ELEC 576 / COMP 576: Fall 2021: Assignment 2

Youssaf Menacer

October 27, 2021

1 Visualizing a CNN with CIFAR10

1.1 CIFAR10 Dataset

By using the zipped file CFAIR10 and the provided code trainCifarStarterCode.py, we import all the images and perform additional pre-processing, such as dividing by 255 (normalizing),and we use one-hot encoding for labels.

1.2 Train LeNet5 on CIFAR10

Here we implement a LeNet5 and train it on CIFAR10.
Here is the configuration of LeNet5: - Convolutional layer with kernel 5 x 5 and 32 fi

lter maps followed by ReLU. - Max Pooling layer subsampling by 2. - Convolutional layer with kernel 5 x 5 and 64 fi

lter maps followed by ReLU. - Max Pooling layer subsampling by 2. - Fully Connected layer that has input 7x7x64 and output 1024. - Fully Connected layer that has input 1024 and output 10 (for the classes). - Soft-max layer (Soft-max Regression + Soft-max Non-linearity). We plot train/test accuracy and train/test loss and get the following result in figure 1.

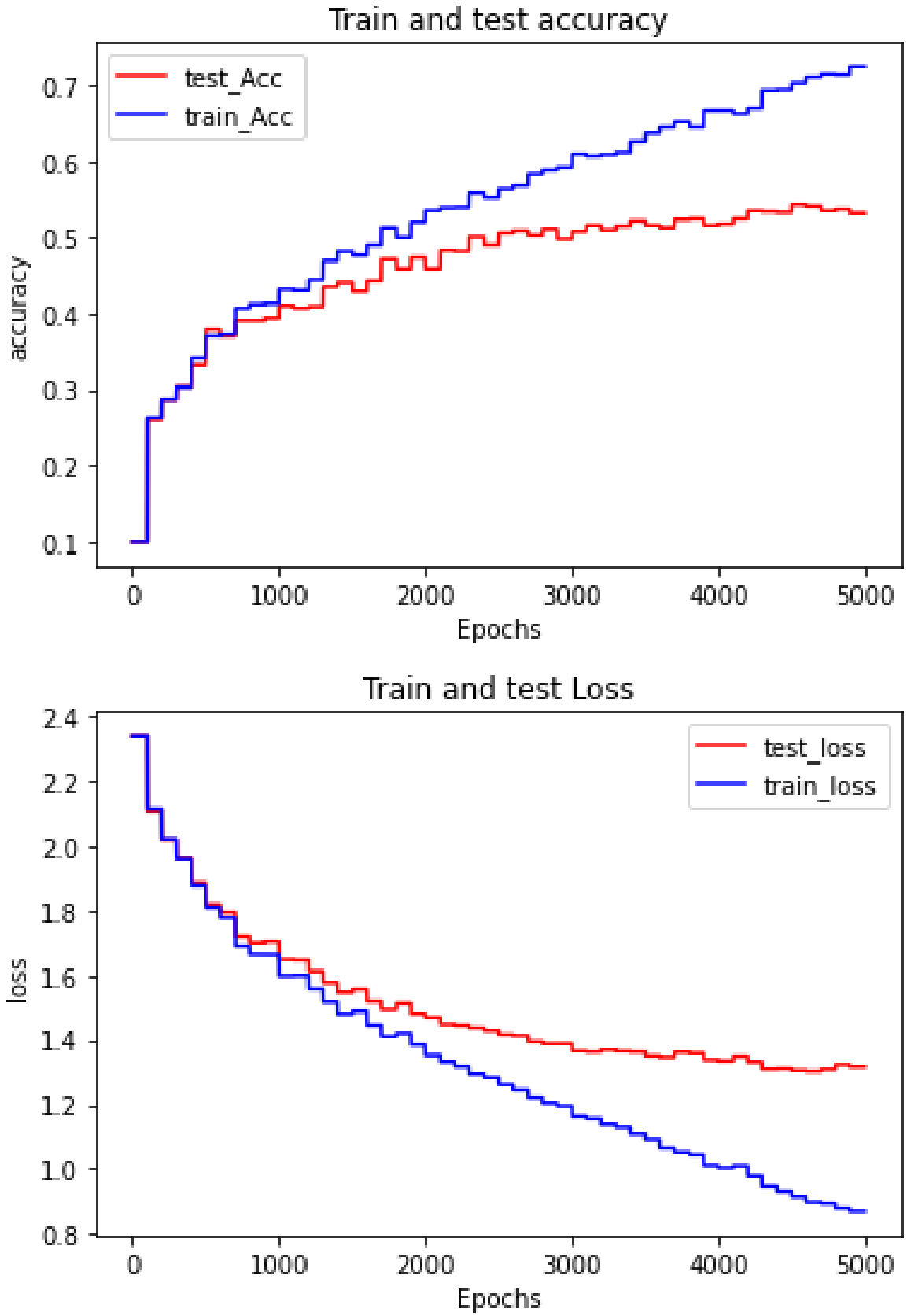


Figure 1: Train/test accuracy and train/test loss of Grayscale CFAIR dataset

For this network, we have CROSS ENTROPY loss function and ADAM OPIMIZER. We use 5500 iterations and a learning rate of $1e-4$ to train LeNet5.
In figure 1, we see that training is going well, but when it gets close to epoch 1000, it performs very poorly. The big gap between test accuracy and train accuracy shows that the model is over fitting.
Figure 2 shows the weights of the first convolutional layer on CIFAIR10 Grayscale for LeNet5.

