M	5.78
ResNeXt-29	2x64d	9.13M	**4.54**
ResNeXt-29	2x64d	9.13M	4.54
ResNeXt-29	8x64d	89.60M	**4.19**
ResNet-50	1x64d	23.52M	5.37
ResNeXt-50	2x40d	17.30M	5.33
ResNeXt-50	8x14d	20.86M	4.80
ResNeXt-50	32x4d	22.98M	**4.54**

Table 2: Ablation experiments on CIFAR-10

Depth. We envaluate effect of depth via ResNet-18 and ResNet-50, it can be seen that the former gains higher accuracy under 200 epochs while maintaining the other same training conditions like optimizer or learning rate scheduler. Therefore, we may come up with the conclusion that **it's not always a good idea to just use deeper network**, not only because of the model complexity and converging time, but also for some possibly redundant layers.

Width. The second part is about width, which refers to the number of channels in a layer. Experiment indicates that the higher the width, the lower the top-1 error. It seems that with parameters of this magnitude, a **wider** network can always achieve better accuracy, for the convergence speed does not slow down much.

Cardinality. Next we investigate increasing complexity by increasing cardinality. We roughly compare the performance of ResNeXt-29 with fixed width 64 but varying cardinality 2 and 8. The latter achieves higher accuracy up to **95.81**%, which is the highest so far. However, the training process is **quite slow** due to the large number of parameters involved (**89.60M**).

Cardinality vs. Width. Last but not least, we evaluate the trade-off between cardinality and width, under preserved complexity as listed in Table 2. Comparing with ResNet-50, the ResNeXt-50 32x4d has a test error of 4.54%, which is **0.83**% lower than the ResNet baseline's 5.37%. With cardinality increasing from 1 to 32 while keeping approximately the same complexity, the test error rate keeps reducing. Furthermore, the 32x4d version also has a much lower training error than the counterpart to the same period, suggesting that the gains are **not from regularization but from stronger representations**, and we may draw the preliminary conclusion that **increasing cardianlity shows better results than going wider**.

1.2 Other Inquries

1.2.1 Training Details

For simplicity, **ResNet-18** with **ReLU** is used as the benchmark model. Same image preprocessing techniques as in the previous section are applied. As for loss function, we choose **CrossEntropy** implemented in PyTorch, .and **Adam** optimizer with initial learning rate 0.001 and reduce-on-plateau is adopted. One more thing, different from the preceding configuration, max training epoch is set to **50**.

1.2.2 Activation Layer

Replace all the activation layers with **Sigmoid, Leaky ReLU, ELU or GELU**. Here are the results.

