Deep Learning

MNIST Digits Classification using PyTorch

Author: Yantian Luo

Institute: Electronic Engineering

ID: 2018310742

Update: April 24, 2019

Contents

Chapter 1 Introduction

MNIST digits dataset is a widely used database for image classification in machine learning field. It contains 60,000 training samples and 10,000 testing samples. Each sample is a 784×1 column vector, which is transformed from an original 28×28 pixels grayscale image.

In this homework, we will continue working on MNIST digits classification problem by utilizing Pytorch framework to implement neural networks. Pytorch provides good abstraction for different modules, and its auto-differentiation feature can save us from the backward details.

Chapter 2 Algorithm Design

In the last two homework, we have implement Linear layer, activation layer, convolution layer and loss layer, in this homework, we only implement an important technique discovered recently and utilize Pytorch framework to implement others.

For every input x_i across a mini-batch, the output can be computed as :

$$BN(x_i) = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.1}$$

where γ and β are learnable parameters, μ_B is the mean of the mini-batch and σ^2 is the variance of the mini-batch.

When we test our samples one by one, the normalization may not work because we don't have mini- batch concept this time. Therefore, during training we should keep running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default momentum of 0.9, which can be mathematically expressed as

$$\begin{aligned} \hat{\mu}_{new} &= momentum \times \hat{\mu} + (1 - momentum) \times \mu_t \\ \hat{\sigma}^2_{new} &= momentum \times \hat{\sigma}^2 + (1 - momentum) \times \sigma_t^2 \end{aligned} \tag{2.2}$$

where where $\hat{\mu}, \hat{\sigma}^2$ are the estimated statistic and μ_t, σ_t^2 is the new observed value.

Chapter 3 Results

3.1 MLP

In this section, we use MLP to work on MNIST digits classification. The network structure is shown in Figure 3.1. And we also compare the results of MLP with BatchNorm and the results without the BatchNorm.

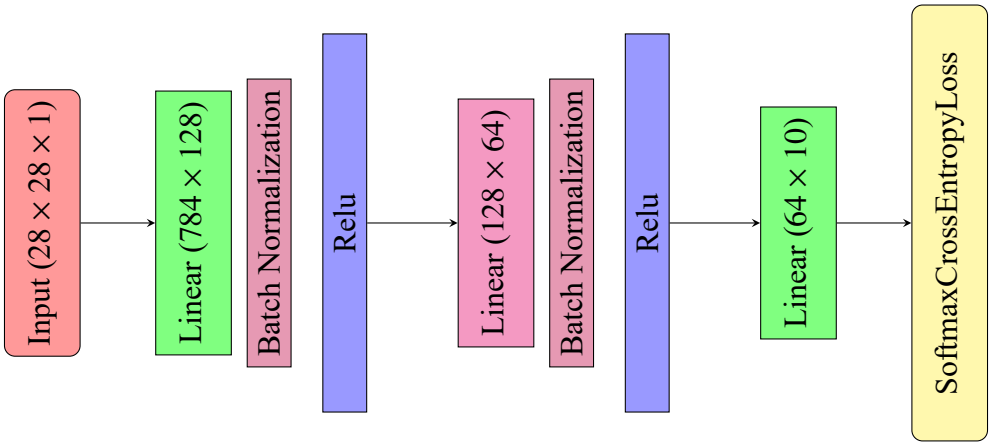

Figure 3.1: MLP Network Structure

The training arguments is as follow:

```
config = {
'learning_rate': 0.1,
'weight_decay': 0.0001,
'momentum': 0.9,
'batch_size': 100,
'max_epoch': 100,
'disp_freq': 50,
}
```

3.1.1 Results with BatchNorm

From the results we can find that only after 5000 iterations, the training accuracy is up to 90%, which is very fast converging, and after one epoch, the test accuracy can

up to 96.56%. Then we draw the train accuracy curve and train loss curve as shown in Figure 3.2 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in Figure 3.3.