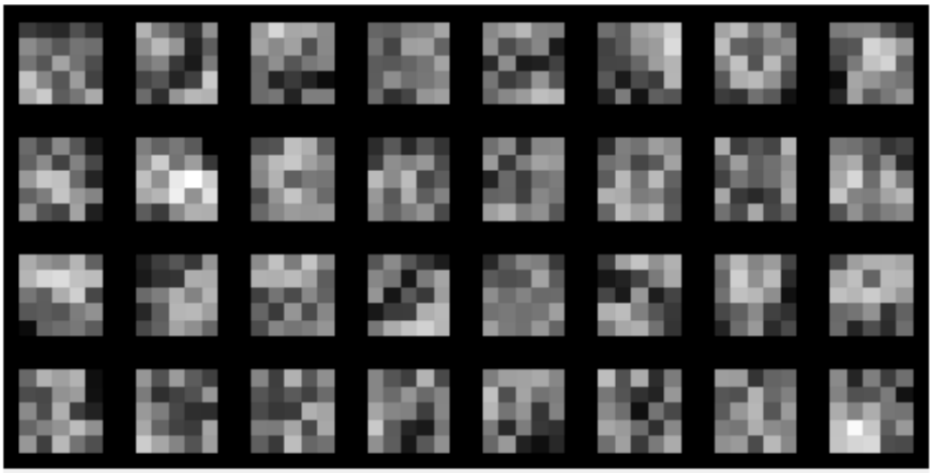
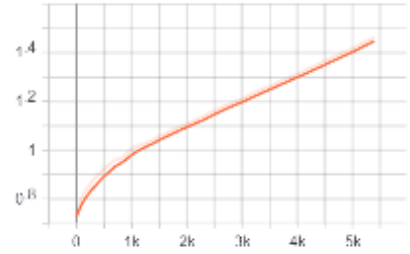
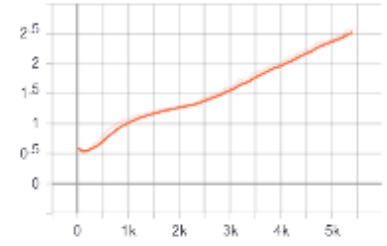
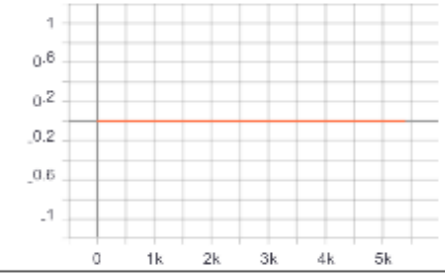
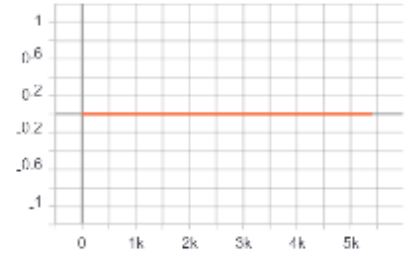
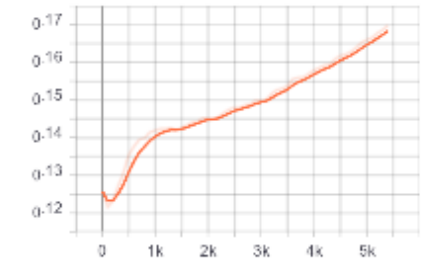
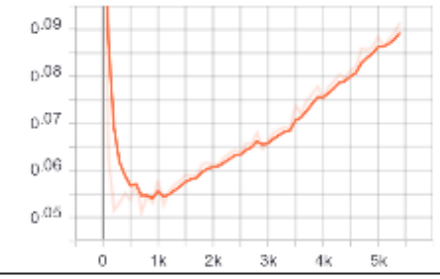
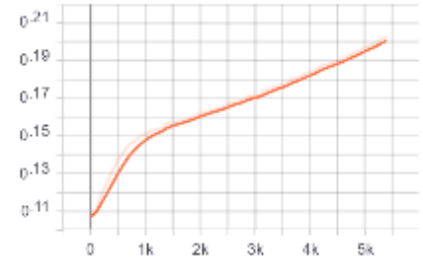
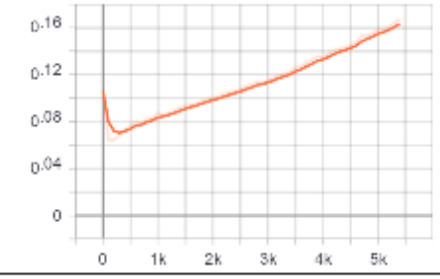
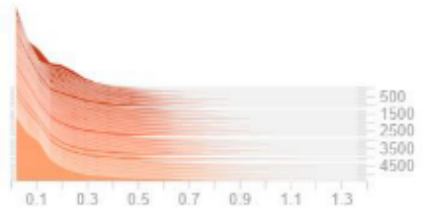
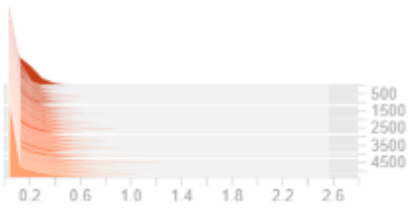


Figure 2: Weights of the first convolutional layer on CIFAIR10 Grayscale for LeNet5

The following table shows statistics of activations in convolutional layers on the test images for LeNet5 network.

