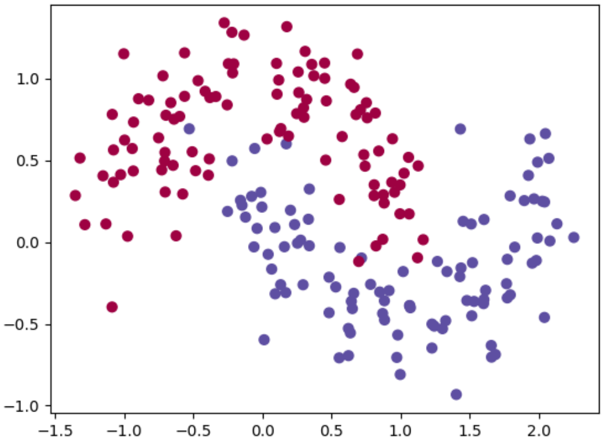
Comp 576 HW 1

Jingyu Fu (NetID: jf70)

1 Backpropagation in Simple Neural Network

1-a



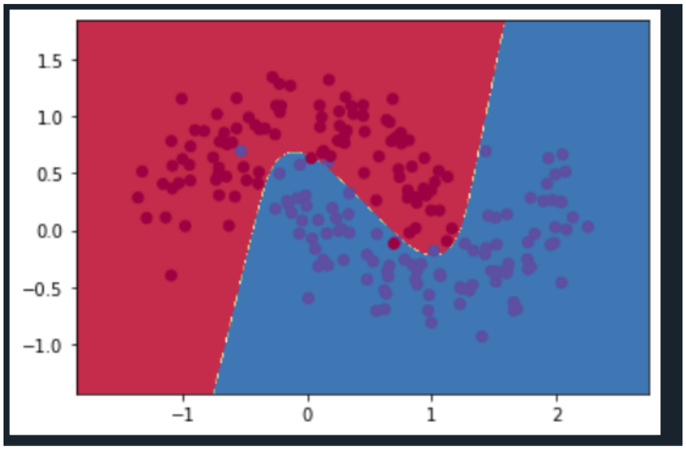
1-e-1 Train with three hidden units for three different functions

Observations:

Three functions all have very similar decision boundary and loss after iteration 19000. Although ReLU has a sharper decision boundary comparing to other two functions, the general trend and shape is still very similar to others.

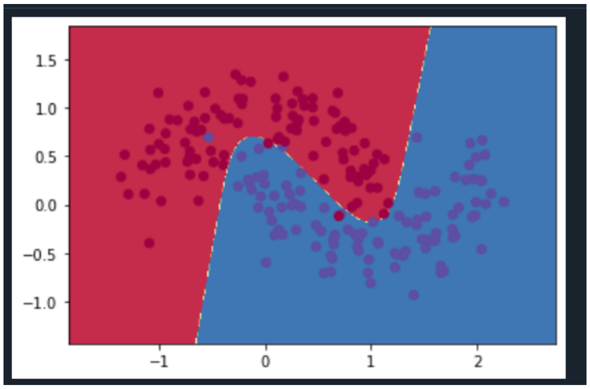
Tanh

Loss after iteration 19000: 0.070758



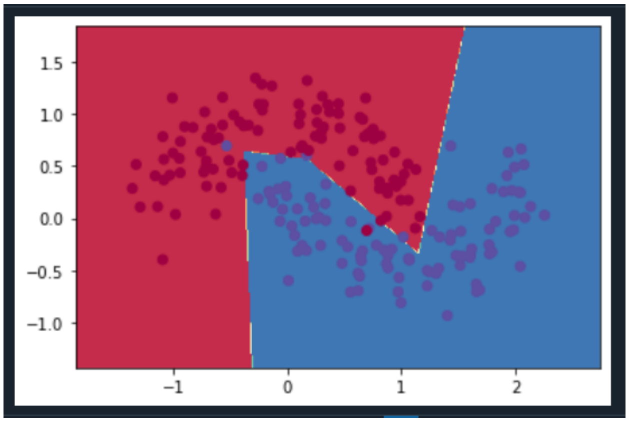
Sigmoid

Loss after iteration 19000: 0.078155



Relu

Loss after iteration 19000: 0.071219

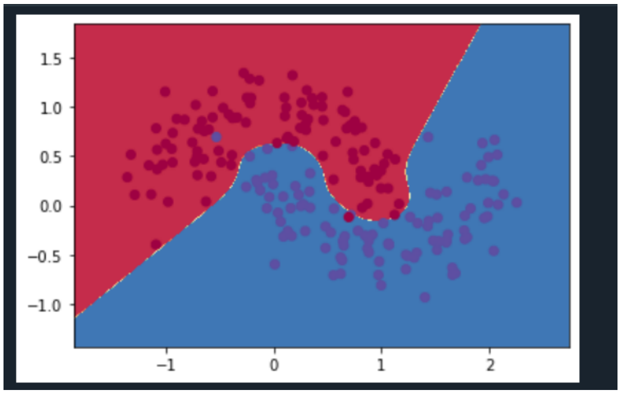


1-e-2. Train with increased hidden units

I tried with 4 hidden and 10 hidden, comparing to previous tests, we could see that higher the hidden units, smaller the loss. But there should be a threshold between 4 hidden and 10 hidden, because the 10 hidden is overfitted.

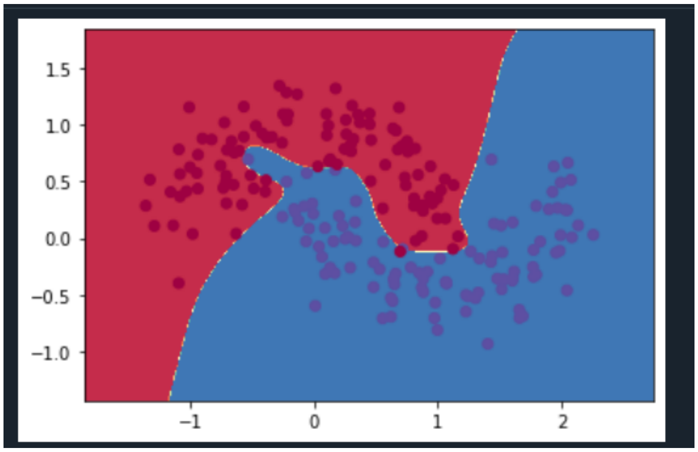
Tanh with 4 hidden

Loss after iteration 19000: 0.050689



Tanh with 10 hidden

Loss after iteration 19000: 0.030565

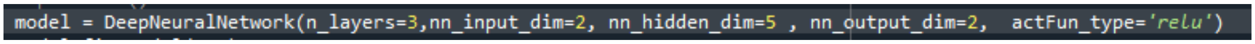


1-f Deeper network:

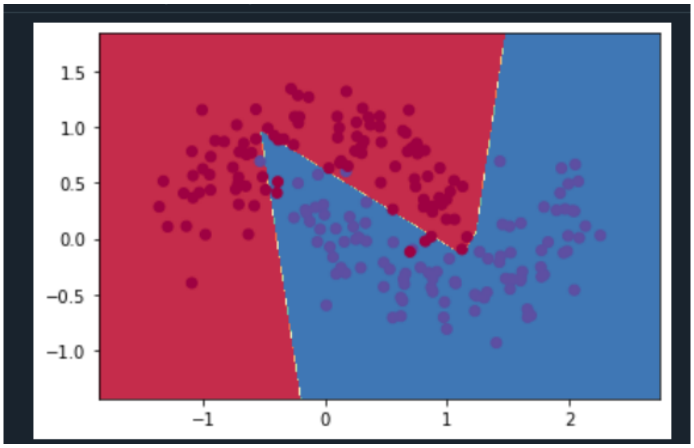
1-f-1: Make Moons dataset

Decision boundary changes as the number of layers and size of layers change. When the number and size of layers increases, the boundary becomes smoother and the loss decreases. But there is a threshold, once passed this threshold, there comes the problem of overfitting. Therefore it is necessary to find a point where it is appropriate, and neither overfitting nor underfitting.

Model 1: 3 layers, relu

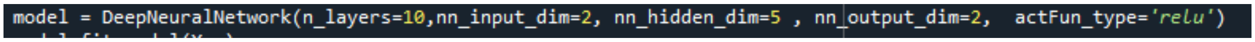


```
model = DeepNeuralNetwork(n_layers=3,nn_input_dim=2, nn_hidden_dim=5 , nn_output_dim=2,  actFun_type='relu')
```

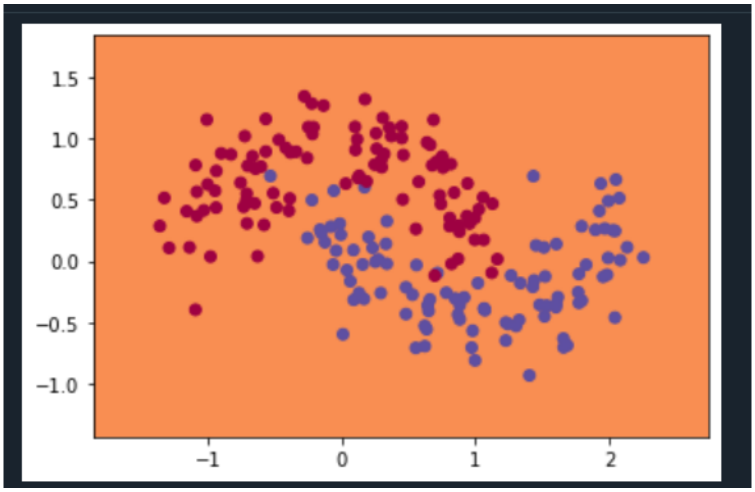


Loss after iteration 19000: 0.108509

Model 2: 10 layers, relu

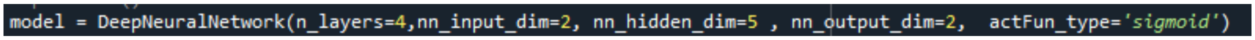


```
model = DeepNeuralNetwork(n_layers=10,nn_input_dim=2, nn_hidden_dim=5 , nn_output_dim=2,  actFun_type='relu')
```

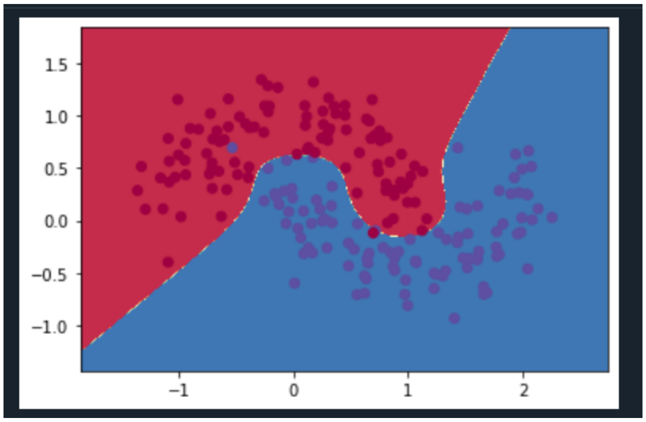


Loss after iteration 19000: 0.693200

Model 3: 4 layers, sigmoid

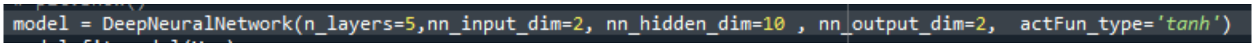


```
model = DeepNeuralNetwork(n_layers=4,nn_input_dim=2, nn_hidden_dim=5 , nn_output_dim=2,  actFun_type='sigmoid')
```

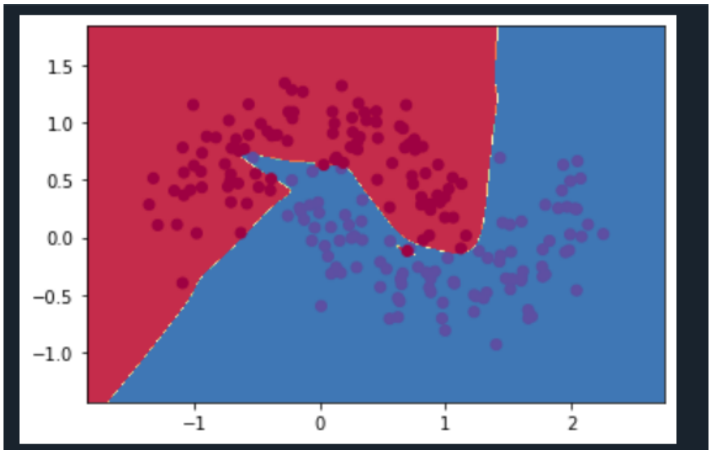


Loss after iteration 19000: 0.058388

Model 4: 5 layers, tanh



```
model = DeepNeuralNetwork(n_layers=5,nn_input_dim=2, nn_hidden_dim=10 , nn_output_dim=2,  actFun_type='tanh')
```

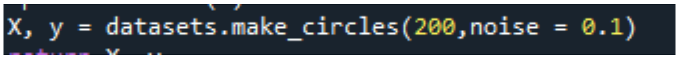


Loss after iteration 19000: 0.079216

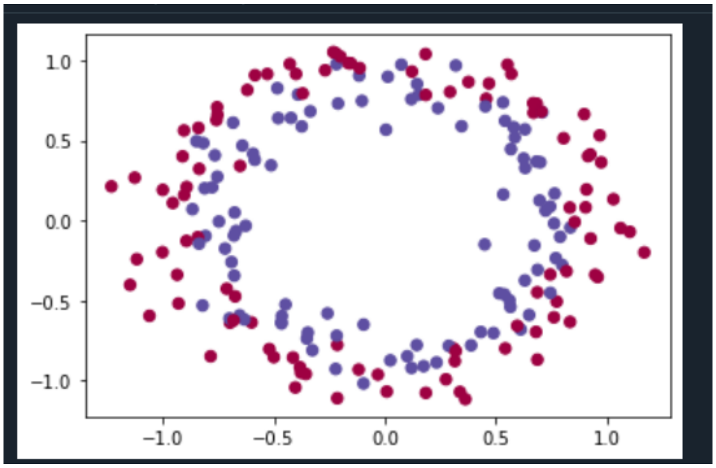
1-f-2: Another dataset

Dataset overview:

The dataset I used is make_circles. This dataset has 200 samples and 0.1 noise, whose ideal result is a circle. It is easy to find that this dataset has a much higher loss comparing to make_moons, which is probably because of the closely clustered data points in this dataset.



