Choice of Parameters in ResNet

Kun Xiao *kx2090@nyu.edu*
Haotian Zheng *hz2687@nyu.edu*
Luo Chengtao *cl6418@nyu.edu*

Abstract

ResNet (Residual Network), as one of the most commonly used models, is widely implemented and used in deep learning and image classification. Furthermore, the parameters in this model are analytically unknown. To tackle all these analytical unknowns, multiple methods and tools are used in our project. In this report, there are many details regarding ResNet, including but not limited to: cross validation, training data, testing data, and model representation. In the very beginning, the accuracy we obtained was 80.8%, which is inherently insufficient. With the implementation of the sorts of methods we mentioned above, the accuracy was enhanced and refined. Eventually, for the resulting model, we achieved and obtained state-of-the-art performance over a set of different images with a maximum accuracy that is approximately 91.0%.

1. Introduction & Motivation

The importance of image classification could be reflected in many different areas where object identification is quite necessary. First of all, in many most problem-solving cases, the acquisition of access to identifying objects is complex and humanely uninterpretable, which is quite unrealizable for humans to straightforwardly brainstorm on the problem that how to classify objects accurately and efficiently. Using computational tools (ex. Neural Network, Logistic Regression, etc.) is considered as a big step in various classification problems when it comes to improving validity and accuracy. Secondly, in image processing, it is indispensable to classify images for a better performance of computer vision methods, including but not limited to: edge detection, face recognition and image denoising. And last but not least, in the area of information extraction, during the data preprocessing, the data should be well classified prior to being put into the same category.

2. Motivation

For the motivation of this project, it is quite obvious that we have to achieve a higher accuracy. After we analyzed the requirements of this project carefully, several defaults were found existing in the initial model. Among these defaults, low accuracy and insufficient hyper-parameters, these two defaults are quite important with a great influence for the final result. To better understand our project in detail, we are going to discuss these two defaults that how the low accuracy influences the final result and the value of the hyper-parameter impacts the final result. We would discuss the potential modifications that are to improve the low accuray we made as well.

First of all, for the very first default about low accuracy, the initial accuracy was approximately 80%, which led to one false classification in every five images when the template was implemented. And there is a requirement for our project that we have to acquire an accuracy that is greater than 90% at least for our final result. For the initial accuracy, it could hardly satisfy the requirement since it is lower than 90%. Secondly, for the value of the hyper-parameter, it is necessary to be adjusted. Since the construction of the model architecture could not be changed, the value of hyperparameters would be an option for us to adjust to improve the accuracy.

Moreover, to improve the accuracy to be much more effective, methods were tried and implemented including using the data augmentation, adjusting the batch size, and changing the kernel size in the pool. Some of these methods are not effective enough, which only has a slight impact on the accuracy of the final result we got. Some of these methods do change the accuracy of the final result significantly. For instance, we added a method called data augmentation to adjust the accuracy. Data augmentation methods would be preferable for us to choose to be adopted to reach a higher accuracy and a lower loss. Moreover, we implement the adjustment of learning rate from 0.01 to 0.001, which turns the accuray to increase rapidly.

Therefore, it was worthwhile to improve the performance of the model by using more training strategies and changing the parameters.

3. Methodology

In this chapter, multiple methods of the project were discussed. For example, methods containing data collection, data preprocessing and implementation were discussed. And then, we would discuss the data collection part and data preprocessing part indicating the information and preprocessing operation of the cifar-10 dataset. Furthermore, for the implementation section, it would state the structure of a Resnet18 model and hyperparameter adjustment. We will discuss and make some explanations about the figures displayed in our project as well.

