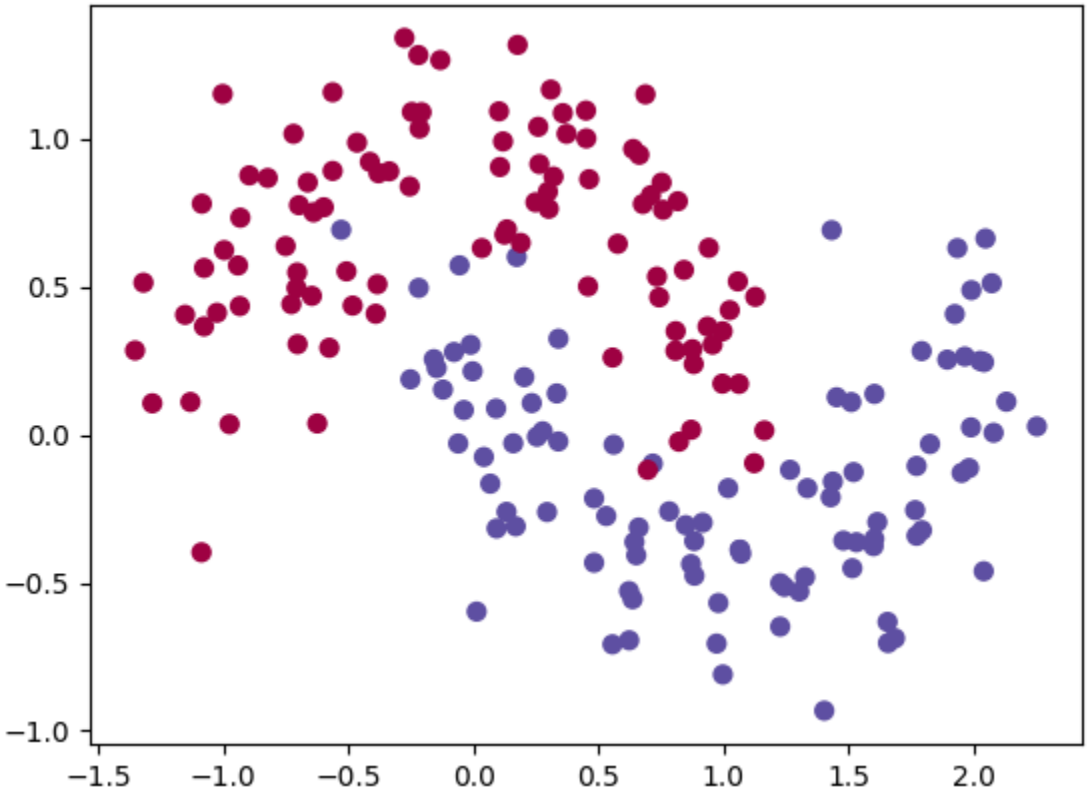
1. Backpropagation in a Simple Neural Network

Data Set:



Implement Function:

Relu: max(0,x)
Sigmoid: 1.0 / (1.0 + exp(-x))
Tanh: (e^x – e^-x) / (e^x + e^-x)

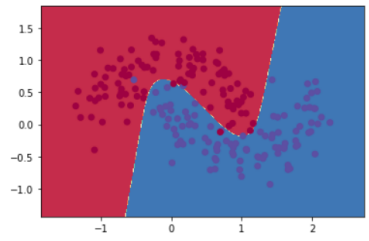
Activation Function Derivatives:
Relu:
X is 0 if X is smaller equal to zero, 1 if otherwise
Sigmoid: Sigmoid(x)(1-Sigmoid(x))
Tanh: 1- Tanh^2(x)

Result:

Sigmoid:

Hidden layer = 3 (default)

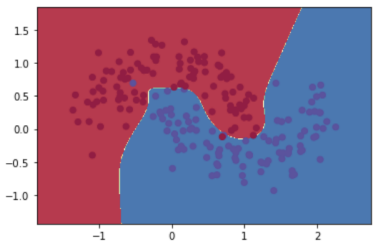
```
Loss after iteration 0: 0.628571
Loss after iteration 1000: 0.088431
Loss after iteration 2000: 0.079598
Loss after iteration 3000: 0.078604
Loss after iteration 4000: 0.078330
Loss after iteration 5000: 0.078233
Loss after iteration 6000: 0.078192
Loss after iteration 7000: 0.078174
Loss after iteration 8000: 0.078166
Loss after iteration 9000: 0.078161
Loss after iteration 10000: 0.078159
Loss after iteration 11000: 0.078158
Loss after iteration 12000: 0.078157
Loss after iteration 13000: 0.078156
Loss after iteration 14000: 0.078156
Loss after iteration 15000: 0.078156
Loss after iteration 16000: 0.078156
Loss after iteration 17000: 0.078156
Loss after iteration 18000: 0.078156
Loss after iteration 19000: 0.078155
```



Increase Hidden Layer:

Hidden layer = 10

```
Loss after iteration 0: 1.015841
Loss after iteration 1000: 0.097373
Loss after iteration 2000: 0.085633
Loss after iteration 3000: 0.079321
Loss after iteration 4000: 0.075821
Loss after iteration 5000: 0.073327
Loss after iteration 6000: 0.070657
Loss after iteration 7000: 0.069038
Loss after iteration 8000: 0.067980
Loss after iteration 9000: 0.067161
Loss after iteration 10000: 0.066505
Loss after iteration 11000: 0.066001
Loss after iteration 12000: 0.065620
Loss after iteration 13000: 0.065332
Loss after iteration 14000: 0.065107
Loss after iteration 15000: 0.064921
Loss after iteration 16000: 0.064752
Loss after iteration 17000: 0.064571
Loss after iteration 18000: 0.064332
Loss after iteration 19000: 0.063966
```

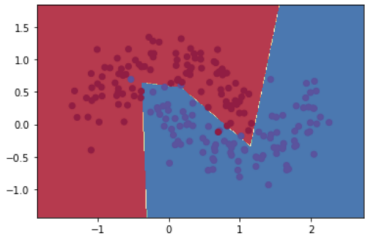


The start loss is greater if given a more hidden layer, but loss after iteration is smaller with a more hidden layer; also the decision boundaries are more distinctive with an increased layer. However, I tried increasing the hidden layer to 20, 25…100, there is no clear change in loss after iteration and decision boundary.

Relu:

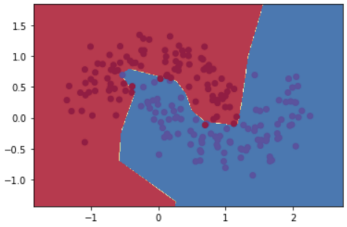
Hidden layer = 3

```
Loss after iteration 0: 0.560274
Loss after iteration 1000: 0.072179
Loss after iteration 2000: 0.071301
Loss after iteration 3000: 0.071159
Loss after iteration 4000: 0.071190
Loss after iteration 5000: 0.071136
Loss after iteration 6000: 0.071276
Loss after iteration 7000: 0.071090
Loss after iteration 8000: 0.071265
Loss after iteration 9000: 0.071084
Loss after iteration 10000: 0.071090
Loss after iteration 11000: 0.071087
Loss after iteration 12000: 0.071086
Loss after iteration 13000: 0.071069
Loss after iteration 14000: 0.071114
Loss after iteration 15000: 0.071074
Loss after iteration 16000: 0.071113
Loss after iteration 17000: 0.071071
Loss after iteration 18000: 0.071090
Loss after iteration 19000: 0.071219
```



Hidden layer = 10

```
Loss after iteration 0: 1.248780
Loss after iteration 1000: 0.057022
Loss after iteration 2000: 0.053784
Loss after iteration 3000: 0.052449
Loss after iteration 4000: 0.058075
Loss after iteration 5000: 0.053869
Loss after iteration 6000: 0.054719
Loss after iteration 7000: 0.054604
Loss after iteration 8000: 0.054592
Loss after iteration 9000: 0.054731
Loss after iteration 10000: 0.054387
Loss after iteration 11000: 0.053418
Loss after iteration 12000: 0.053223
Loss after iteration 13000: 0.053050
Loss after iteration 14000: 0.054486
Loss after iteration 15000: 0.053775
Loss after iteration 16000: 0.053579
Loss after iteration 17000: 0.054931
Loss after iteration 18000: 0.054375
Loss after iteration 19000: 0.053887
```

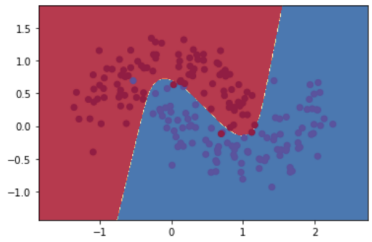


The start loss is greater if given a more hidden layer, but loss after iteration is smaller with a more hidden layer; also the decision boundaries are more distinctive. The loss after iteration is smaller than Sigmoid when increasing hidden layer to 10

Tanh:

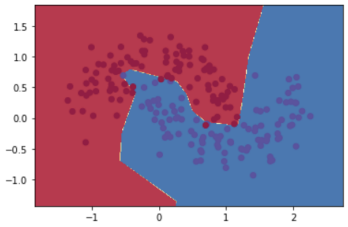
Hidden layer = 3

```
Loss after iteration 0: 0.432387
Loss after iteration 1000: 0.068947
Loss after iteration 2000: 0.069051
Loss after iteration 3000: 0.071218
Loss after iteration 4000: 0.071253
Loss after iteration 5000: 0.071278
Loss after iteration 6000: 0.071293
Loss after iteration 7000: 0.071303
Loss after iteration 8000: 0.071308
Loss after iteration 9000: 0.071312
Loss after iteration 10000: 0.071314
Loss after iteration 11000: 0.071315
Loss after iteration 12000: 0.071315
Loss after iteration 13000: 0.071316
Loss after iteration 14000: 0.071316
Loss after iteration 15000: 0.071316
Loss after iteration 16000: 0.071316
Loss after iteration 17000: 0.071316
Loss after iteration 18000: 0.071316
Loss after iteration 19000: 0.071316
```

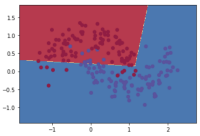


Hidden layer = 10

```
Loss after iteration 0: 1.248780
Loss after iteration 1000: 0.057022
Loss after iteration 2000: 0.053784
Loss after iteration 3000: 0.052449
Loss after iteration 4000: 0.058075
Loss after iteration 5000: 0.053869
Loss after iteration 6000: 0.054719
Loss after iteration 7000: 0.054604
Loss after iteration 8000: 0.054592
Loss after iteration 9000: 0.054731
Loss after iteration 10000: 0.054387
Loss after iteration 11000: 0.053418
Loss after iteration 12000: 0.053223
Loss after iteration 13000: 0.053050
Loss after iteration 14000: 0.054486
Loss after iteration 15000: 0.053775
Loss after iteration 16000: 0.053579
Loss after iteration 17000: 0.054931
Loss after iteration 18000: 0.054375
Loss after iteration 19000: 0.053887
```



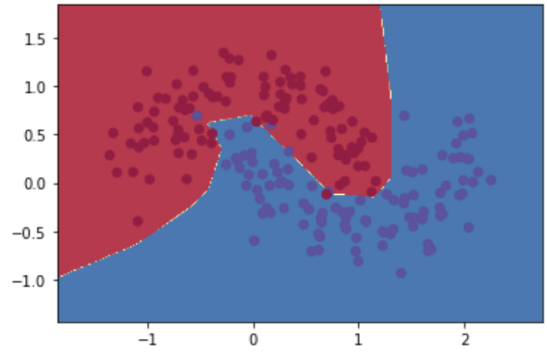
Same with sigmoid and relu, tanh loss starts greater if increase layer but becomes smaller and smaller than 3 hidden layers after iteration. Also the boundaries become more distinctive. However I noticed if I increase hidden layer to 5, the result shows:



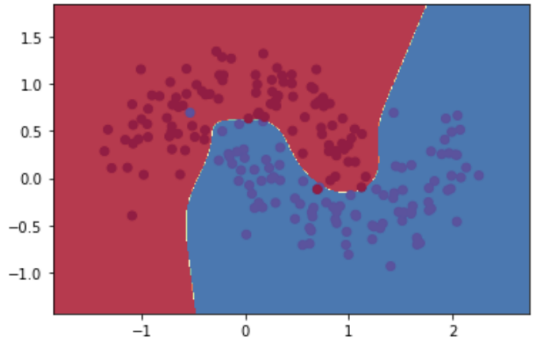
and the loss becomes a lot greater, around 0.225 even after max iteration.

f) Even more fun

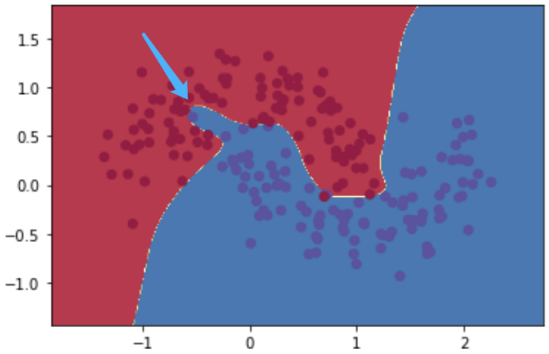
Relu, hidden layer =10, output = 20



Sigmoid, hidden layer = 10, output = 20

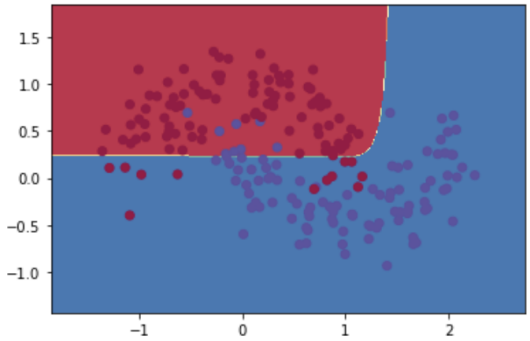


Tanh, hidden layer = 10, output = 20



Interesting here that Tanh's boundary correctly collected that one blue dot

Sigmoid hidden layer =2



Similar to previous results, too small of a hidden layer results in bad results.
To conclude: I believe that Relu have the best result

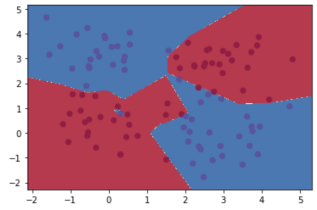
Data Set: Make_blobs

In n_layer_neural_network, the only part I changed is adding a addLayer func.

```
def addLayer(self, input_dim, output_dim, actFun_type=None):
    if actFun_type is None:
        self.layers.append(Layer(input_dim, output_dim))
    else:
        actFun = lambda o: self.actFun(o, actFun_type)
        diff_actFun = lambda o: self.diff_actFun(o, actFun_type)
        self.layers.append(Layer(input_dim, output_dim, actFun, diff_actFun))
```

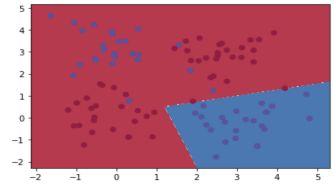
So it is easier to add layers.

```
Loss after iteration 1000: 0.155676
Loss after iteration 2000: 0.148611
Loss after iteration 3000: 0.129126
Loss after iteration 4000: 0.108939
Loss after iteration 5000: 0.086914
Loss after iteration 6000: 0.118445
Loss after iteration 7000: 0.112961
Loss after iteration 8000: 0.077659
Loss after iteration 9000: 0.052670
Loss after iteration 10000: 0.058778
Loss after iteration 11000: 0.396121
Loss after iteration 12000: 0.049931
Loss after iteration 13000: 0.052580
Loss after iteration 14000: 0.102994
Loss after iteration 15000: 0.048979
Loss after iteration 16000: 0.050148
Loss after iteration 17000: 0.054091
Loss after iteration 18000: 0.049652
Loss after iteration 19000: 0.053663
```



I tried using the Make_blobs data set with 30 hidden layers.

```
Loss after iteration 1000: 0.390815
Loss after iteration 2000: 0.390411
Loss after iteration 3000: 0.390319
Loss after iteration 4000: 0.390454
Loss after iteration 5000: 0.390324
Loss after iteration 6000: 0.390358
Loss after iteration 7000: 0.390195
Loss after iteration 8000: 0.390375
Loss after iteration 9000: 0.390297
Loss after iteration 10000: 0.390235
Loss after iteration 11000: 0.390194
Loss after iteration 12000: 0.390410
Loss after iteration 13000: 0.390291
Loss after iteration 14000: 0.390253
Loss after iteration 15000: 0.390253
Loss after iteration 16000: 0.390252
Loss after iteration 17000: 0.390166
Loss after iteration 18000: 0.390165
Loss after iteration 19000: 0.390172
```



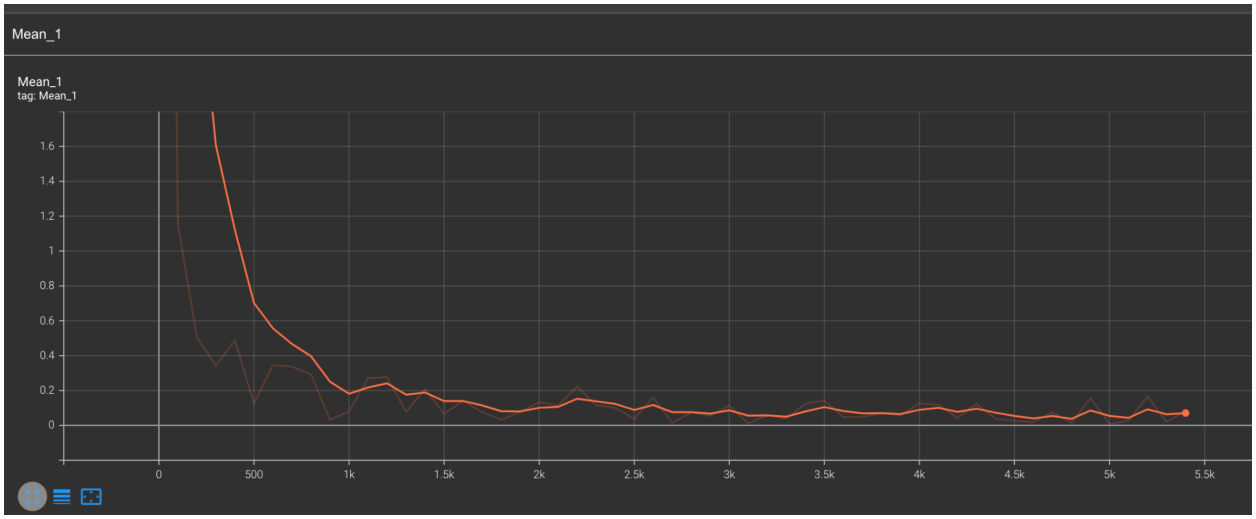
2 hidden layers
Similar to previous results, 2 hidden layers is not enough.

2. Training a Simple Deep Convolutional Network on MNIST

Training Result:

step 0, training accuracy 0.1
step 100, training accuracy 0.84
step 200, training accuracy 0.98
step 300, training accuracy 0.96
step 400, training accuracy 0.94
step 500, training accuracy 0.96
step 600, training accuracy 0.86
step 700, training accuracy 0.94
step 800, training accuracy 0.94
step 900, training accuracy 0.96
step 1000, training accuracy 0.96
step 1100, training accuracy 0.96
step 1200, training accuracy 0.94
step 1300, training accuracy 0.98
step 1400, training accuracy 1
step 1500, training accuracy 0.98
step 1600, training accuracy 0.94
step 1700, training accuracy 0.98
step 1800, training accuracy 0.94
step 1900, training accuracy 0.92
step 2000, training accuracy 1
step 2100, training accuracy 0.96
step 2200, training accuracy 0.98
step 2300, training accuracy 0.98
step 2400, training accuracy 0.98
step 2500, training accuracy 1
step 2600, training accuracy 1
step 2700, training accuracy 0.98
step 2800, training accuracy 0.98
step 2900, training accuracy 0.96
step 3000, training accuracy 0.98
step 3100, training accuracy 0.98
step 3200, training accuracy 0.98
step 3300, training accuracy 0.98
step 3400, training accuracy 0.96
step 3500, training accuracy 1
step 3600, training accuracy 0.98
step 3700, training accuracy 1
step 3800, training accuracy 1
step 3900, training accuracy 1
step 4000, training accuracy 0.98
step 4100, training accuracy 0.98
step 4200, training accuracy 0.98
step 4300, training accuracy 0.98
step 4400, training accuracy 1
step 4500, training accuracy 0.98
step 4600, training accuracy 1
step 4700, training accuracy 0.98
step 4800, training accuracy 0.98
step 4900, training accuracy 1
step 5000, training accuracy 0.98
step 5100, training accuracy 0.98
step 5200, training accuracy 1
step 5300, training accuracy 0.98
step 5400, training accuracy 1
test accuracy 0.9874
The training takes 246.051883 second to finish

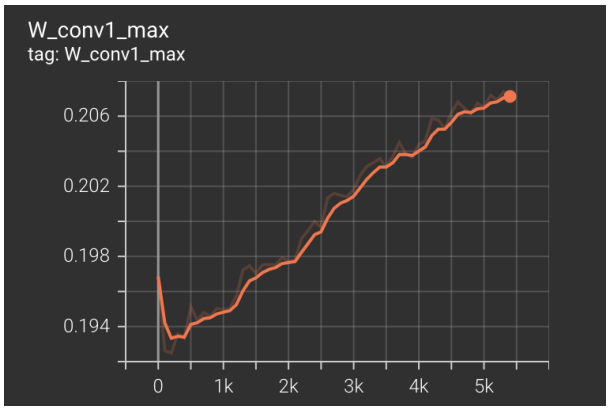
Visualize Training（MNIST):