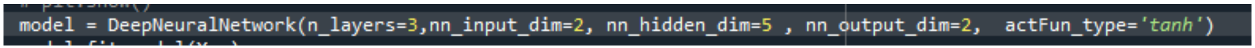
```
X, y = datasets.make_circles(200,noise = 0.1)
```



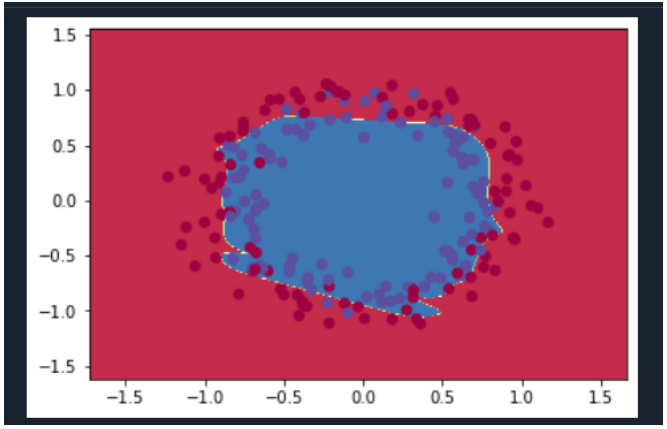
Observations:

Three layers with activation function of tanh has a very high loss, which is 0.86. When I increased the hidden dim, the loss decreased and boundary became little bit smoother. Activation function sigmoid with 5 layers has a lossvalue between those previous two models and a more reasonable boundary shape.

Model 1:

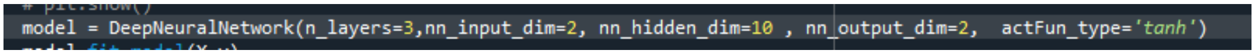


```
model = DeepNeuralNetwork(n_layers=3,nn_input_dim=2, nn_hidden_dim=5 , nn_output_dim=2,  actFun_type='tanh')
```

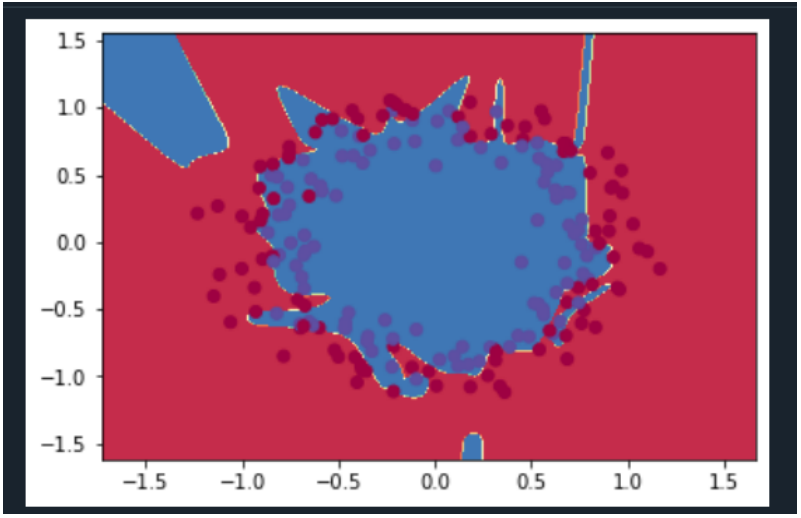


Loss after iteration 19000: 0.861026

Model 2:

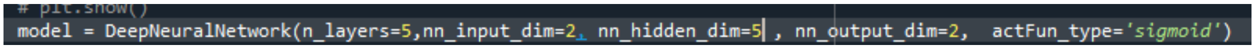


```
# plt.show()
model = DeepNeuralNetwork(n_layers=3,nn_input_dim=2, nn_hidden_dim=10 , nn_output_dim=2,  actFun_type='tanh')
model.fit_model(X,y)
```

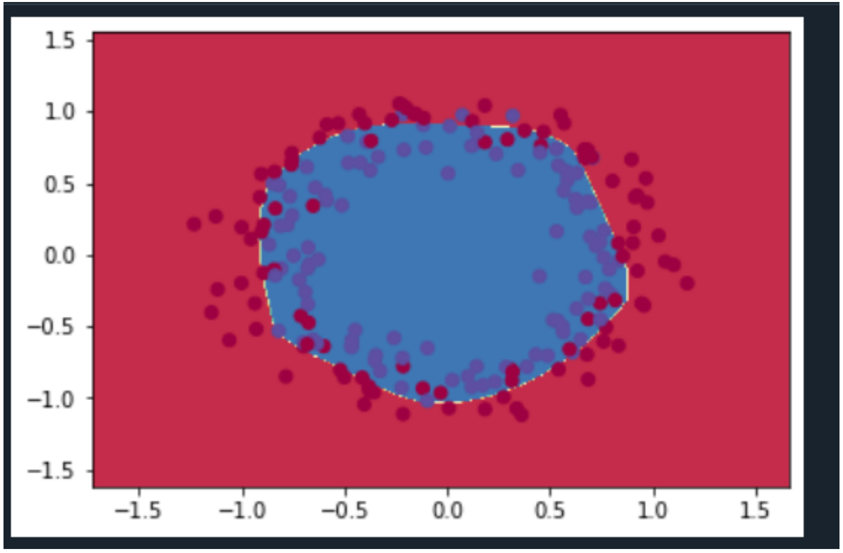


Loss after iteration 19000: 0.299084

Model 3:



```
# plt.show()
model = DeepNeuralNetwork(n_layers=5,nn_input_dim=2, nn_hidden_dim=5 , nn_output_dim=2,  actFun_type='sigmoid')
```



Loss after iteration 19000: 0.414022

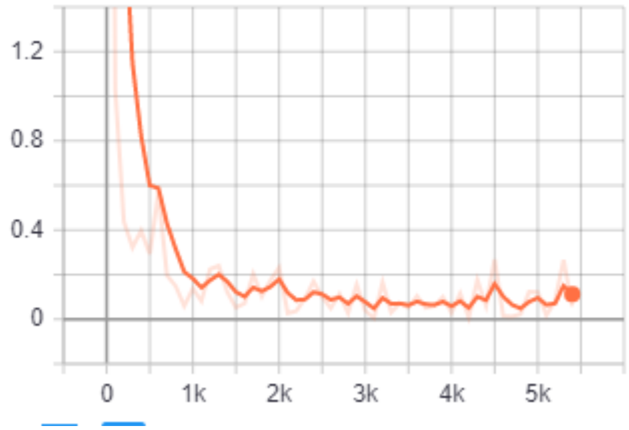
2 Training a Simple Deep Convolutional Network on MNIST

2-a-5

Final test accuracy: 0.9866

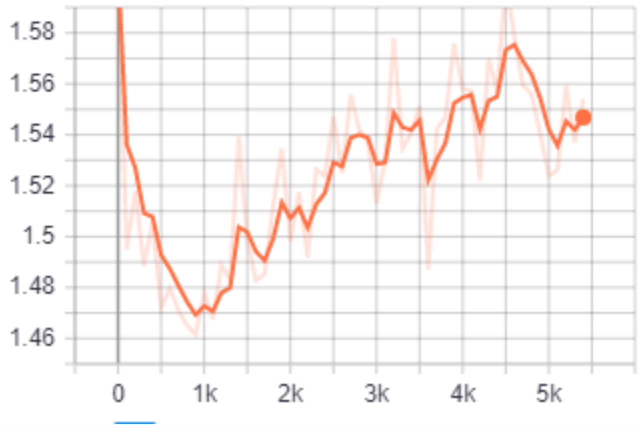
Training time: 391.691699

2-a-6:

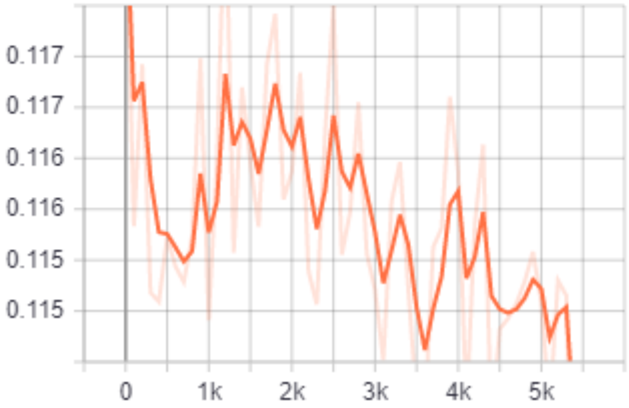


2-b

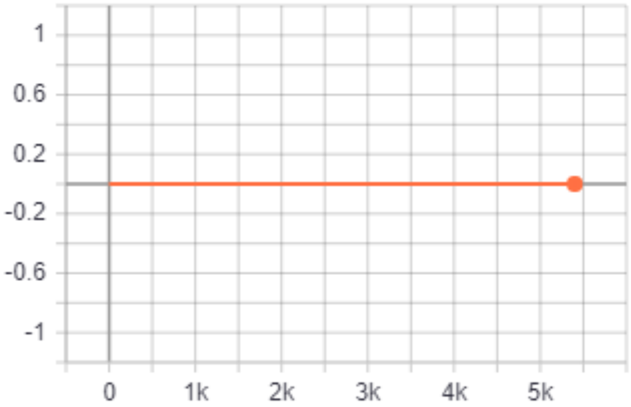
conv1_activation_max



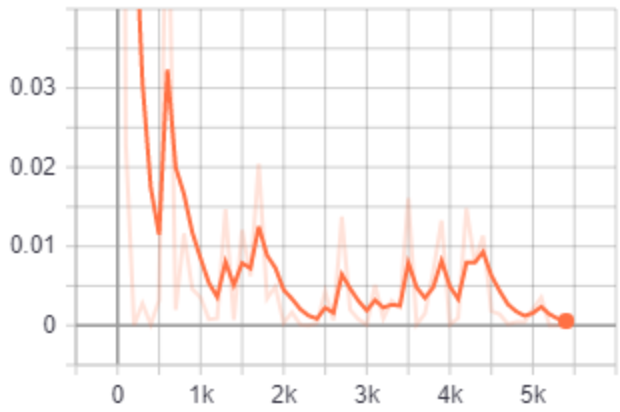
conv1_activation_mean



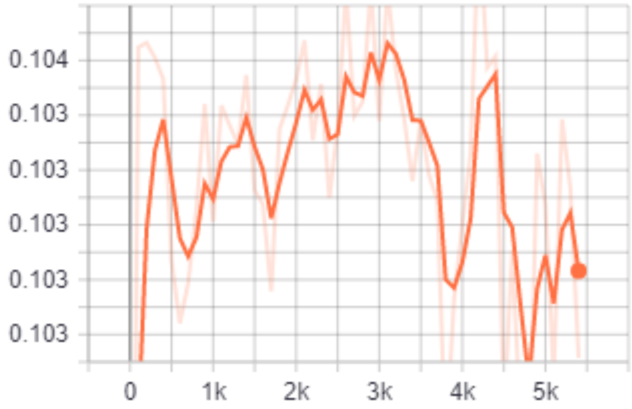
conv1_activation_min



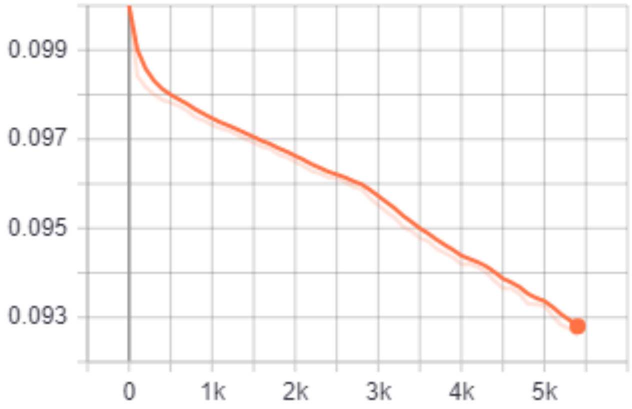
conv1_activation_std

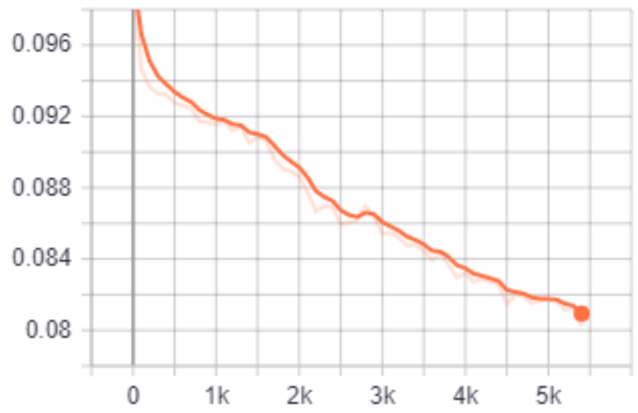


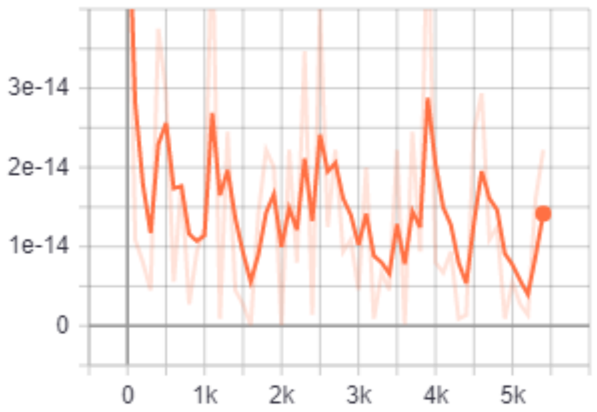
conv1_bias_max

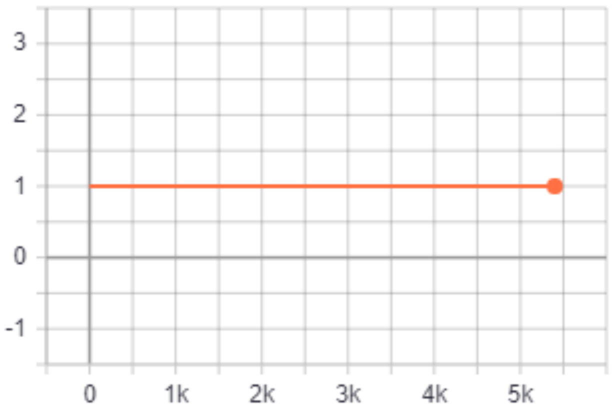


conv1_bias_mean

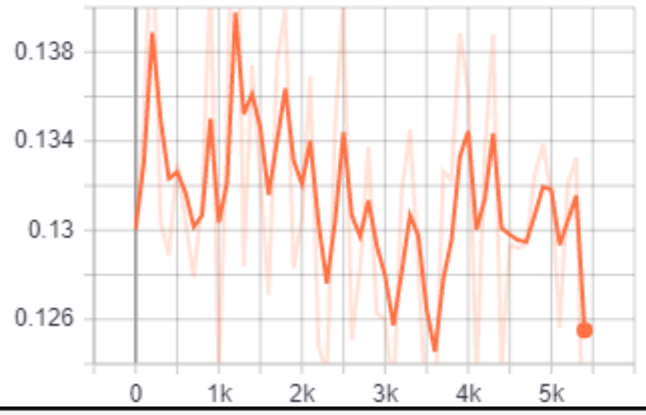



conv1_bias_min

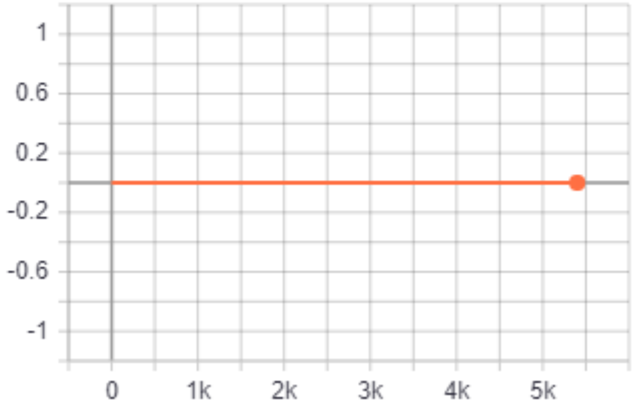

conv1_bias_std


conv1_input_max

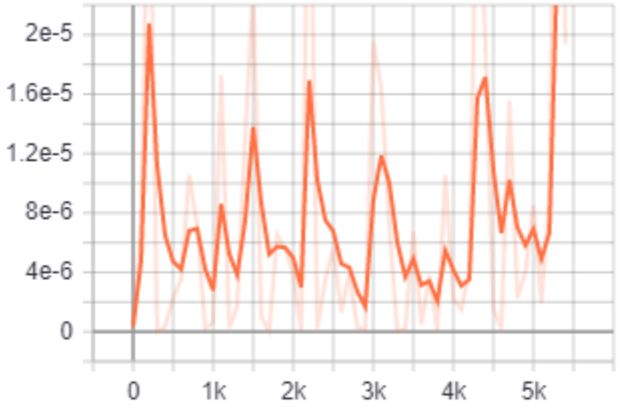
conv1_input_mean

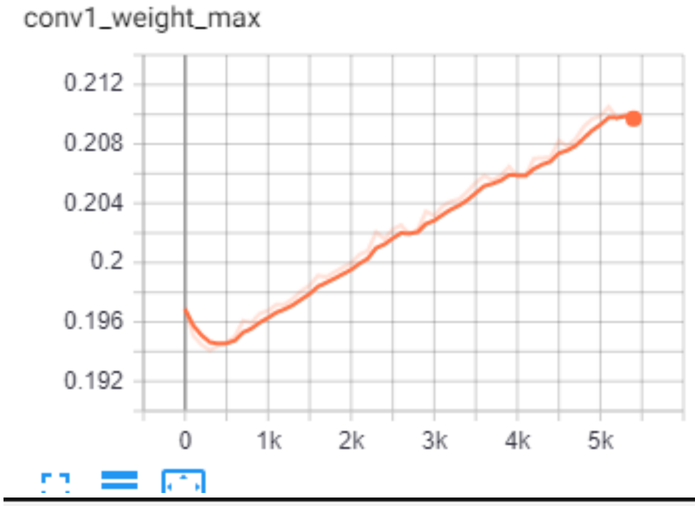


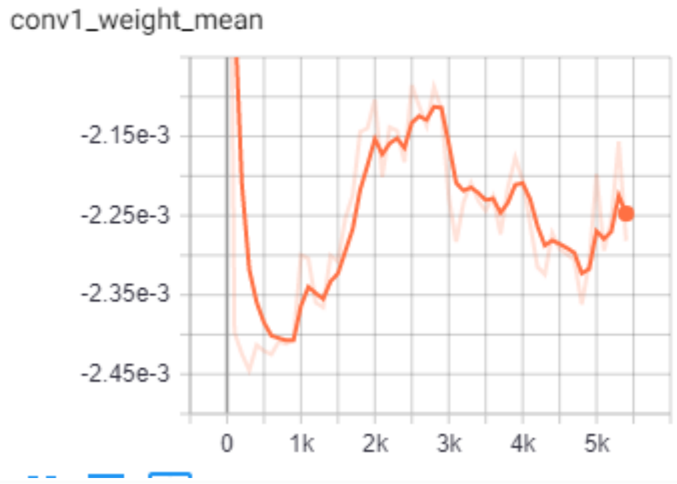
conv1_input_min

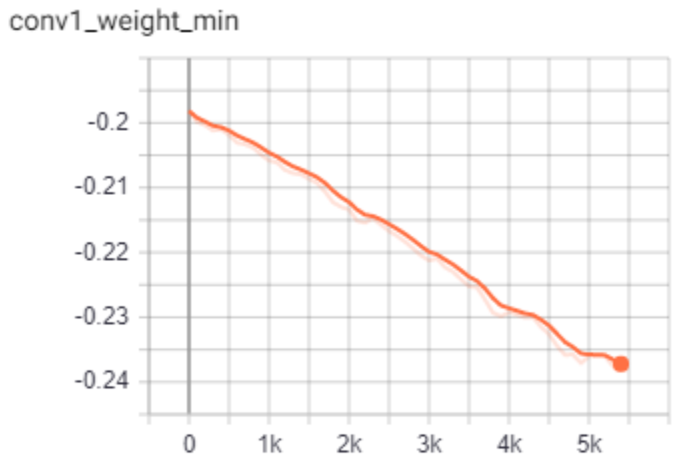


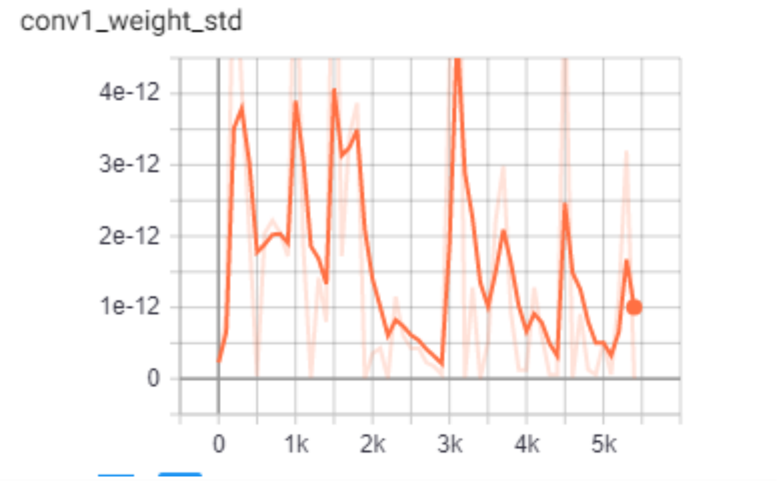
conv1_input_std

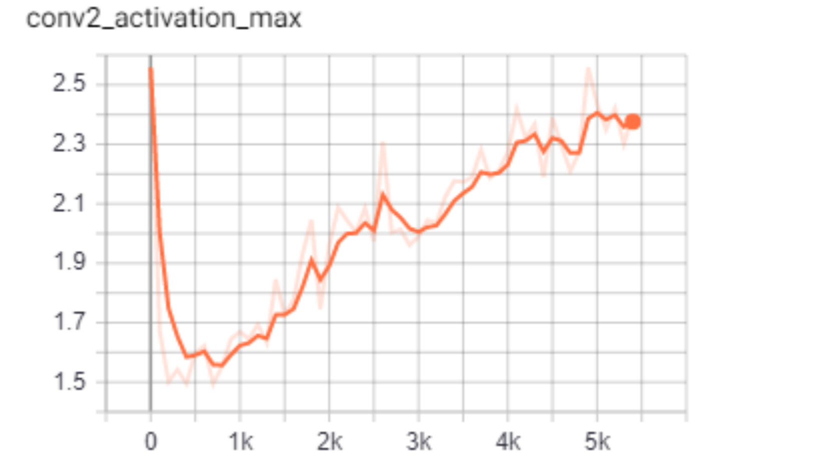