3.1 Introduction of the Dataset Called cifar-10 with 10 different classes

Alex Krizhevsky stated that the CIFAR-10, as a public dataset, is made of 60,000 (32x32) color images in ten different classes, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck., and each class contains 6000 images. Furthermore, the dataset is divided into two sections, 50,000 training images and 10,000 test images. All these elements in this dataset are divided into five different training batches and one test batch. Each of them consists of 10,000 images. Additionally, the test batch is made of 1000 randomly-selected images from each class. Moreover, the training batches contain the remaining images in random order; however, some training batches might contain more images from one class than the other. In addition, the training batches contain exactly 5000 images from each class. Figure 3.1.1 shown below presents the general information of this dataset. [1]

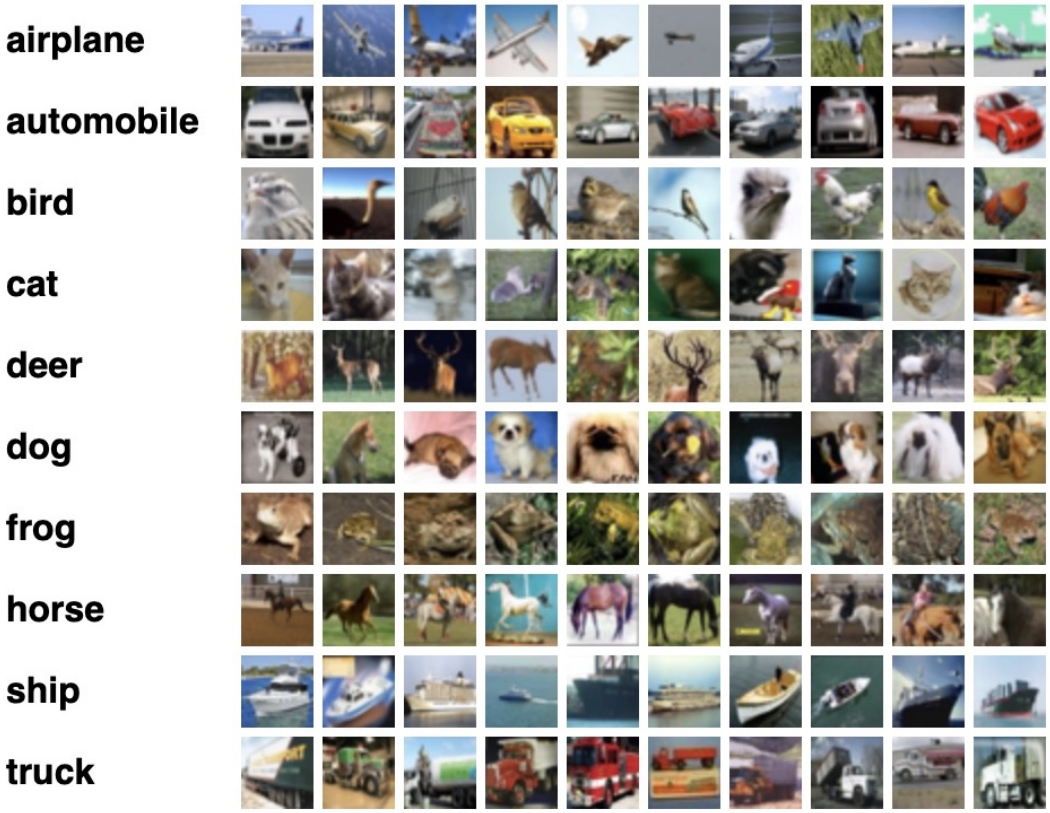


Figure 3.1.1

3.2 Data Preprocessing

To improve the performance of the model, data preprocessing was quite necessary, and in this

case, data augmentation strategies were used in the training dataset, including the methods RandomCrop, RandomHorizontalFlip and RandomGrayscale. The method RandomCrop that contained 2 parameters, the target pixel and padding, resized a subset of an origin image to a specific size, and in this project, the target pixel that was selected was 32, the origin pixel of the image and the padding was 4. The method RandomHorizonFlip randomly selected some images from the origin dataset with a given probability and flipped the selected images horizontally. In this project, the probability was 0.5. For the method RandomGrayscale, it randomly transferred the colored images to gray images with a given probability. In this project, the probability was set to be 0.5.

3.3 Implementation

This section introduced the structure of the Resnet18 model, values of different parameters and some specific parameters that were added into the model, which were momentum and weight decay. Moreover, the reason for not adopting dropout was stated.

3.3.1 The ResNet-18 is a multi-layered residual network with 17 convolutional layers and 1 average pooling layer. There are 4 residual layers, each of which contains at least one residual block of two convolutional layers, excluding the input layer. The convolution layer has a fixed number of channels and one kernel of size K × K. Every output of each block has a skip connection from the previous block's input. The output of each block has an activation function ReLU, except for the last block. The final output is activated via the Softmax function. The average pooling layer has a kernel of size P × P. The figure described the basic structure of the model.