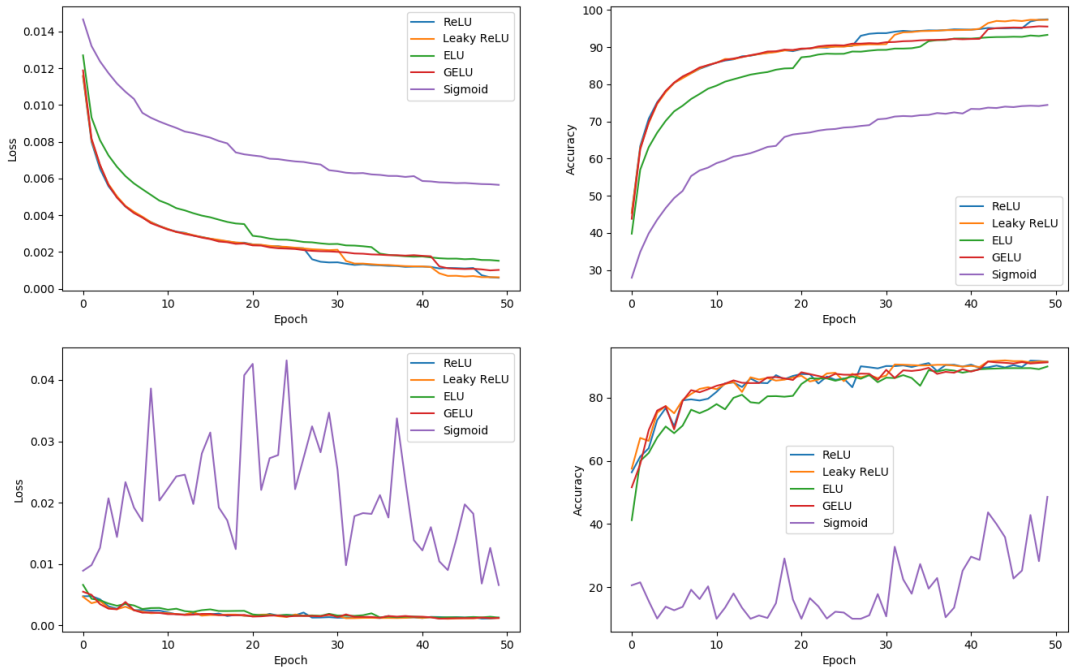

Figure 10: Different activation layers (**Upper:** Training. **Lower:** Test.)

Sigmoid activation layer looks out of place, while the others are united in convergent behavior. Among ReLU family, Leaky ReLU is the relatively fastest one and ELU is on the contrary, and from test accuracy perspective, GELU is the most **stable** one with visually minimum volatility.

For more code details, please refer to the attached **main_activation.py** file.

1.2.3 Loss Function

We attempt some other criterion like **BCELoss, MSELoss, HuberLoss** to see the effect of the loss function. Here presented are the results.

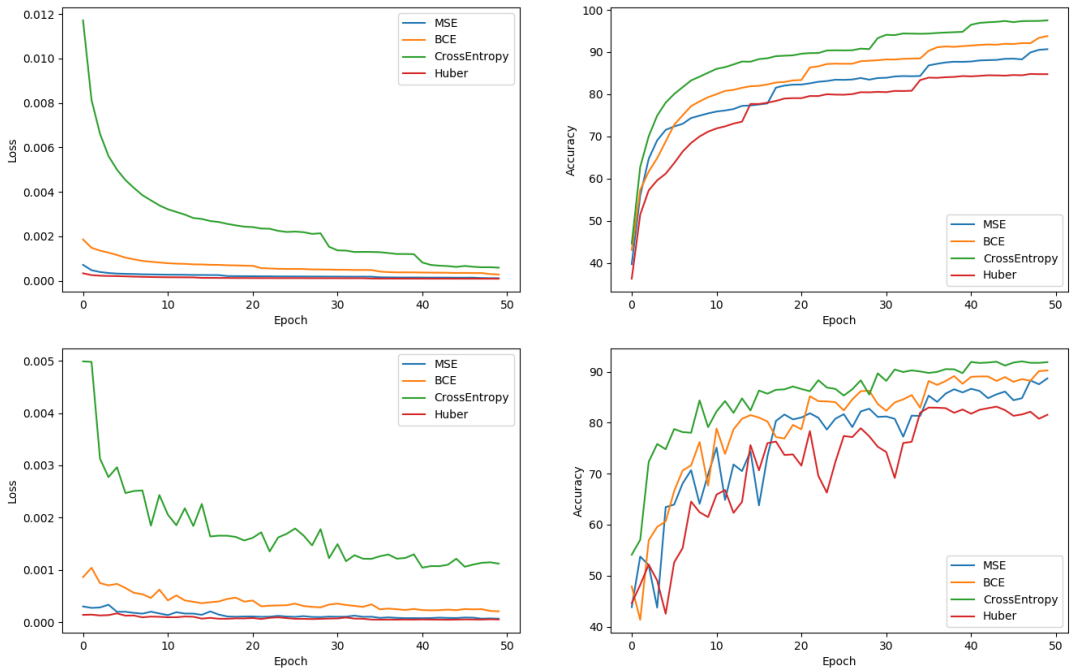

Figure 11: Different Loss Functions (**Upper:** Training. **Lower:** Test.)

From the log image, phenomenon can be observed that the stratification is quite significant in each subfigure, manifesting the classification capability ranking should be: CrossEntropy > BCELoss > MSELoss > HuberLoss.

For more code details, please refer to the attached **main_criterion.py** file.

1.2.4 Optimizer

This part we try different optimizers, including **SGD, Adagrad, Adadelta, Adam**, and results are shown in the following figure.