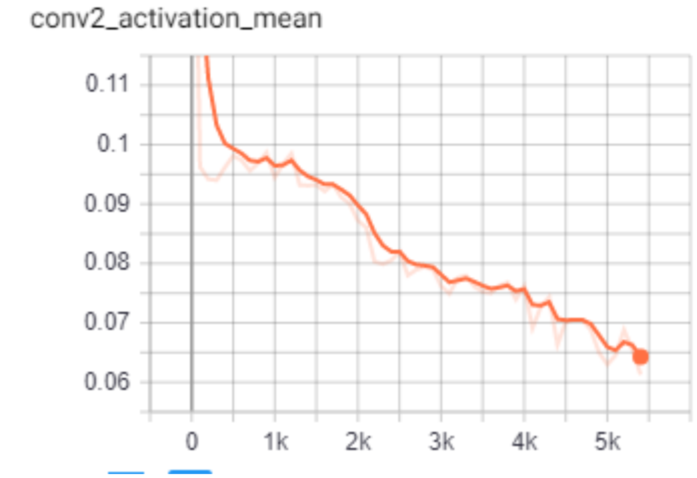

conv1_weight_max
0.212
0.208
0.204
0.2
0.196
0.192
0
1k
2k
3k
4k
5k


conv1_weight_mean
-2.15e-3
-2.25e-3
-2.35e-3
-2.45e-3
0
1k
2k
3k
4k
5k

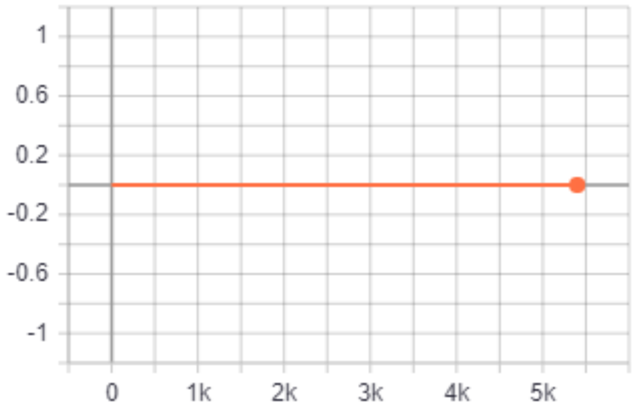

conv1_weight_min
-0.2
-0.21
-0.22
-0.23
-0.24
0
1k
2k
3k
4k
5k


conv1_weight_std
4e-12
3e-12
2e-12
1e-12
0
0
1k
2k
3k
4k
5k

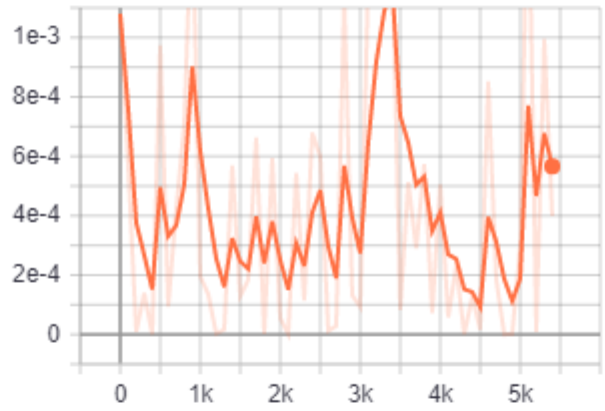

conv2_activation_max
2.5
2.3
2.1
1.9
1.7
1.5
0
1k
2k
3k
4k
5k


conv2_activation_mean
0.11
0.1
0.09
0.08
0.07
0.06
0
1k
2k
3k
4k
5k

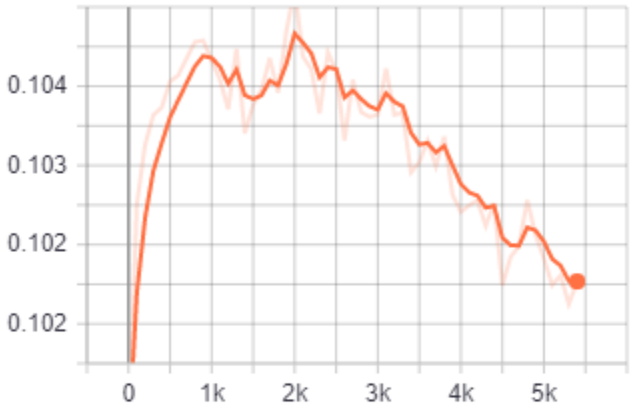
conv2_activation_min



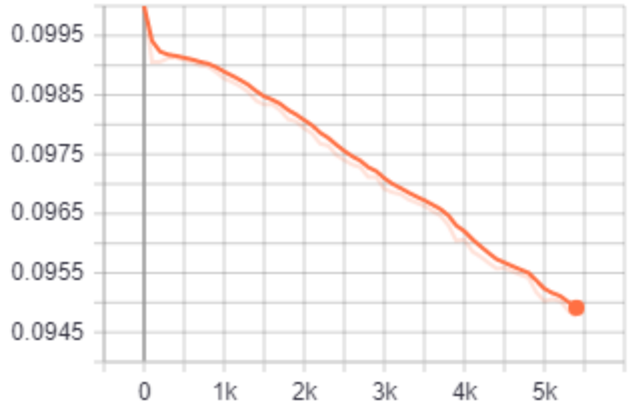
conv2_activation_std



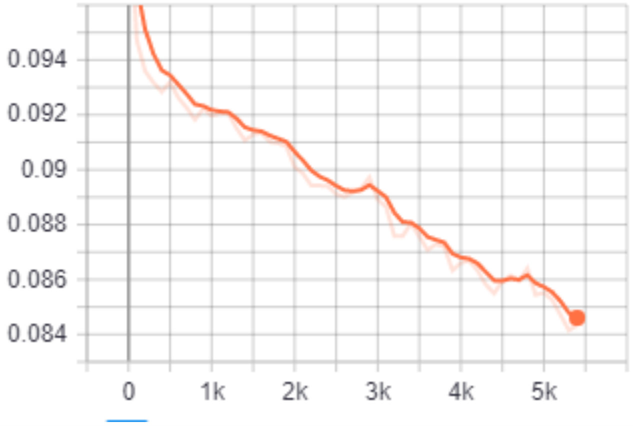
conv2_bias_max



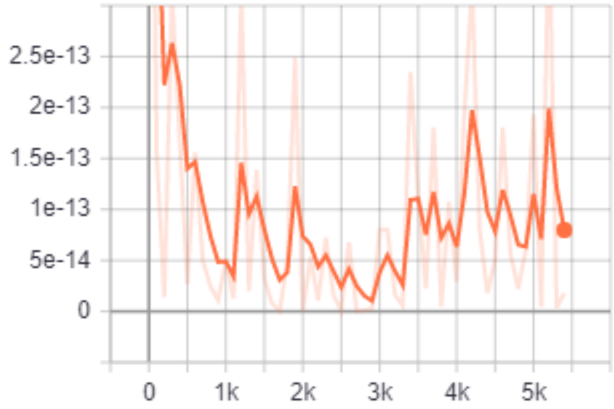
conv2_bias_mean

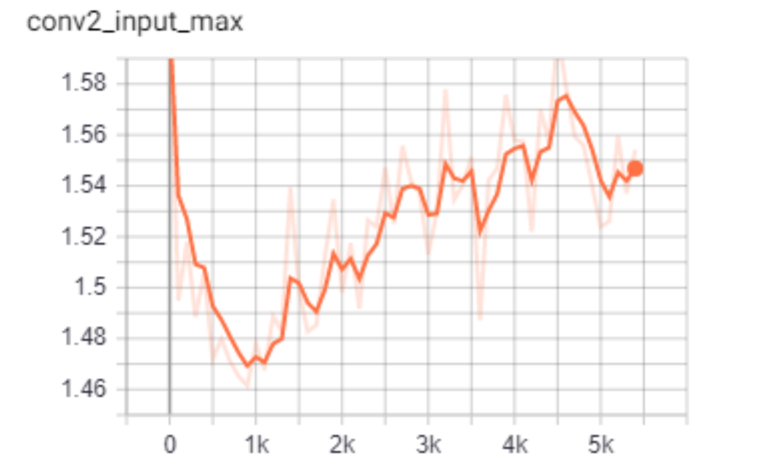


conv2_bias_min

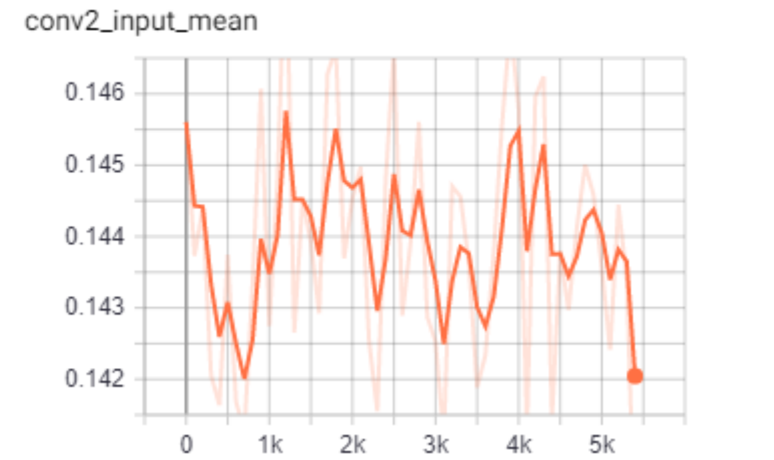


conv2_bias_std

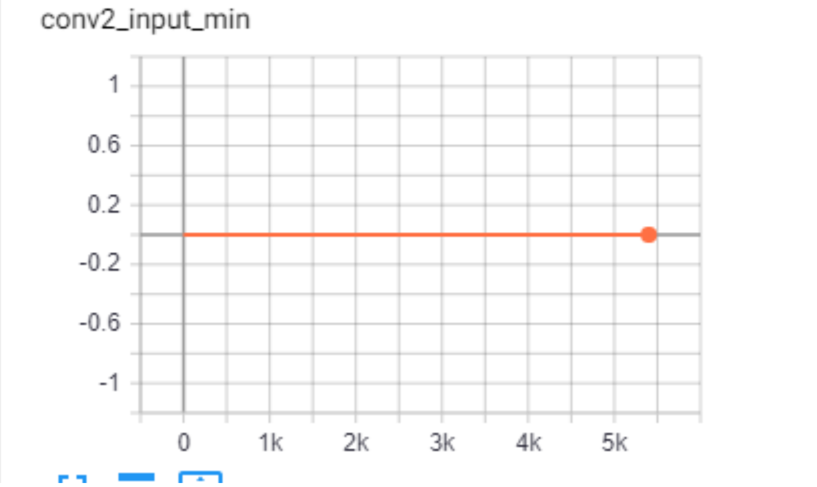




