Report For Deep Learning Mini-Project

Lin Yuan[1] **Xinyu Zhang**[2]

[1]ly2279 N15583133
[2]zx3630 N21619909

Abstract

The aim of this project is to create the modified residual network (ResNet) architecture that achieves the highest test accuracy on the CIFAR-10 image classification dataset while keeping model's parameter count to a maximum of 5 million. We did a series of training of models includes ResNet18. ResNet34, ResNet50. ResNet101 and ResNet152 first as baseline. Then, we proposed two self-modified model structure training with both SGD and Adam to compare the result. We have open-sourced our code to https://github.com/LinYuanNYU/DL_MiniProject.git

Introduction

ResNet (He et al. 2015) is a deep convolutional neural networks which sloved the exploding and vanishing gradient descent problem, so that increase the depth of these networks further in order to make the model more robust and enhance its performance. The architecture of ResNet is assuming that a neural network unit can learn any function, asymptotically, then it can learn the identity function as well.the gradients can flow directly through the skip connections backwards from later layers to initial filters.

The CIFAR-10 dataset consists of 6000 images per class in 10 classes totaling 60000 32x32 color images. Each training batch and test batch in the dataset has 10,000 photos, and there are five training batches total.

By randomly rotating and cropping the provided picture data, (Wang et al. 2017) carry out some basic data augmentation. To do this, they define a torchvision transform. The PyTorch documentation has information on all the transformations that are employed to pre-process and enhance data. We download and choose the training and test datasets and prepare data loaders for further use.

We utilized codes open-sourced by (Liu 2013) and trained two self-defined ResNet with two kinds of optimizers: SGD with momentum and Adam. Also, we utilize CrossEntropyLoss as the Loss function in training both structures.

Traditional ResNet Series

In this section, we trained ResNet18 and ResNet34 for 50 epochs, and ResNet101, ResNet152 for 40 epochs and we get results shown in Figure 1-4 and Table 1

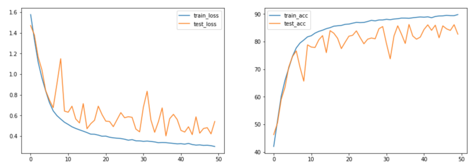


Figure 1: Losses and Acces for ResNet18

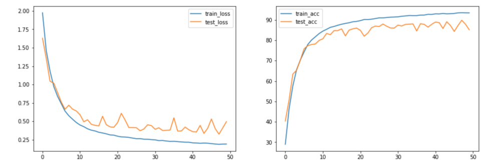


Figure 2: Losses and Acces for ResNet34

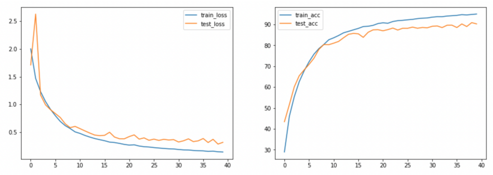


Figure 3: Losses and Acces for ResNet101

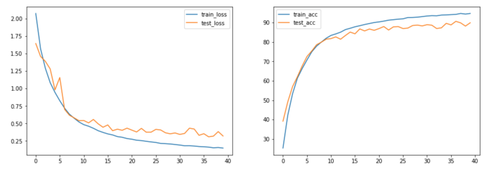


Figure 4: Losses and Acces for ResNet152

Model	Train Acc	Test Acc
ResNet18	89.810%	86.14%
ResNet34	93.500%	89.83%
ResNet101	94.718%	90.66%
ResNet152	94.958%	90.85%

Table 1: Classical ResNet on CIFAR-10 We Trained

Self-Modified Structure

In this section, we proposed two modified ResNets since we need a network with less than 5 million parameters.

Method 1: Decrease number of blocks in each Conv section into 1

For the first one, we modified the structure based on ResNet18. For each Residual block, ResNet18 has 2 blocks, we changed it into one block. The structure is shown in Figure 5.