Statistics of activations in convolutional layers on the test images on LeNet5 network	
Convolutional Layer 1	Convolutional Layer 2
Conv1: Max of activation after ReLU 	Conv2: Max of activation after ReLU 
Conv1: Min of activation after ReLU 	Conv2: Min of activation after ReLU 
Conv1: Mean of activation after ReLU 	Conv1: Mean of activation after ReLU 
Conv1: Std of activation after ReLU 	Conv1: Std of activation after ReLU 
Conv1: Histogram of activation after ReLU 	Conv1: Histogram of activation after ReLU 

Now, we try
- Convolutional layer with kernel 5 x 5 and 32 filter maps followed by ReLU. - Max Pooling layer subsampling by 2 and Batch normalization. - Convolutional layer with kernel 3 x 3 and 64 filter maps followed by ReLU. - Max Pooling layer subsampling by 2. Then, Batch normalization and Dropout. - Convolutional layer with kernel 3 x 3 and 64 filter maps followed by ReLU. - Max Pooling layer subsampling by 2 and Batch normalization. - Fully Connected layer that has input 4x4x64 and output 256. Then Batch normalization and Dropout. - Fully Connected layer that has input 256 and output 10. - Soft-max layer (Soft-max Regression + Soft-max Non-linearity).

we get the following figure

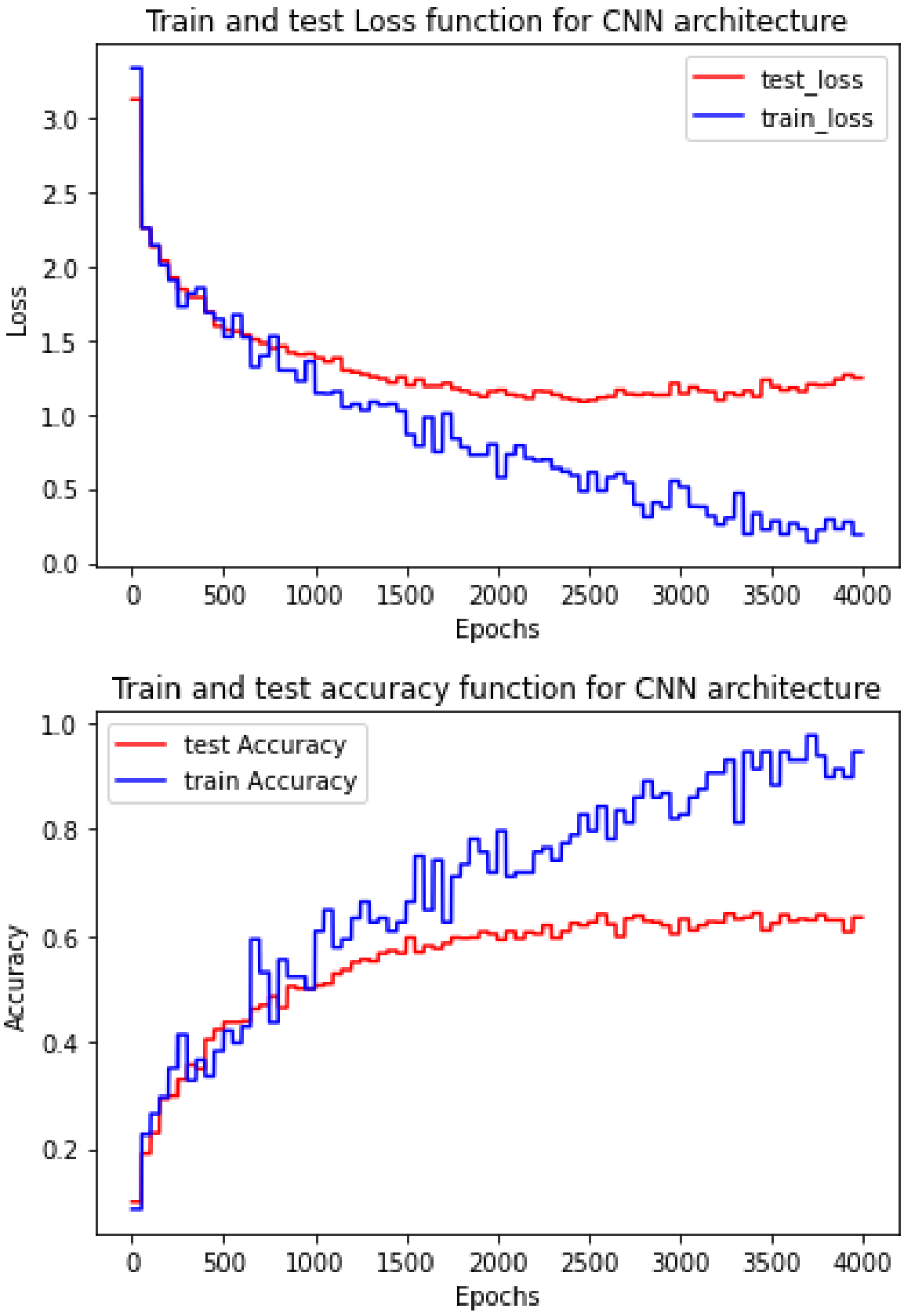


Figure 3: Train/test accuracy and train/test loss of Grayscale CFAIR dataset

Comparing figure 3 to figure 1, we see clearly that the test accuracy gets better, but we still have over fitting.
We use the new CNN to check the weights of the first convolutional layer on CIFAIR10 Grayscale for LeNet5, figure 4.