conv2_input_max

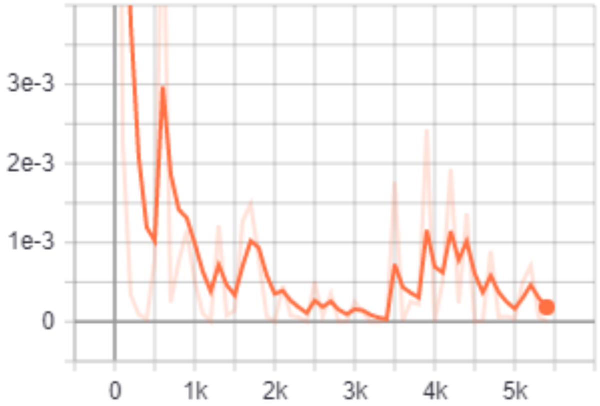


conv2_input_mean

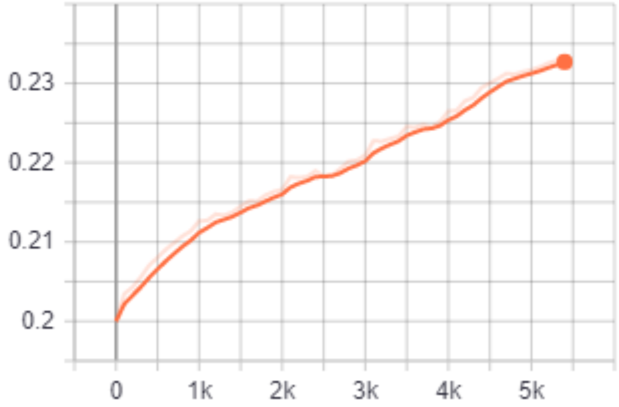


conv2_input_min

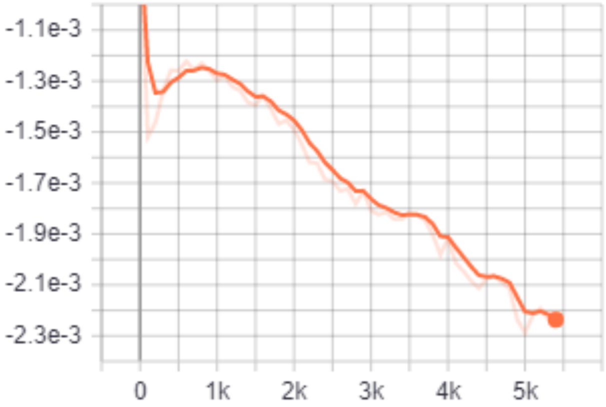
conv2_input_std



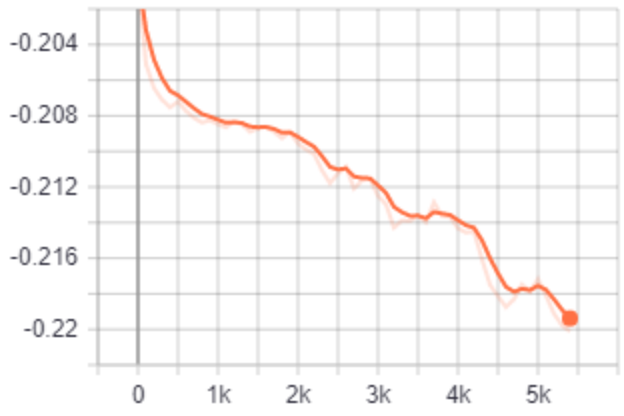
conv2_weight_max



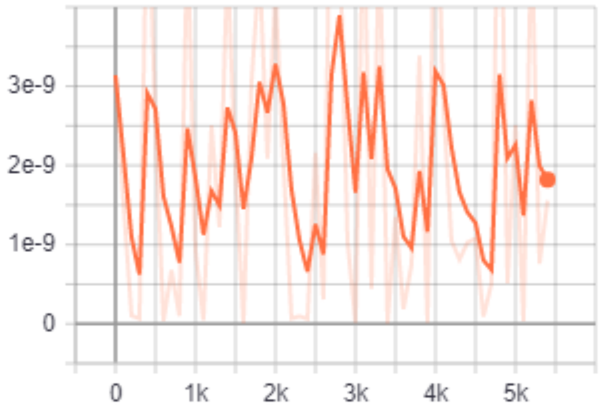
conv2_weight_mean



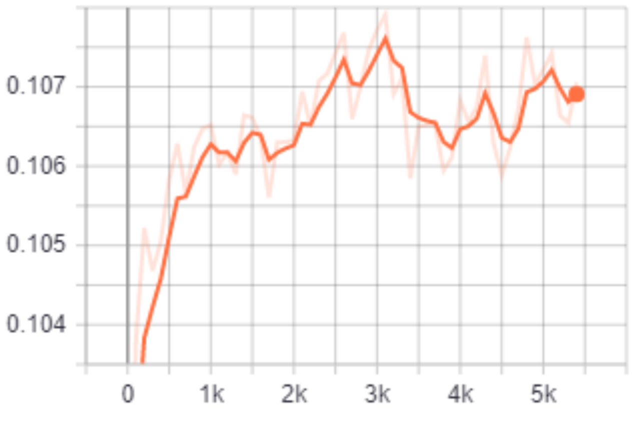
conv2_weight_min

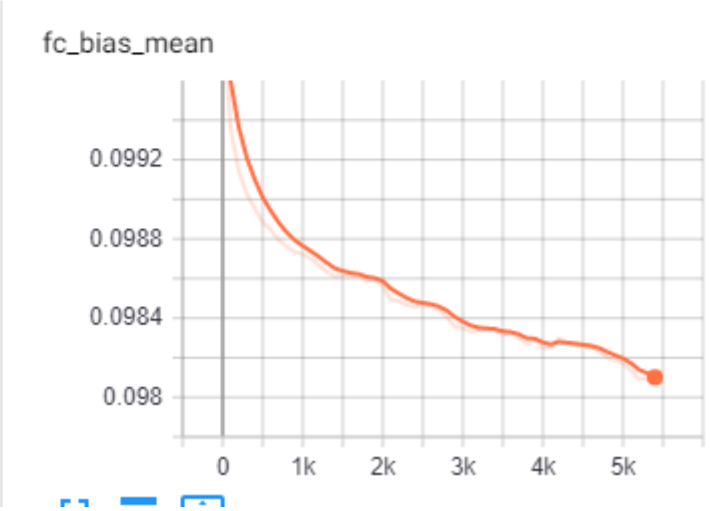


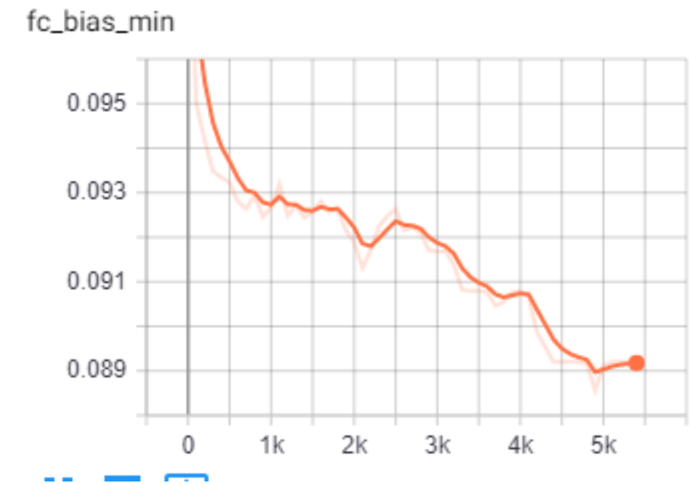
conv2_weight_std

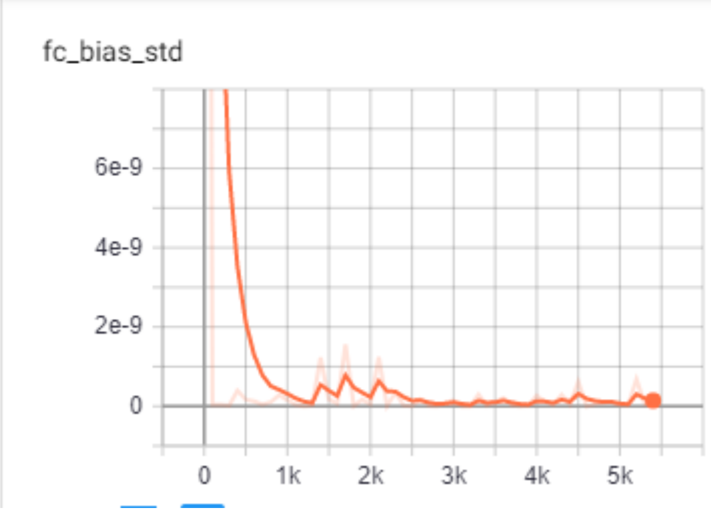


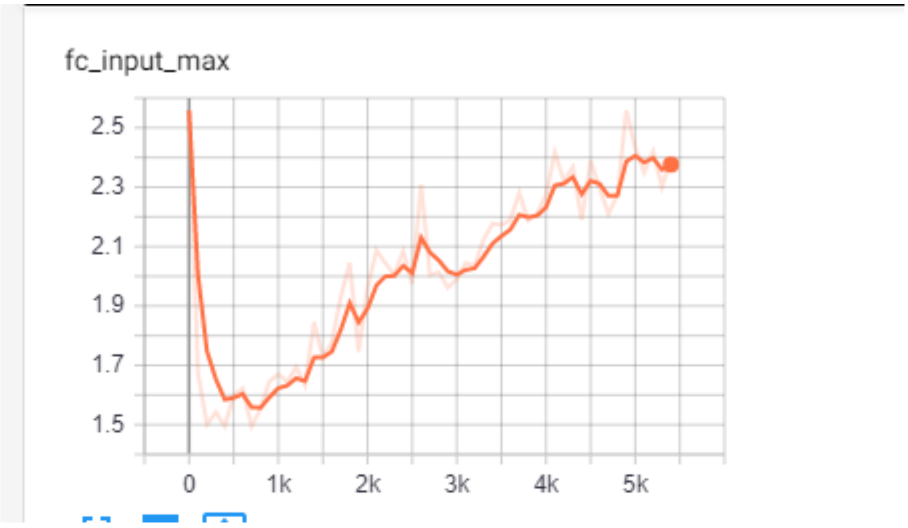
fc_bias_max



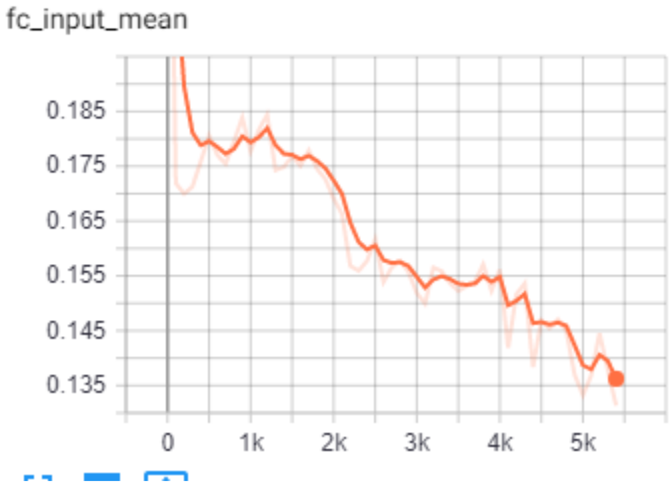

fc_bias_mean
0.0992
0.0988
0.0984
0.098
0
1k
2k
3k
4k
5k


fc_bias_min
0.095
0.093
0.091
0.089
0
1k
2k
3k
4k
5k

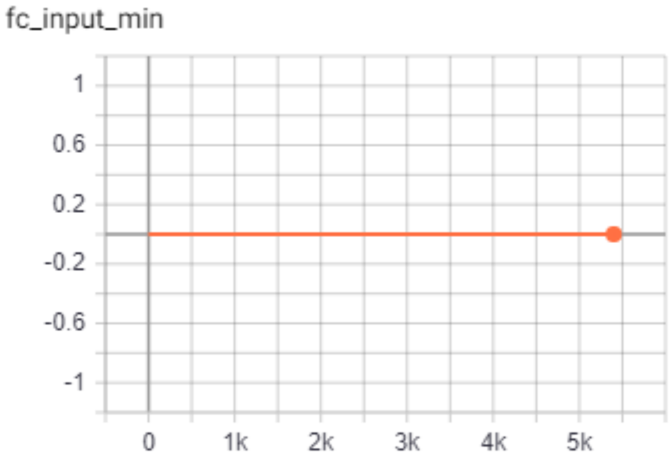

fc_bias_std
6e-9
4e-9
2e-9
0
0
1k
2k
3k
4k
5k



fc_input_max

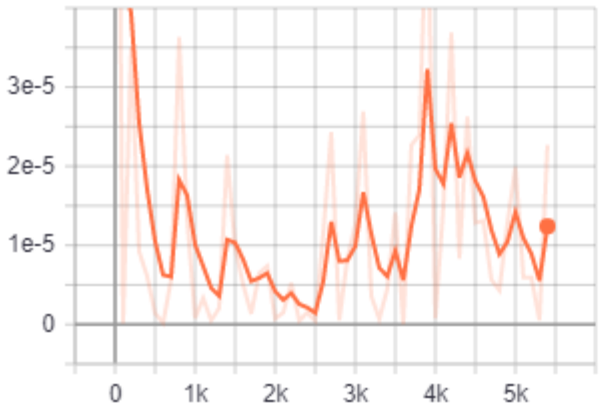


fc_input_mean

