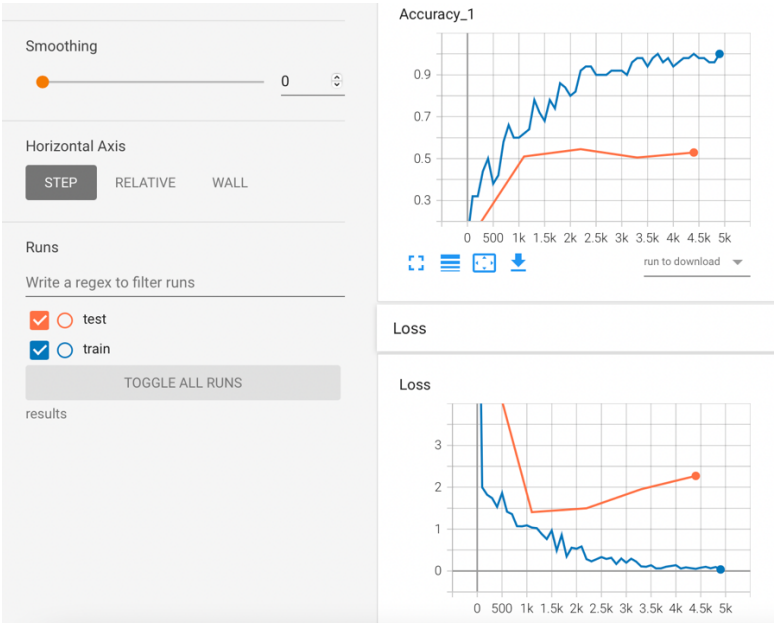
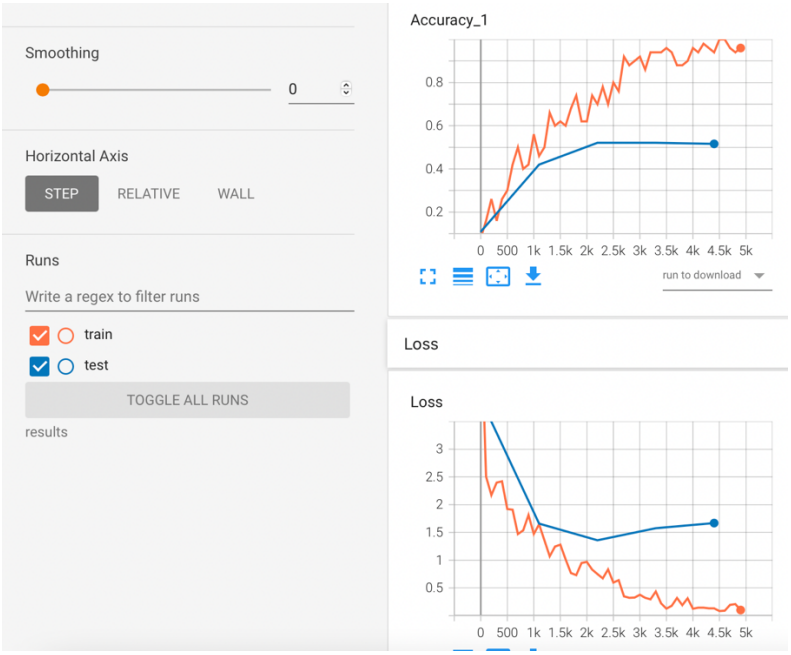
Task 1

Relu, Adam, Normal initializations:



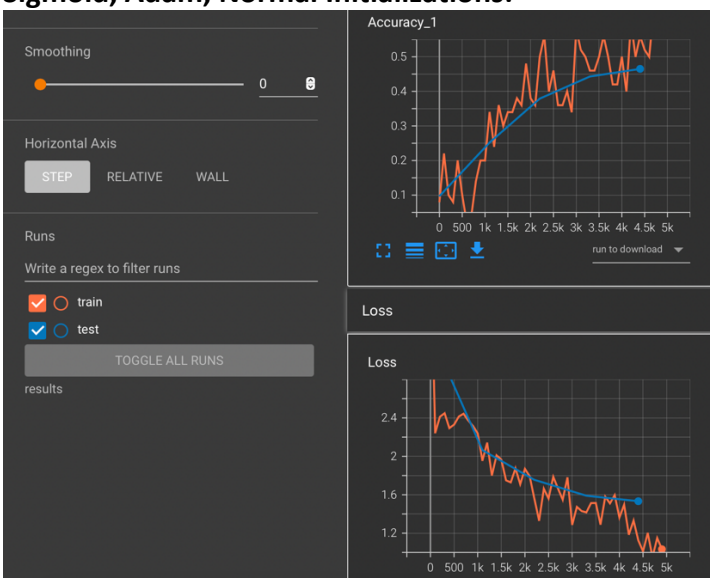
Training accuracy is close to 96%, and test accuracy is close to 52.5%.

Tanh, Adam, Normal Initializations:



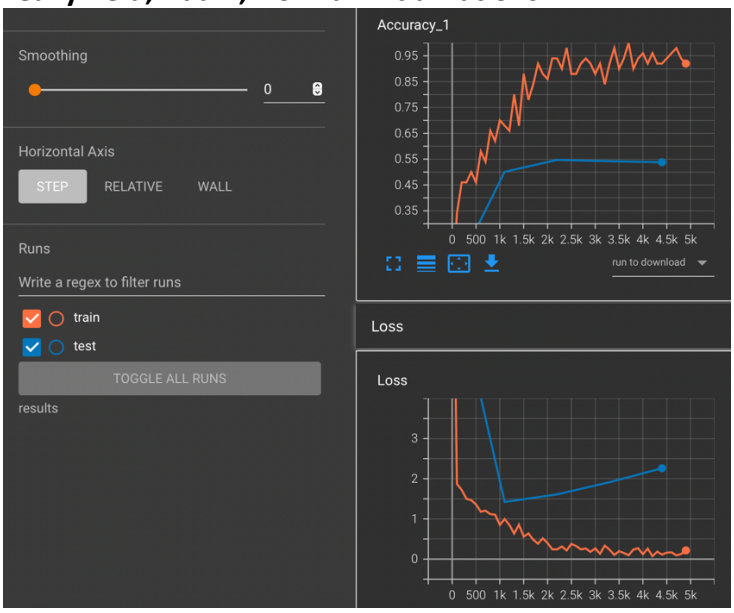
Training accuracy is close to 96%, and test accuracy is close to 54.3%.

Sigmoid, Adam, Normal Initializations:



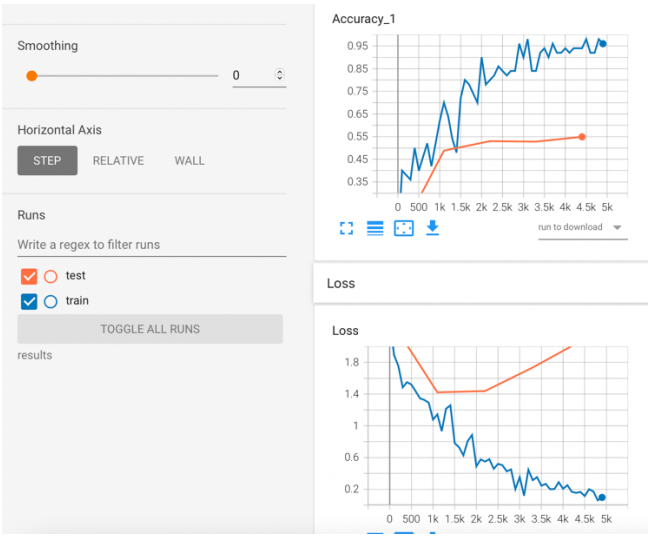
Training accuracy is close to 66%, and test accuracy is close to 49%.