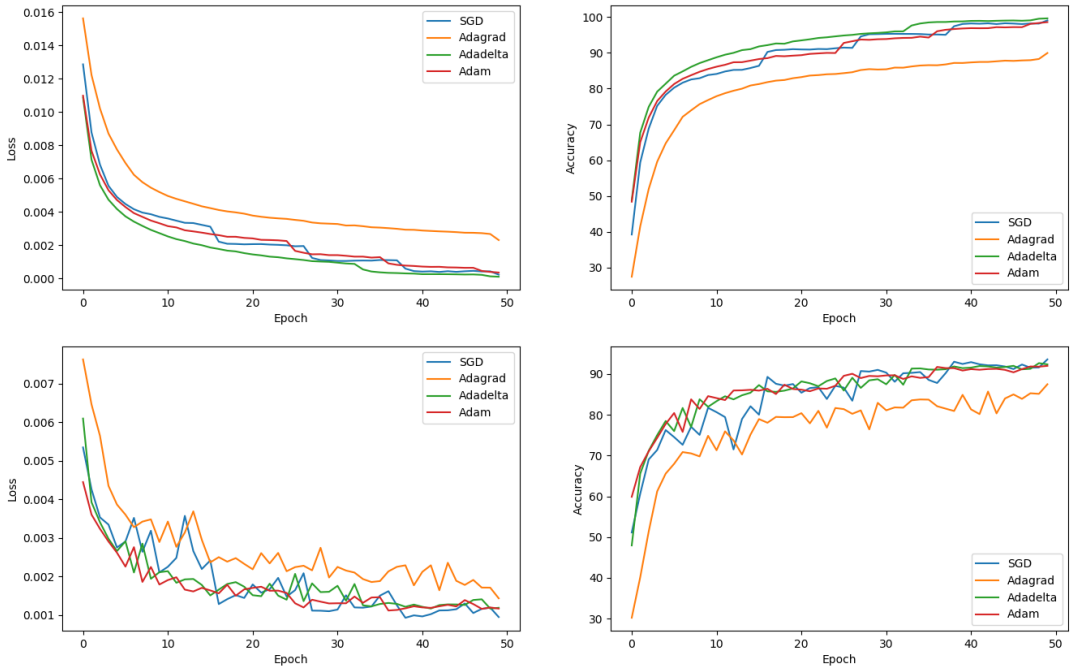

Figure 12: Different Optimizers (**Upper:** Training. **Lower:** Test.)

Adagrad performs worse in terms of every subfigure, while others are on the same level.

1.3 Network Interpretation

Ramprasaath R. Selvaraju et al. proposed a technique for producing ”visual explanations” for decisions from a large class of **CNN-based models**, making them more transparent. Their approach - **Gradient-weighted Class Activation Mapping (Grad-CAM)**, uses the gradients of any target concept, flowing into the **final convolutional layer** to produce a coarse localization map highlighting important regions in the image for predicting the concept.

The following figure shows the results of Grad-CAM for ResNet-18.

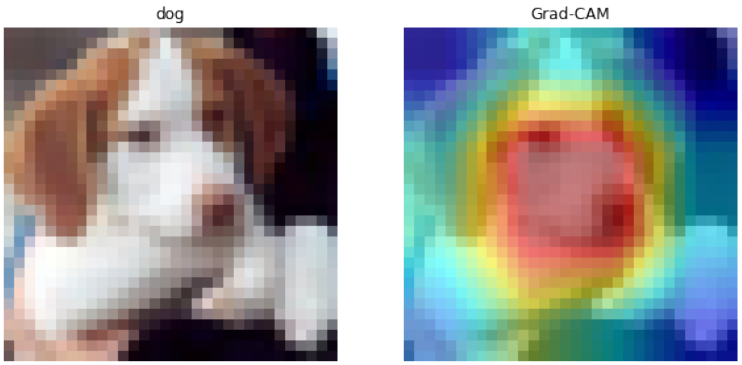

Figure 13: Grad-CAM for ResNet-18

1.4 Best Model

Our best model is **WideResNet-28-10** with extra **Cutout** augmentation, other preprocessings are just as usual. The test accuracy achieves **96.65**% in 200 epochs, and the powerful effect of Cutout augmentation reflects in the training process saliently compared with any other techniques.