3.3.2 Adjusting the values of parameters could improve the performance of neural networks

In this project, learning rate, batch size and channel were discussed. And the table described different parameters and the related value candidates.

Learning rate	0.01	0.005	0.001
Batch size	32	64	128
Channel	42	64	80

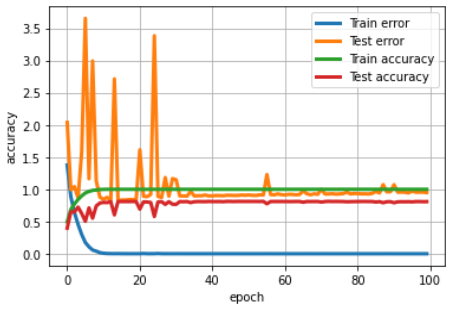
3.3.3 The momentum and the weight decay are set to 0.9 and 5e-4 in the stochastic gradient

descent optimizer. Furthermore, the dropout is removed from the model since the batch-normalization layer was added, which prevents the predicted outputs from being overfitted and adding dropout may reduce accuracy.

4. Result and Analysis

In this chapter, training results of the improved model would be shown below and compared to the figure which was the result of the initial model without adjustment. Table indicated the initial value choice for the model. Section 4.1 showed the results after data augmentation, and Section 4.2 described the results of parameter choices, including the value of learning rate, batch size, block, channel, kernel size of conv layer and kernel size in pooling layer, were shown as the loss and accuracy curve as well as the graphes, and the related analysis was stated. Section 4.3 summarized the final result including the final architecture, the number of trainable parameters, loss and accuracy curve and testing error and accuracy.

Figure : training result of initial model




```
87
Epoch 87, Train loss 0.000120, Test loss 1.073363, Train acc 1.000000, Test acc 0.791600
88
Epoch 88, Train loss 0.000222, Test loss 0.960645, Train acc 0.999980, Test acc 0.806628
89
Epoch 89, Train loss 0.000113, Test loss 0.962424, Train acc 1.000000, Test acc 0.807822
90
Epoch 90, Train loss 0.000127, Test loss 1.071712, Train acc 1.000000, Test acc 0.791501
91
Epoch 91, Train loss 0.000145, Test loss 0.955041, Train acc 1.000000, Test acc 0.805334
92
Epoch 92, Train loss 0.000096, Test loss 0.957037, Train acc 1.000000, Test acc 0.807026
93
Epoch 93, Train loss 0.000086, Test loss 0.953548, Train acc 1.000000, Test acc 0.806330
94
Epoch 94, Train loss 0.000096, Test loss 0.944070, Train acc 1.000000, Test acc 0.807723
95
Epoch 95, Train loss 0.000104, Test loss 0.973704, Train acc 1.000000, Test acc 0.804638
96
Epoch 96, Train loss 0.000090, Test loss 0.955004, Train acc 1.000000, Test acc 0.809614
97
Epoch 97, Train loss 0.000077, Test loss 0.954974, Train acc 1.000000, Test acc 0.809713
98
Epoch 98, Train loss 0.000076, Test loss 0.956690, Train acc 1.000000, Test acc 0.807623
99
Epoch 99, Train loss 0.000069, Test loss 0.949431, Train acc 1.000000, Test acc 0.808818
```