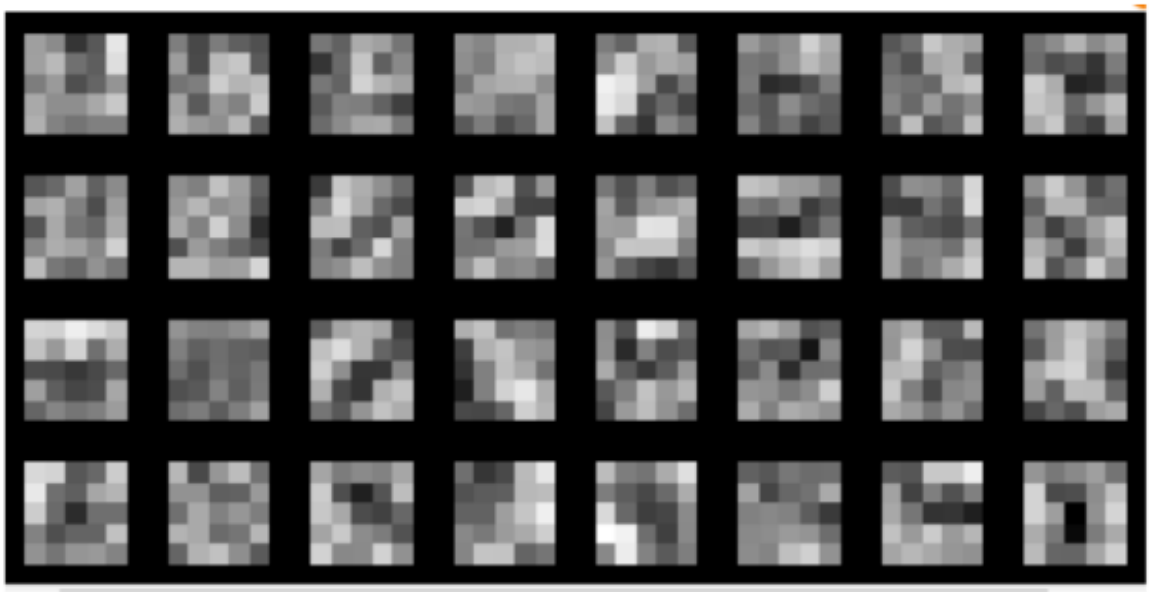
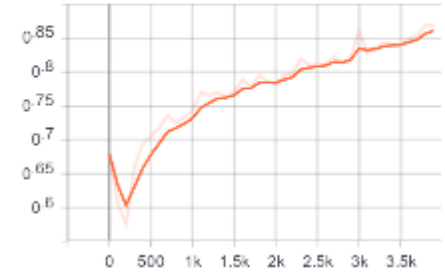
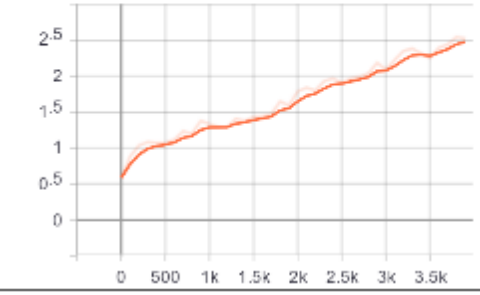
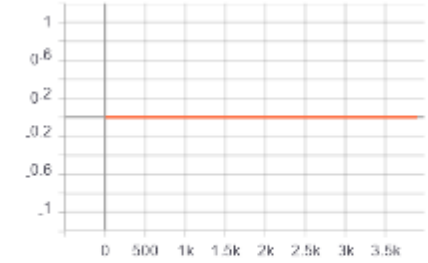
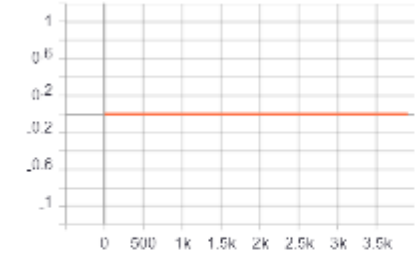
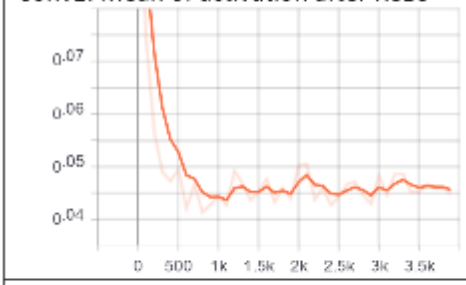
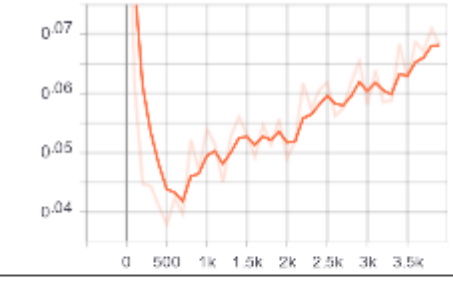
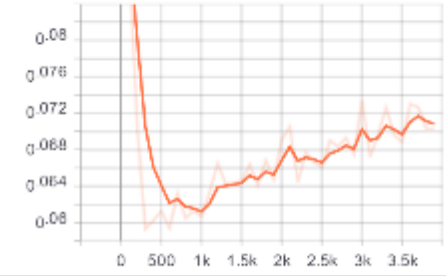
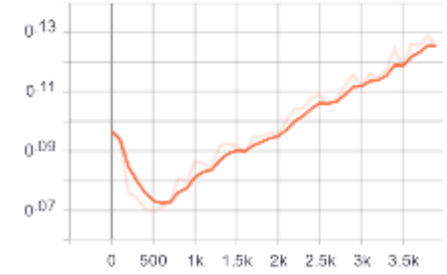
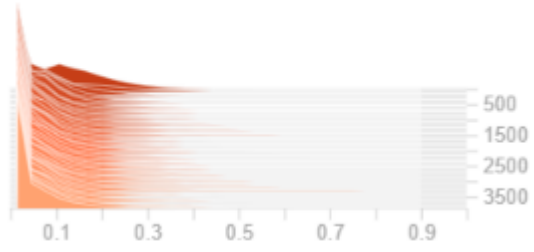
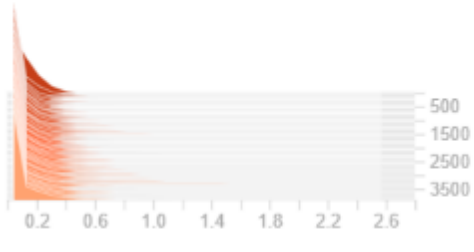