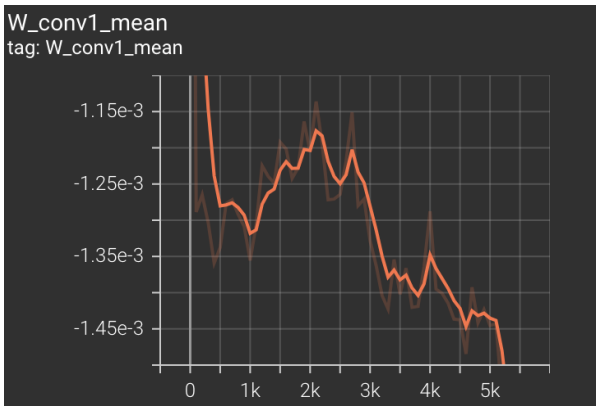


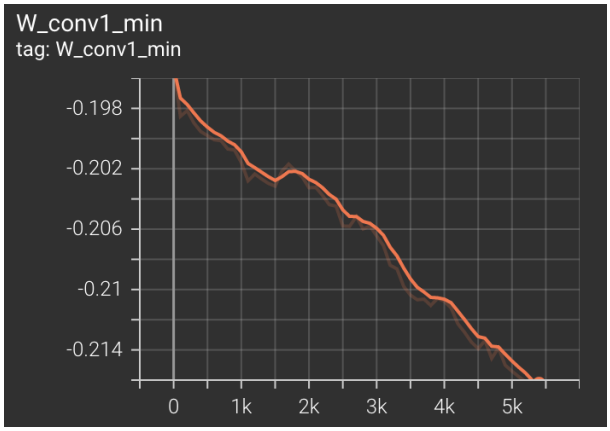
b) More on Visualizing Your Training

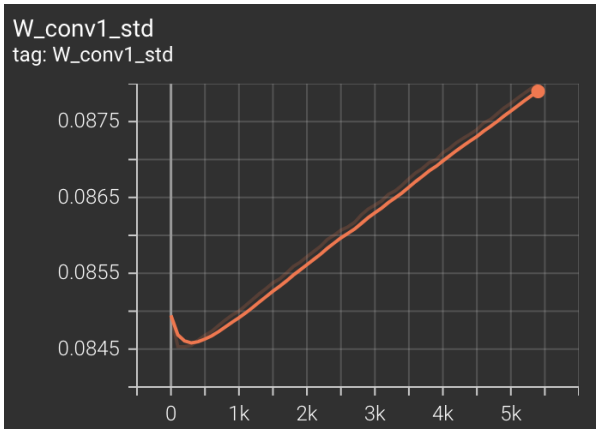
Statistic Monitor After (Optimization = Adam, Activation = Relu)

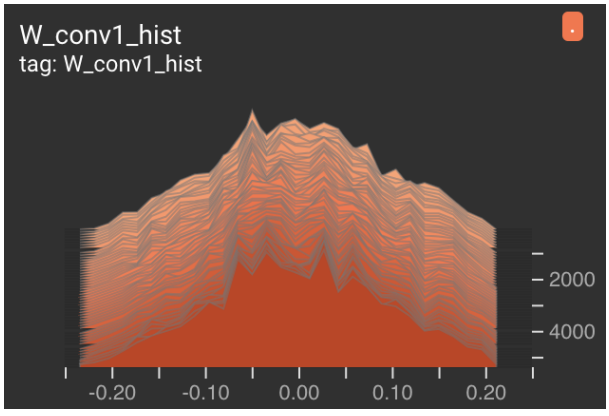
Convolution layer 1 (Weight Max, Min, Mean, STD, Hist):



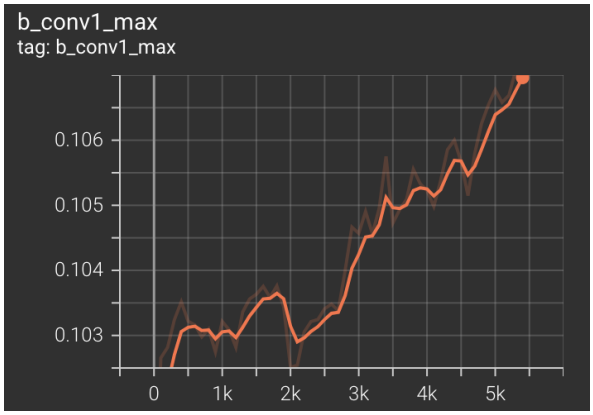


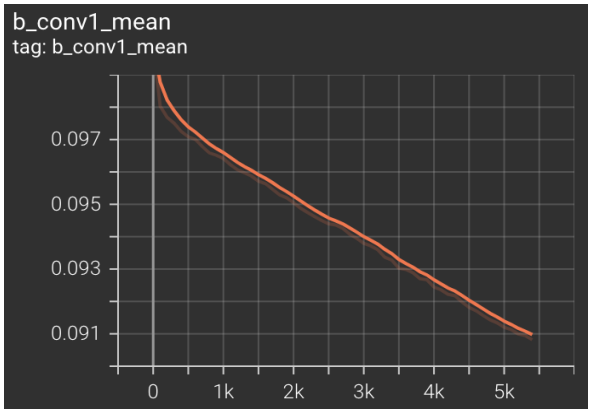


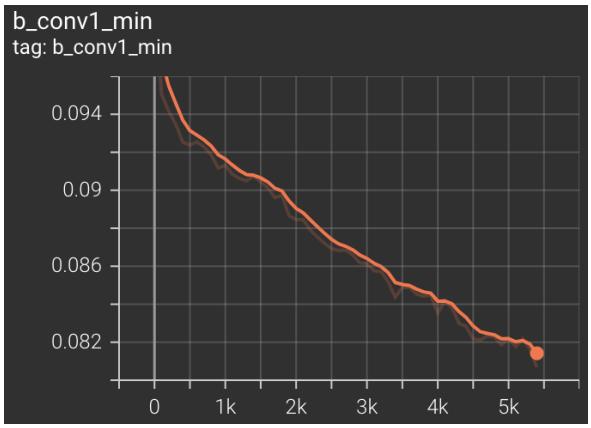


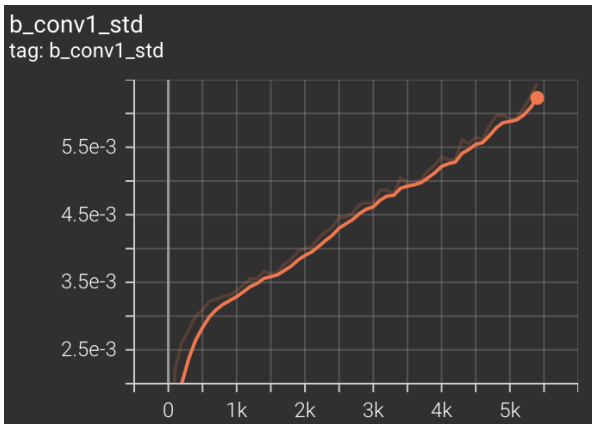


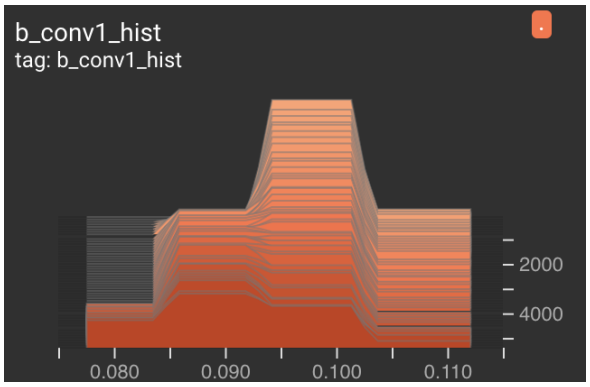
Convolution layer 1 (Bias Max, Min, Mean, STD, Hist)



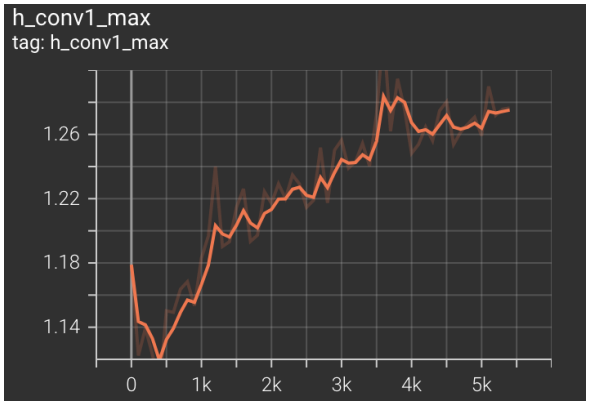


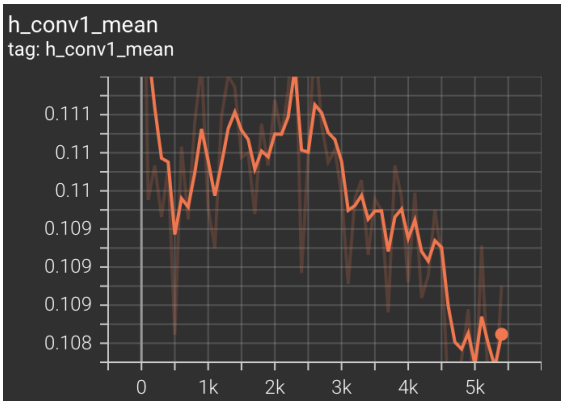


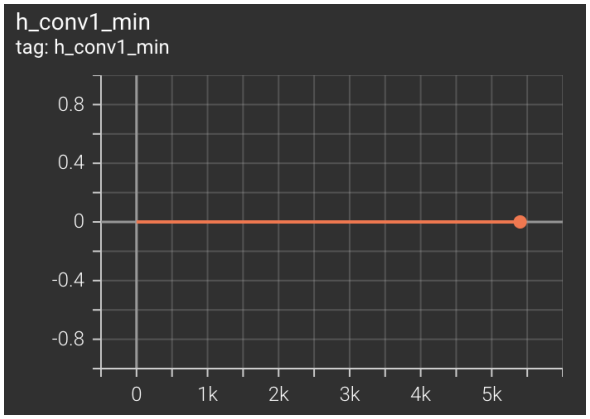


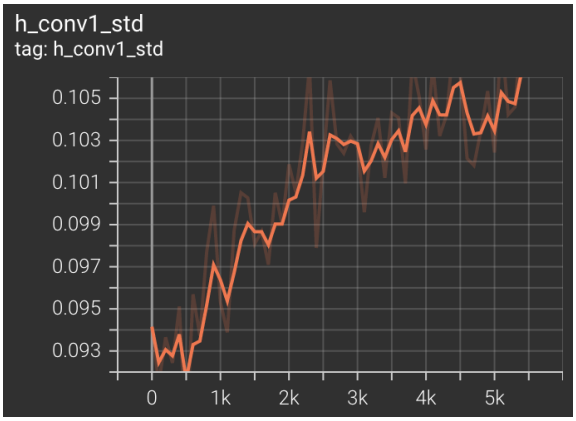


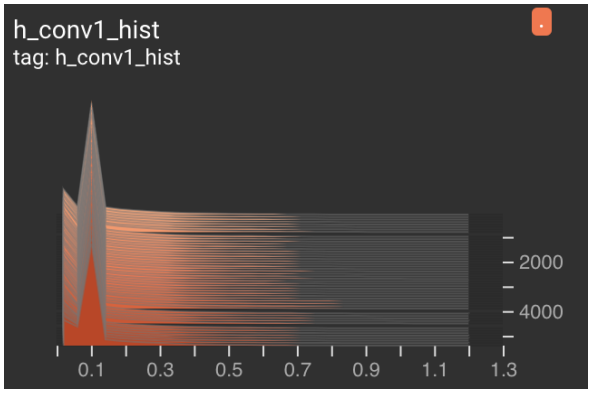
Convolution layer 1 (Relu Max, Min, Mean, STD, Hist)