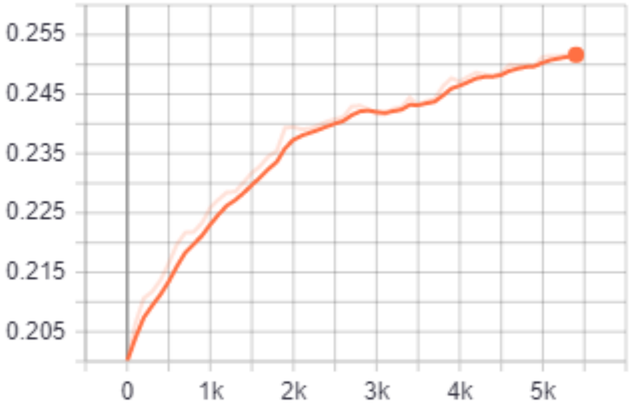


fc_input_min

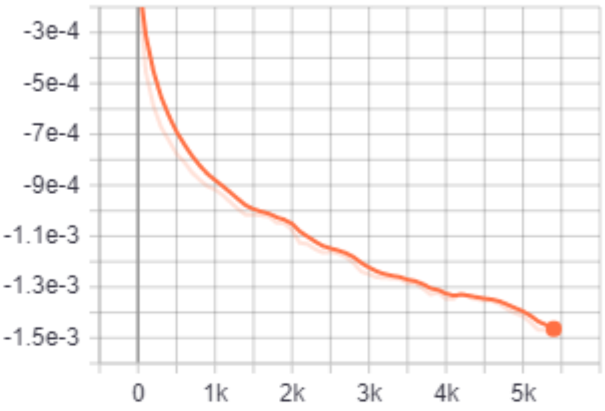
fc_input_std

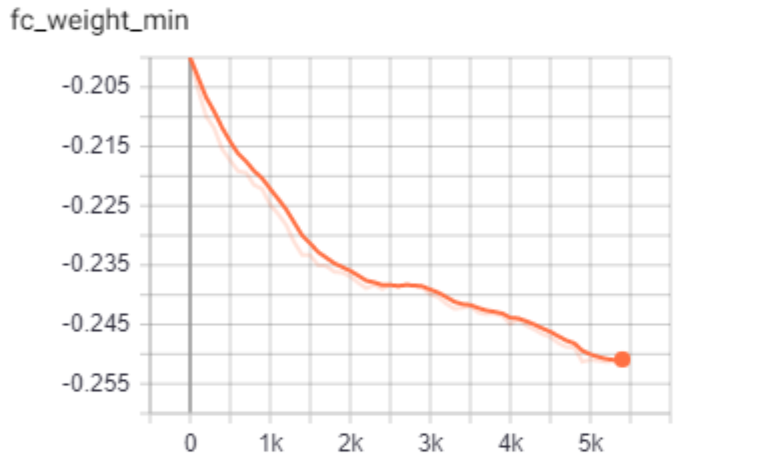


fc_weight_max

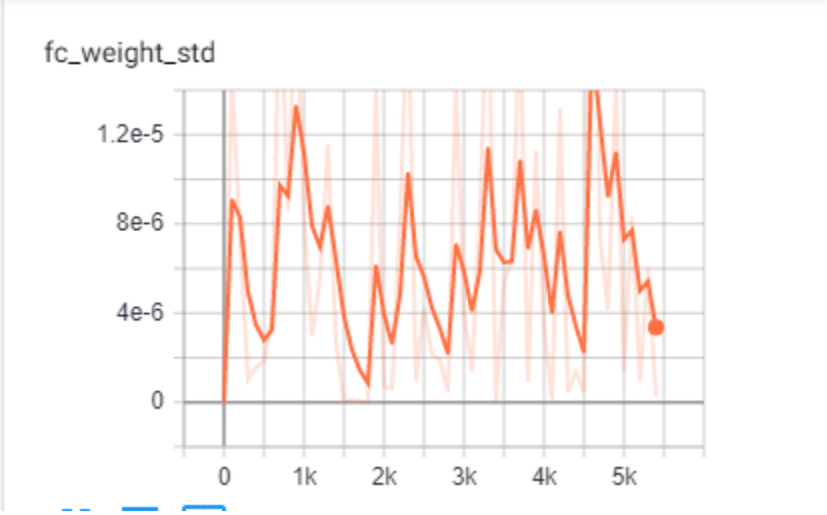


fc_weight_mean

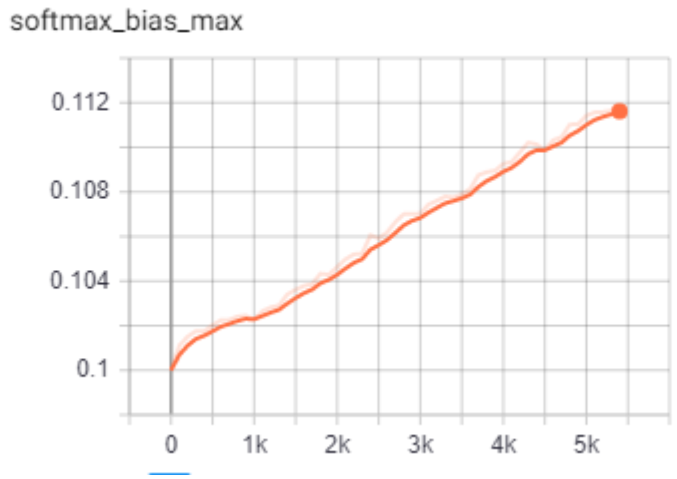




fc_weight_min

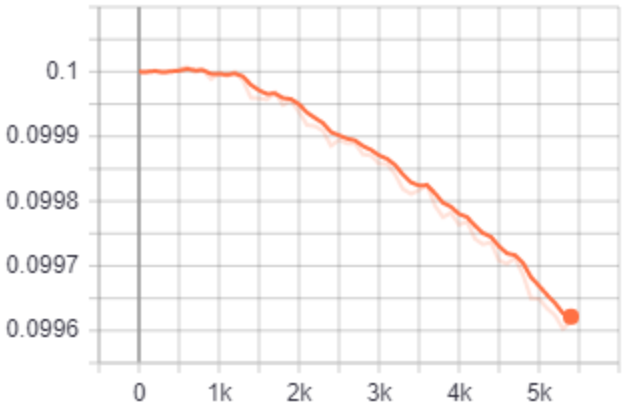


fc_weight_std

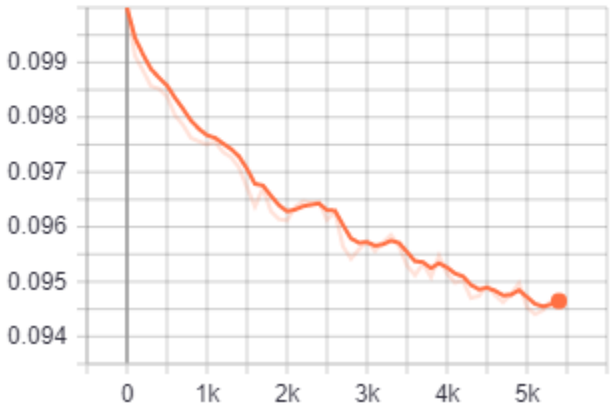


softmax_bias_max

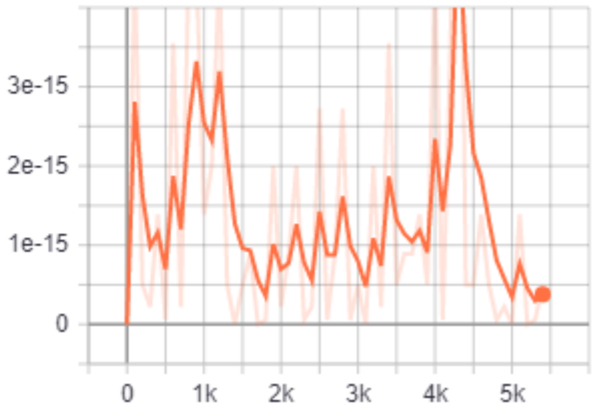
softmax_bias_mean

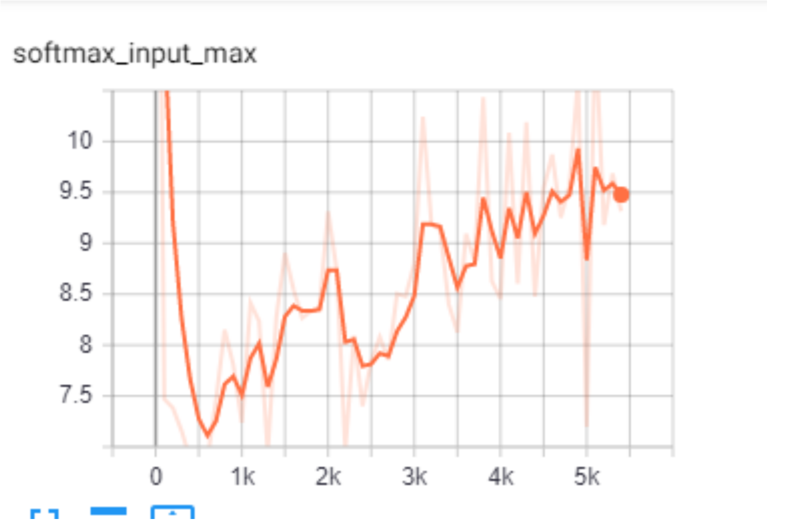


softmax_bias_min

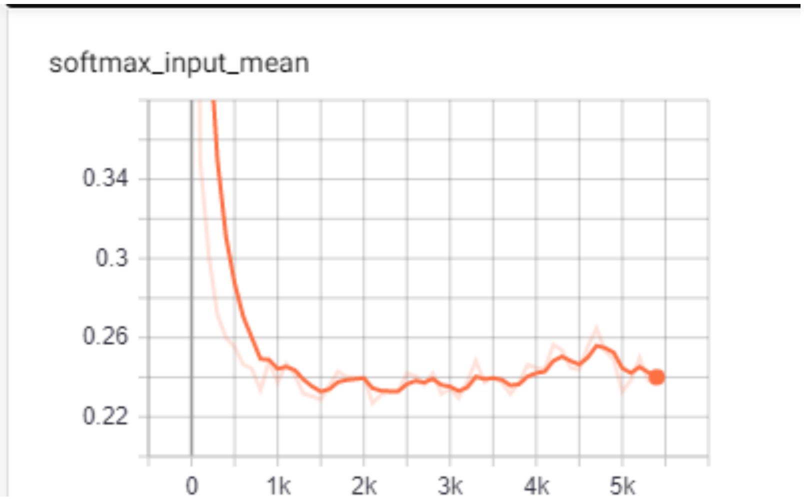


softmax_bias_std

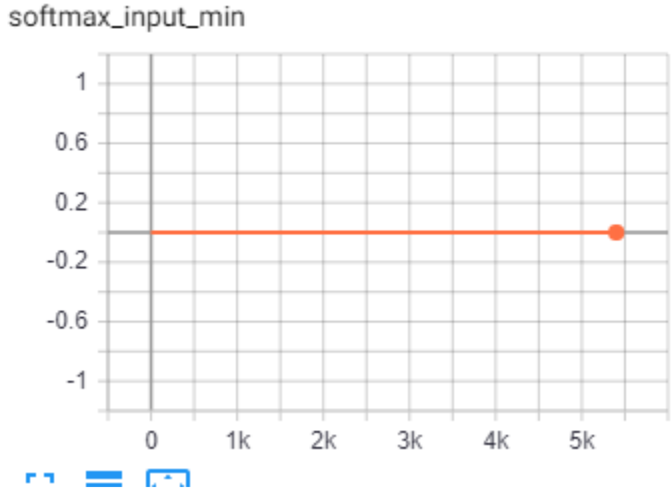




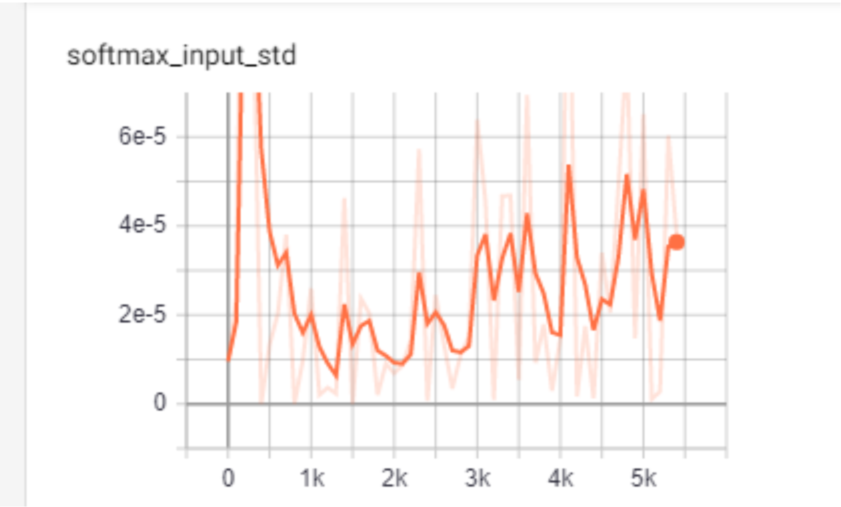
softmax_input_max

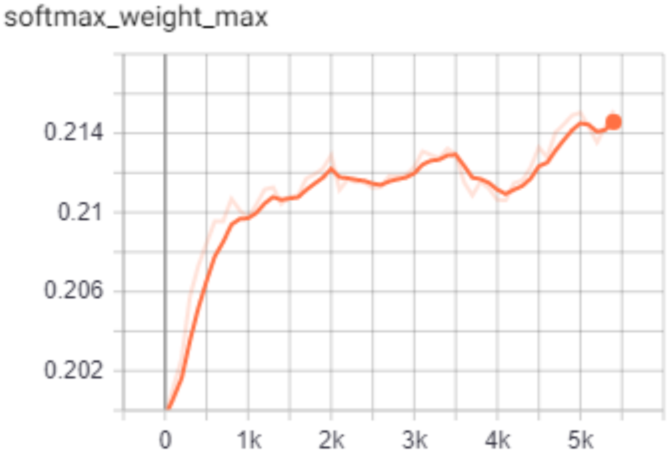


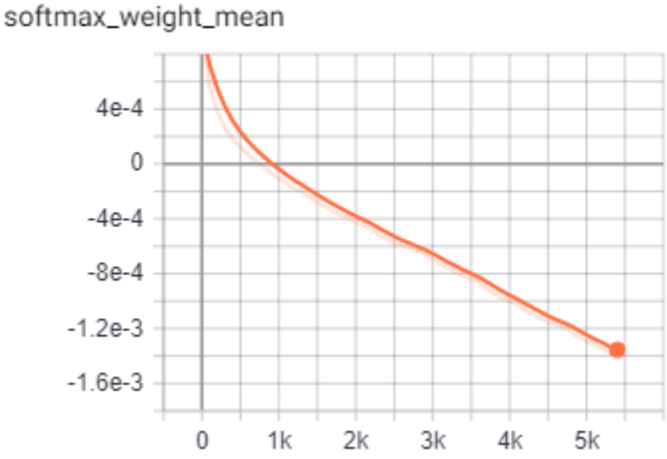
softmax_input_mean

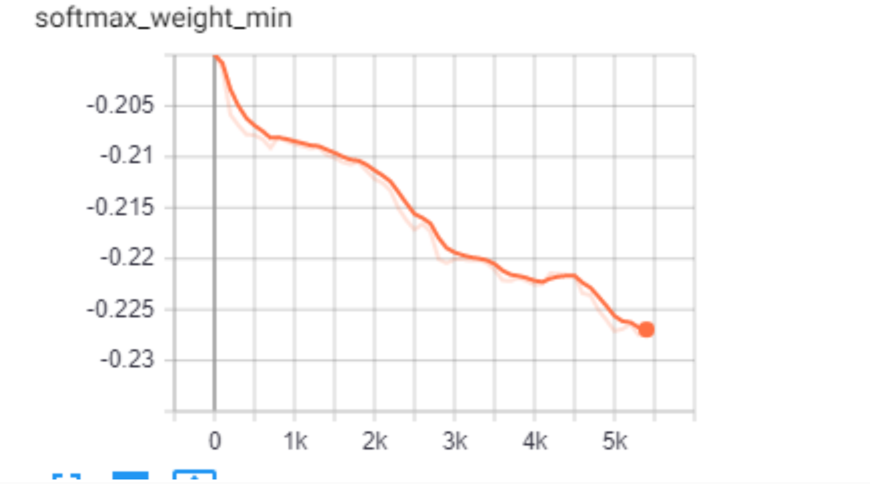


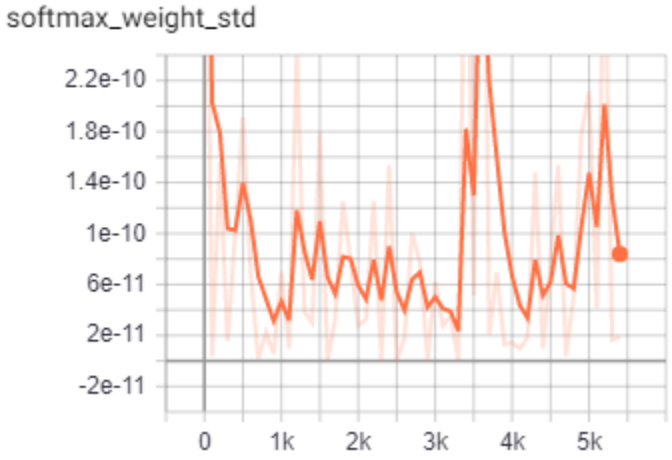
softmax_input_min

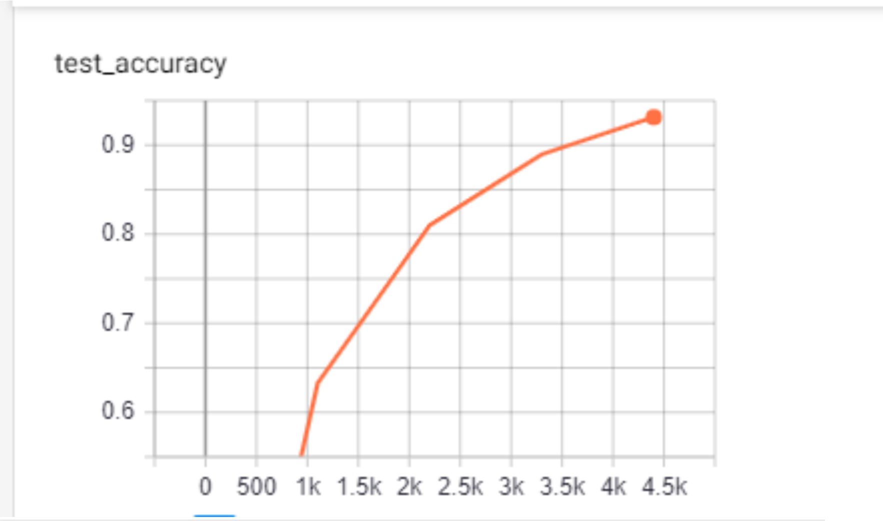