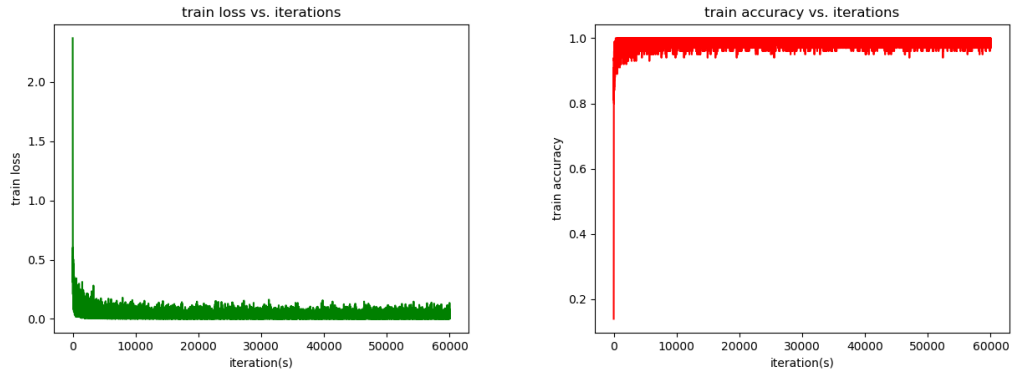

Figure 3.2: train loss curve and train accuracy curve using MLP with BatchNorm

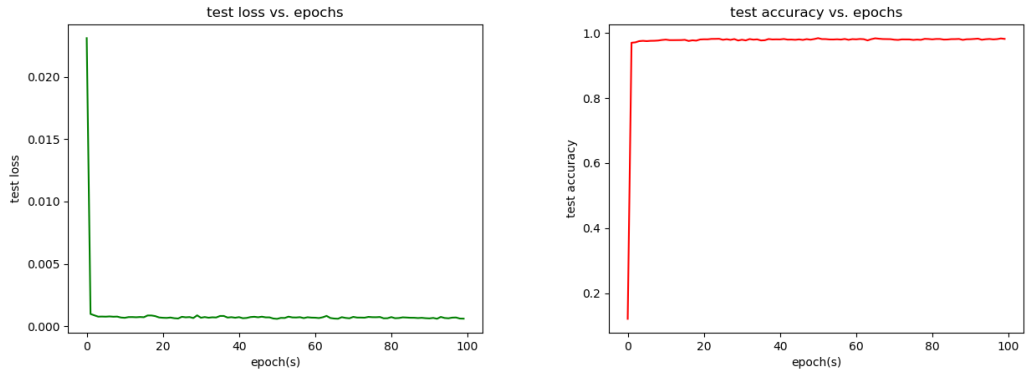

Figure 3.3: test loss curve and test accuracy curve using MLP with BatchNorm

3.1.2 Results without BatchNorm

From the results we can find that after 15000 iterations, the training accuracy can up to 90%, which is slower than the results with BatchNorm, and after one epoch, the test accuracy can up to 95.46%, which is also smaller than the results with BatchNorm. Then we draw the train accuracy curve and train loss curve as shown in Figure 3.4 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in Figure 3.5.

3.1.3 Comparisons

Here we list the comparisons of the last two results as shown in Table 3.1. From the table, we can find that:

- The result of MLP with BN has faster converging speed than MLP without BN;

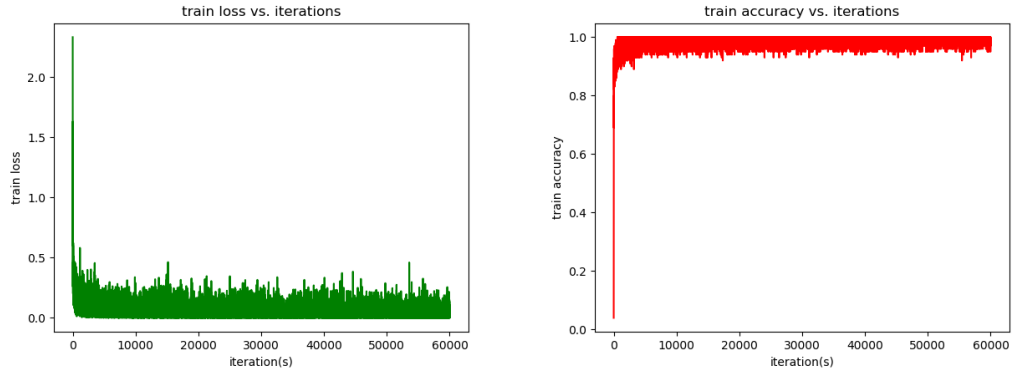

Figure 3.4: train loss curve and train accuracy curve using MLP without BatchNorm