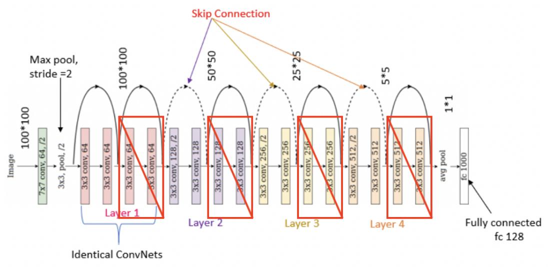
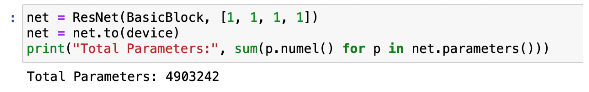


Figure 5: Method 1: Decrease number of blocks in each Conv section into 1

From codes shown in Figure 6, we can verify that the number of parameter the model has are 4.9 million. We proposed two models and trained them with both SGD and Adam optimizer.



```
: net = ResNet(BasicBlock, [1, 1, 1, 1])
  net = net.to(device)
  print("Total Parameters:", sum(p.numel() for p in net.parameters()))

  Total Parameters: 4903242
```

Figure 6: Number of Parameters: 4.9M

Training with SGD

After training for 60 epochs, which uses about 15 minutes on RTX8000, we get results shown in Figure 7 and 8. For this part, I use SGD with momentum as optimizer and an initial learning rate of 0.1. Also, a learning rate scheduler is introduced. Specifically, I use CosineAnnealingLR provided by Pytorch.

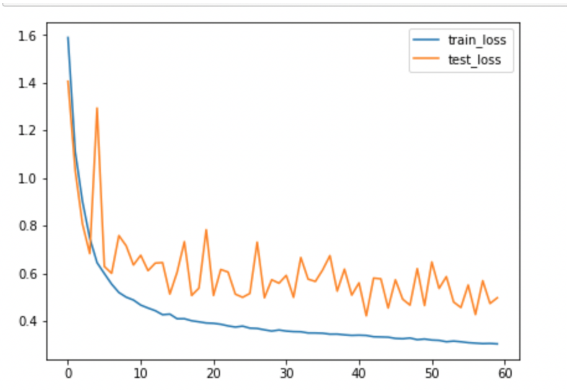


Figure 7: Losses for Self Modified ResNet V1 using SGD

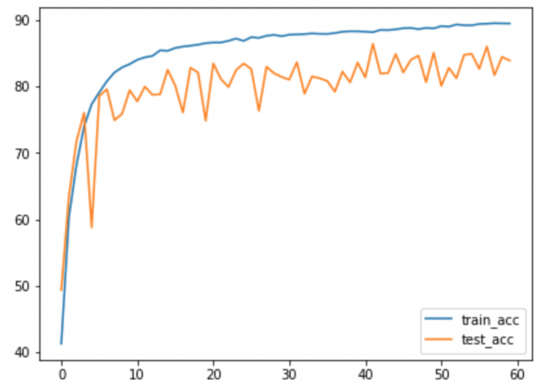
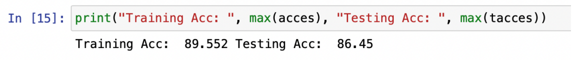


Figure 8: Accuracies for Self Modified ResNet V1 using SGD

Using codes in Figure 9, we obtain the result shown in table:



```
In [15]: print("Training Acc: ", max(acces), "Testing Acc: ", max(tacces))

         Training Acc:  89.552 Testing Acc:  86.45
```

Figure 9: Maximum Accuracy of Self Modified ResNet using SGD

Model	Train Acc	Test Acc
Self-Modified ResNet(SGD)	89.552%	86.45%

Table 2: Result for Self modified ResNet V1 using SGD

Training with Adam

Same as above, with the same model structure, I use Adam optimizer this time. The result is shown in Figure 10 and Figure 11.

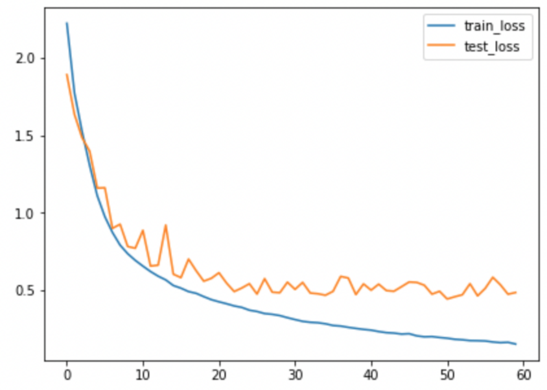


Figure 10: Losses for Self Modified ResNet V1 using Adam

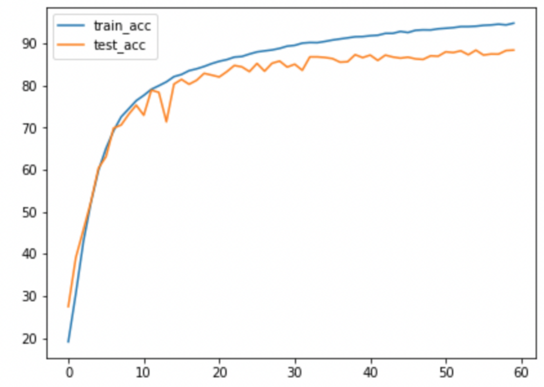
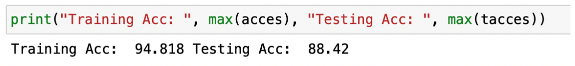