softmax_input_std
6e-5
4e-5
2e-5
0
0
1k
2k
3k
4k
5k

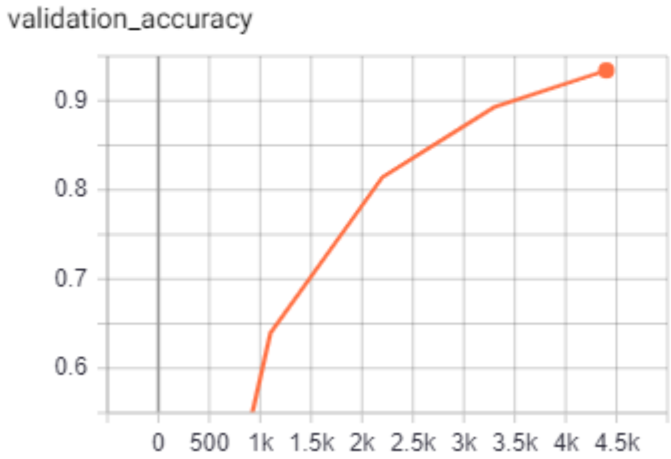

softmax_weight_max
0.214
0.21
0.206
0.202
0
1k
2k
3k
4k
5k

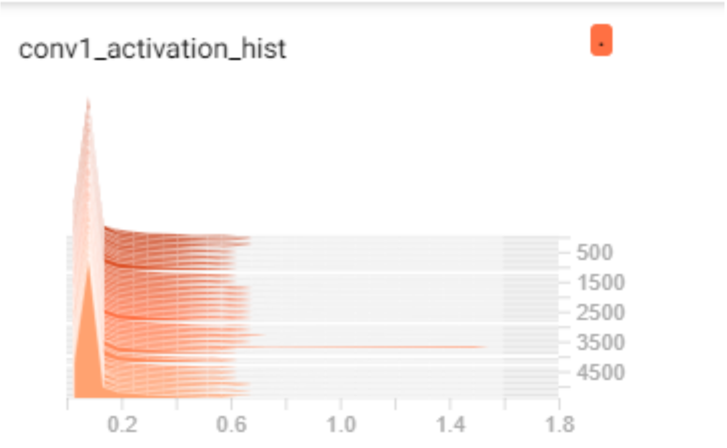

softmax_weight_mean
4e-4
0
-4e-4
-8e-4
-1.2e-3
-1.6e-3
0
1k
2k
3k
4k
5k

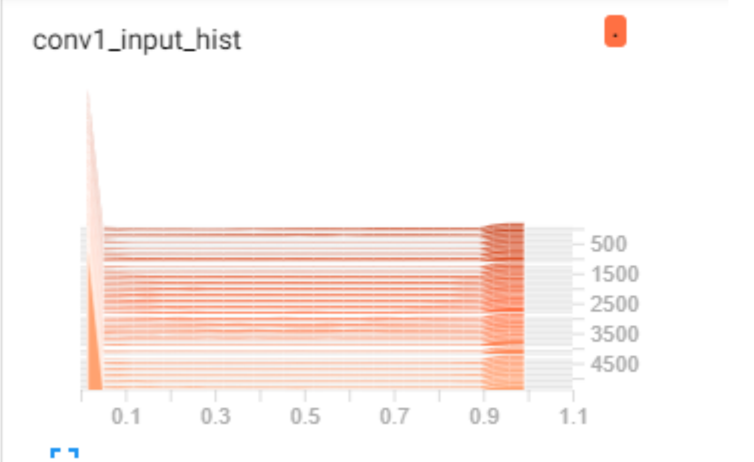

softmax_weight_min

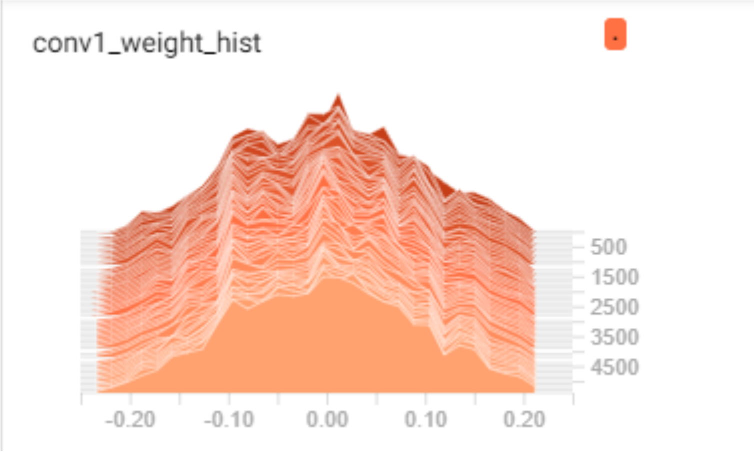

softmax_weight_std

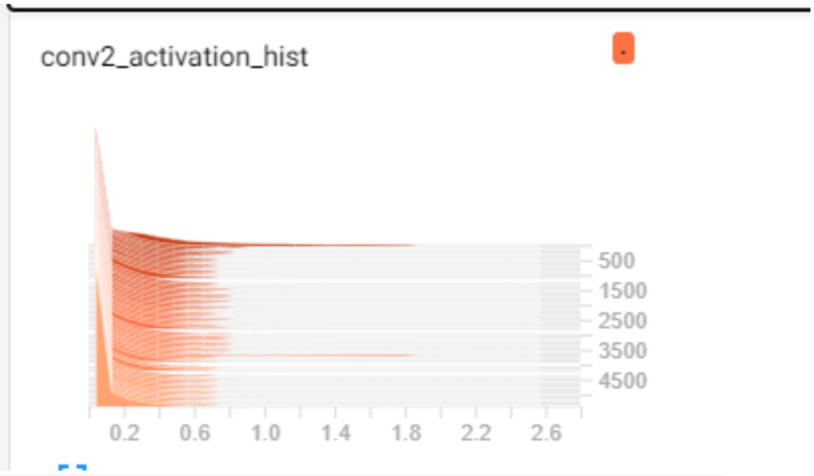

test_accuracy
0.9
0.8
0.7
0.6
0
500
1k
1.5k
2k
2.5k
3k
3.5k
4k
4.5k

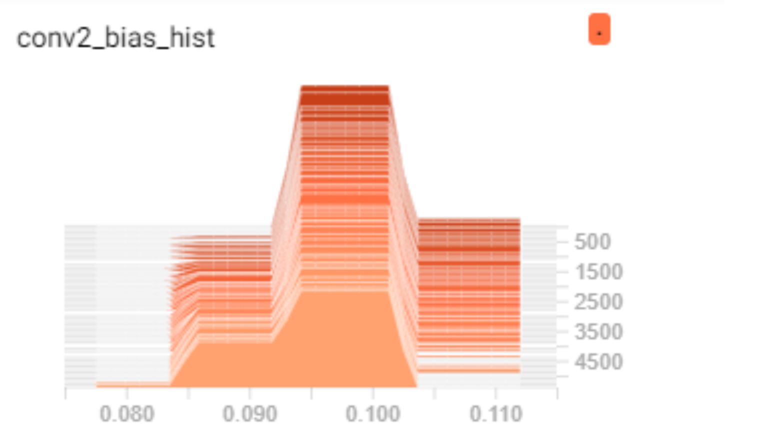

validation_accuracy
0.9
0.8
0.7
0.6
0
500
1k
1.5k
2k
2.5k
3k
3.5k
4k
4.5k

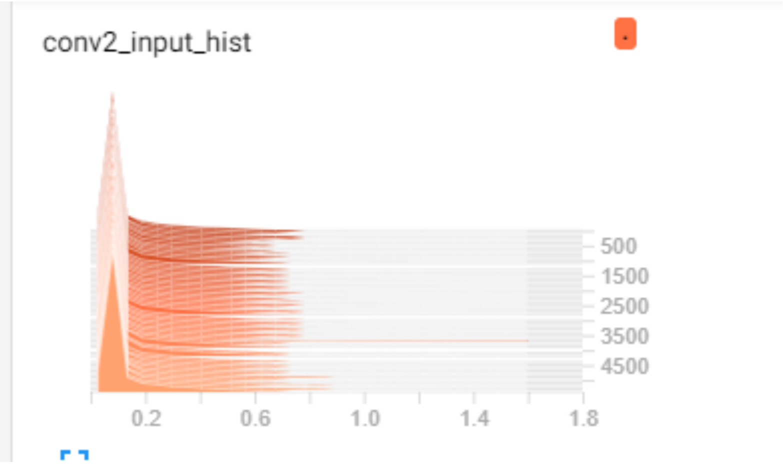

conv1_activation_hist

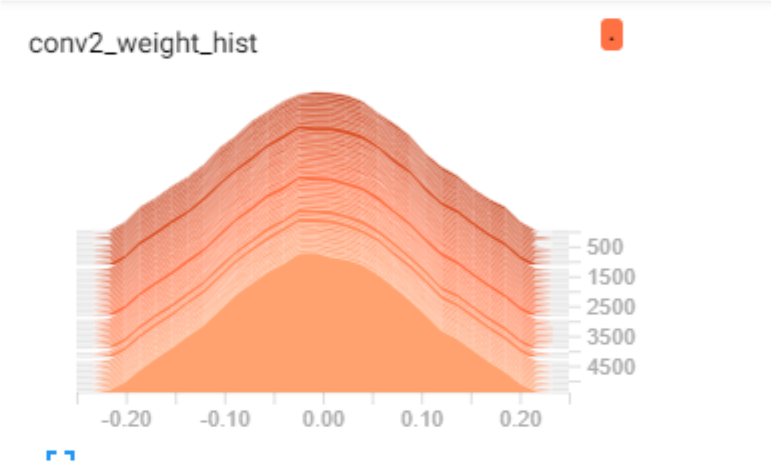

conv1_input_hist

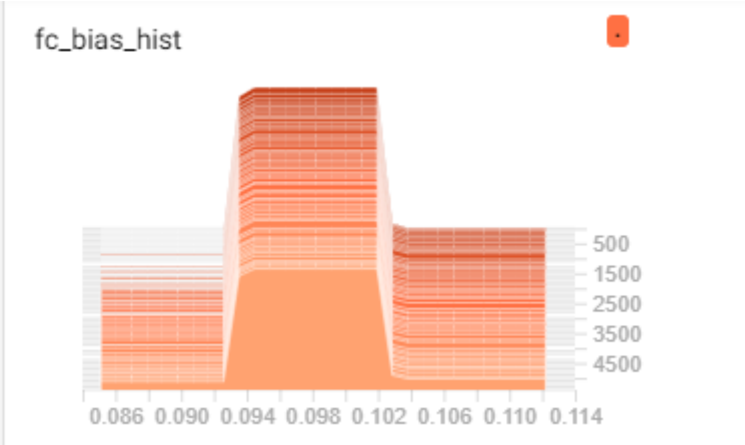

conv1_weight_hist

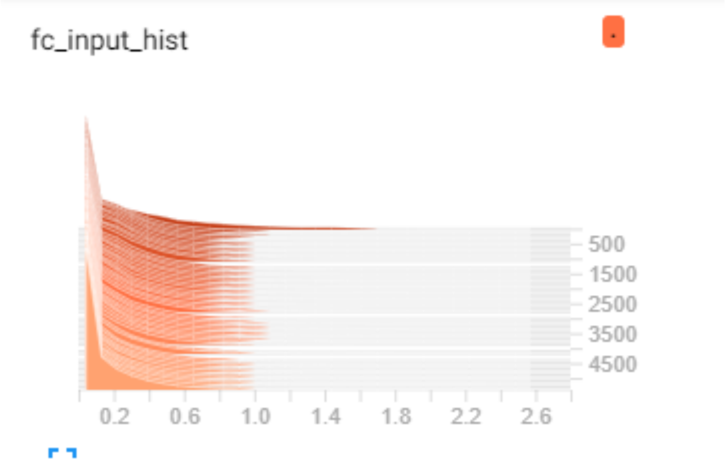

conv2_activation_hist

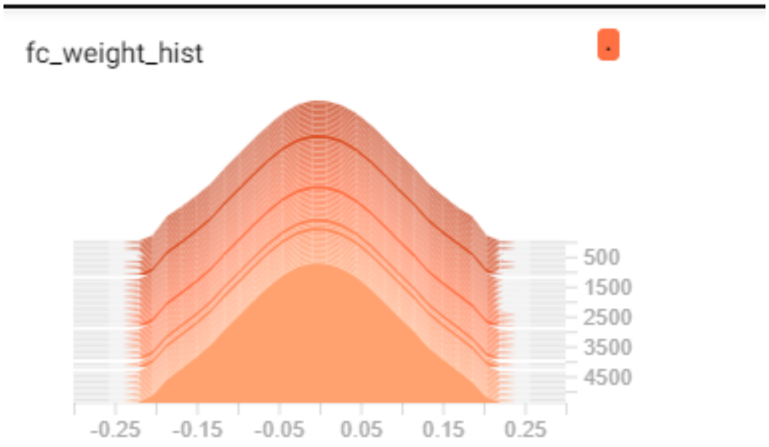