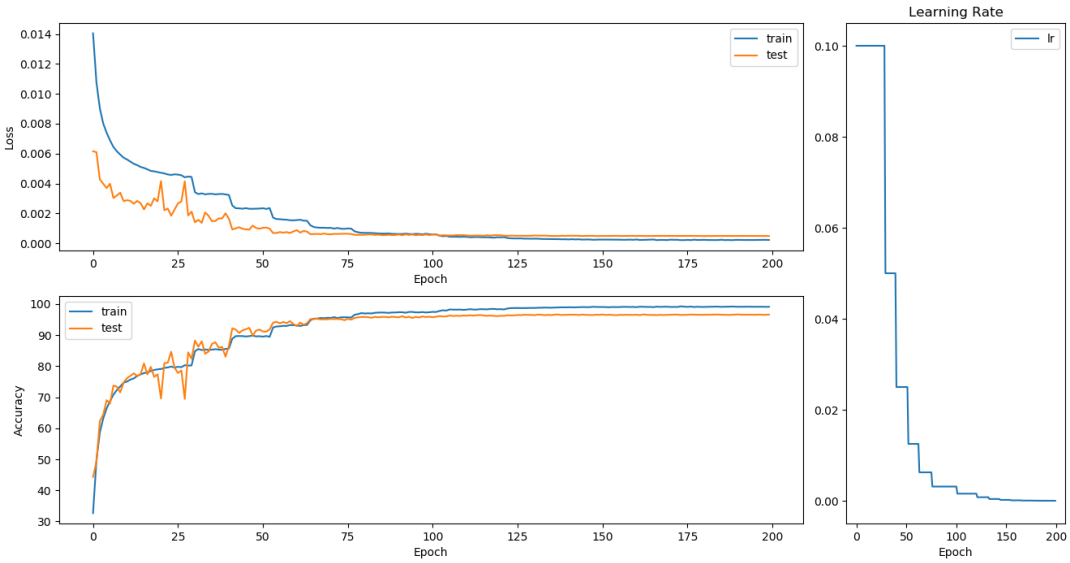

Figure 14: WRN-28-10 with **Cutout**

2 Batch Normalization

Batch Normalization aims to **reduce internal covariate shift**, and in doing so aims to accelerate the training of deep neural nets. It accomplishes this via a normalization step that fixes the means and variances of layer inputs. Batch Normalization also has a beneficial effect on the gradient flow through the network, by reducing the dependence of gradients on **the scale of the parameters or of their initial values**. This allows for **use of much higher learning rates without the risk of divergence**. Furthermore, batch normalization regularizes the model and reduces the need for Dropout.

The implementation of Batch Normalization for a minibatch $\mathcal{B}$ is as follows:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i)$$

Where γ and β are **learnable parameters**, ϵ is a small constant to avoid division by zero.

Recent research results show that BN reparametrizes the underlying optimization problem to make its landscape significantly more smooth. So along this line, we are going to measure:

1. Loss landscape or variation of the value of the loss.
2. Gradient predictiveness or the change of the loss gradient.
3. Maximum difference in gradient over the distance.

2.1 Loss Landscape

By setting the learning rate to 2e-3, 1e-3, 5e-4 and 1e-4 and training each for 20 epochs, the fill-between plot of the loss landscape can be shown in the following figure. This is a loss-step plot, since the batch size is 128, there will be nearly 8,000 steps in total.

