Deep Learning Mini-Project

Kaiyu Pei(kp2690), Keng-Ming Chang(kc4536), Jincheng Tian(jt4434)

New York University

https://github.com/gamingzhang/ece7123_mini_project

Overview

In this mini-project, we are tasked with coming up with a modified residual network (ResNet) architecture with the highest test accuracy on the CIFAR-10 image classification dataset, under the constraint that the model has no more than 5 million parameters. We started with ResNet-18 and then designed 5 different models. During the design process, we learned that starting with a good network architecture is a good strategy when we try to design our own models. Our final model has 4,992,586 parameters and gains 95.66% test accuracy on CIFAR-10.

Introduction

First, we need to design the building blocks of our ResNet. There are two types of residual block in the original ResNet paper [1]: BasicBlock and BottleNect. BottleNect can be used if we train a deep ResNet. Consider that our model has no more than 5 million parameters, we use BasicBlock in our model.

Second, we think about the architecture of our ResNet. There are five models proposed by the authors in the original paper [1]: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152. ResNet-18(Appendix 1) has 11,173,962 parameters (fig. 1) and gets 93.02% test accuracy on CIFAR-10 dataset [2]. We can start with this good architecture and make some modification.

```
================================================================
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 1 Total Parameters of ResNet-18

Model design

Model 1. We try to reduce the number of parameters of ResNet-18 and see if it still can get high test accuracy on CIFAR-10. ResNet-18 has 4 layers, and each layer has two Basic Blocks. We decide to remove one Basic Block in each layer:

Number of blocks: [2,2,2,2] → [1,1,1,1]

Now the number of parameters becomes 4,903,242 (fig.2). We train this model for 200 epochs and the best model gets 94.35% test accuracy (fig. 3). Seems that we already find a good architecture.

```
================================================================
Total params: 4,903,242
Trainable params: 4,903,242
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 2 Total Parameters of Model 1

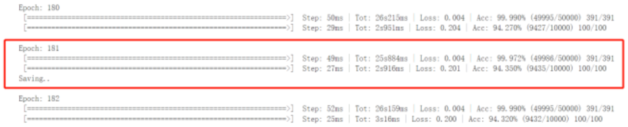


Fig. 3 Test accuracy of Model 1

Model 2. We want to know if we can get a better model if we increase the number of basic blocks. From the summary of the ResNet-18, we know that most of the trainable parameters are from the last layer. So, we try to remove the last layer of ResNet-18 and then add more basic blocks in first three layer. In our model 2, the number of the blocks is [2,2,4,0]:

Number of blocks: [1,1,1,1] → [2,2,4,0]

Pool size in the average pool layer: 4 → 8

This model has 5,139,018 parameters (fig. 4). We train this model for 200 epochs and surprisingly, this model gets 100% train accuracy and 95.46% test accuracy at epoch 197 (fig. 5). Unfortunately, we cannot use this model as our final model because the number of parameters exceed 5M. But maybe we can design a model better than Model 1 based on this model.

```
----------------------------------------------------------------
Total params: 5,139,018
Trainable params: 5,139,018
Non-trainable params: 0
```

Fig. 4 Total Parameters of Model 2

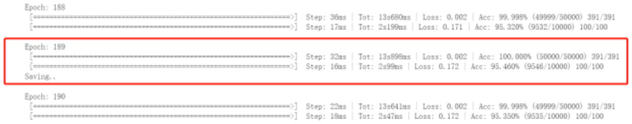


Fig. 5 Test Accuracy of Model 2

Model 3. We want to design a model for no more than 5 million parameters based on Model 2. We decrease the number of channels in the layer 3 from 256 to 250:
Number of blocks: [2,2,4,0]
Number of channels in 3rd layer: 256 → 250
This model has 4,939,902 parameters (fig. 6). We train it for 245 epochs, and it achieves 100% train accuracy and 95.03% test accuracy at epoch 214. (fig. 7)

```
================================================================
Total params: 4,939,902
Trainable params: 4,939,902
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 6 Total Parameters of Model 3

```
Epoch: 213
 [=============================================================>]  Step: 28ms | Tot: 13s771ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=============================================================>]  Step: 19ms | Tot: 2s154ms | Loss: 0.197 | Acc: 94.910% (9491/10000) 100/100

Epoch: 214
 [=============================================================>]  Step: 32ms | Tot: 13s862ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=============================================================>]  Step: 18ms | Tot: 2s138ms | Loss: 0.199 | Acc: 95.030% (9503/10000) 100/100
Saving..

Epoch: 215
 [=============================================================>]  Step: 30ms | Tot: 13s814ms | Loss: 0.002 | Acc: 99.996% (49998/50000) 391/391
 [=============================================================>]  Step: 35ms | Tot: 2s30ms | Loss: 0.196 | Acc: 94.940% (9494/10000) 100/100
```

Fig. 7 Test Accuracy of Model 3

Model 4. We try to improve model 3's performance by adding more parameters. We made some slight modifications:
the number of channels in layer 2: 128 → 130
the number of channels in layer 3: 250 → 251
Now the model has 4,993,025 parameters (fig. 8). We train this model for 400 epochs, and it gets 95.43% test accuracy at epoch 307(fig. 9).