Leaky Relu, Adam, Normal Initializations:



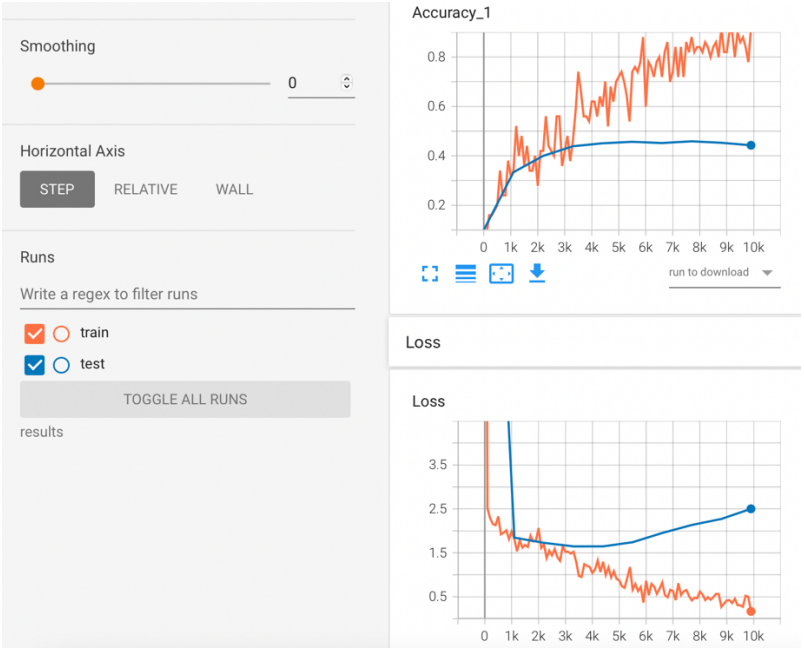
Training accuracy is close to 92%, and test accuracy is close to 52.8%.

Relu, Adam, Glorot Uniform Initializations:



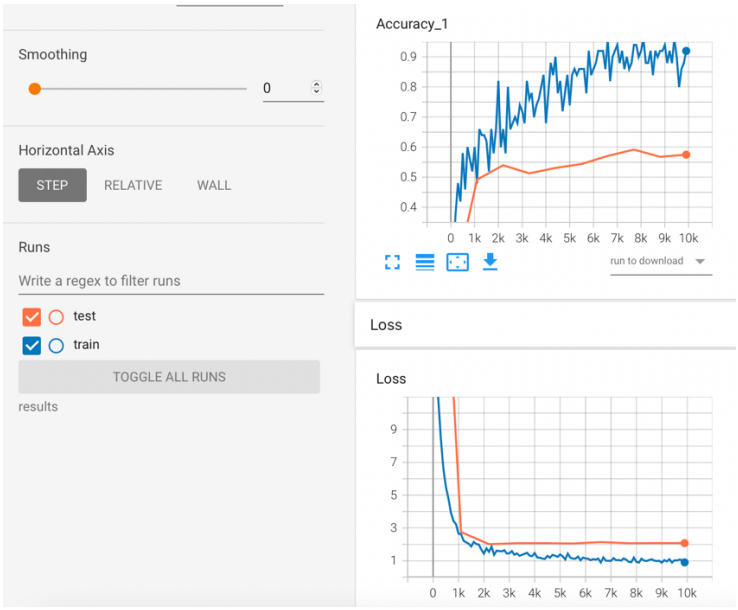
Training accuracy is close to 96%, and test accuracy is close to 53.3%.

Relu, Momentum, Normal Initializations:



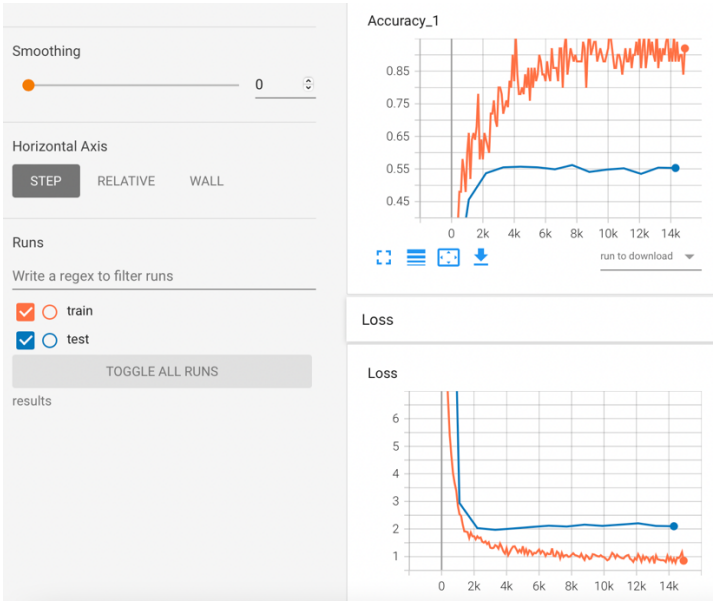
Training accuracy is close to 92%, and test accuracy is close to 45.4%.

Relu, Adam, Normal Initializations, Added Regularization:



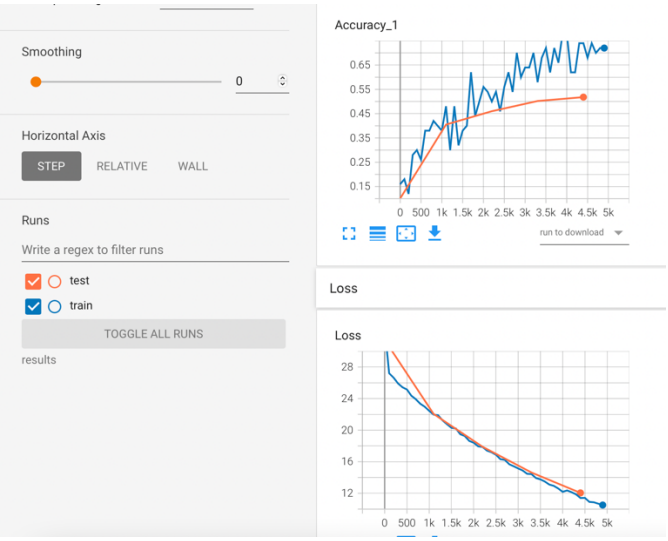
Training accuracy is close to 92%, and test accuracy is close to 58.5%.

Relu, Adam, Normal Initializations, Added Regularization, increases iterations:



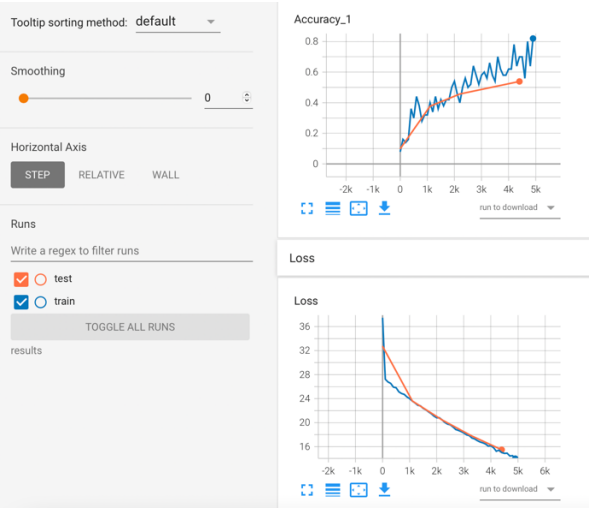
Training accuracy is close to 92%, and test accuracy is close to 56%.