Figure 4: Weights of the first convolutional layer on CIFAIR10 Grayscale for LeNet5

We also show the statistis for this new CNN, so we get the following table:

Statistics of activations in convolutional layers on the test images on new CNN network

Convolutional Layer 1	Convolutional Layer 2
Conv1: Max of activation after ReLU	Conv2: Max of activation after ReLU
	
Conv1: Min of activation after ReLU	Conv2: Min of activation after ReLU
	
Conv1: Mean of activation after ReLU	Conv1: Mean of activation after ReLU
	
Conv1: Std of activation after ReLU	Conv1: Std of activation after ReLU
	
Conv1: Histogram of activation after ReLU	Conv1: Histogram of activation after ReLU
	

2 Visualizing and Understanding Convolutional Networks

Summary of the paper

The author talks about a visualization technique, using a Deconvolutional Network, which gives insight of large Con-Net models. He uses visualization technique to test the ImageNet model on other dataset and realized this latter generalizes well on other datasets. Using a deconvnet, author shows a method to map the feature activities of a convnet back to the input pixel space, realizing what input pattern originally caused a given activation in the feature maps. In this network, we attach s deconvnet to each of the convnet layer providing a continuous push back to image pixels by repeatedly applying unpool and filter to reconstruct the activity in the layer responsible for influencing the chosen activation.

A small transformation of image could impact at the first layer of the model, but a lesser impact at the top feature layer. Moreover, visualization can interfere in selecting the right network architecture. At the end, there are some experiments on ImageNet dataset showing how visualization helps to see the role of convolutional part of the ImageNet model in getting state of the art performance and ability of feature extraction layers to generalize ImageNet to other datasets.

3 Build and Train an RNN on MNIST

The goal here is to use RNN for object recognition.

3.1 Setup an RNN

By using and modifying the starter code, lstmMNISTStarterCode.py,we get the following results,