Figure 11: Accuracies for Self Modified ResNet V1 using Adam

Using codes in Figure 12, we obtain the result shown in table



```
print("Training Acc: ", max(acces), "Testing Acc: ", max(tacces))
Training Acc:  94.818 Testing Acc:  88.42
```

Figure 12: Maximum Accuracy of Self Modified ResNet V1 using Adam

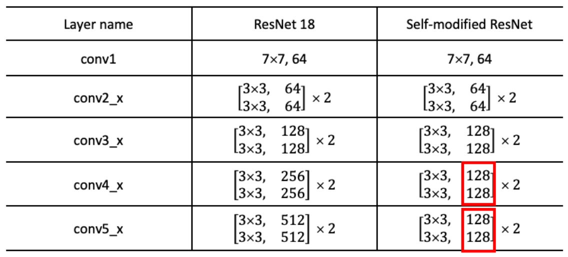
Model	Train Acc	Test Acc
Self-Modified ResNet(Adam)	94.818%	88.42%

Table 3: Result for Self modified ResNet V1

From the above experiments, we can conclude that by using Adam optimizer, we improved the performance of our model by 2% to 88.42%.

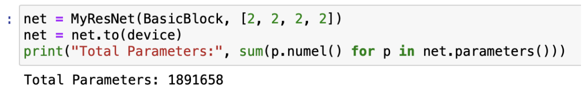
Method 2: Decrease 3_{rd} Dimension of Conv Layers into 128

As we know, the majority of trainable parameters is introduced at the point when we connect the last convolution layer and the first Linear layer. By reducing the dimension of latter part of Conv blocks, we can drastically reduce the number of trainable parameters. A structural comparison of our model with classical ResNet-18 is shown in figure 13. With such modification, we further decrease the amount of parameters down to 1.8 million(as shown in Figure 14).



Layer name	ResNet 18	Self-modified ResNet
conv1	7×7, 64	7×7, 64
conv2_x	$\begin{bmatrix}3\times3, & 64\\3\times3, & 64\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3, & 64\\3\times3, & 64\end{bmatrix}\times 2$
conv3_x	$\begin{bmatrix}3\times3, & 128\\3\times3, & 128\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3, & 128\\3\times3, & 128\end{bmatrix}\times 2$
conv4_x	$\begin{bmatrix}3\times3, & 256\\3\times3, & 256\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3, & \boxed{128}\\3\times3, & \boxed{128}\end{bmatrix}\times 2$
conv5_x	$\begin{bmatrix}3\times3, & 512\\3\times3, & 512\end{bmatrix}\times 2$	$\begin{bmatrix}3\times3, & \boxed{128}\\3\times3, & \boxed{128}\end{bmatrix}\times 2$

Figure 13: Convolutional part of Self-modified ResNet V2



```
: net = MyResNet(BasicBlock, [2, 2, 2, 2])
  net = net.to(device)
  print("Total Parameters:", sum(p.numel() for p in net.parameters()))

  Total Parameters: 1891658
```

Figure 14: Number of Parameters: 1.8 million

Training with SGD

Similar to the above experiments in Method 1, I trained the model for 60 epochs and using a initial learning rate of 0.1 along with a Cosine learning rate scheduler, which obviously has a T_max of 60 as well. Eventually, we have the result shown in Figure 15 and 16. The performance is quite surprisingly good. We have a accuracy over 94% very quickly(less than 15 minute training time on RTX8000).