Learning rate	Batch size	Block	Channel
0.01	64	[2,2,2,2]	64

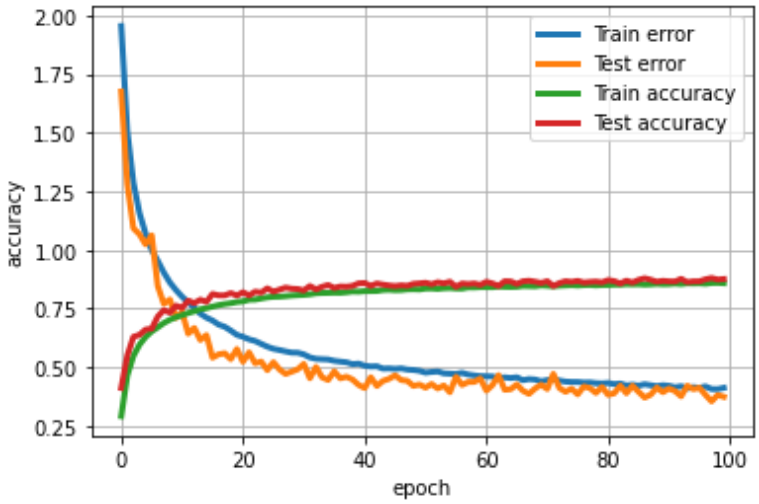
-

4.1 Data Augmentation

The data augmentation methods that were implemented were RandomCrop, RandomHorizonFLip

and RandomGrayscale. The Figure attached below showed a more stable loss and accuracy curves compared to the ones of the initial model, and both the training accuracy and testing accuracy had a great increase from 0.80 to 0.87; however, this accuracy was not good enough to meet the requirement. Hence, there are more improvements needed to be made to reach a higher accuracy by adding other methods.

Figure :result of using data augmentation




```
Epoch 83, Train loss 0.421626, Test loss 0.388598, Train acc 0.854020, Test acc 0.868830
Epoch 84, Train loss 0.424778, Test loss 0.420109, Train acc 0.853281, Test acc 0.860569
Epoch 85, Train loss 0.420801, Test loss 0.394663, Train acc 0.853780, Test acc 0.871915
Epoch 86, Train loss 0.426804, Test loss 0.369990, Train acc 0.852122, Test acc 0.878881
Epoch 87, Train loss 0.422587, Test loss 0.382377, Train acc 0.853121, Test acc 0.872412
Epoch 88, Train loss 0.418535, Test loss 0.412632, Train acc 0.855099, Test acc 0.864053
Epoch 89, Train loss 0.422710, Test loss 0.392807, Train acc 0.853621, Test acc 0.868033
Epoch 90, Train loss 0.420233, Test loss 0.408716, Train acc 0.854120, Test acc 0.865645
Epoch 91, Train loss 0.413701, Test loss 0.397749, Train acc 0.856318, Test acc 0.865147
Epoch 92, Train loss 0.418192, Test loss 0.375324, Train acc 0.854200, Test acc 0.876294
Epoch 93, Train loss 0.409316, Test loss 0.420673, Train acc 0.857377, Test acc 0.862361
Epoch 94, Train loss 0.414493, Test loss 0.405695, Train acc 0.856738, Test acc 0.865744
Epoch 95, Train loss 0.411580, Test loss 0.412007, Train acc 0.857317, Test acc 0.865744
Epoch 96, Train loss 0.417472, Test loss 0.381579, Train acc 0.855519, Test acc 0.874403
Epoch 97, Train loss 0.406402, Test loss 0.353913, Train acc 0.858816, Test acc 0.880076
Epoch 98, Train loss 0.406001, Test loss 0.380825, Train acc 0.859375, Test acc 0.873308
Epoch 99, Train loss 0.411478, Test loss 0.373297, Train acc 0.857836, Test acc 0.875597
torch.return_types.max(
values=tensor([ 8.9686,  5.6093,  7.8231,  9.3273,  9.9837,  4.0641,  6.8967,  6.6708,
         7.2406,  6.8415,  6.4041,  3.4051,  6.4611,  7.8706,  9.0741, 10.4333],
       device='cuda:0', grad_fn=<MaxBackward0>),
indices=tensor([7, 5, 8, 0, 8, 4, 7, 0, 3, 5, 3, 8, 3, 5, 1, 7], device='cuda:0'))
tensor([7, 5, 8, 0, 8, 2, 7, 0, 3, 5, 3, 8, 3, 5, 1, 7], device='cuda:0')
tensor(0.1216, device='cuda:0', grad_fn=<NllLossBackward0>)
```