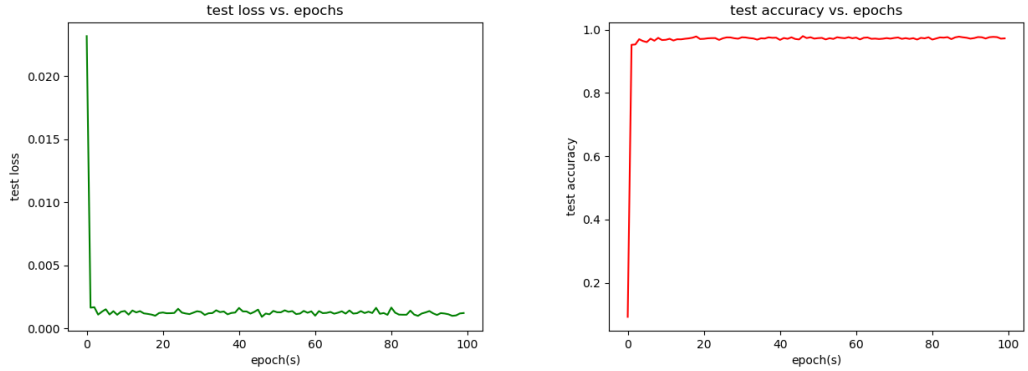

Figure 3.5: test loss curve and test accuracy curve using MLP without BatchNorm

- The test accuracy of MLP with BN is also bigger than MLP without BN, and the test loss of MLP with BN is less than MLP without BN;
- The parameters number of MLP with BN is a little more than MLP without BN, because the BN layer has trainable parameters, but it's a very little number;
- The training time of two cases is close and MLP without BN has a little shorter training time than MLP with BN because MLP with BN has a little more parameters than MLP without BN;

Therefore, we can find, using BatchNorm in MLP is better than normal MLP.

Table 3.1: The comparisons of the results with BatchNorm and without BatchNorm

Cases	Converge speed	test Acc	test loss	training time	#(parameters)
With BN	faster	0.983	0.0006	17min	109,568
Without BN	slower	0.976	0.0010	14min	109,184

3.2 CNN

In this section, we use CNN to work on MNIST digits classification. The network structure is shown in Figure 3.6. And we also compare the results of CNN with Batch-

Norm and the results without the BatchNorm.

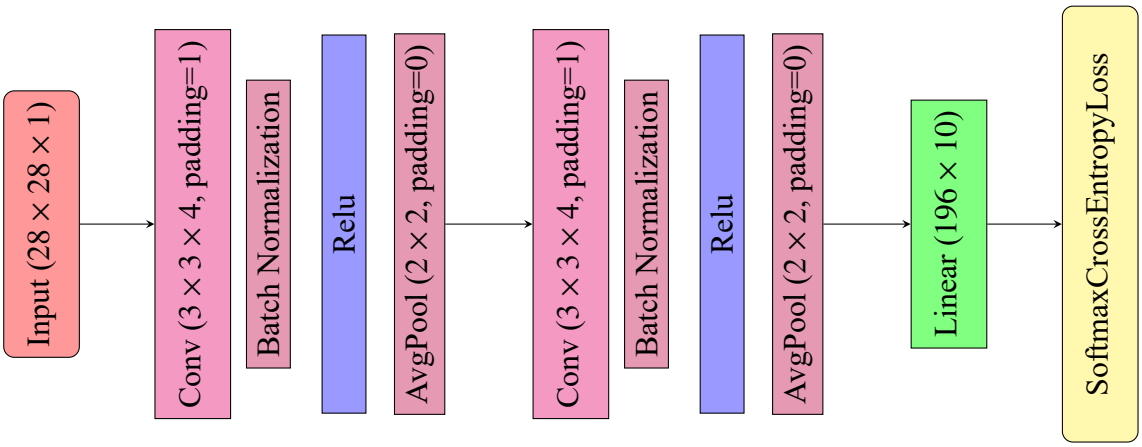

Figure 3.6: CNN Network Structure

The training arguments is as follow:

```
config = {
'learning_rate': 0.1,
'weight_decay': 0.0001,
'momentum': 0.9,
'batch_size': 100,
'max_epoch': 100,
'disp_freq': 50,
}
```