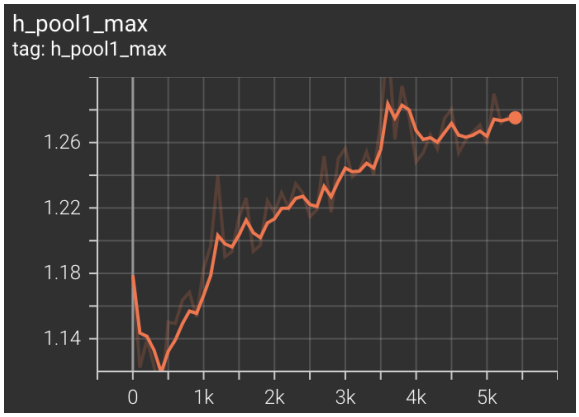


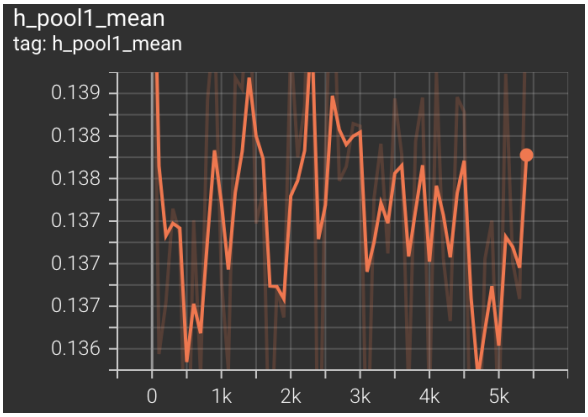


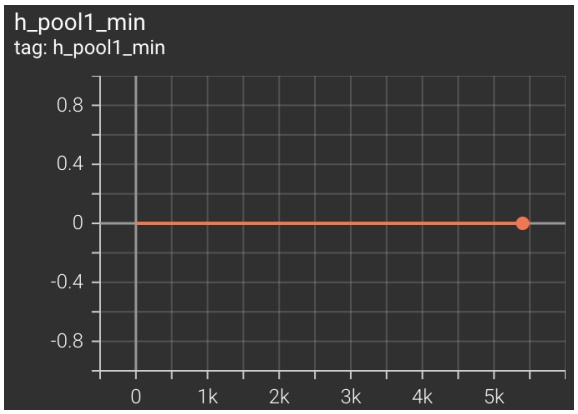


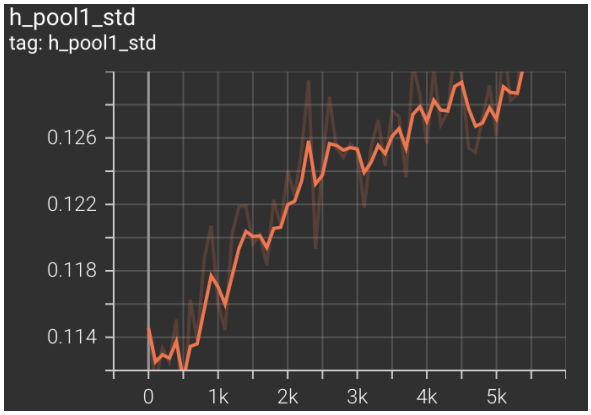


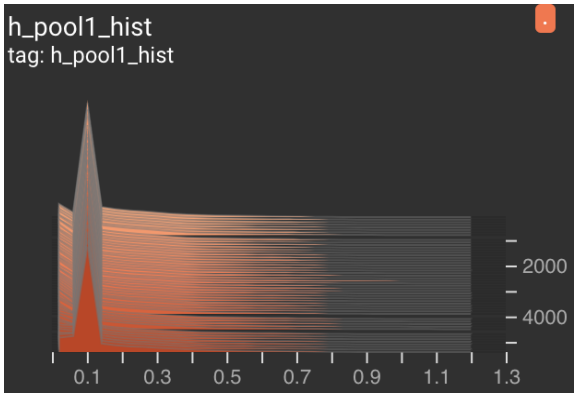
Convolutional layer 1(Conv2D Max, Min, Mean, STD, Hist)



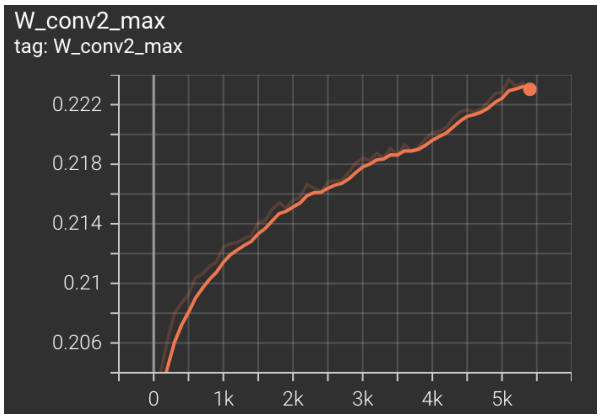


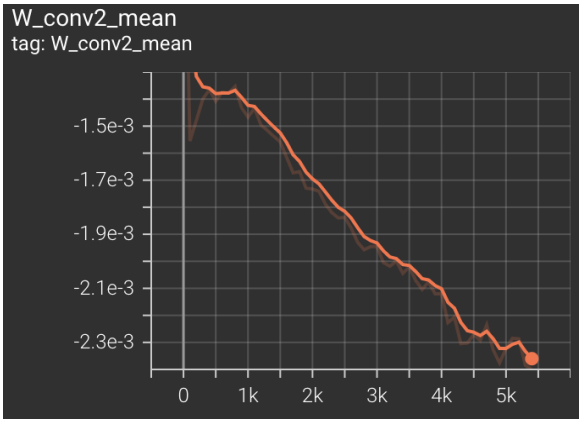


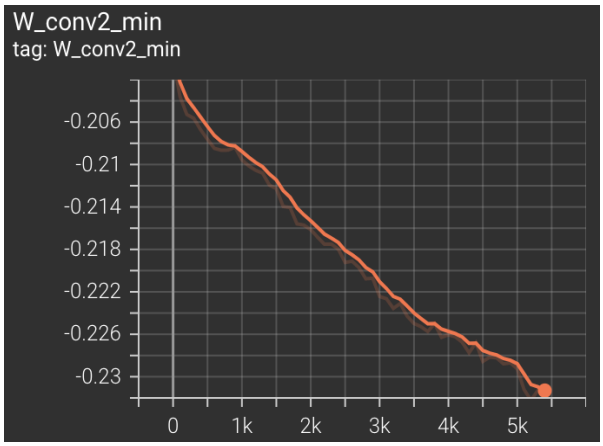


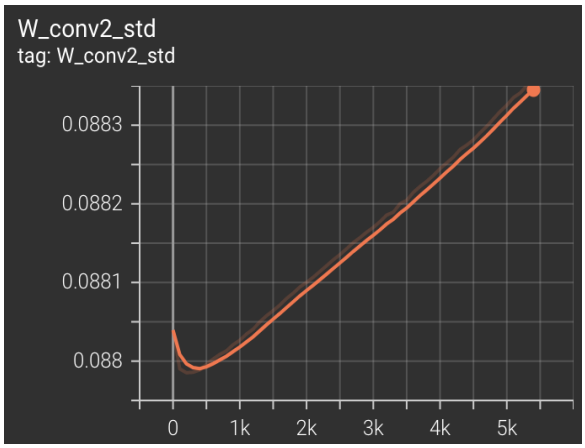


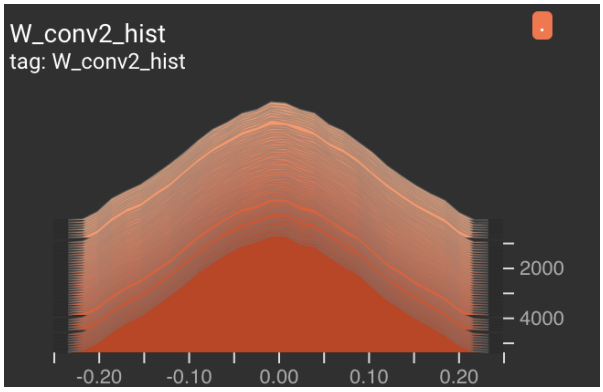
Convolution layer 2 (Weight Max, Min, Mean, STD, Hist)



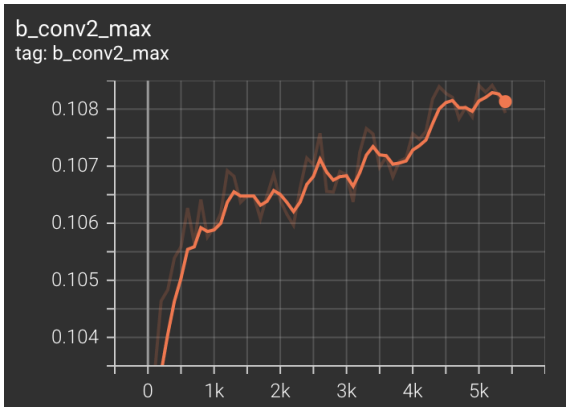


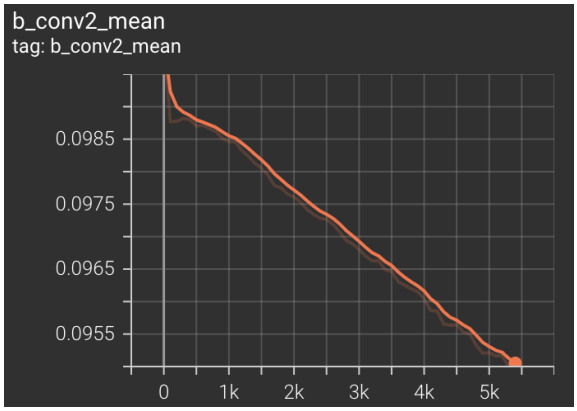


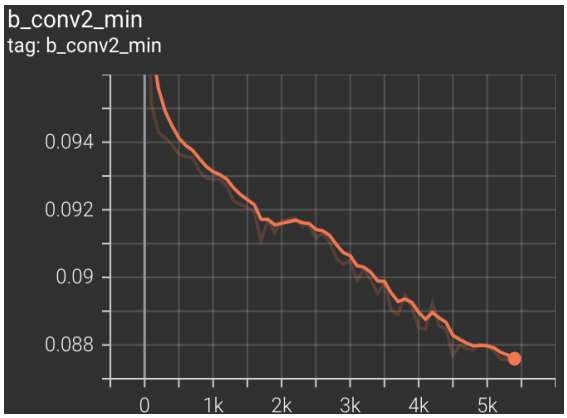


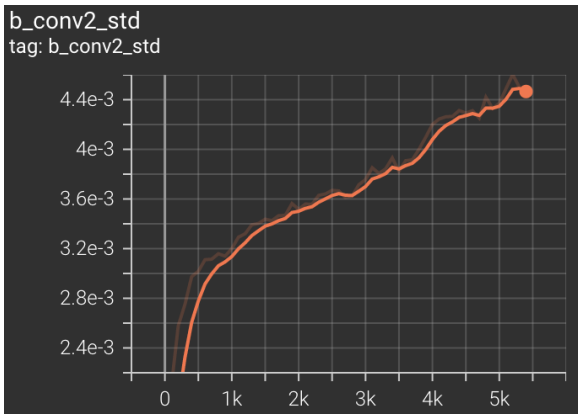


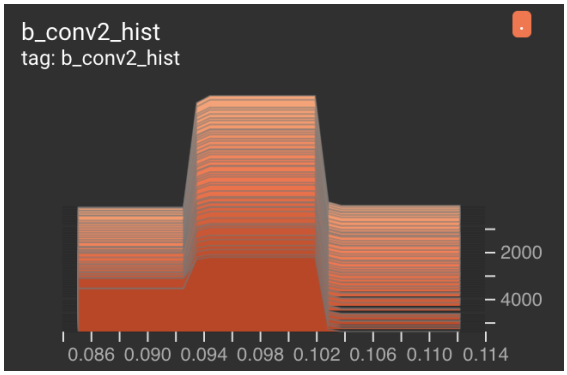
Convolution Layer 2 (Bias Max, Min, Mean, STD, Hist)