Relu, Momentum(0.8),learning rate=1e-2, Normal Initializations, Added Regularization:



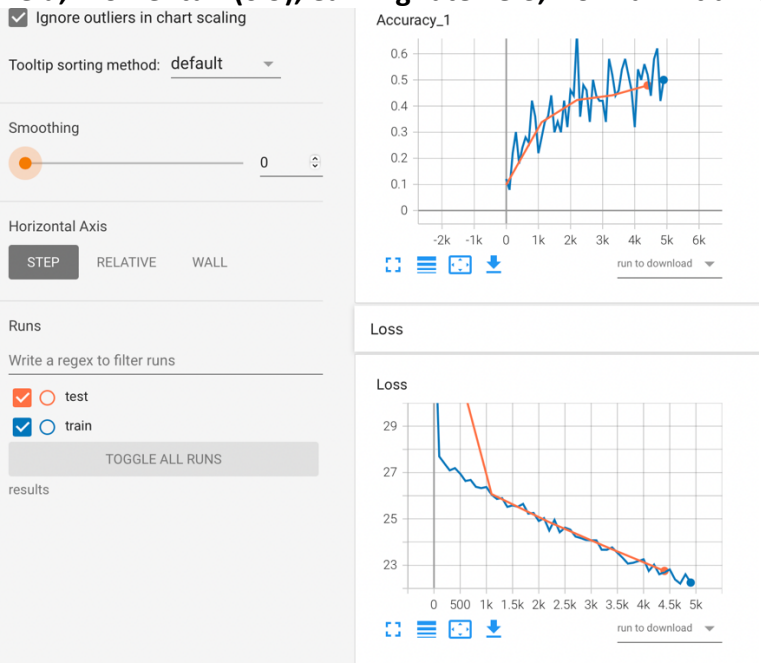
Training accuracy is close to 72%, and test accuracy is close to 53%.

Relu, Momentum(0.7),learning rate 1e-2, Normal Initializations, Added Regularization:



Training accuracy is close to 52%, and test accuracy is close to 52%.

Relu, Momentum(0.9),learning rate 1e-3, Normal Initializations, Added Regularization:

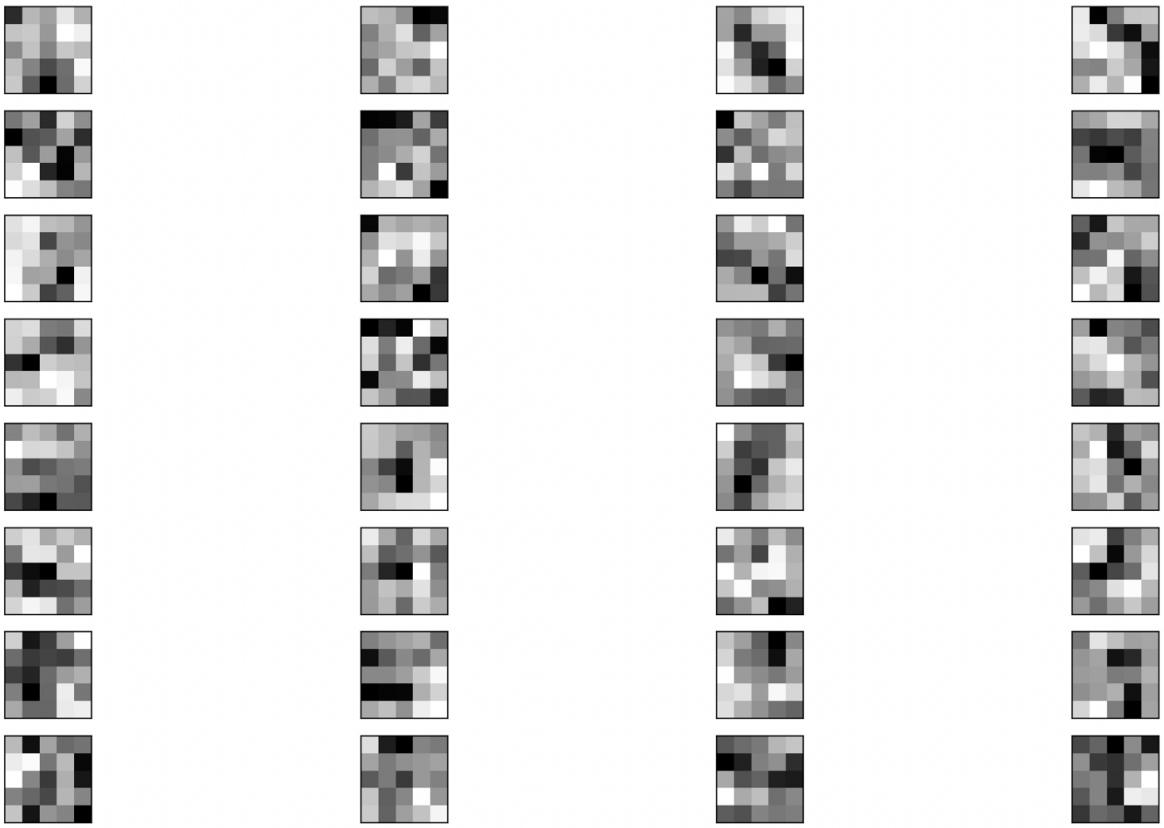


Training accuracy is close to 50%, and test accuracy is close to 48%.

I tried changing several hyperparameters including learning rate, activation functions, optimizers, initializations, but the test accuracy did not go beyond ~55%. I tried using tanh, relu, sigmoid, leaky relu for activations, also I tried training with different batch sizes ranging from 16,32,50,100. I also used different initializations like normal initializations, glorot uniform. For applying regularization, I tried with different values of dropout, added regularization terms to cross entropy. I tried using adam, adagrad, momentum optimizers, and tried tuning with different learning rates and different momentum values. But the overfitting problem could not be improved. I trained my model using the template provided as well as created a new template for tensorflow version 2. I have not shown some of the results which were very poor, but the code is commented and is present in my submission.

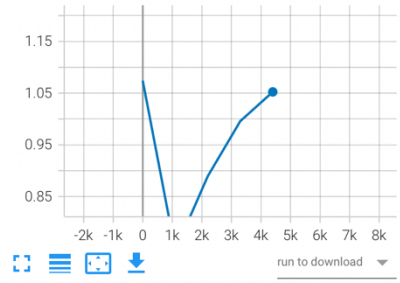
Version 1 template: CIFAR_V1.py
Version 2 template: CIFAR_V2.py

I think that the training data is low that's why the model is overfitting and even the different regularization techniques are also not helping. It is also possible that choosing a different convolutional neural network architecture can help.

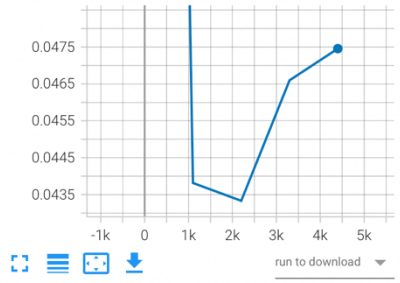


Since the dataset is very blurry, the gabor filters shown above are also difficult to comprehend. But still, many of the filters clearly demonstrate the detection of gradients (light grey to white to black).