3.2.1 Results with BatchNorm

From the results we can find that only after 10000 iterations, the training accuracy is up to 92%, which is very fast converging, and after one epoch, the test accuracy can up to 96.83%. Then we draw the train accuracy curve and train loss curve as shown in Figure 3.7 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in Figure 3.8.

3.2.2 Results without BatchNorm

From the results we can find that after 35000 iterations, the training accuracy can up to 92%, which is slower than the results with BatchNorm, and after one epoch, the test accuracy can up to 95.95%, which is also smaller than the results with BatchNorm. Then we draw the train accuracy curve and train loss curve as shown in Figure 3.9 and

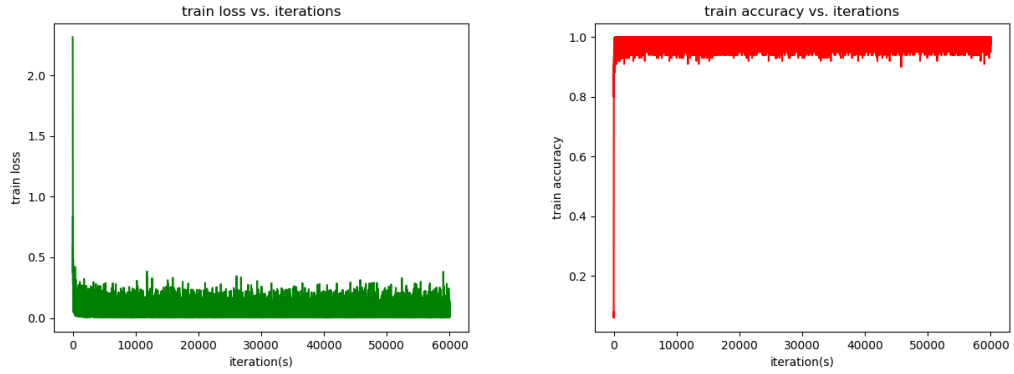

Figure 3.7: train loss curve and train accuracy curve using CNN with BatchNorm

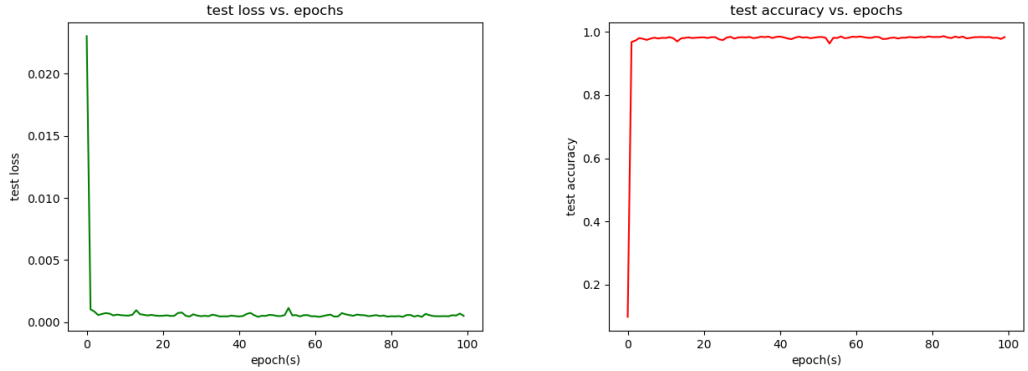

Figure 3.8: test loss curve and test accuracy curve using CNN with BatchNorm

we draw the test accuracy curve and test loss curve with respect to epoch as shown in Figure 3.10.