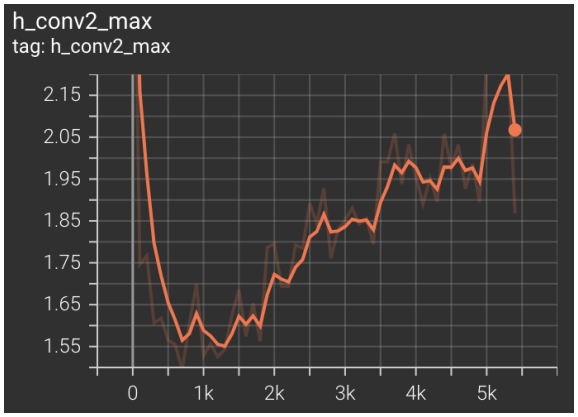


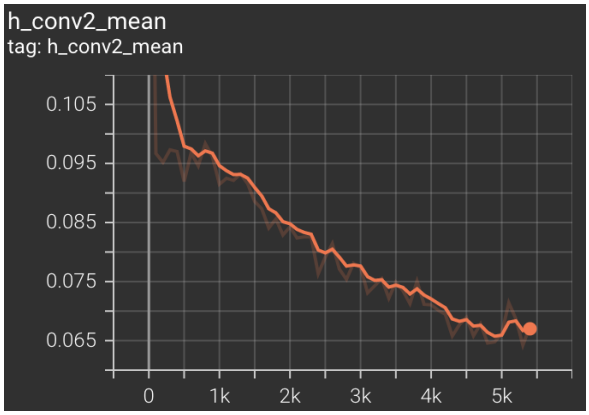


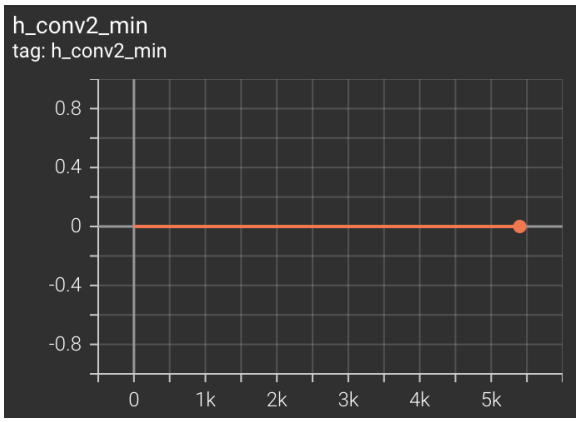


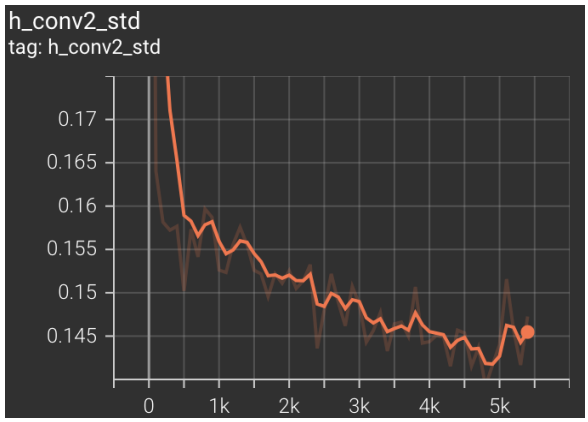


Convolution layer 2(Relu Max, Min, Mean, STD)

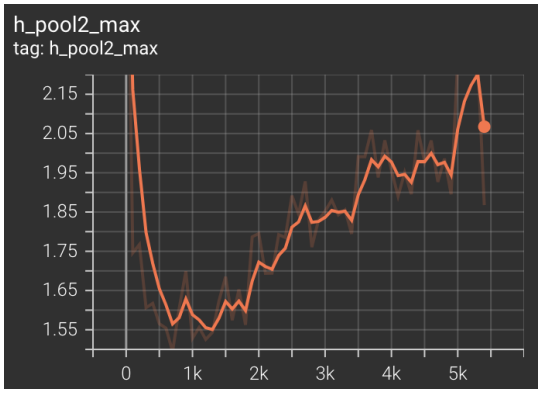


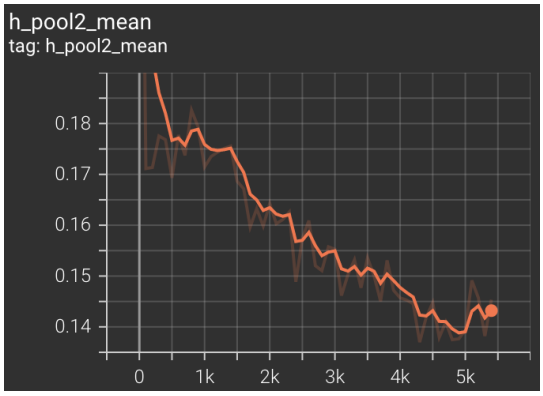


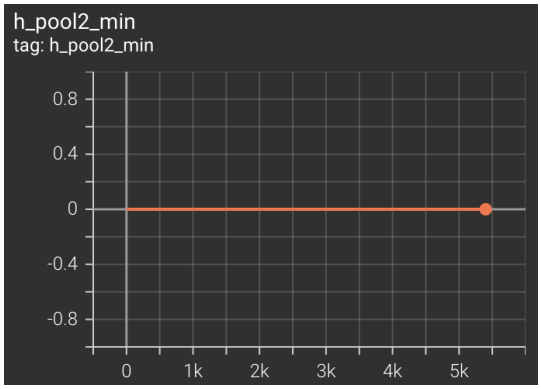


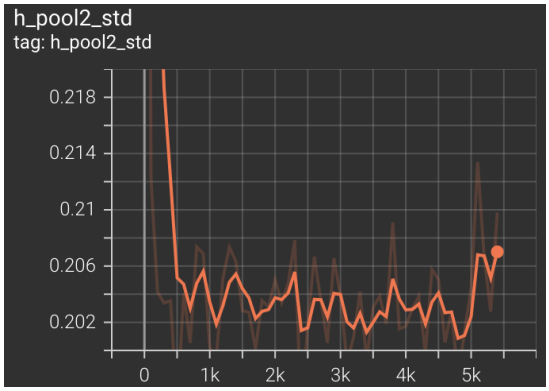


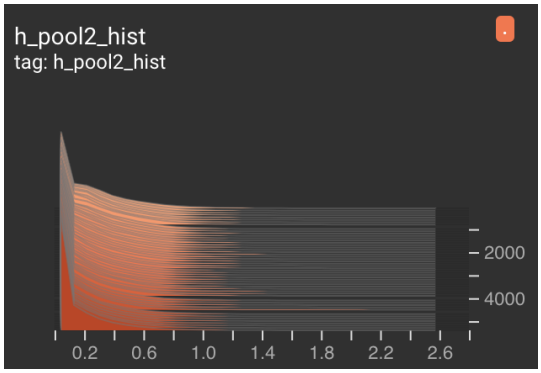
Convolution layer 2 (Conv2D Max, Min, Mean, STD, Hist)











C) Changing Optimization and Activation function (Time for more Fun)

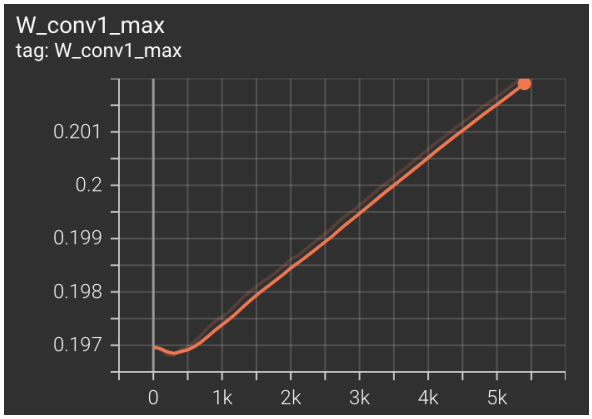
- I changed my activation function to SIGMOID in Conv layer 1 and Conv layer2, and changed my optimization function to AdagradOptimizer. The result shows a significantly lower accuracy with using Relu and adam.

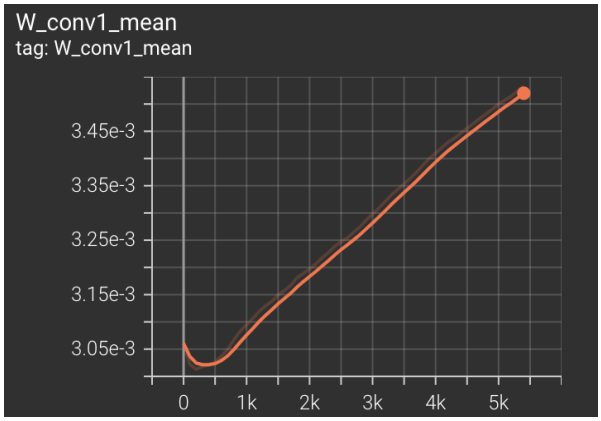
Result:

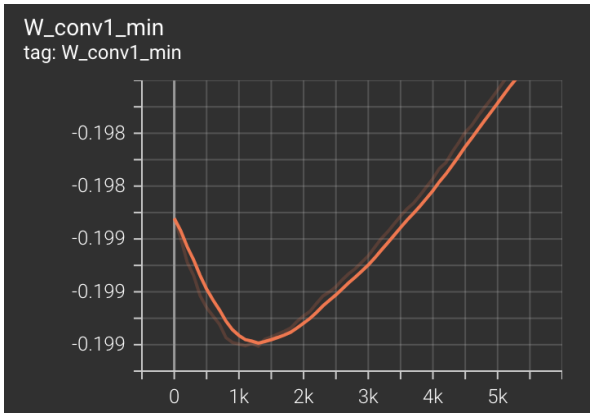
step 0, training accuracy 0.1
step 100, training accuracy 0.04
step 200, training accuracy 0.12
step 300, training accuracy 0.08
step 400, training accuracy 0.04
step 500, training accuracy 0.12
step 600, training accuracy 0.12
step 700, training accuracy 0.16
step 800, training accuracy 0.16
step 900, training accuracy 0.1
step 1000, training accuracy 0.08
step 1100, training accuracy 0.18
step 1200, training accuracy 0.16
step 1300, training accuracy 0.28
step 1400, training accuracy 0.18
step 1500, training accuracy 0.24
step 1600, training accuracy 0.28
step 1700, training accuracy 0.16
step 1800, training accuracy 0.26
step 1900, training accuracy 0.24
step 2000, training accuracy 0.3
step 2100, training accuracy 0.24
step 2200, training accuracy 0.34
step 2300, training accuracy 0.26
step 2400, training accuracy 0.34
step 2500, training accuracy 0.3
step 2600, training accuracy 0.36
step 2700, training accuracy 0.4
step 2800, training accuracy 0.46
step 2900, training accuracy 0.36
step 3000, training accuracy 0.3
step 3100, training accuracy 0.54
step 3200, training accuracy 0.48
step 3300, training accuracy 0.44
step 3400, training accuracy 0.38
step 3500, training accuracy 0.4
step 3600, training accuracy 0.5
step 3700, training accuracy 0.42
step 3800, training accuracy 0.48
step 3900, training accuracy 0.44
step 4000, training accuracy 0.46
step 4100, training accuracy 0.54
step 4200, training accuracy 0.52
step 4300, training accuracy 0.44
step 4400, training accuracy 0.66
step 4500, training accuracy 0.32
step 4600, training accuracy 0.54
step 4700, training accuracy 0.36
step 4800, training accuracy 0.48
step 4900, training accuracy 0.42
step 5000, training accuracy 0.66
step 5100, training accuracy 0.64
step 5200, training accuracy 0.62
step 5300, training accuracy 0.56
step 5400, training accuracy 0.6
test accuracy 0.5456
The training takes 318.305866 second to finish