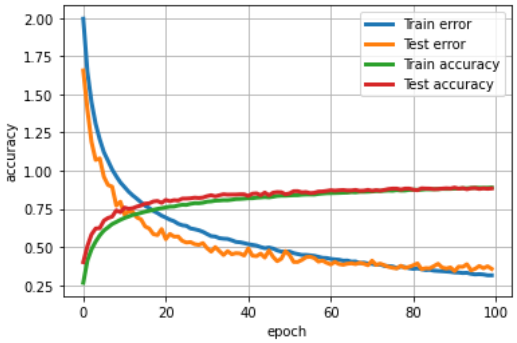
4.2 Hyperparameter value adjustment

4.2.1 Learning rate

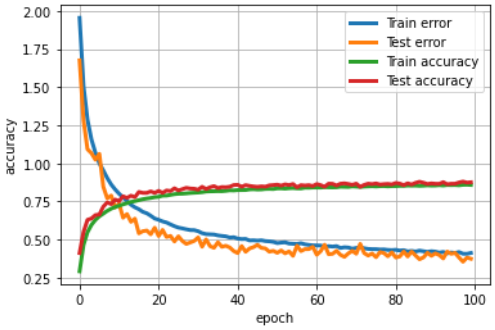
The value choices of learning rate were 0.01, 0.005, 0.001. The related loss and accuracy curves were shown as figures attached below. From the loss and accuracy curves, it was hard to tell which value was suitable since the trend the curves moved and final point the curves reached were similar. Therefore, the table describing the testing accuracy determined the value choice, and it can be recognized that the value of 0.001. It had a higher testing accuracy than the other two values.

Therefore, the optimal value of learning rate was 0.001.

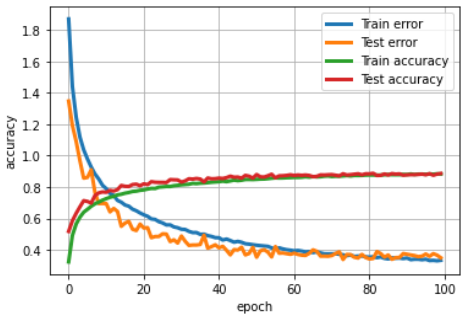
lr=0.01



lr=0.005



lr=0.001

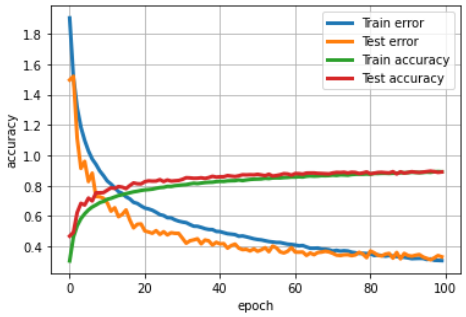


Learning rate	0.01	0.005	0.001
Testing accuracy	0.8755	0.8836	0.8865

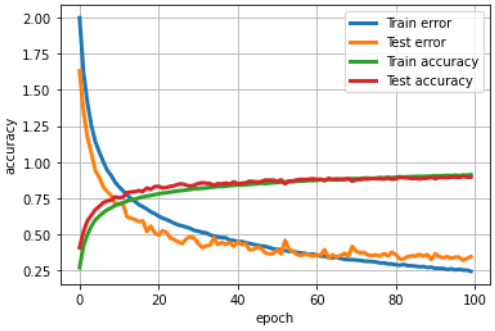
4.2.2 Batch size

The value choice of batch size were 32, 64, 128. And the related loss and accuracy curves were shown as figures shown below. The same condition as the learning rate, the table describing the testing accuracy determined the value choice, and the value of 64 had a higher testing accuracy than the other 2. Therefore, the optimal value of batch size was 32.