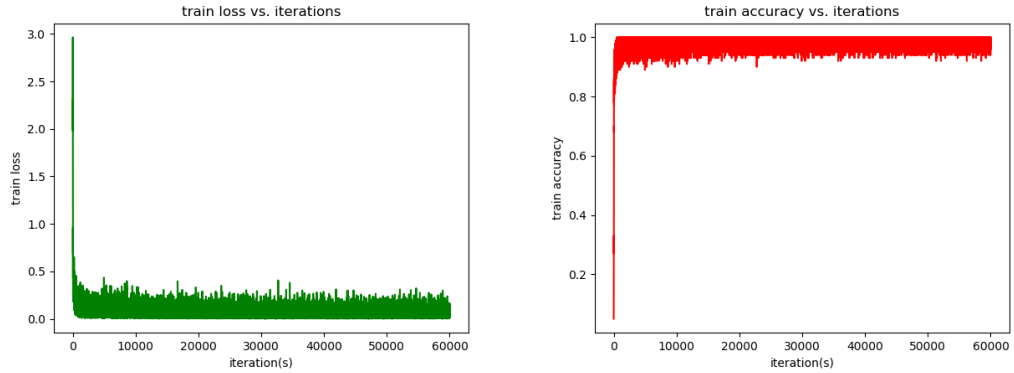

Figure 3.9: train loss curve and train accuracy curve using CNN without BatchNorm

3.2.3 Comparisons

Here we list the comparisons of the last two results as shown in Table 3.2. From the table, we can find that:

- The result of CNN with BN has faster converging speed than CNN without BN;

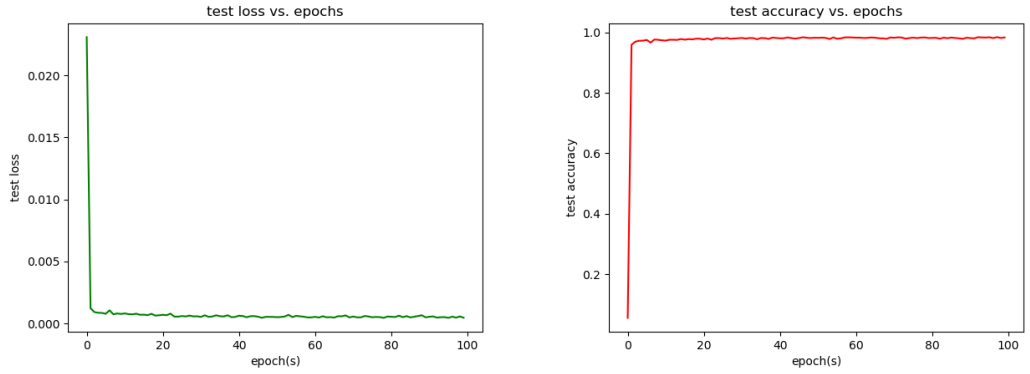

Figure 3.10: test loss curve and test accuracy curve using CNN without BatchNorm

- The test accuracy and loss of the two experiments is close;
- The parameters number of CNN with BN is a little more than CNN without BN, because the BN layer has trainable parameters, but it's a very little number.
- The training time of CNN without BN is a shorter than CNN with BN because CNN with BN has a little more parameters than CNN without BN;

Therefore, we can find, using BatchNorm in CNN is better than normal CNN.

Table 3.2: The comparisons of the results with BatchNorm and without BatchNorm

Cases	Converge speed	test Acc	test loss	training time	#(parameters)
With BN	faster	0.983	0.0005	38min	2159
Without BN	slower	0.983	0.0005	24min	2143

3.3 Conclusions

From the experiments we can find that CNN with BatchNorm has best test accuracy and test loss, and parameters number is small which is not easy to overfit. But the training time is very longer. MLP with BatchNorm has shorter training time and test accuracy is also relatively good. But the parameters number is very large which is easy to result in overfitting. Therefore, choose CNN with BatchNorm is a better way to slove the MNIST digits classification.

Chapter 4 mini-ResNet

In this chapter, we design our own mini-ResNet to solve MNIST digits classification task.