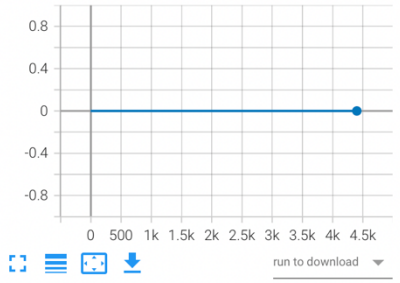
h_Conv1_MAX



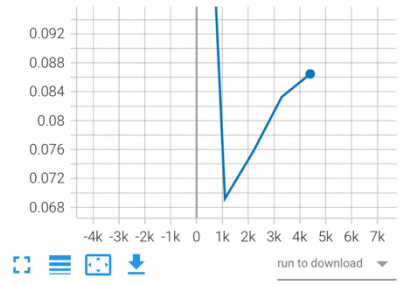
h_Conv1_MEAN



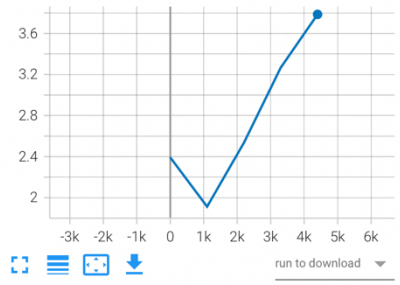
h_Conv1_MIN



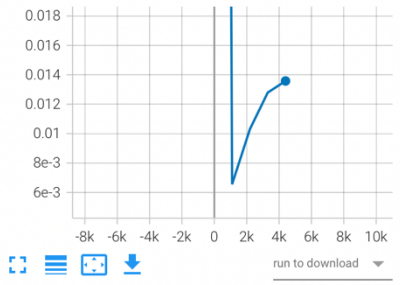
h_Conv1STD



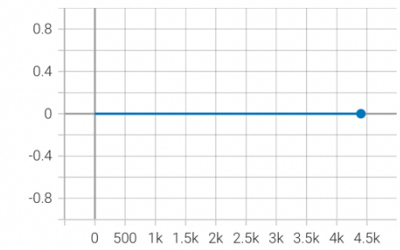
h_Conv2_MAX



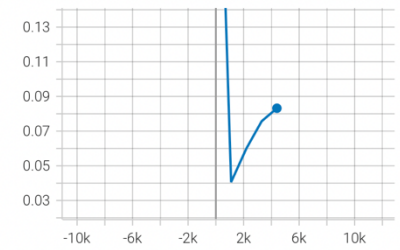
h_Conv2_MEAN



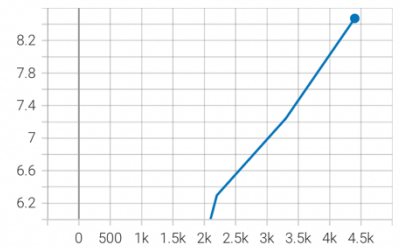
h_Conv2_MIN



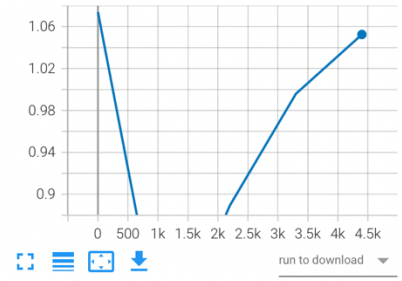
h_Conv2STD



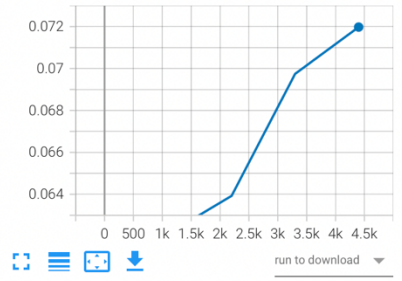
h_fc1_MAX



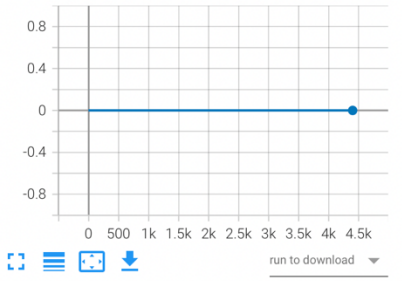
h_pool1_MAX



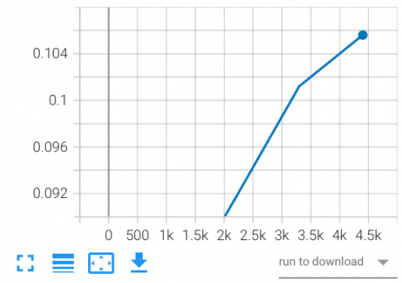
h_pool1_MEAN



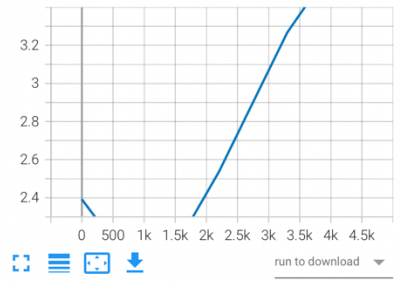
h_pool1_MIN



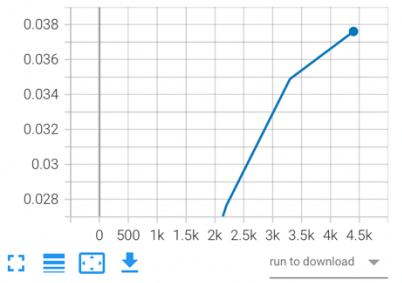
h_pool1STD



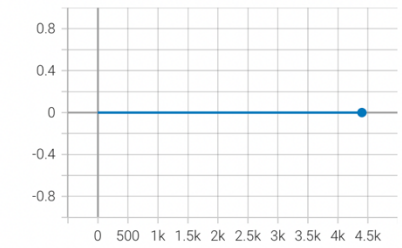
h_pool2_MAX



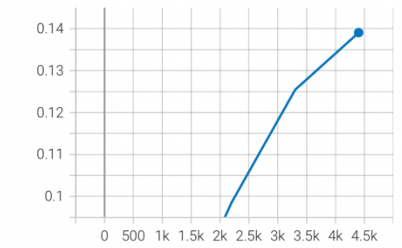
h_pool2_MEAN



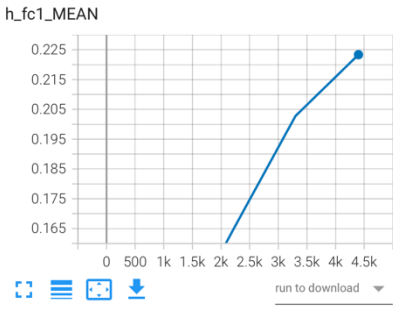
h_pool2_MIN



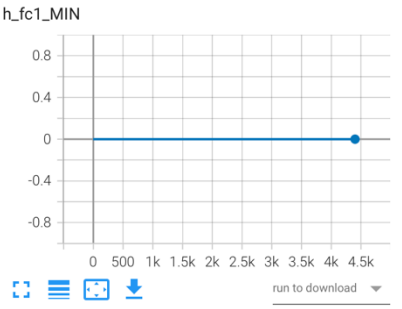
h_pool2STD



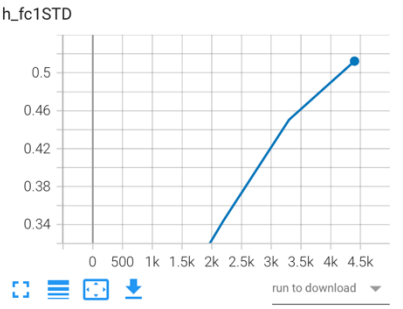
h_fc1_MEAN



h_fc1_MIN



h_fc1STD



The above are activations for Relu. The minimum of the convolution activations are 0 which is expected because its Relu. The graphs show standard deviation in both the convolution layers decreases to low value initially then slowly starts increasing again, the same is visible from the from the histogram as well. From the histograms, it is seen that most of the change happens from the iteration 0 to iteration 1100, after that the shapes are relatively constant. This is also confirmed by the accuracy and loss graphs, the accuracy changes the most at the 1100th iteration, after that it is relatively constant.