```
================================================================
Total params: 4,993,025
Trainable params: 4,993,025
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 8 Total Parameters of Model 4

```
Epoch: 306
 [=============================================================>]  Step: 76ms | Tot: 44s718ms | Loss: 0.002 | Acc: 100.000% (50000/50000) 391/391
 [=============================================================>]  Step: 37ms | Tot: 4s318ms | Loss: 0.179 | Acc: 95.310% (9531/10000) 100/100

Epoch: 307
 [=============================================================>]  Step: 76ms | Tot: 44s571ms | Loss: 0.002 | Acc: 99.994% (49997/50000) 391/391
 [=============================================================>]  Step: 26ms | Tot: 4s212ms | Loss: 0.178 | Acc: 95.430% (9543/10000) 100/100
Saving..

Epoch: 308
 [=============================================================>]  Step: 65ms | Tot: 44s860ms | Loss: 0.002 | Acc: 99.996% (49998/50000) 391/391
 [=============================================================>]  Step: 31ms | Tot: 4s299ms | Loss: 0.178 | Acc: 95.330% (9533/10000) 100/100
```

Fig. 9 Test Accuracy of Model 4

Model 5. In model 2,3,4, we notice that layer 3 has the most blocks. We want to know if we can get a better model if layer 2 has the most blocks. Here is the modification:
the number of channels in layer 2: 130 → 128
the number of channels in layer 3: 251 → 250
Number of blocks: [2,2,4,0] → [4,5,3,0]
This model has 4,992,586 parameters (fig. 10). We train this model for 200 epochs and it gets 95.66% test accuracy at epoch 189. (fig. 11)

```
================================================================
Total params: 4,992,586
Trainable params: 4,992,586
Non-trainable params: 0
----------------------------------------------------------------
```

Fig. 10 Total Parameters of Model 5

```
Epoch: 188
 [=============================================================>]  Step: 93ms | Tot: 56s925ms | Loss: 0.002 | Acc: 99.994% (49997/50000) 391/391
 [=============================================================>]  Step: 42ms | Tot: 4s27ms | Loss: 0.171 | Acc: 95.510% (9551/10000) 100/100 ...........

Epoch: 189
 [=============================================================>]  Step: 92ms | Tot: 56s950ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=============================================================>]  Step: 39ms | Tot: 4s27ms | Loss: 0.171 | Acc: 95.660% (9566/10000) 100/100 ==>.........
Saving..

Epoch: 190
 [=============================================================>]  Step: 96ms | Tot: 56s920ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=============================================================>]  Step: 43ms | Tot: 4s26ms | Loss: 0.169 | Acc: 95.550% (9555/10000) 100/100 ...........
```

Fig. 11 Test Accuracy of Model 5

Lessons

The most important lesson we have learned during the design process is that we can start with a good network architecture if we want to design our own models to solve our own problems. A good network architecture provides us good hyperparameters, such as the number of layers, the number of basic blocks, the number of channels, etc. We can just fine-tune the network for our own goal, and we might have a good result. If we build our model from scratch, we must think about all the possible hyperparameters, and it might take a lot of time to train the model on the dataset to find a good set of hyperparameters.

Result

We have tried 5 different architecture and all of them achieve more than 94% accuracy (see table 1). We choose model 5 as our final model. It has 4,992,586 parameters and achieves 95.66% test accuracy. See table 2 and Appendix 2 for detailed architectures.

Model	Number of blocks	Number of parameters	test accuracy	Final model
ResNet-18	[2,2,2,2]	11,173,962	93.02%	
Model 1	[1,1,1,1]	4,903,242	94.35%	
Model 2	[2,2,4,0]	5,139,018	95.46%	
Model 3	[2,2,4,0]	4,939,902	95.03%	
Model 4	[2,2,4,0]	4,993,025	95.43%	
Model 5	[4,5,3,0]	4,992,586	95.66%	√

Table 1 Original ResNet-18 and all 5 models we designed

Layer(type)	Output Size	Architecture
Conv1	$64 \times 32 \times 32$	kernel size: 3×3 Output channel: 64 Stride=1
Layer1	$64 \times 32 \times 32$	$\begin{bmatrix} 3 \times 3 & 64 \\ 3 \times 3 & 64 \end{bmatrix} \times 4$
Layer2	$128 \times 16 \times 16$	$\begin{bmatrix} 3 \times 3 & 128 \\ 3 \times 3 & 128 \end{bmatrix} \times 5$
Layer3	$256 \times 8 \times 8$	$\begin{bmatrix} 3 \times 3 & 256 \\ 3 \times 3 & 256 \end{bmatrix} \times 3$
Avg_pool	256	Pool Size 8×8
Linear	10	Input channel 256 Output channel 10

Table 2 Architecture for Model 5

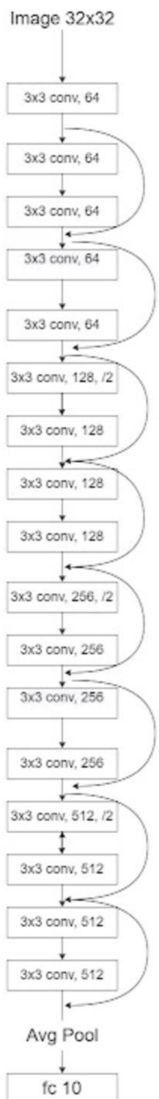
References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Deep Residual Learning for Image Recognition. arXiv:1512.03385

[2] Train CIFAR10 with PyTorch
https://github.com/kuangliu/pytorch-cifar

Appendix

1. ResNet-18 Architecture



2. Final Model Architecture