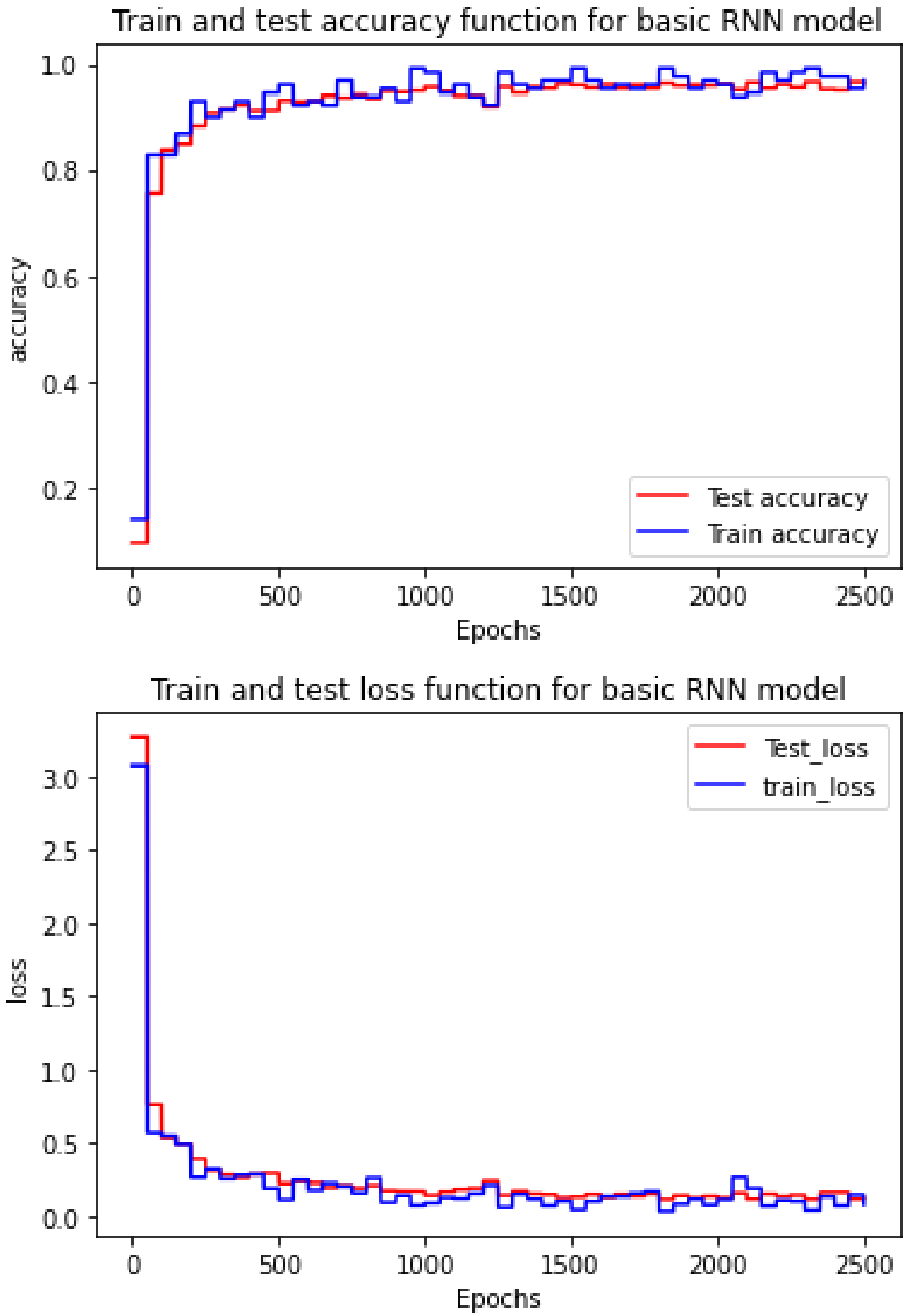


Figure 5: Train/test accuracy and train/test loss of Grayscale MNIST dataset

For the inputs, we have 28 pixel of images. The RNN network has 128 neurons, learning rate 0.001 and we train the network for 5500 times.
Figure 5 shows the loss/accuracy on training/test data. We see an accuracy of 96.87% on test data and 96.32% accuracy on training data with iteration number of 5500.

3.2 How about using an LSTM or GRU

3.2.1 LSTM

Figure 6 shows the loss/accuracy of the LSTM network on both training and testing dataset. We just follow the same configuration of the learning rate, 0.001, and hidden neurons as RNN. We see clearly that the network performs better than basic RNN network (figure 5). This time the test accuracy is 98.35% and 99.22% on training accuracy.

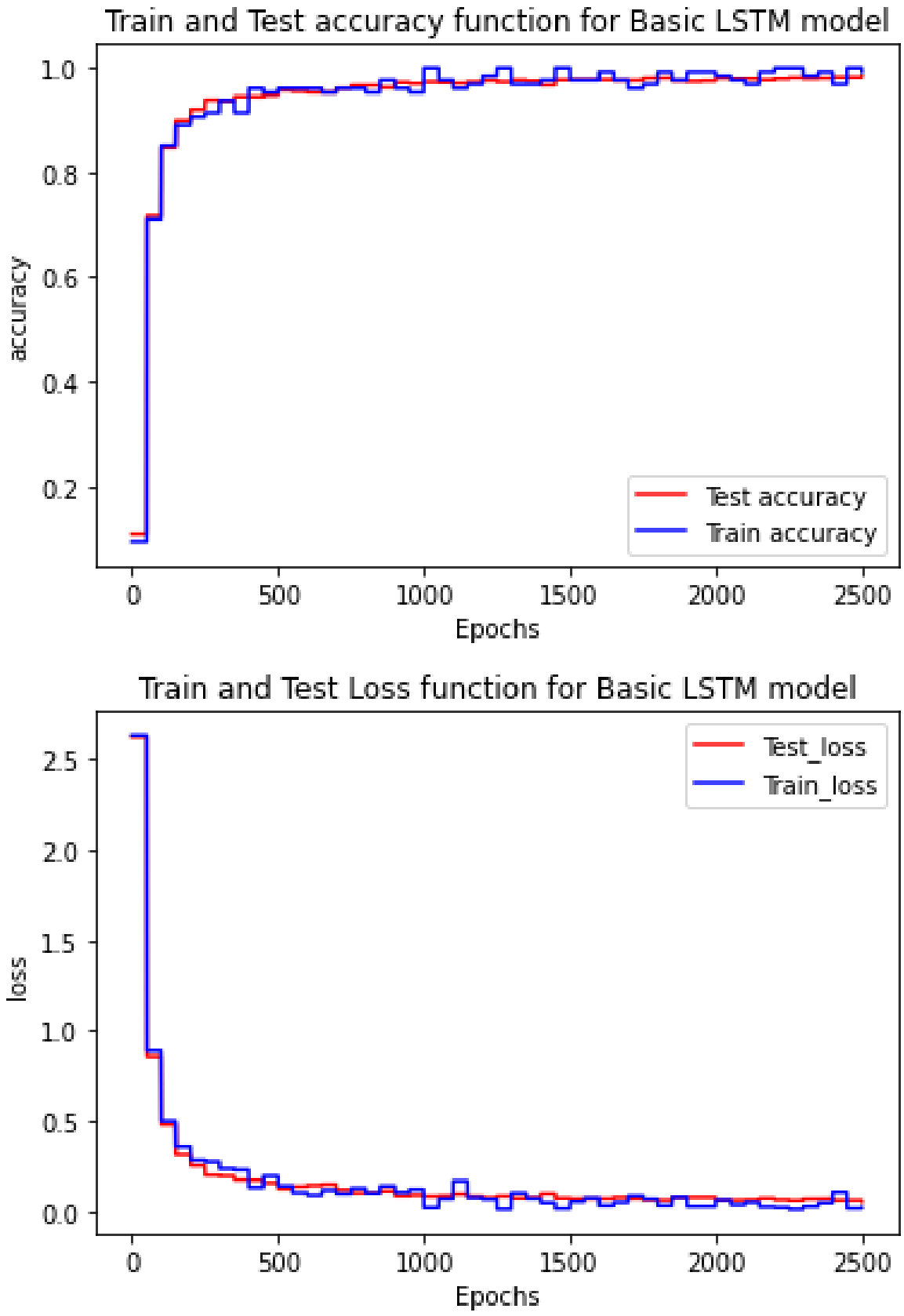


Figure 6: Train/test accuracy and train/test loss of Grayscale MNIST dataset for LSTM architecture

Here we record the accuracy and loss every 50 iteration.