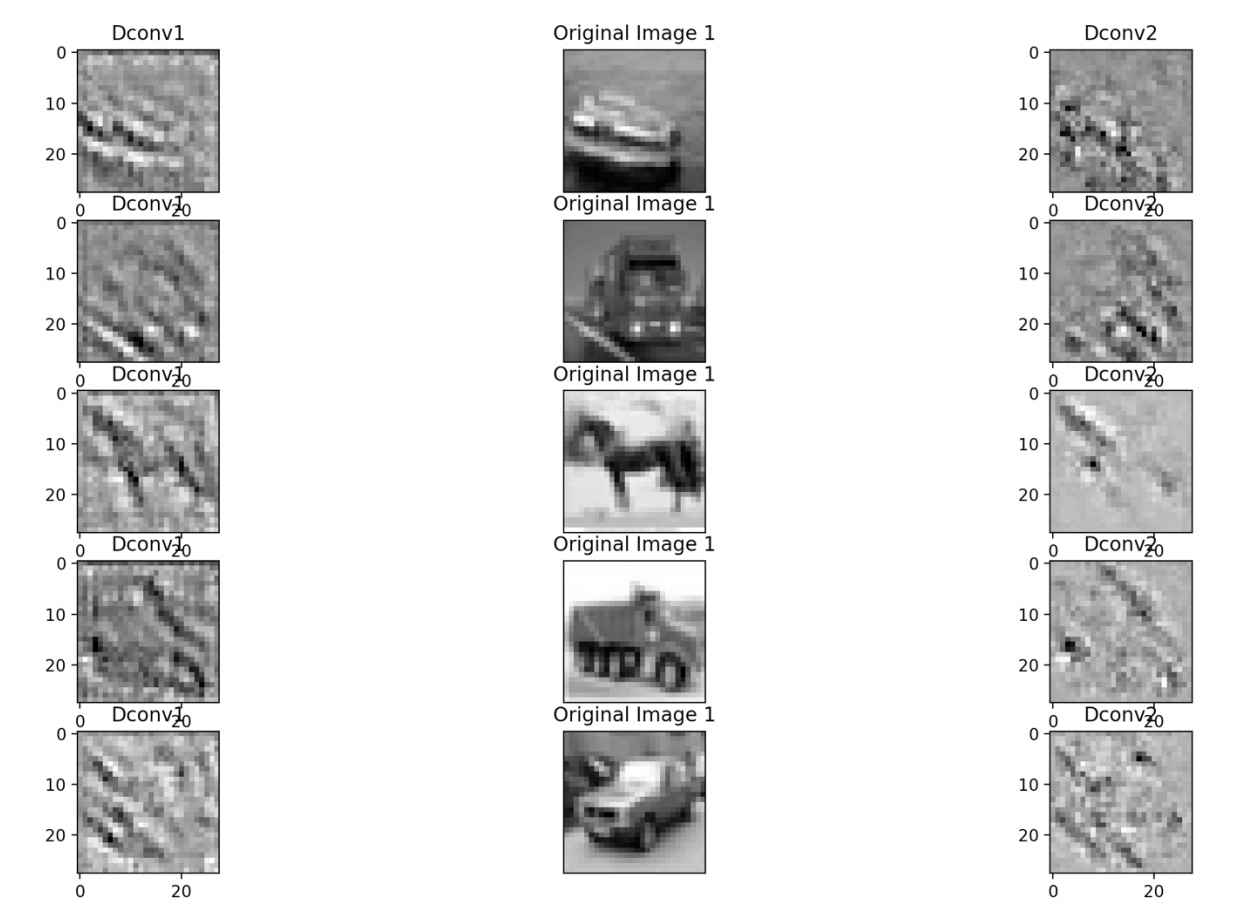
Task 2

Attached the summary file as: Paper Summary

Files:Dconvnet.py, Dconvnet_visualizations.py

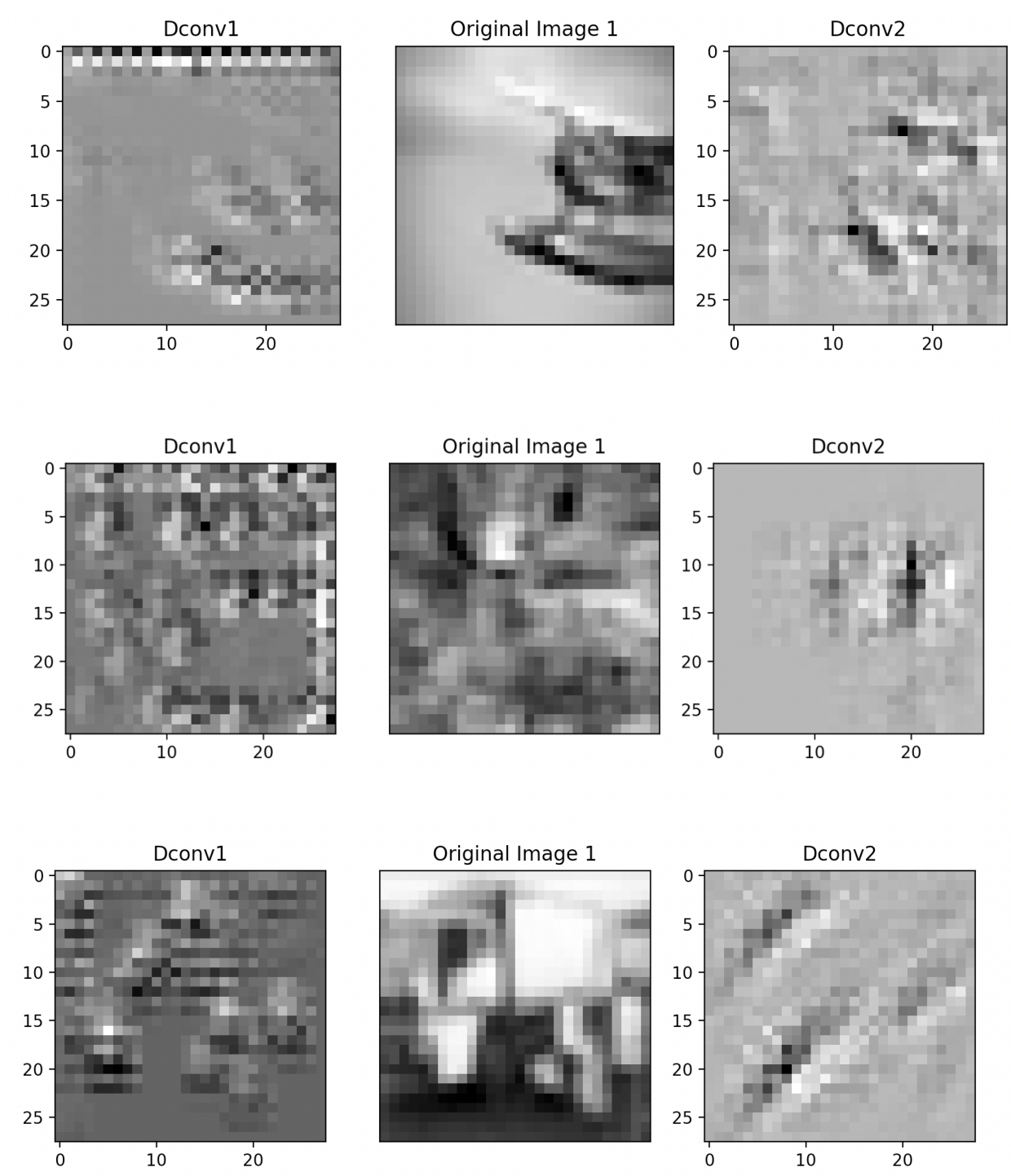
Dconvnet.py is just doing the training of the cnn and storing the parameters (weights and biases), then in dconvnet_visualizations.py, I am loading those parameters and then directly passing the test images, since no training is required here, I am directly visualizing the two convolution layers.

Below are the outputs after visualizing 5 images from test set:



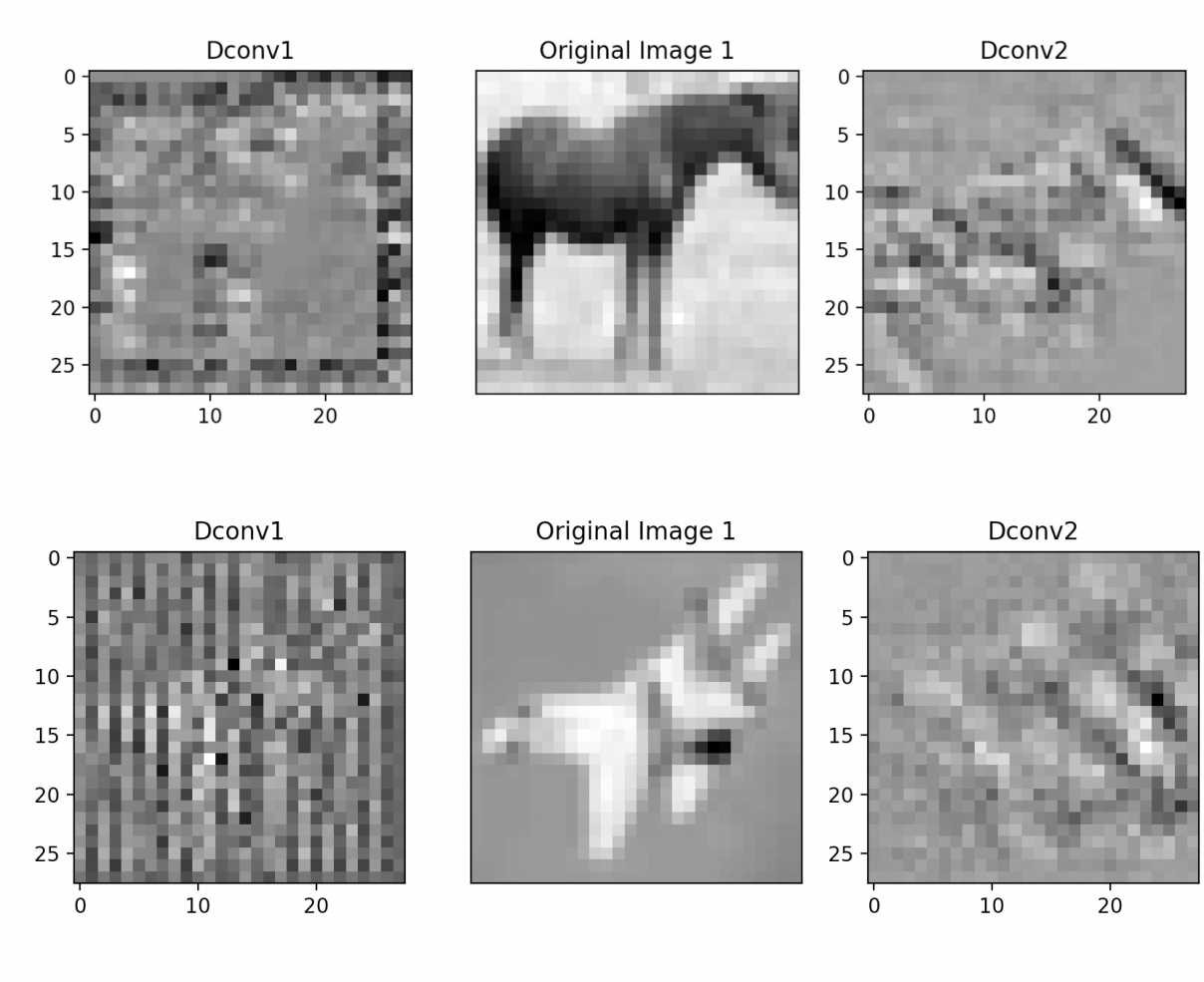
Dconv1 illustrates the output after mapping the activation output of convolutional layer 1 back to input space. Dconv2 illustrates the output after mapping the activation output of convolutional layer 2 back to input space. Since the original images are themselves very blurry, the reconstruction is also not very good.

The above results were when I deconvoluted all the layers back to input space. Below results are when I deconvolute only the highest activation map back to input space.
Code files: DCONVET2.PY



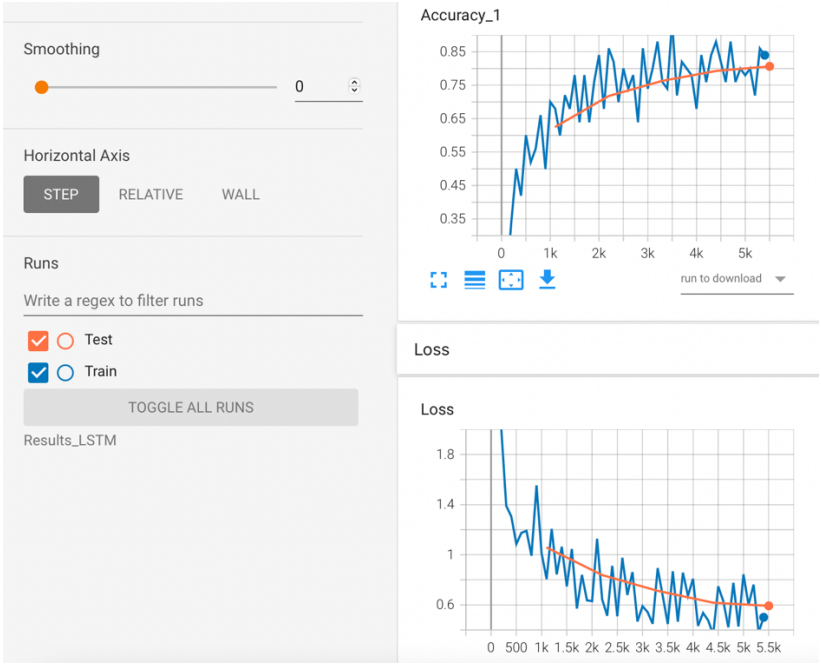
I had to do this using two separate files because getting the maximum activation and setting others to 0 was an issue in tensorflow version1.

Below are outputs are after considering the top 9 activations.

