Assingment 1 : Deep Learning for CV - Building a CNN

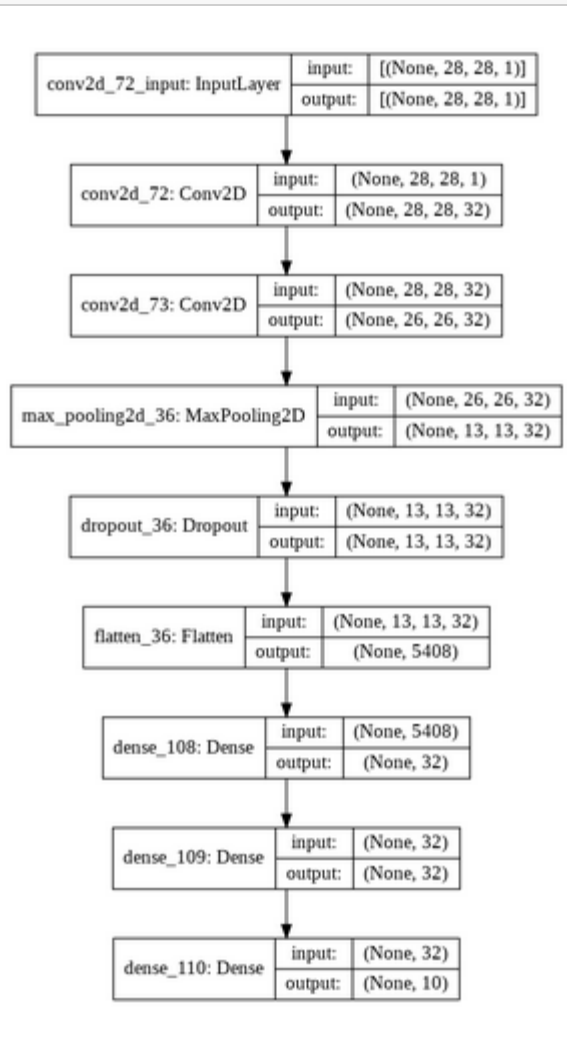
Index Number : 17000475

Dataset - MINIST

Aim - Classify digit images from 0 to 9

1. Architecture of the CNN Model

```
In [ ]: from keras.utils.vis_utils import plot_model
        plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```



2. Source Code

Import Modules

```
In [ ]: from __future__ import absolute_import, division, print_function, unicode_literals
        import tensorflow as tf
        import matplotlib.pyplot as plt
        import numpy as np
        import datetime
        from tensorflow import keras
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        from keras.utils import np_utils
        from tensorboard.plugins.hparams import api as hp
```

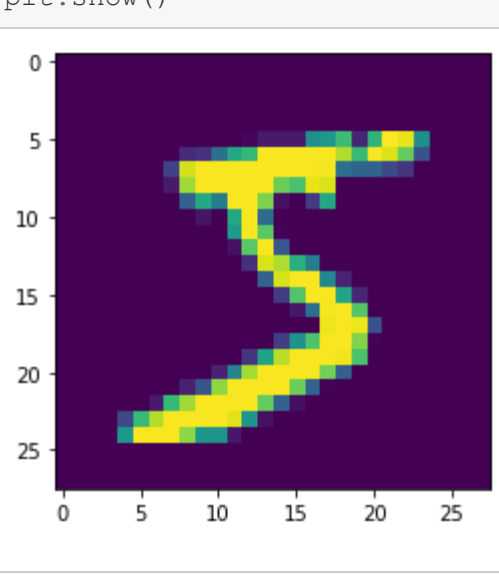
Dataset Preparation and Inspect

```
In [ ]: mnist = keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
In [ ]: print("train image shape:",x_train.shape)
        print("train label shape:",y_train.shape)
        print("test image shape:",x_test.shape)
        print("test label shape:",y_test.shape)
```

```
train image shape: (60000, 28, 28)
train label shape: (60000,)
test image shape: (10000, 28, 28)
test label shape: (10000,)
```

```
In [ ]: plt.figure()
        plt.imshow(x_train[0])
        # plt.colorbar()
        # plt.grid(False)
        plt.show()
```