3.2.2 GRU

In figure 7, we use the same set ups as the basic RNN, so the learning rate is 0.001 and the number of neurons is 128. Hence we get,

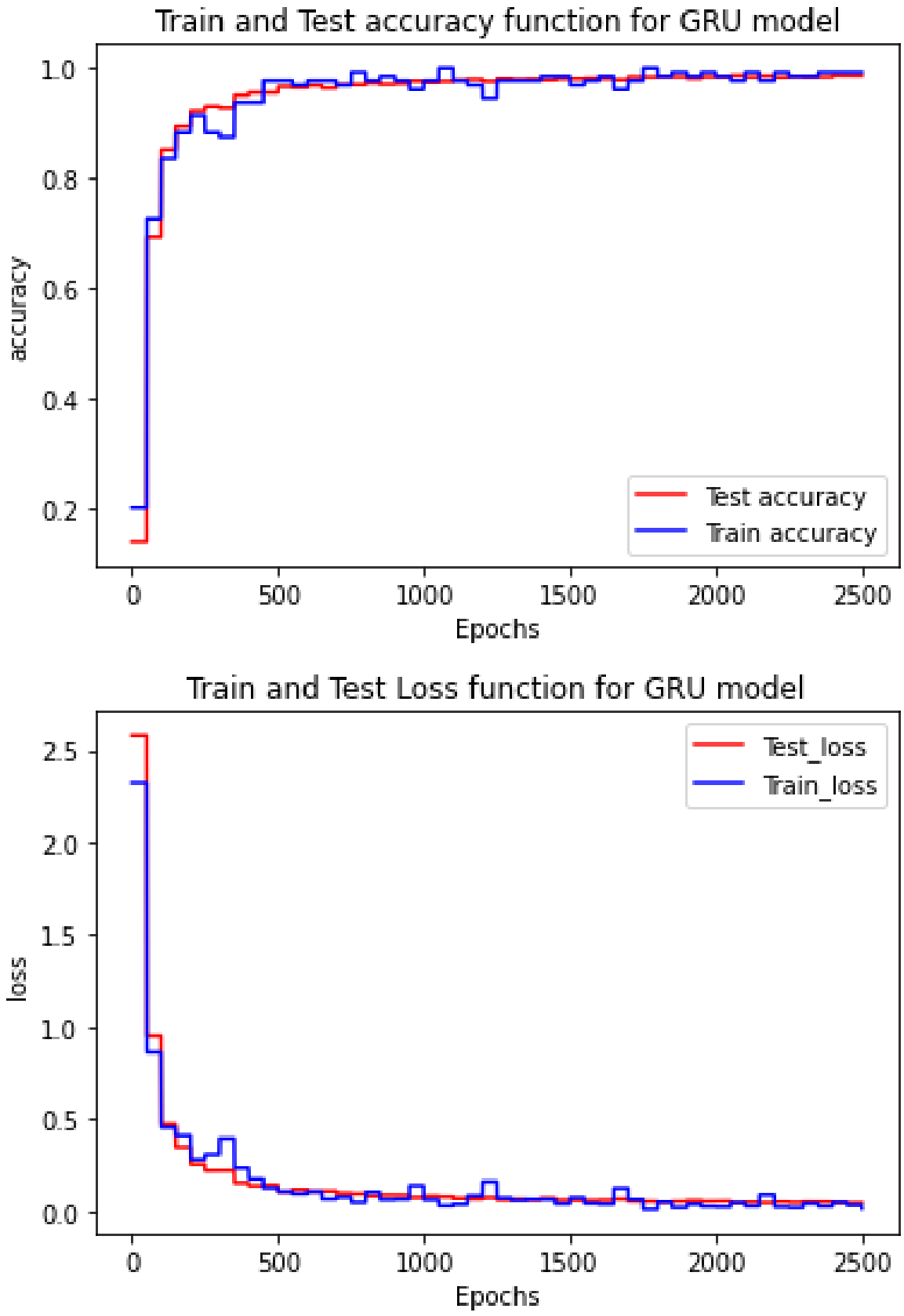


Figure 7: Train/test accuracy and train/test loss of Grayscale MNIST dataset for GRU architecture

We see that the training accuracy here is 99.22% and the test accuracy is 98.59%, which better and faster than both LSTM and RNN.
Here we record the results every 50 iteration.

3.3 Compare against the CNN

3.3.1 RNN

Using different hidden units 32, then 64, we get the following results

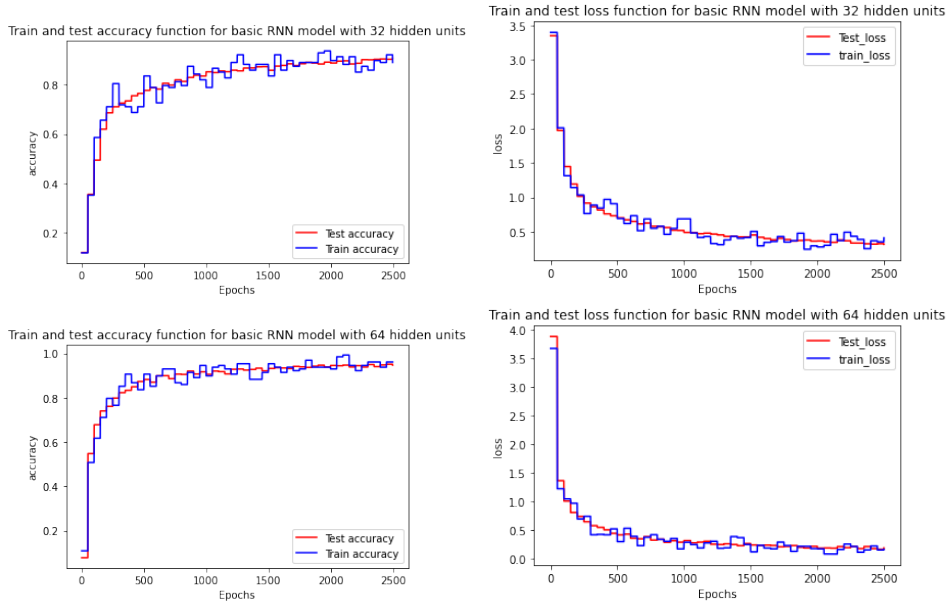


Figure 8: Train/test accuracy and train/test loss of Grayscale MNIST dataset for RNN architecture for 32/64 hidden units

For figure 8, we choose 32 hidden units, and we get 89.06% for the training accuracy and 90.79% for the test accuracy. However, the results get way better when we increase the number of hidden units to 64, and we get 96.09% for training accuracy and 94.64% for test accuracy.
The results here are captured after every 50 iteration.