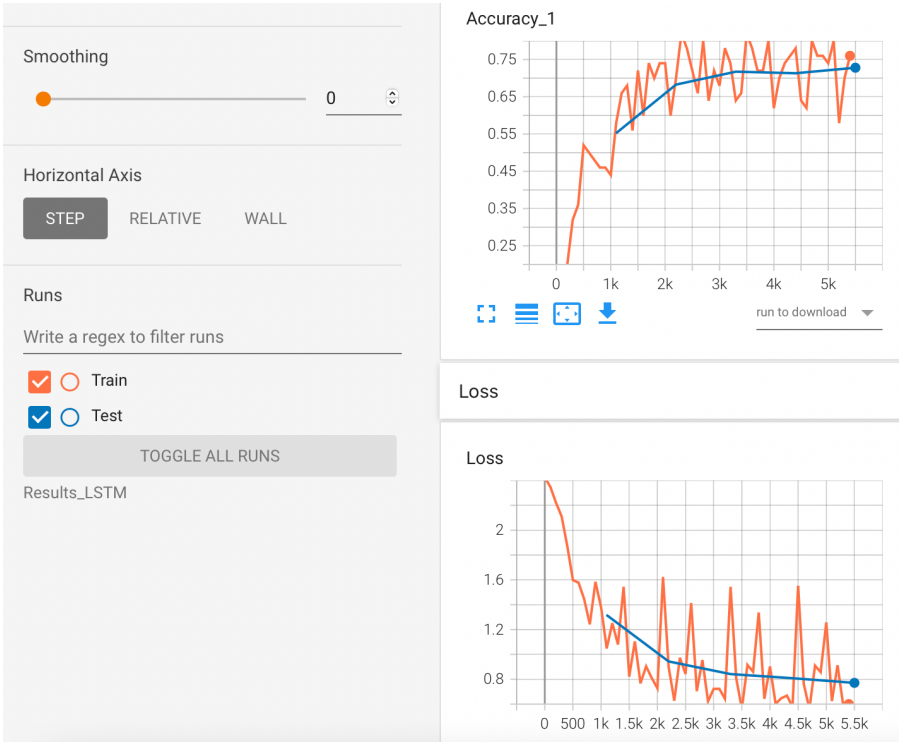


Task 3
Files: RNN.py
Simple RNN

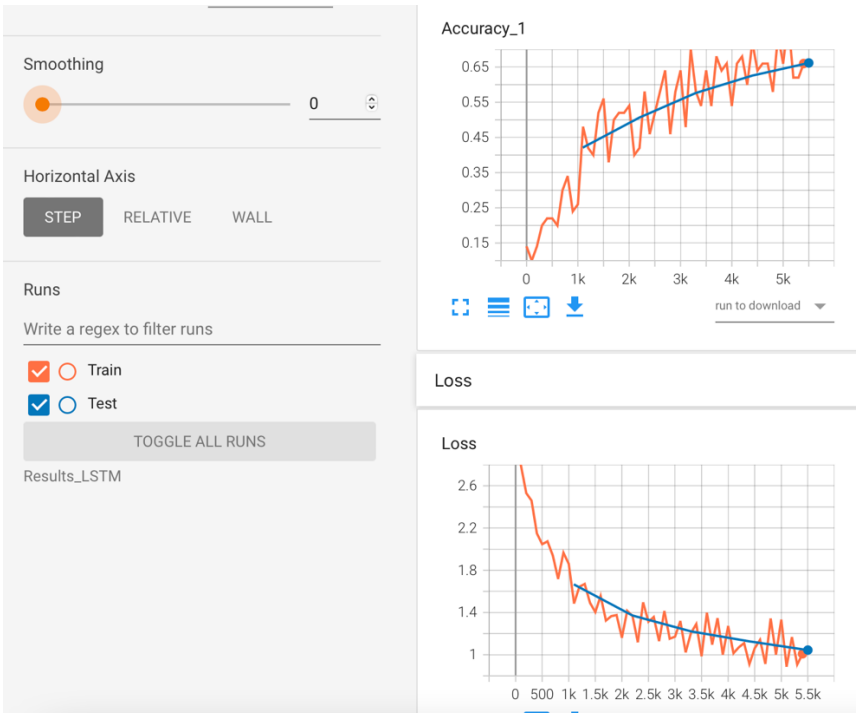
Hidden=20



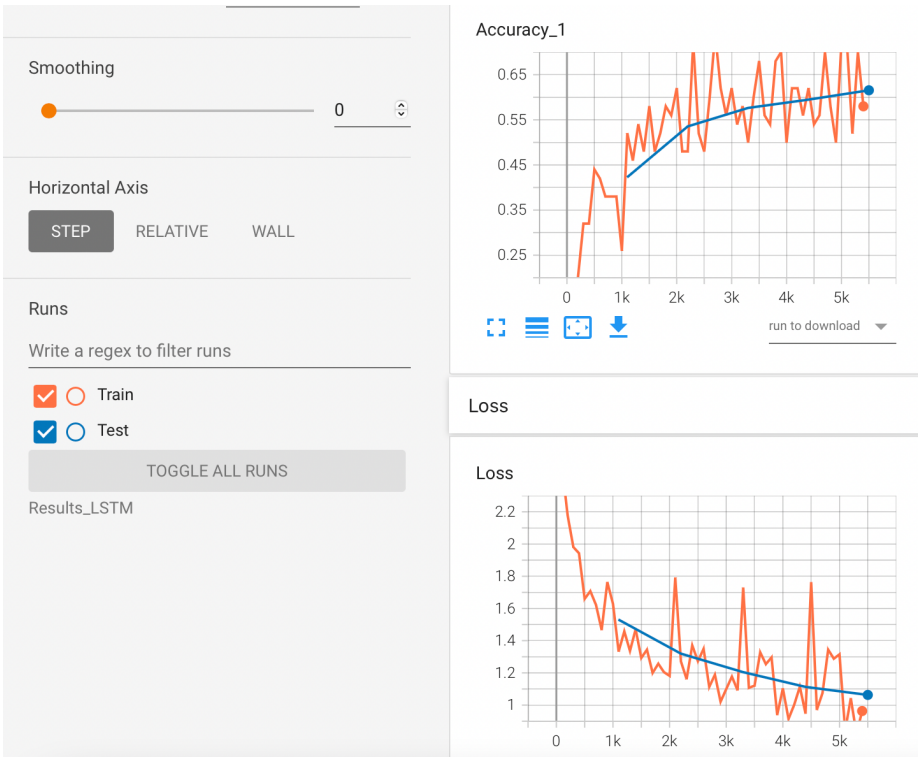
Hidden=15



Hidden=10


Smoothing
0
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
Train
Test
TOGGLE ALL RUNS
Results_LSTM
Accuracy_1
0.65
0.55
0.45
0.35
0.25
0.15
0
1k
2k
3k
4k
5k
run to download
Loss
Loss
2.6
2.2
1.8
1.4
1
0
500
1k
1.5k
2k
2.5k
3k
3.5k
4k
4.5k
5k
5.5k

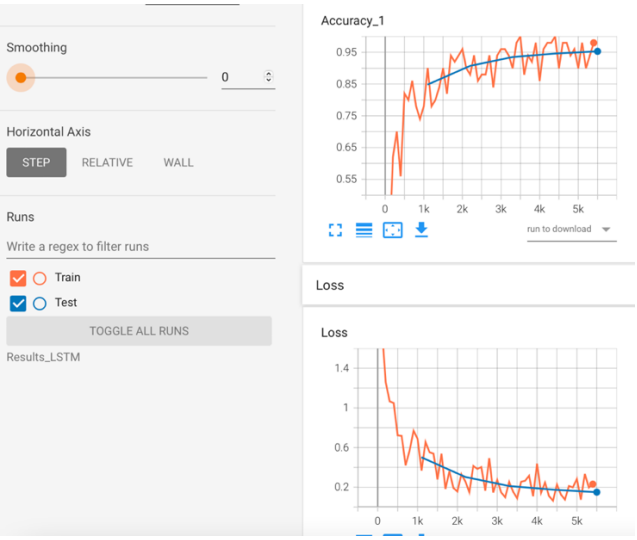
Hidden=5


Smoothing
0
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
Train
Test
TOGGLE ALL RUNS
Results_LSTM
Accuracy_1
0.65
0.55
0.45
0.35
0.25
0
1k
2k
3k
4k
5k
run to download
Loss
Loss
2.2
2
1.8
1.6
1.4
1.2
1
0
1k
2k
3k
4k
5k

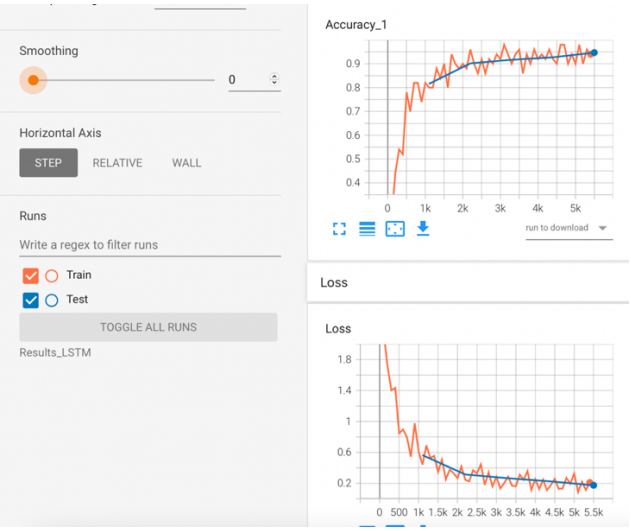
As I decrease the number of hidden layers the accuracy decreases, and the loss increase for both training and testing. It is expected because more number of neurons can better capture features. Since this is a simple RNN, I think the model is not able to remember the data for long duration.