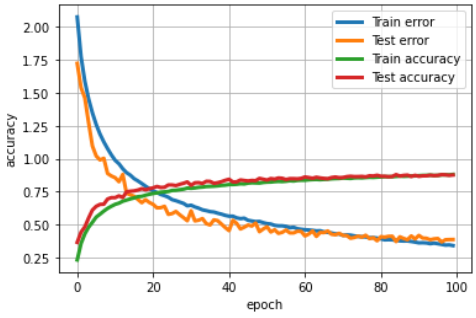
batch size=32



batch size=64



batch size=128

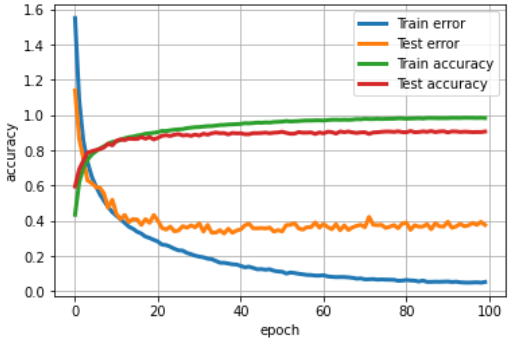


Batch size	32	64	128
Testing accuracy	0.8965	0.8909	0.8766

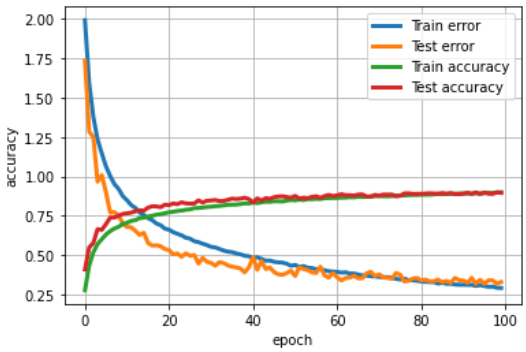
4.2.3 Channel

The value choices of the channel were 42, 64, 80. From the loss and accuracy curves, there existed a greater gap between training and testing error curve in value 42 than the other 2, however, the value 42 reached a higher testing accuracy from the table. The testing error curve and training error curve of value 42 did not go in opposite directions, leading to no overfitting. Therefore, 42 might be the optimal choice for the channel based on our testing.

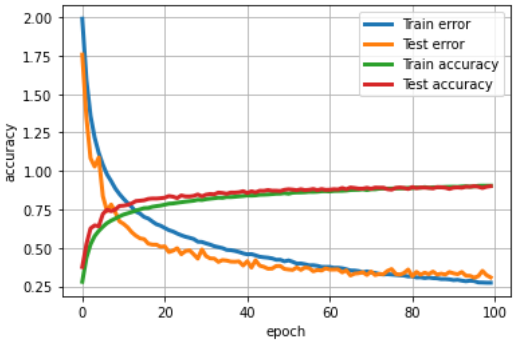
channel=42



channel=64



channel=80

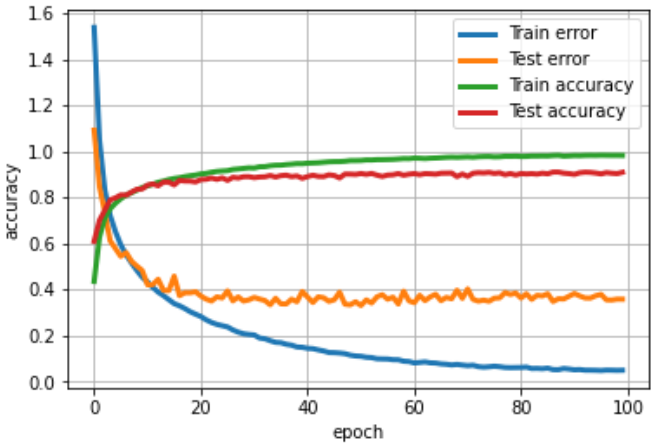


Channel	42	64	80
Testing accuracy	0.9057	0.8947	0.9016

4.3 Summary

The final architecture of the model was similar to the origin. However, data augmentation strategies were used and hyperparameters values were adjusted, a better performance of the model was achieved. The error and accuracy curve was shown as the figures shown below. And the final test error and test accuracy are shown in the table as well. The number of trainable parameters was 4815940, which was less than 5M. Moreover, in order to reduce the number of parameters to meet the requirement that the number of trainable parameters must be less than 5M, We changed the in_planes value from 64 to 42. After adjusting the in_planes value, the final result was shown as the curve figure and the table attached below.

final result



Final training error	**0.050407**
Final testing error	**0.358776**
Final training accuracy	**0.982726**
Final testing accuracy	**0.909545**

5. References

[1] Alex Krizhevsky. (n.d.). *The CIFAR-10 dataset*. CIFAR-10 and CIFAR-100 datasets. Retrieved March 24, 2022, from https://www.cs.toronto.edu/~kriz/cifar.html