3.3.2 LSTM

Using different hidden units 32, then 64, we get the following results

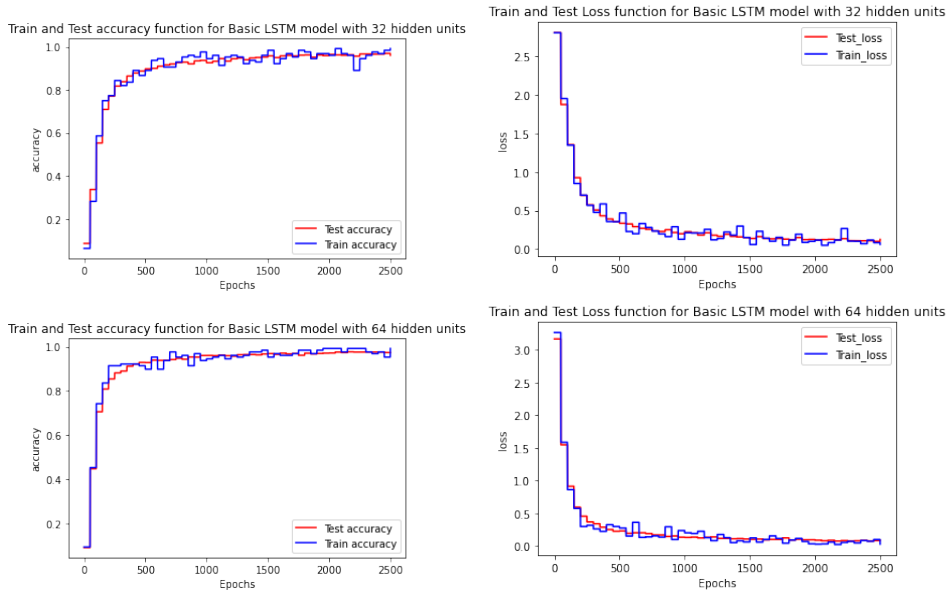


Figure 9: Train/test accuracy and train/test loss of Grayscale MNIST dataset for LSTM architecture for 32/64 hidden units

For figure 9, we choose 32 hidden units, and we get 99.22% for the training accuracy and 96.11% for the test accuracy. However, the results get way better when we increase the number of hidden units to 64, and we get 99.22% for training accuracy and 97.35% for test accuracy.
The results here are captured after every 50 iteration.

3.3.3 GRU

Using different hidden units 32, then 64, we get the following results

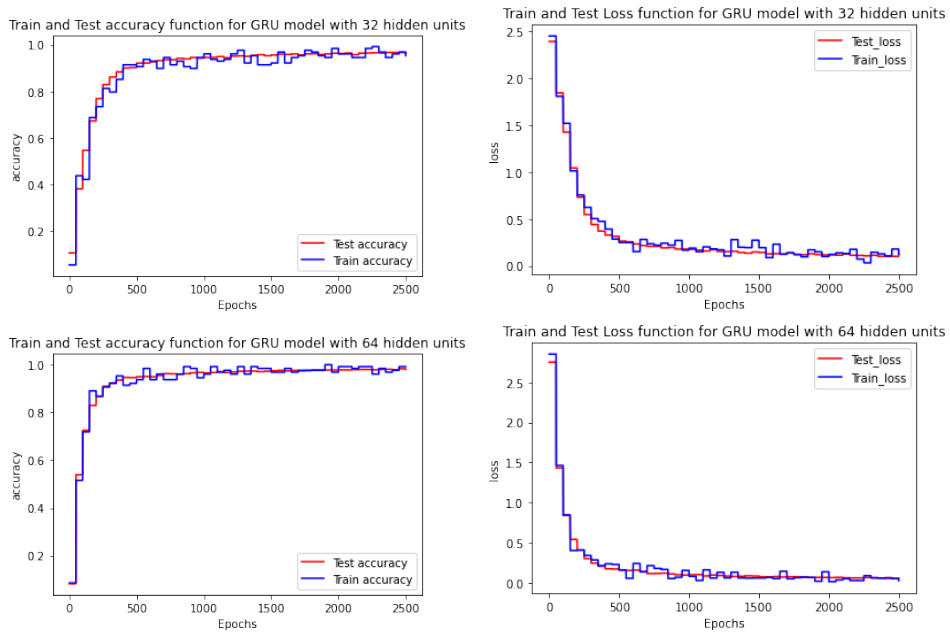


Figure 10: Train/test accuracy and train/test loss of Grayscale MNIST dataset for GRU architecture for 32/64 hidden units

For figure 10, we choose 32 hidden units, and we get 96.88% for the training accuracy and 96.72% for the test accuracy. However, the results get way better when we increase the number of hidden units to 64, and we get 99.22% for training accuracy and 98.1% for test accuracy.
The results here are captured after every 50 iteration.

4 Resources

- Lecture notes
- StackOverflow
-CNN
-CNN for object recognition
-RNN