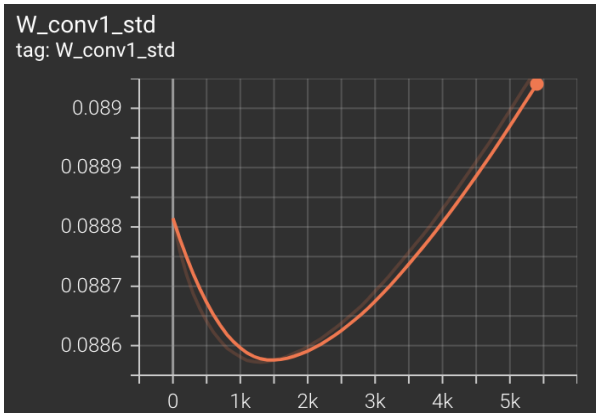
Statistic Monitor After Changing:
Optimization = Adagrad, Activation = Sigmoid

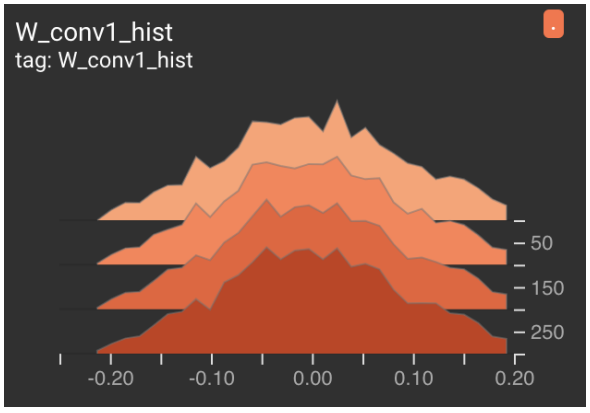
Convolution layer 1 (Weight Max, Min, Mean, STD, Hist)



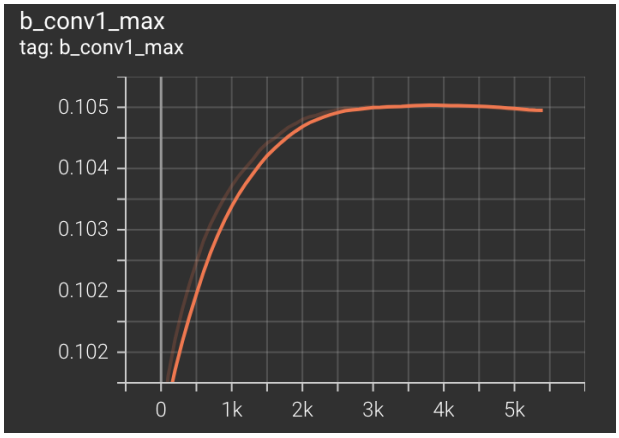


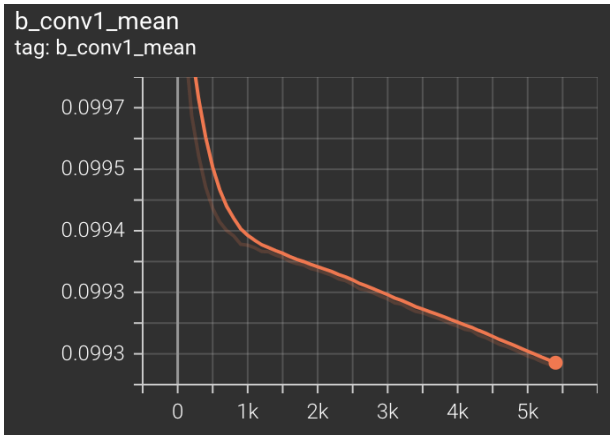


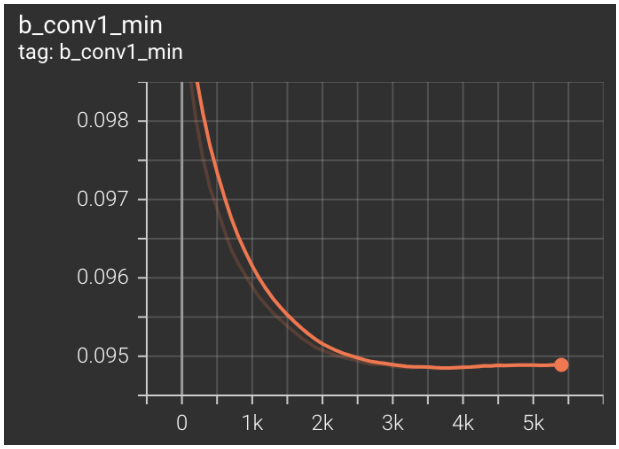


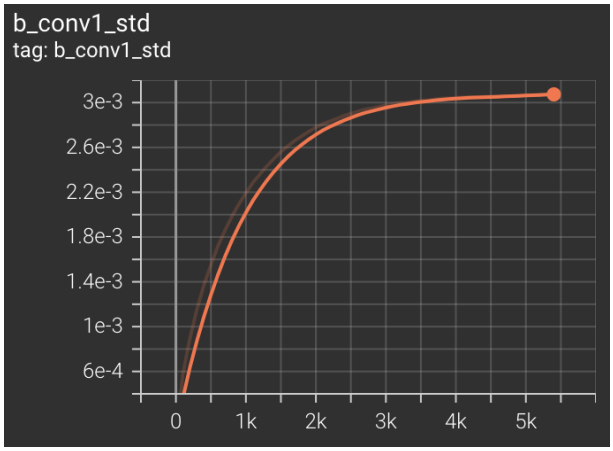


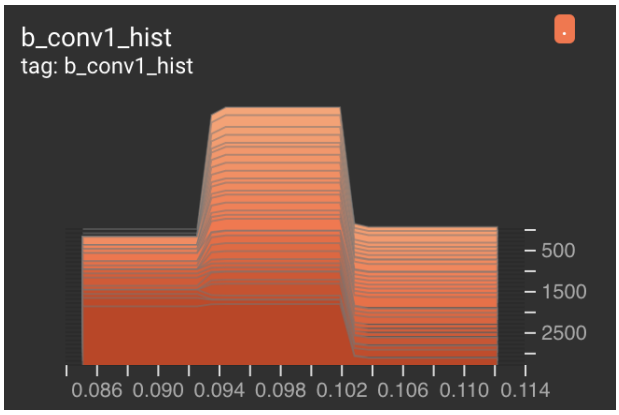
Convolution layer 1(Bias Max, Min, Mean, STD, Hist)



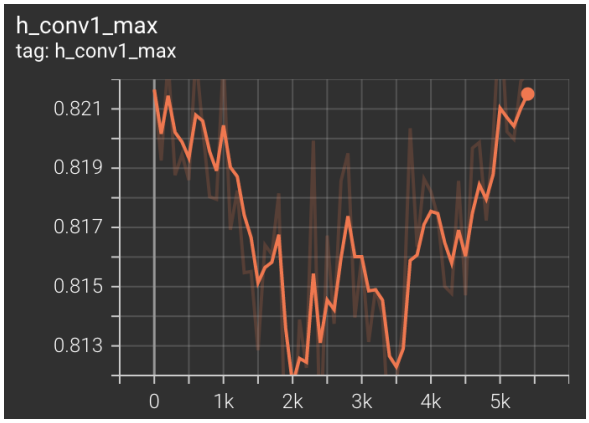


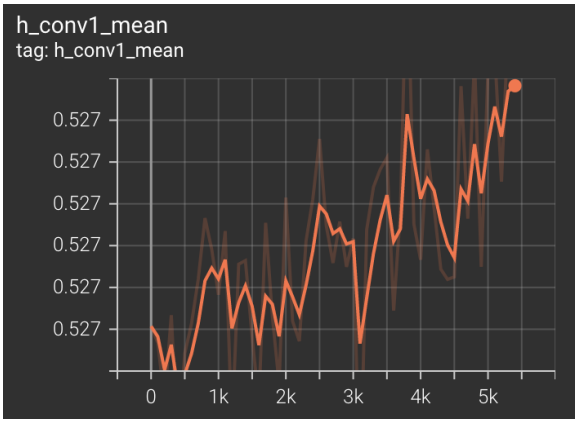


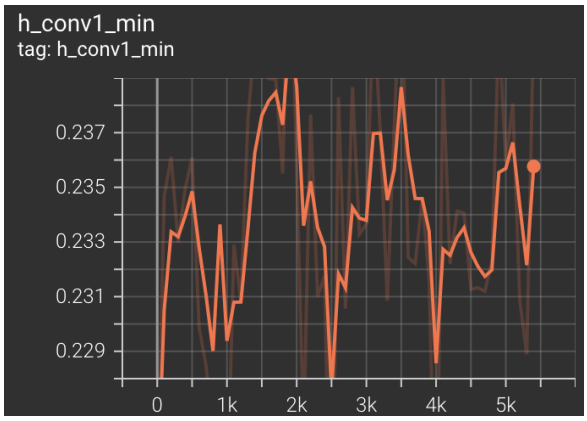


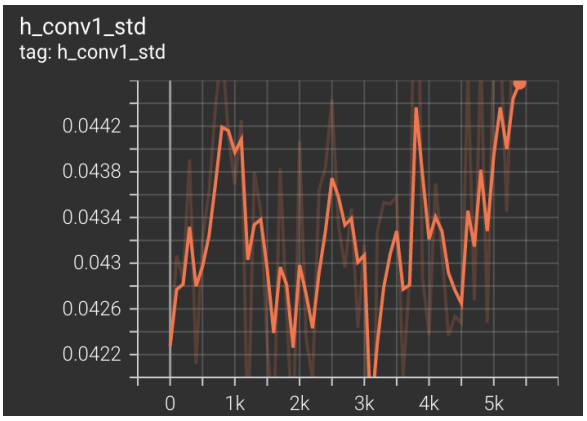


Convolution layer 1(Relu Max, Min, Mean, STD)