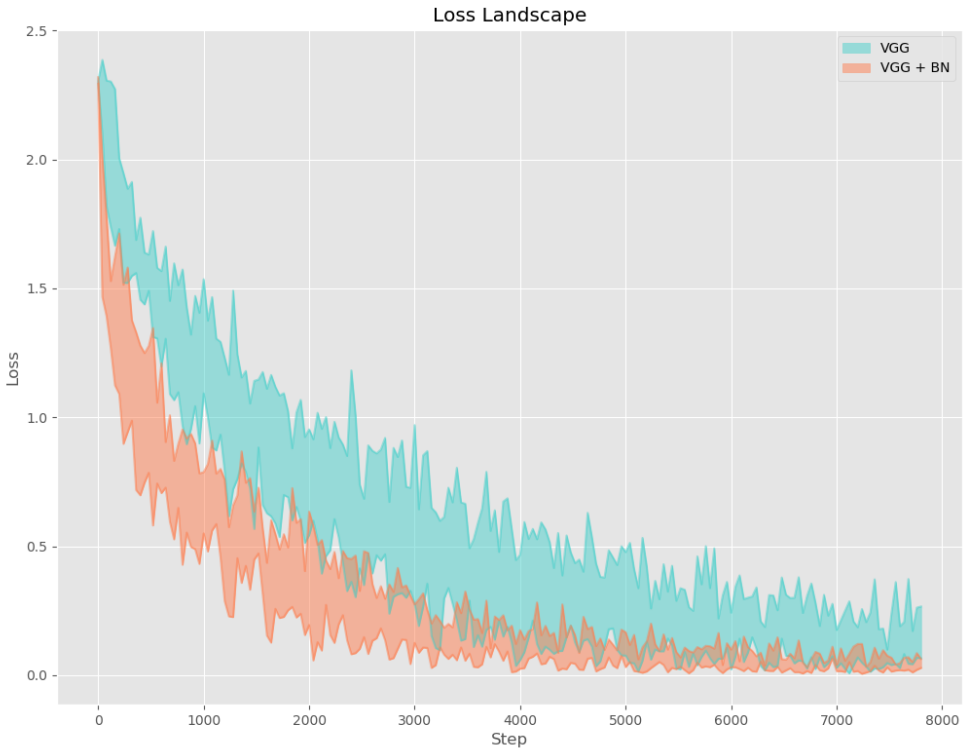

Figure 15: Loss landscape for VGG-A with and without BatchNorm

From figure above, we can easily draw the conclusion that BatchNorm really helps to reduce the loss variance and thus boost the training speed.

2.2 Gradient Predictiveness

Keeping the same hyper-parameters, we intend to record the change of gradient in last linear layer where the metric chosen is the **L2 norm** of the gradient difference. Results are presented in the following figure.

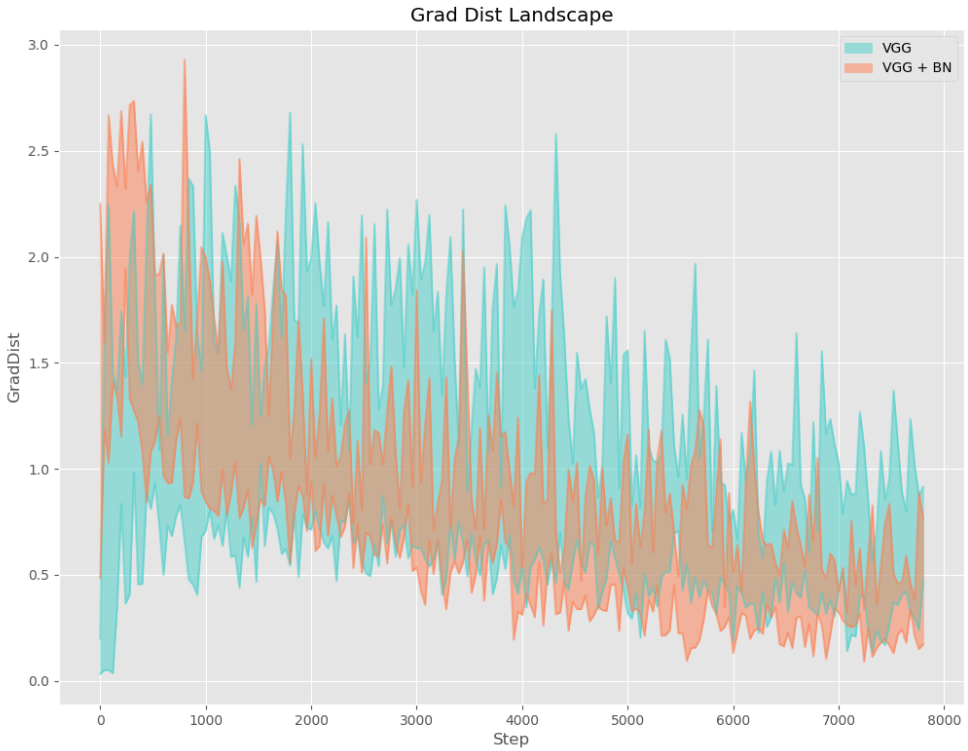

Figure 16: Gradient predictiveness for VGG-A with and without BatchNorm

As for the general trend of the gradient predictiveness, we can find that VGG with BatchNorm is batter and more stable than VGG without BatchNorm because of the lower L2 distance and more smooth volatility.