conv2_bias_hist


conv2_input_hist

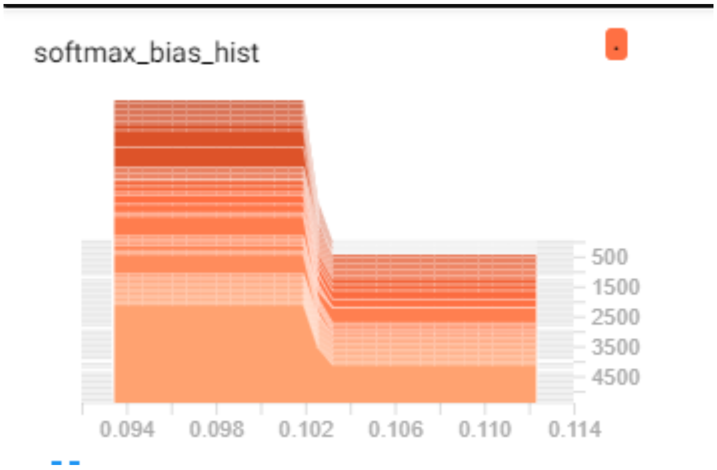

conv2_weight_hist
500
1500
2500
3500
4500
-0.20
-0.10
0.00
0.10
0.20


fc_bias_hist
500
1500
2500
3500
4500
0.086
0.090
0.094
0.098
0.102
0.106
0.110
0.114

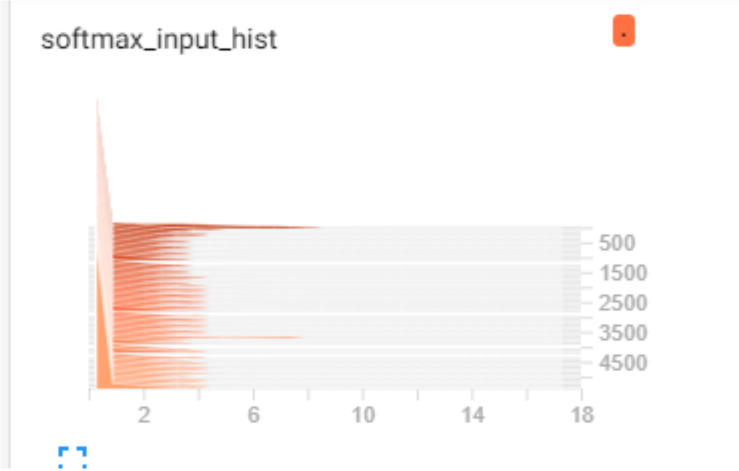

fc_input_hist
500
1500
2500
3500
4500
0.2
0.6
1.0
1.4
1.8
2.2
2.6

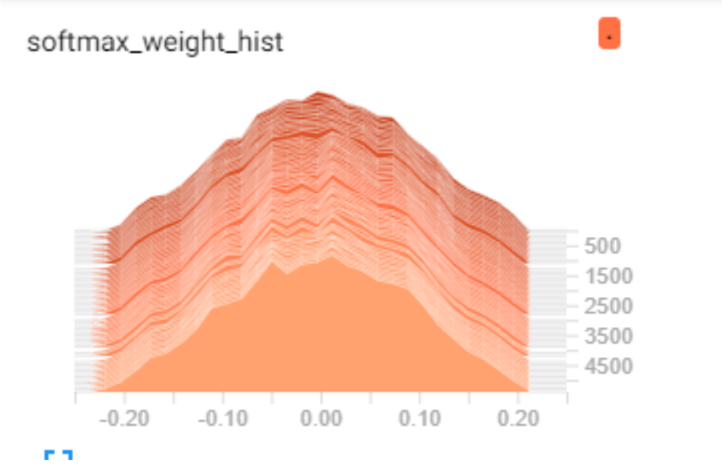


fc_weight_hist



softmax_bias_hist

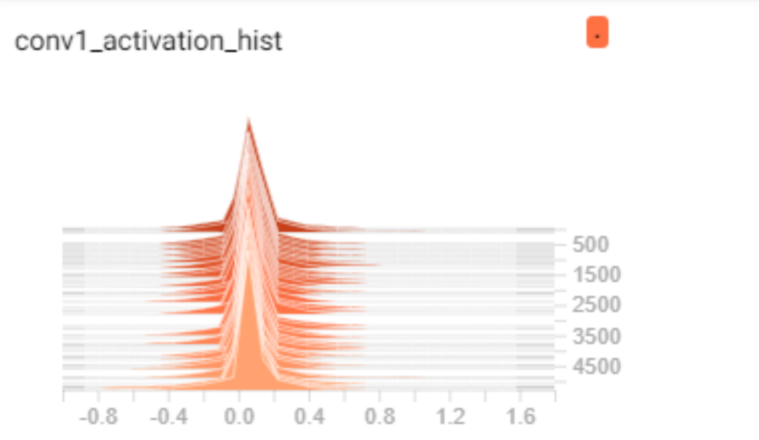


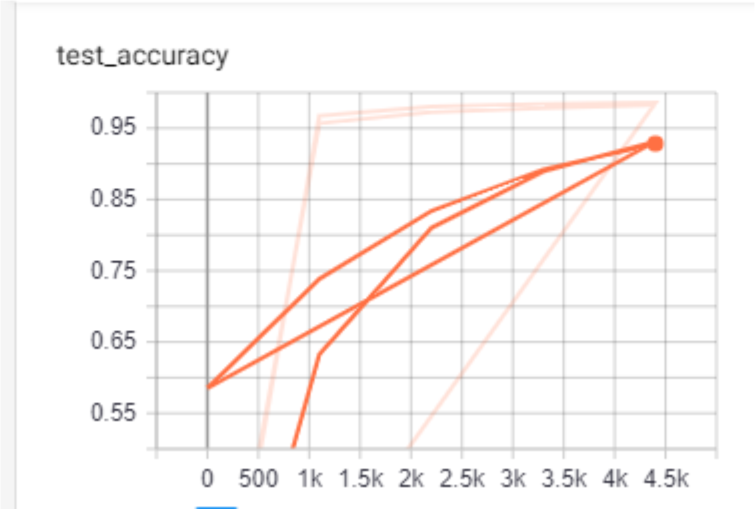


2-c:

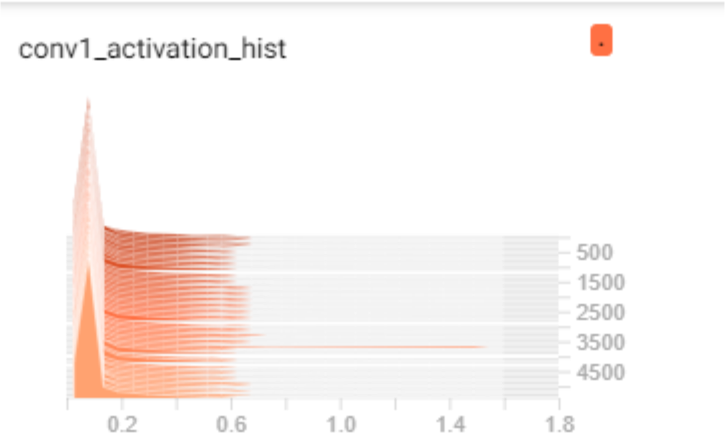
Using tanh as activation function, test accuracy increases faster and network learns faster.

with tanh:

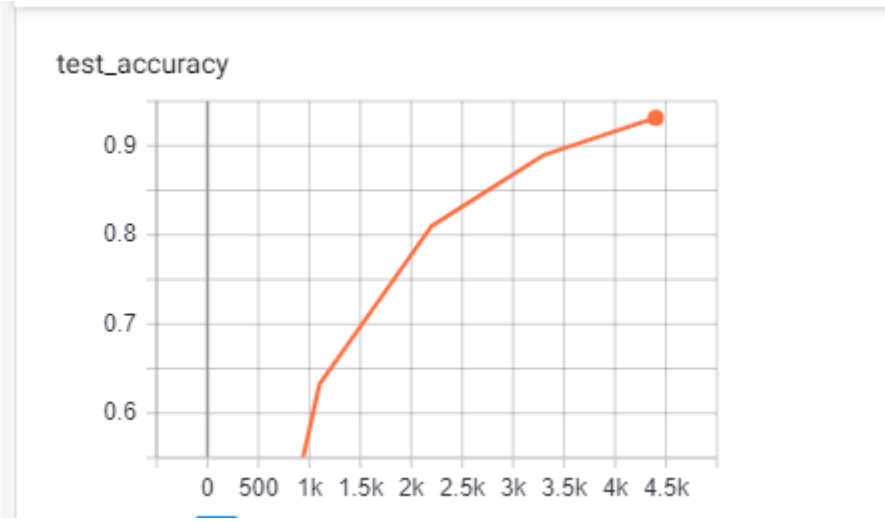

conv1_activation_hist
500
1500
2500
3500
4500
-0.8
-0.4
0.0
0.4
0.8
1.2
1.6


test_accuracy
0.95
0.85
0.75
0.65
0.55
0
500
1k
1.5k
2k
2.5k
3k
3.5k
4k
4.5k

With relu:



conv1_activation_hist



test_accuracy