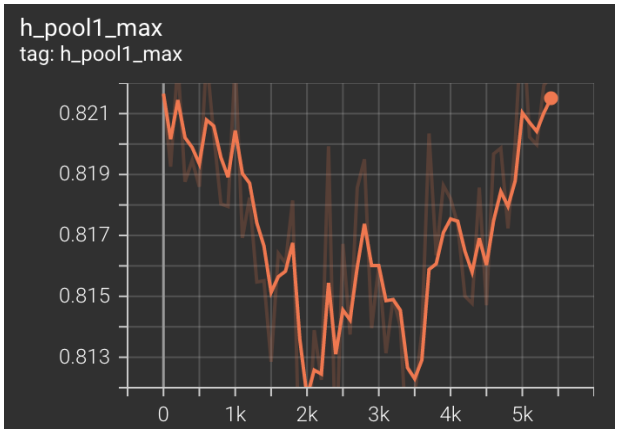


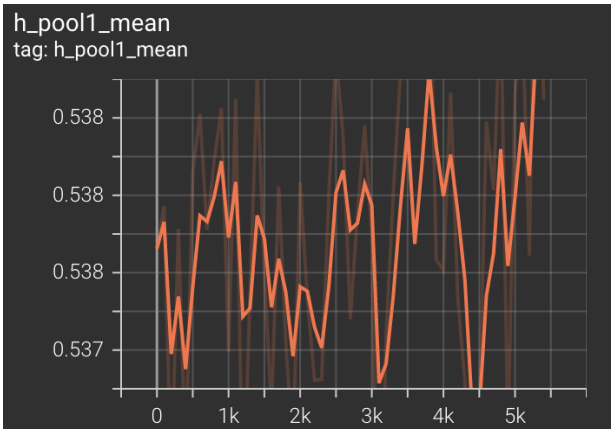


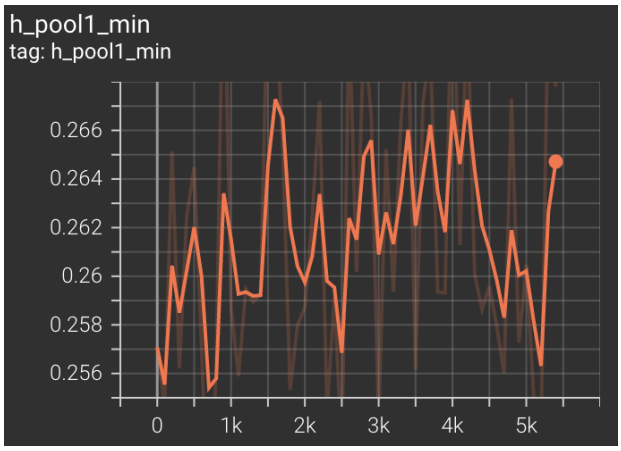


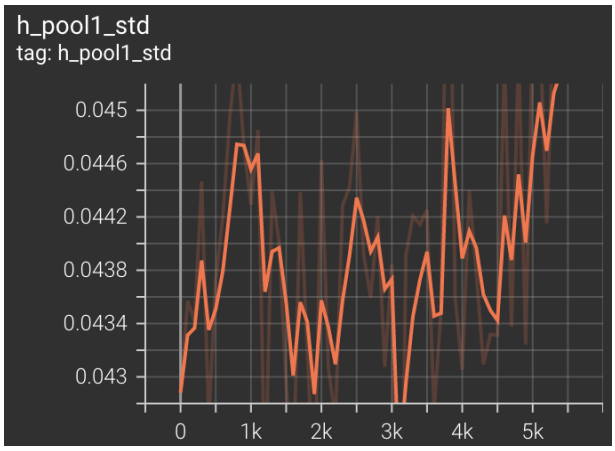


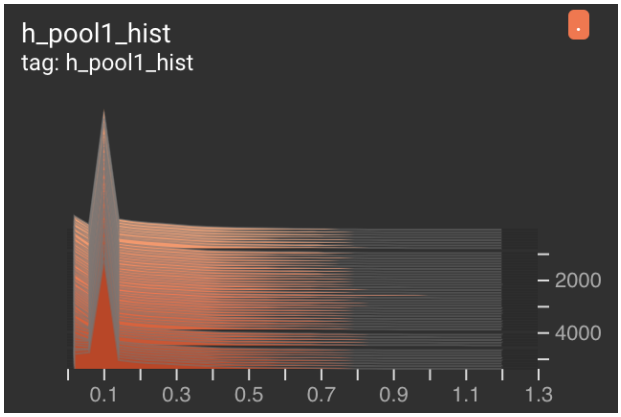
Convolution layer 1(Conv2D Max, Min, Mean, STD, Hist)



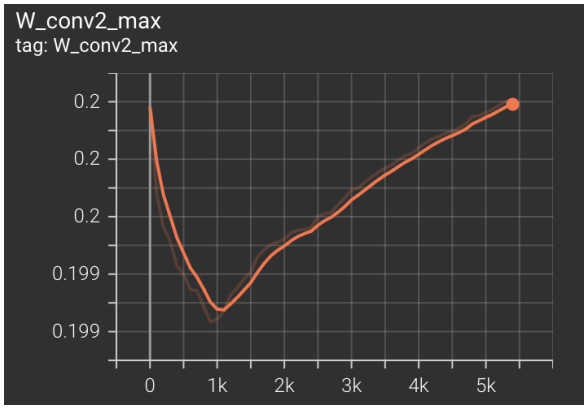


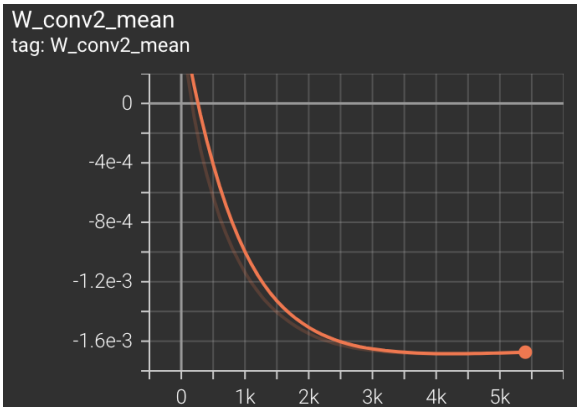


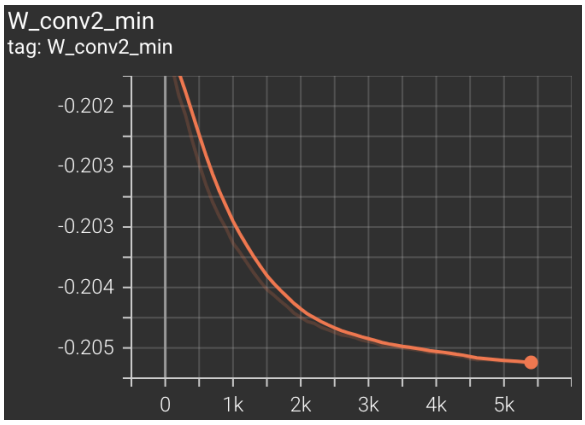


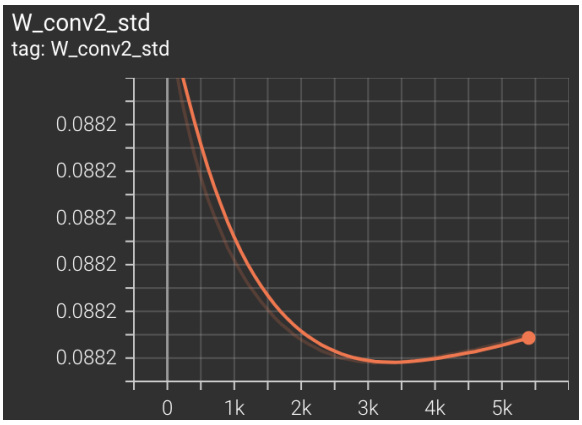


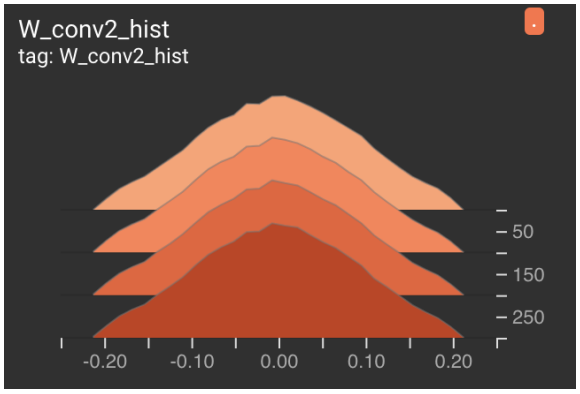
Convolution layer 2(Weight Max, Min, Mean, STD, Hist)



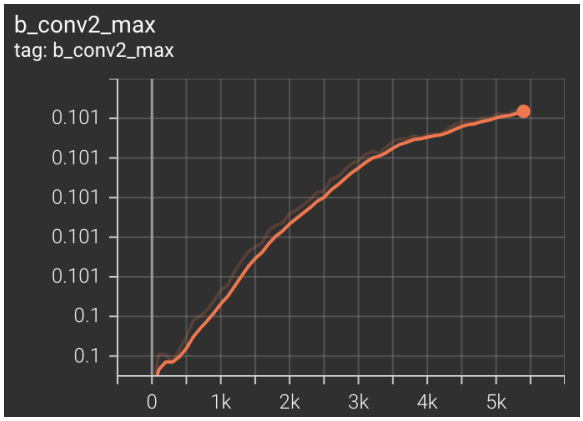


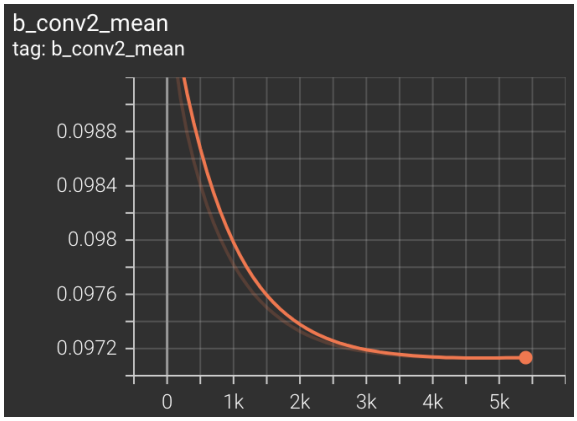


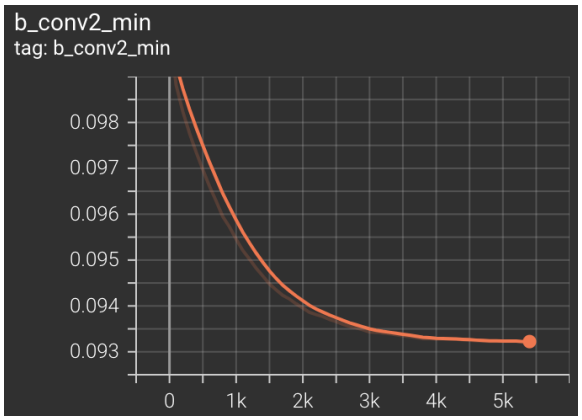


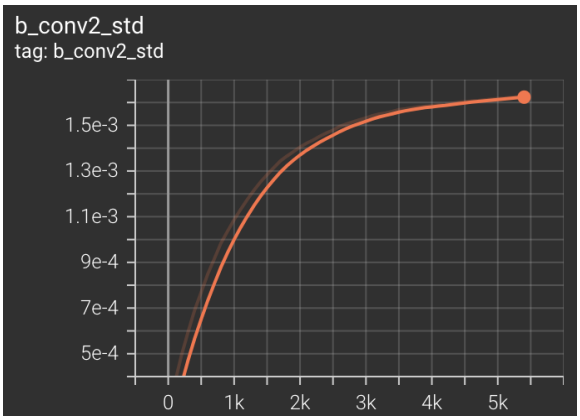


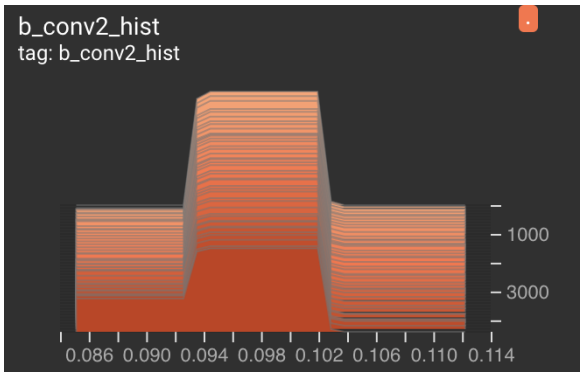
Convolution layer 2(Bias Max, Min, Mean, STD, Hist)