4.1 ResNet block

The most important mudule of ResNet is ResNet block, which is shown in Figure 4.1. From the figure, we can find the forward operation can be computed as follow:

$$y = \mathcal{F}(x) + x \tag{4.1}$$

And the backward operation can be computed by pytorch automaticlly.

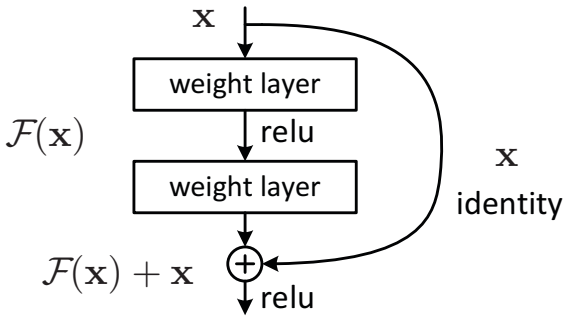

Figure 4.1: Residual learning: a building block

The two kinds of ResNet block are shown in Figure 4.2 and in MNIST digits classification task we only use the left one.

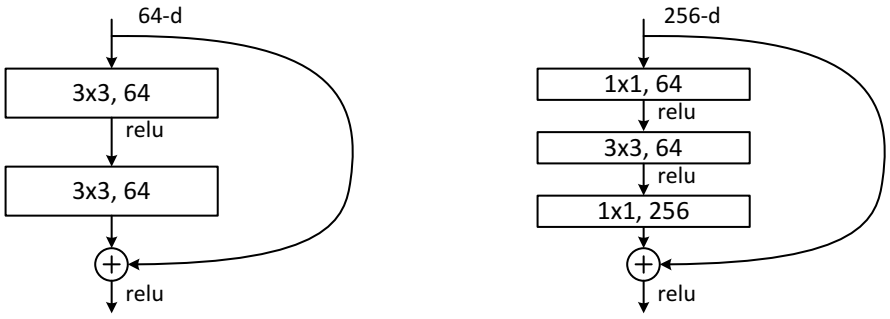

Figure 4.2: Two kinds of deeper residual function F

4.2 Network Structure

Network Structure (eight layers) designed for MNIST digits classification task is shown in Figure 4.3. Left is the overall structure and right is the detailed structure of one ResNet block.