2.3 Beta Smoothness

Recall that f is β-smooth if its gradient is β-Lipschitz. It is worth noting that, due to the existence of non-linearities, one should not expect the β-smoothness to be bounded in an absolute, global sense.

$$\|\nabla f\left(w_{t+1}\right)-\nabla f\left(w_t\right)\| \leq \beta_t\left\|w_{t+1}-w_t\right\|$$

And we know that the update of w provided by SGD from t to $t+1$ is

$$w_{t+1}=w_t-lr*\nabla f\left(w_t\right)$$

Thus we can derive β from the following formula:

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

Or just use

$$\beta_t=\max \frac{\|\nabla f\left(w_{t+1}\right)-\nabla f\left(w_t\right)\|}{\|w_{t+1}-w_t\|}$$

for Adam optimizer.

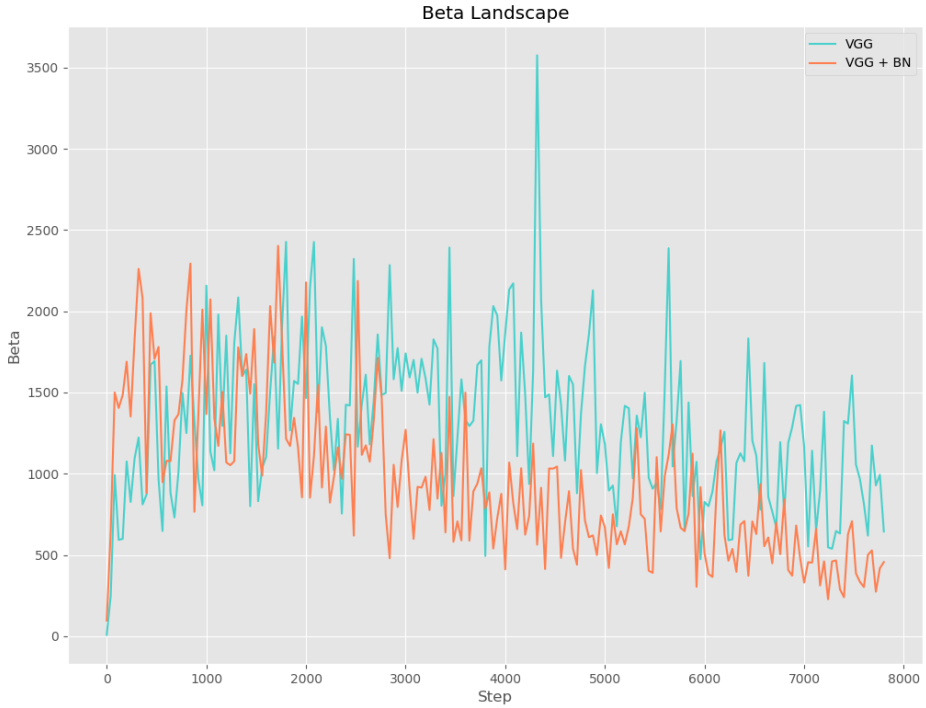

Figure 17: Beta Smoothness for VGG-A with and without BatchNorm

3 DessiLBI

The great success of deep neural networks is built upon their over-parameterization, which smooths the optimization landscape without degrading the generalization ability. Despite the benefits of over-parameterization, a huge amount of parameters makes deep networks cumbersome in daily life applications. On the other hand, training neural networks without over-parameterization faces many practical problems, e.g., being trapped in local optimal. Though techniques such as pruning and distillation are developed, they are expensive in fully training a dense network as backward selection methods, and there is still a void on systematically exploring forward selection methods for learning structural sparsity in deep networks. To fill in this gap, this paper proposes a new approach based on differential inclusions of inverse scale spaces. Specifically, our method can generate a family of models from simple to complex ones along the dynamics via coupling a pair of parameters, such that over-parameterized deep models and their structural sparsity can be explored simultaneously. This kind of differential inclusion scheme has a simple discretization, dubbed Deep structure splitting Linearized Bregman Iteration (DessiLBI), whose global convergence in learning deep networks could be established under the Kurdyka-Łojasiewicz framework.