LSTM Cell

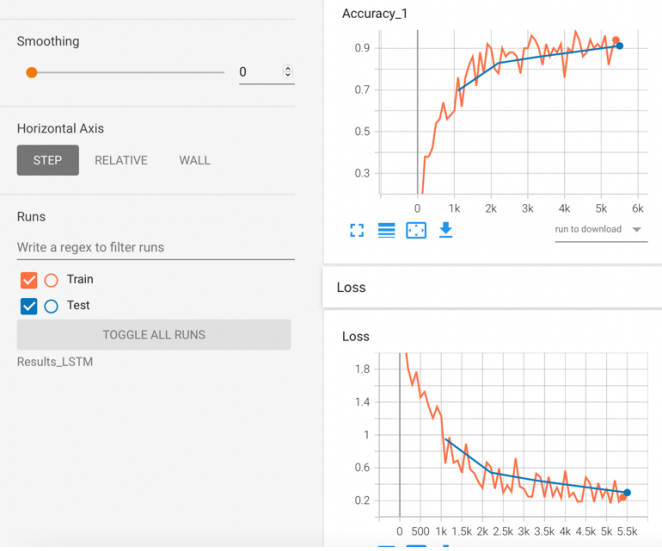
Hidden=20



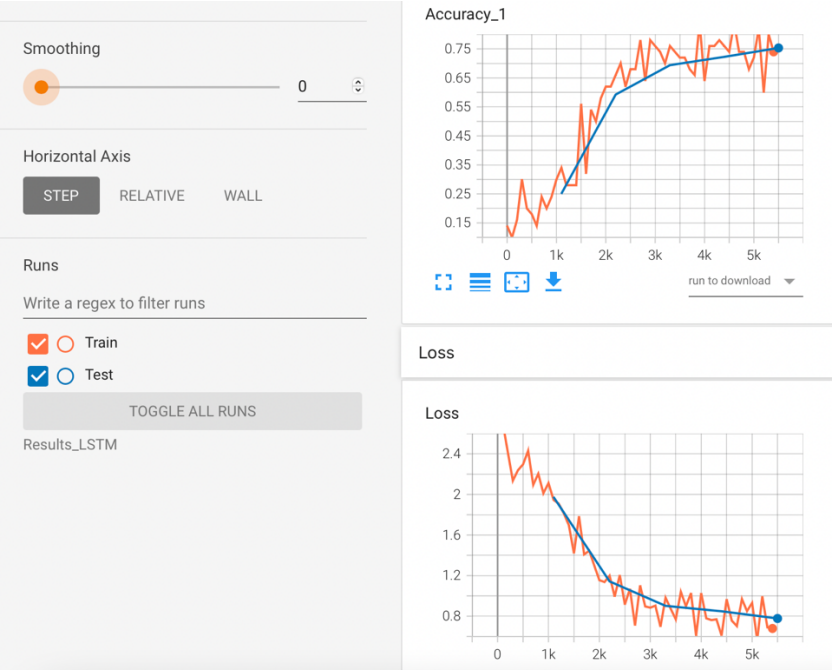
Hidden=15



Hidden=10



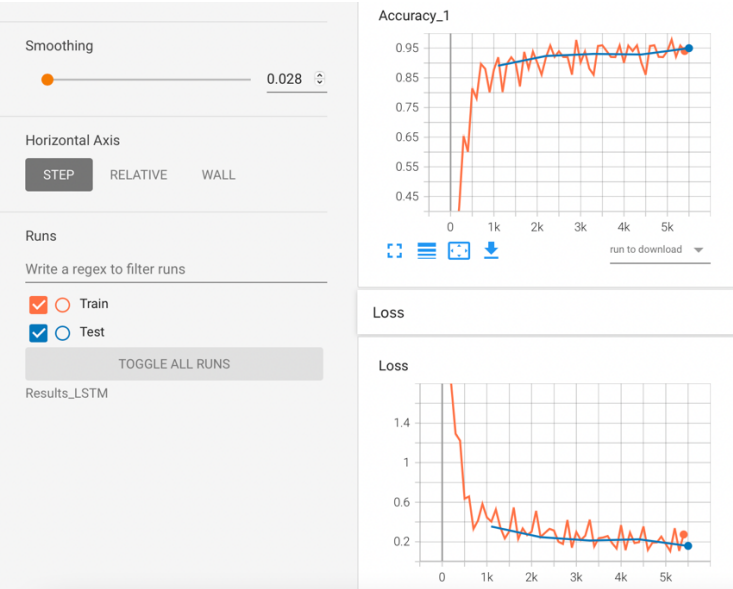
Hidden=5



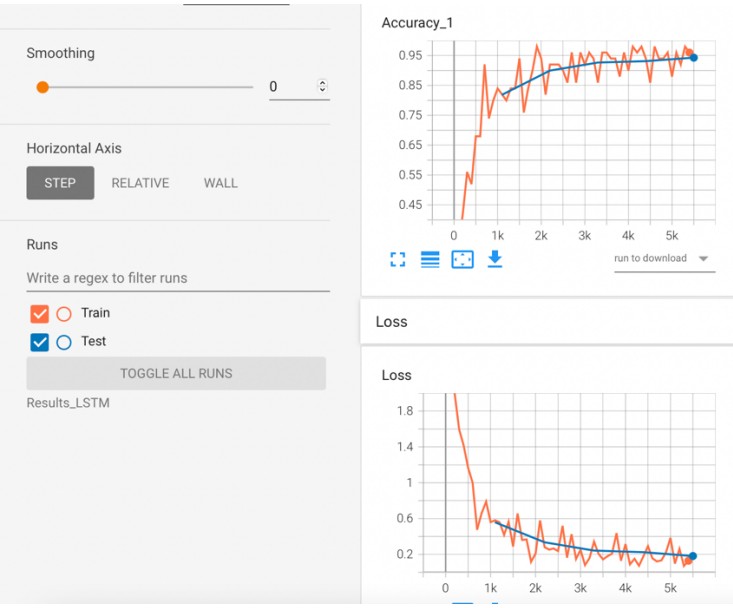
For LSTM, the test and train accuracies are all the same for hidden neurons =10,15,20. It seems like there is almost no benefit of increasing the number of neurons beyond 10, it will only increase the simulation time. For neurons=10, the accuracy achieved is ~90% which is very close to 95% for neurons=15,20.LSTM achieves higher accuracy than simple RNN and I think that is because of the memory cell in the LSTM. It is able to remember long term data. The training and test accuracy does decrease when the neurons decreases to 5, there is almost 20% drop.

GRU

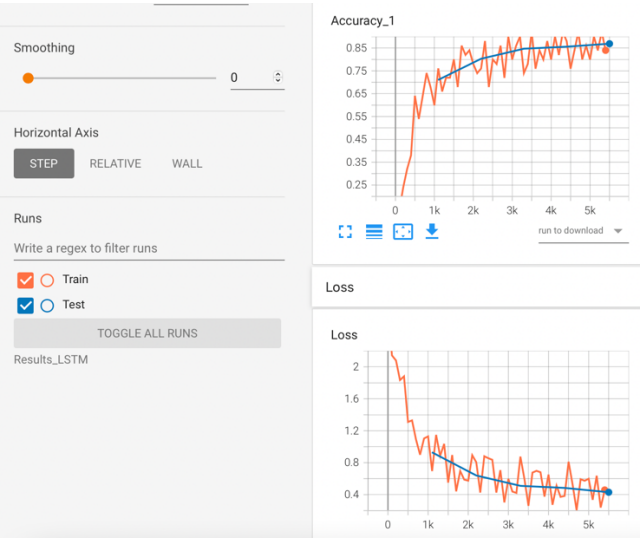
Hidden=20



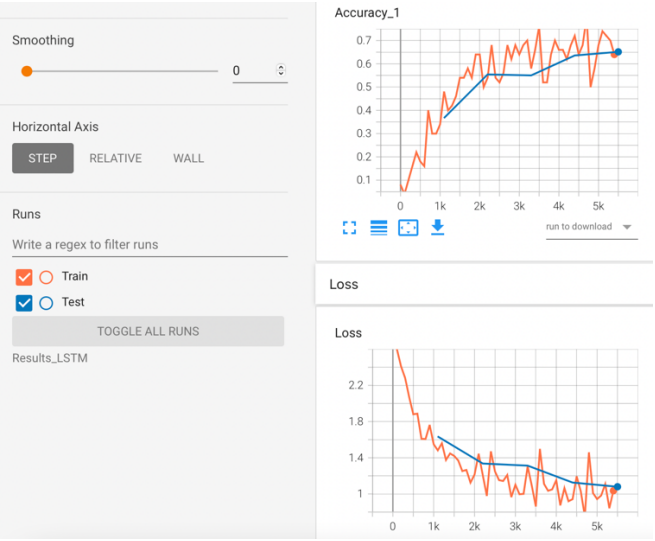
Hidden =15



Hidden=10



Hidden=5



For GRU, the train and test accuracy are almost the same (~95%) for neurons=15,20. These results are like that of LSTM. However, for neurons=10, GRU achieves ~85% accuracy on both training and testing whereas LSTM achieves ~90%. For neurons =5, GRU achieves ~65% train/test accuracies whereas LSTM achieved ~75%. So, LSTM is performing better than GRU in this case as well. But I think the performance difference is not much, also LSTM is internally more complex than GRU.

Part C

The CNN used in the previous assignment had 2 convolutional layers and 2 fully connected layers, so the network was bigger than the network used for RNN. The RNN network just had 1 hidden layer with 28 time steps. Each input image in the case of RNN was decomposed into 28 time steps and then model was trained. In the case of CNN, the entire image was given as input at once. The training accuracies of CNN model reached close 1 at a faster rate compared to all the RNN models. The accuracy of simple RNN was lower than CNN even after increasing the neurons to 20. For LSTM and GRU, they were able to achieve accuracy close to the CNN model. But I think RNN are more suitable for sequential data and CNN are more suitable for images based datasets, even though RNN did provide good levels of accuracy for MNIST, it might always not be the same.