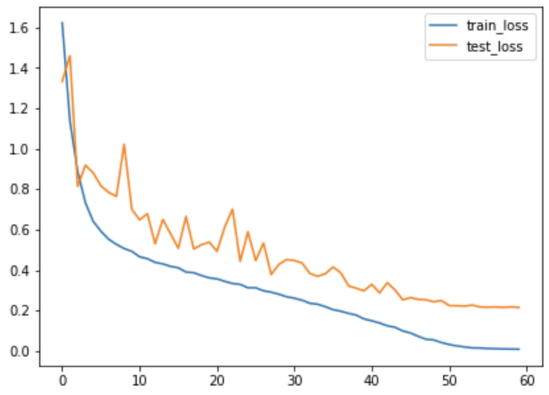


Figure 15: Losses for Self Modified ResNet V2 using SGD

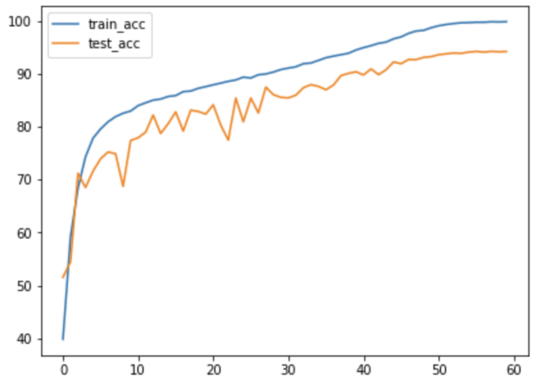
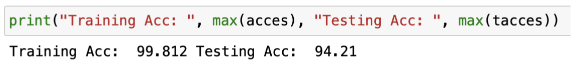


Figure 16: Accuracies for Self Modified ResNet V2 using SGD

Using codes in Figure 12, we obtain the result shown in table



```
print("Training Acc: ", max(acces), "Testing Acc: ", max(tacces))
Training Acc:  99.812 Testing Acc:  94.21
```

Figure 17: Maximum Accuracy of Self Modified ResNet V2 using SGD

Model	Train Acc	Test Acc
Self-Modified ResNet(Adam)	99.812%	94.21%

Table 4: Result for Self modified ResNet V2 using SGD

Training with Adam

Same as above, with the same model structure, I use Adam optimizer this time. The result is shown in Figure and Figure 11.

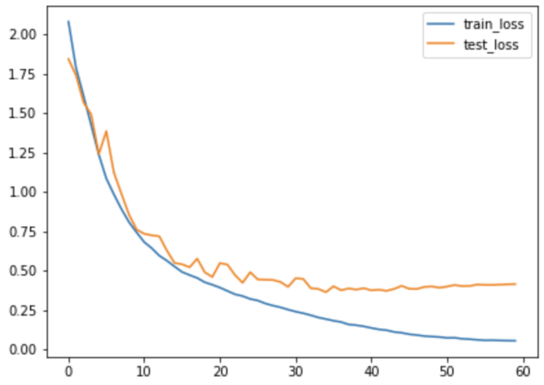


Figure 18: Losses for Self Modified ResNet V2 using Adam

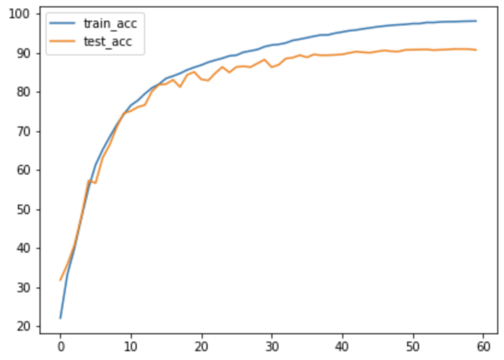
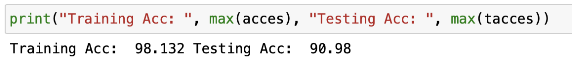


Figure 19: Accuracies for Self Modified ResNet V2 using Adam

Using codes in Figure 12, we obtain the result shown in table



```
print("Training Acc: ", max(acces), "Testing Acc: ", max(tacces))
Training Acc:  98.132 Testing Acc:  90.98
```

Figure 20: Maximum Accuracy of Self Modified ResNet V2 using Adam

Model	Train Acc	Test Acc
Self-Modified ResNet(Adam)	98.132%	90.98%

Table 5: Result for Self modified ResNet V2 using Adam

Conclusion

Above all, we proposed two ways of modifications: 1. decrease number of convolutional blocks down to 1 for each convolution section; 2. Decrease 3rd Dimension of Convolution Layers into 128 for latter part of the network. For each method, we trained with two optimizers from scratch. Eventually we have result shown in Table 6. As we can see, we achieved a highest accuracy of 94% on test set with the second method using SGD.

Model	Train Acc	Test Acc
Self-Modified ResNet V1 (SGD)	89.552%	86.45%
Self-Modified ResNet V1 (Adam)	94.818%	88.42%
Self-Modified ResNet V2 (SGD)	99.812%	94.21%
Self-Modified ResNet V2 (Adam)	98.132%	90.98%

Table 6: Result for Self modified ResNet V2

References

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385.

Liu, K. 2013. pytorch-cifar. https://github.com/kuangliu/pytorch-cifar.

Wang, F.; Jiang, M.; Qian, C.; Yang, S.; Li, C.; Zhang, H.; Wang, X.; and Tang, X. 2017. Residual Attention Network for Image Classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.