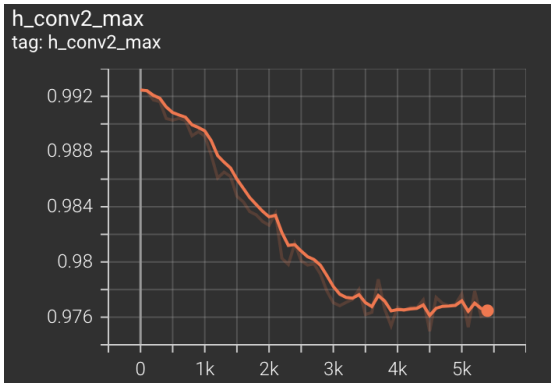


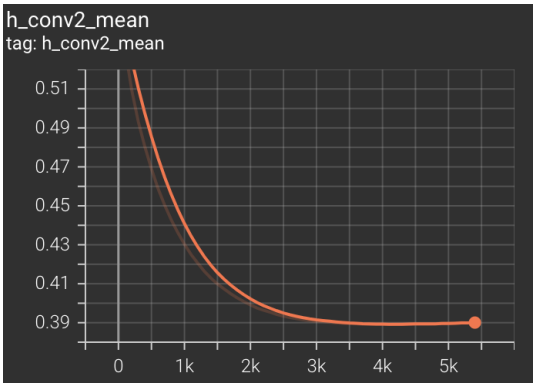


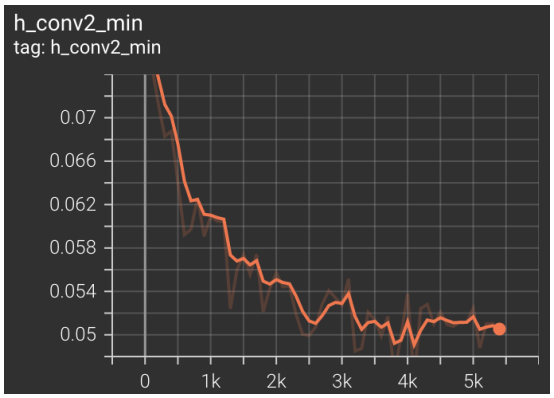


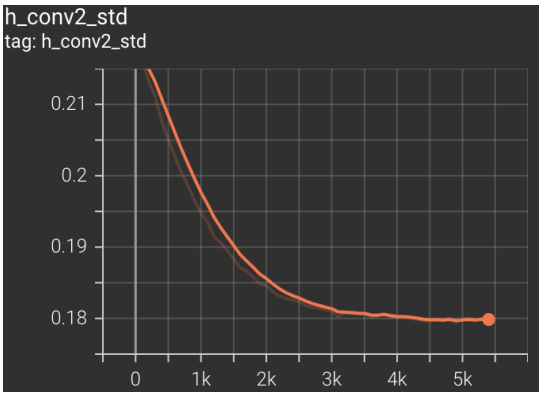


Convolution layer 2(Relu Max, Min, Mean, STD)

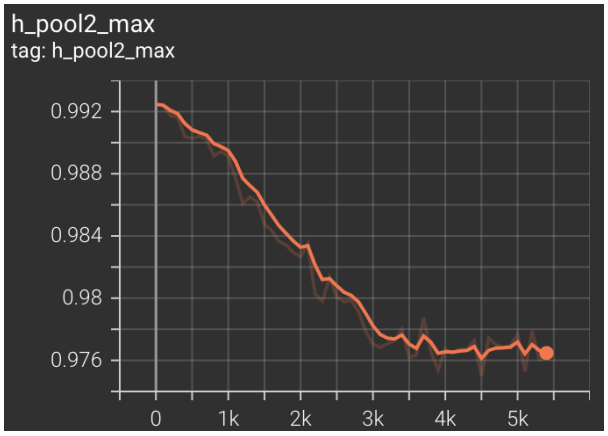


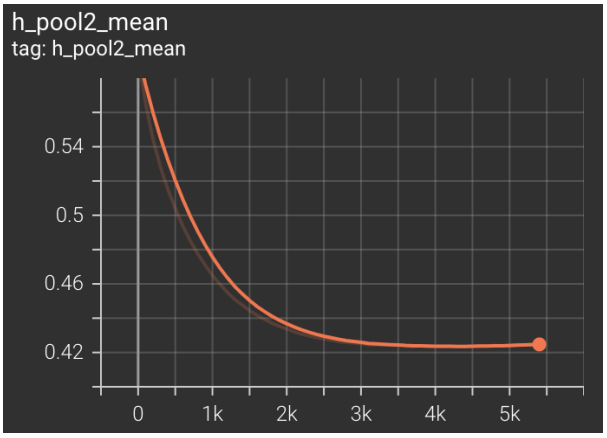


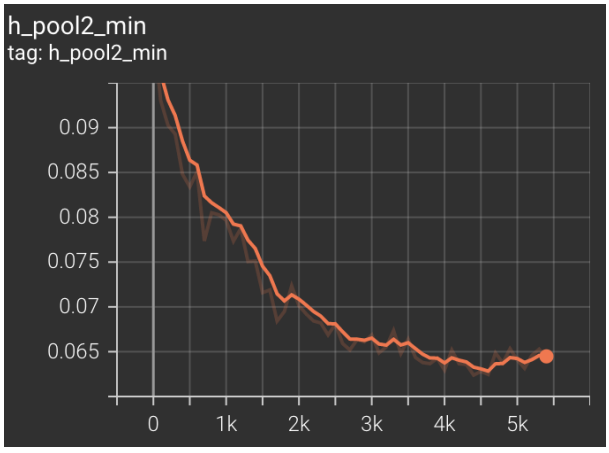


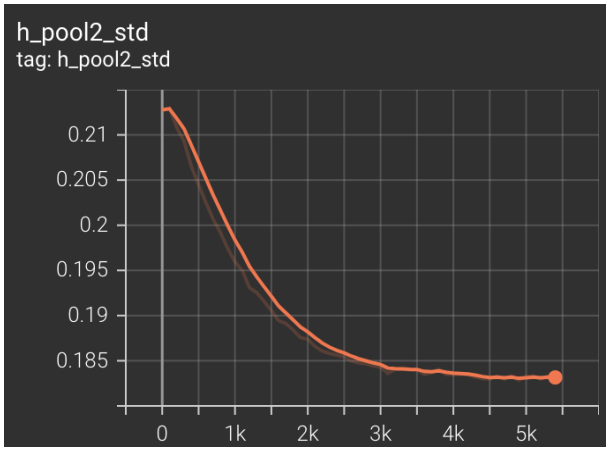


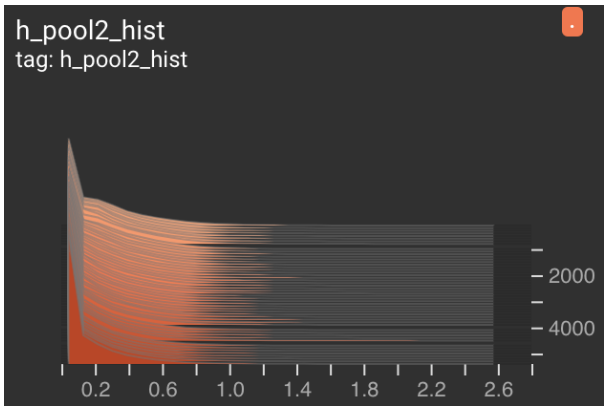
Convolution layer 2(Conv2D Max, Min, Mean, STD, Hist)





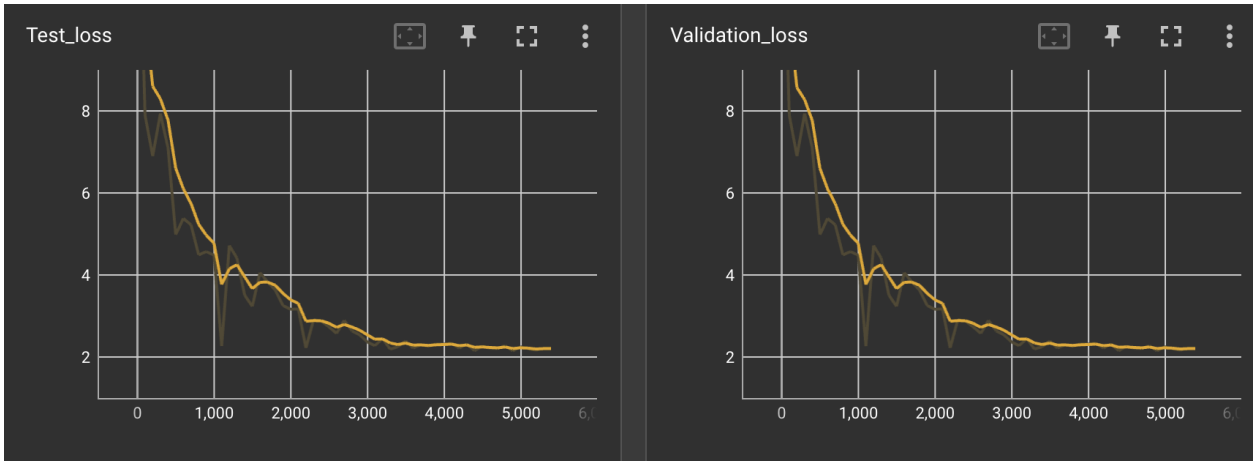




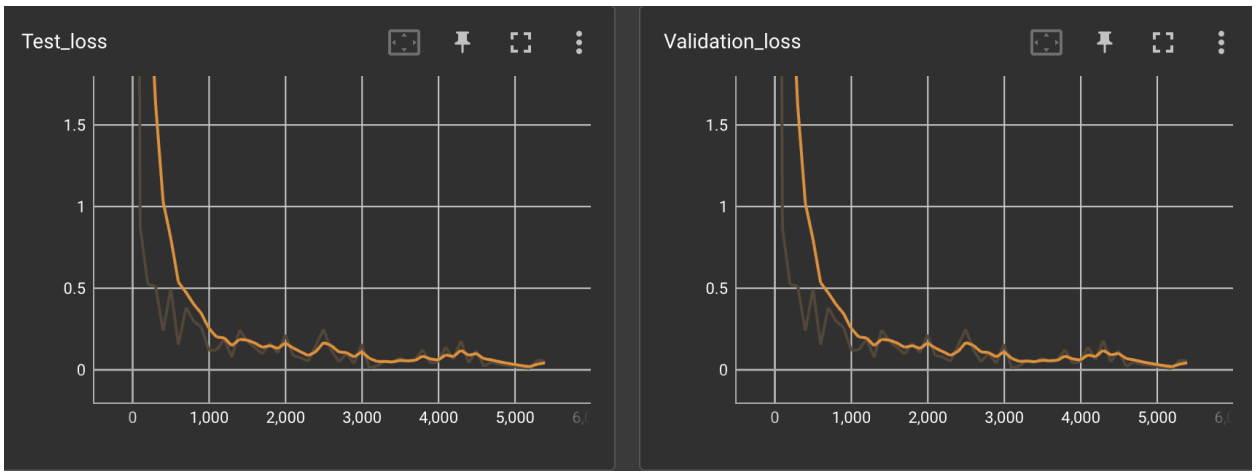


Validation & Test Loss:

Activation = Sigmoid
Optimization = Adagrad



Activation = Relu
Optimization = Adam



Observation comment:

We can see directly that losses are dropping in both combinations. However, with Relu activation function and Adam optimization, the test loss drops faster than using Sigmoid activation function and adagrad optimization, as shown in the training result section.
Similarly, for validation loss, we can see the loss drops faster by around 1000 iterations compared to using sigmoid & adagrad.

Reference:

262588213843476. “Tensorflow Tutorial ‘Deep Mnist for Experts.’” *Gist*, https://gist.github.com/saitodev/c4c7a8c83f5aa4a00e93084dd3f848c5.

Balawejder, Maciej. “Neural Network from Scratch Using Numpy.” *Medium*, Analytics Vidhya, 6 Mar. 2022, https://medium.com/analytics-vidhya/neural-network-mnist-classifier-from-scratch-using-numpy-library-94bbcfed7eae.

Person. “Implementing a Neural Network from Scratch in Python.” *Denny's Blog*, 14 Sept. 2022, https://dennybritz.com/posts/wildml/implementing-a-neural-network-from-scratch/.