3.1 LeNet on MNIST

Applying SLBI and SGD for LeNet on MNIST dataset respectively, we can compare the performance of the two optimizers. The results are shown in the following figure.

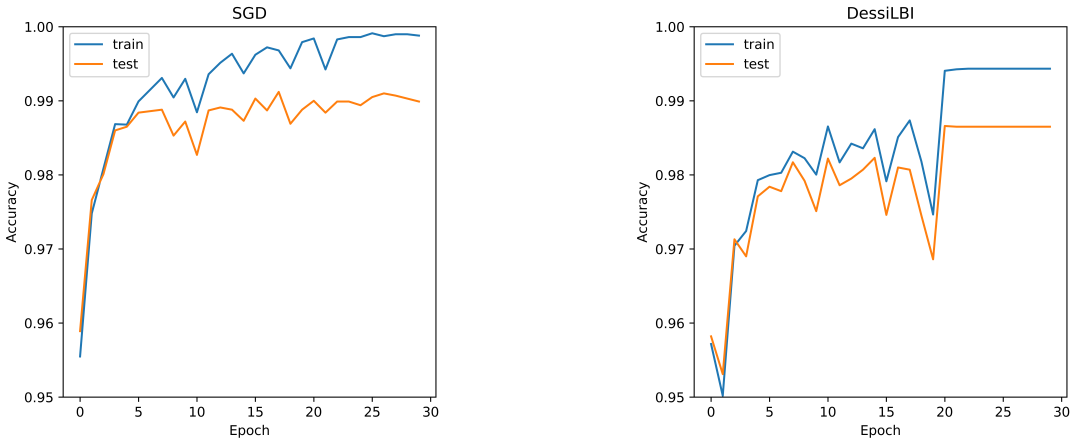

Figure 18: Comparison between SGD and SLBI for LeNet on MNIST

They are of similar accuracy, while SLBI converges after about 20 epochs because of the learning rate. Now let's prune some of the parameters on the third convolutional layer at different rates.

Pruning Ratio	0%	10%	20%	40%	60%	80%
Accuracy	98.65%	98.65%	98.65%	98.64%	98.60%	98.30%

Table 3: Accuracy of SLBI on LeNet on MNIST with different pruning ratios

As you can see, the pruning is successful and safe until the ratio reaches nearly 60%, which is a very high level of pruning, indicating the high degree of redundancy in model parameters.

Here is the visualization on the third convolutional layer before and afterpruning.

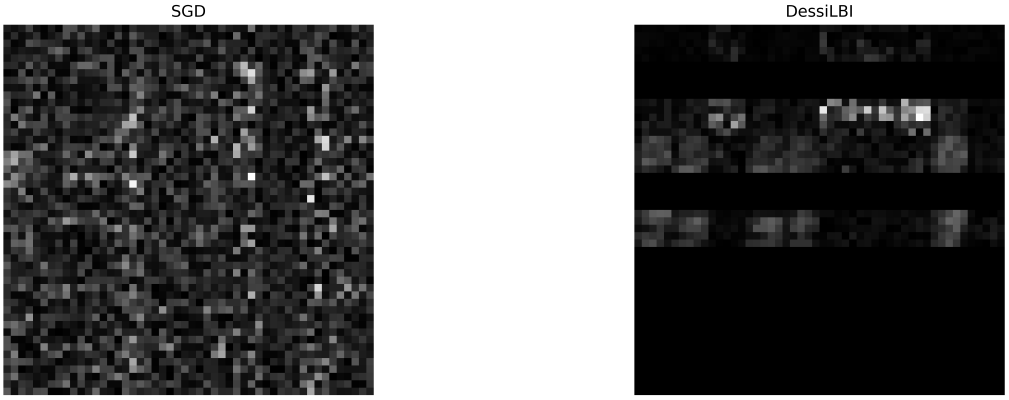

Figure 19: Visualization on 3rd Conv before and after pruning

3.2 Combination of SLBI and Adam

Referring to the official implementation of Adam, we modify the core script (step method) of SLBI to use Adam instead of SGD. Then we train the ResNet on CIFAR-10 dataset with both modified SLBI and Adam. The results are shown as following.