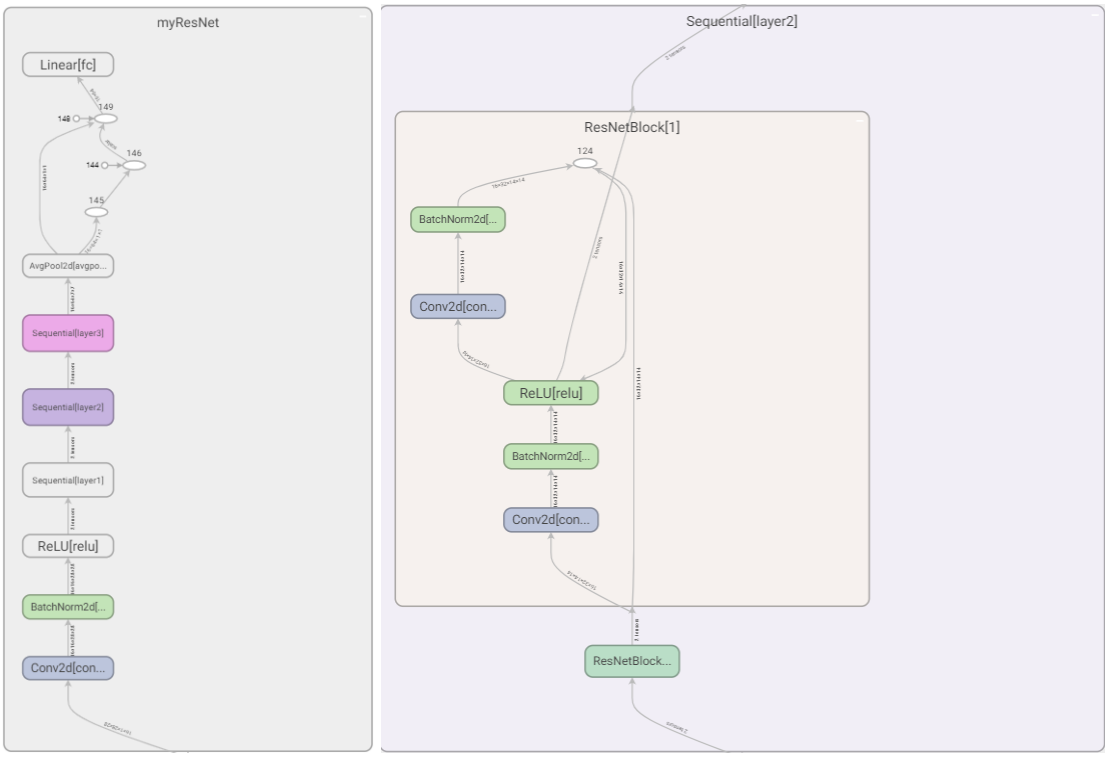

Figure 4.3: MyResNet Network Structure

4.3 Results

Only after 5 epochs, test accuracy can up to 99.06%. We can find it's very powerful. But the training time is too long for my cpu computer. At last, we train for 60 epochs and the best test accuracy is 99.46% and the best test loss is 0.0004, which is much better than last experiments before. At last we draw the train accuracy curve and train loss curve as shown in Figure 4.4 and we draw the test accuracy curve and test loss curve with respect to epoch as shown in Figure 4.5.

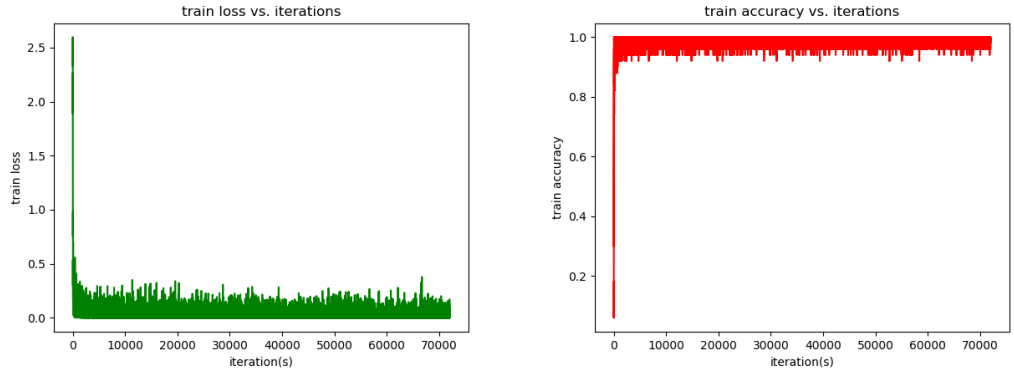

Figure 4.4: train loss curve and train accuracy curve using mini-ResNet

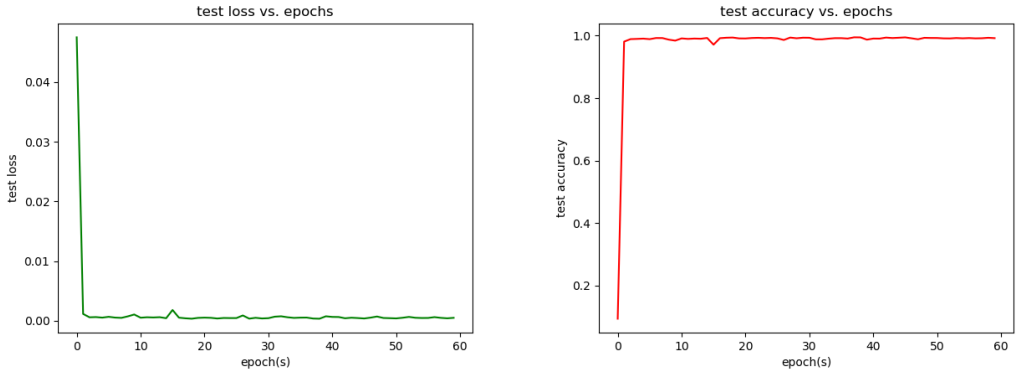

Figure 4.5: test loss curve and test accuracy curve using mini-ResNet