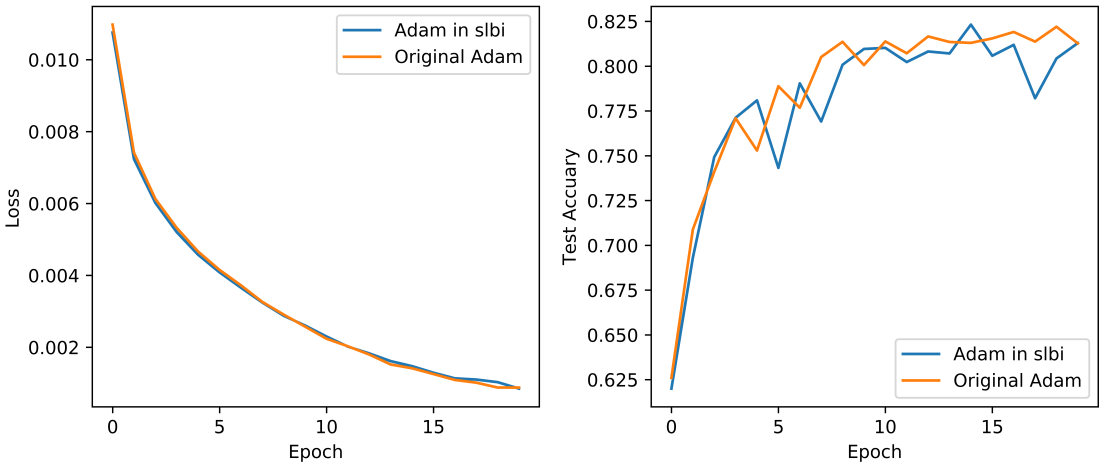

Figure 20: Comparison between Adam and SLBI for ResNet on CIFAR-10

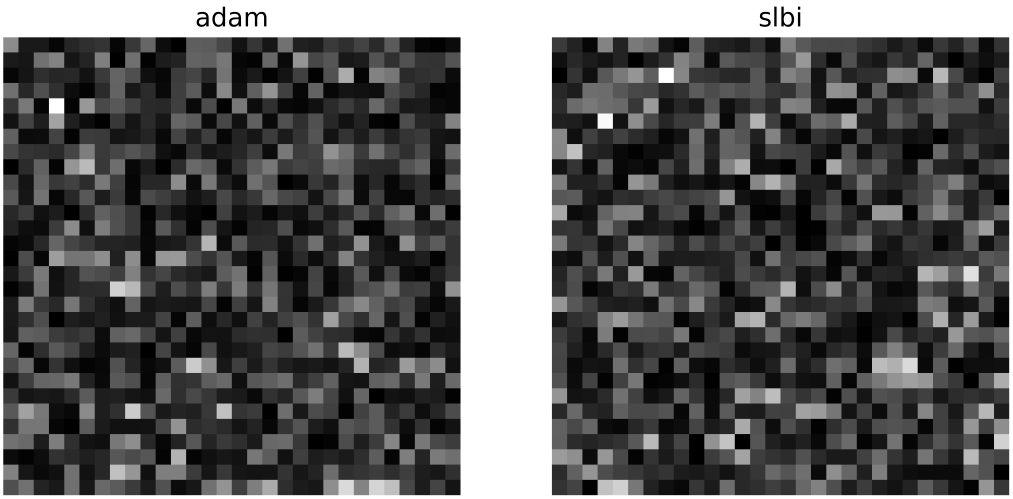

Figure 21: Visualization on 3rd Conv before and after pruning

There's no significant difference between them, which means the light-weight over-parameterization in this case.