```
In [ ]: fig, axes = plt.subplots(3, 6, figsize=(8,8))
        fig.subplots_adjust(hspace=0.5, wspace=0.5)

        for i, ax in enumerate(axes.flat):

          ax.imshow(x_train[i])
          ax.set_xticks([])
          ax.set_yticks([])

        plt.show()
```

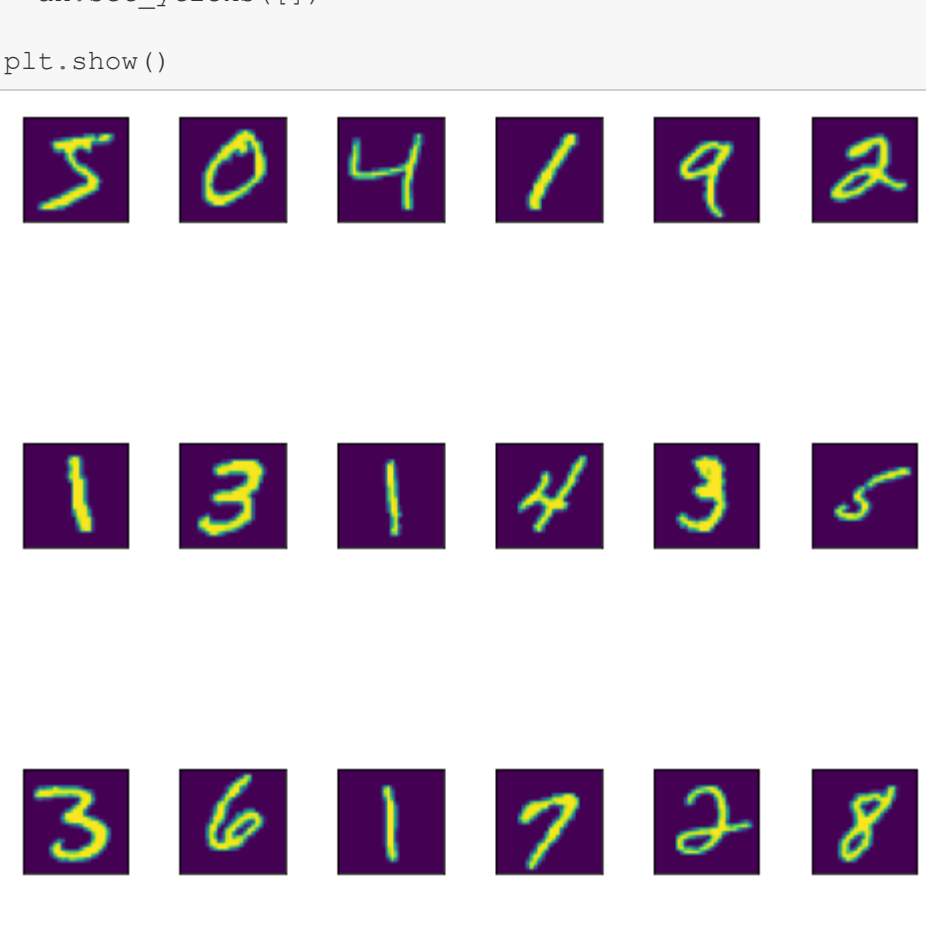


Image pre-processing

```
In [ ]: # convert MINIST Dataset 3d dimention to 4d because keras required 4d input for convolution operation
        # MINIST image depth will include as new dimention. Since it has grayscale depth is 1

        x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
        x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
```

```
In [ ]: #Normalization - Rescaling pixels on scale 0 to 1
        x_train = x_train / 255.0
        x_test = x_test / 255.0
```

Tensorboard Setup

```
In [ ]: #initializing Tensorboard extension
        %load_ext tensorboard
        #creating a logs directory
        #log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
```

```
In [ ]: # Clear any logs from previous runs
        !rm -rf ./logs/
```

```
In [ ]: x_train.shape[1:]
Out[ ]: (28, 28, 1)
```

Hyperparameter Selection

```
In [ ]: HP_NUM_UNITS_1 = hp.HParam('num_units_1', hp.Discrete([16,32,64]))
        HP_NUM_UNITS_2 = hp.HParam('num_units_2', hp.Discrete([16,32,64]))
        HP_NUM_UNITS_3 = hp.HParam('num_units_3', hp.Discrete([32,64]))
        HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
        HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam']))

        METRIC_ACCURACY = 'accuracy'

        with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
          hp.hparams_config(
            hparams=[HP_NUM_UNITS_1,HP_NUM_UNITS_2, HP_DROPOUT,HP_NUM_UNITS_3, HP_OPTIMIZER],
            metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
          )
```

```
In [ ]: def train_test_model(hparams):
          # number of possible label values
          model = Sequential()
          #input_shape = (height,width,channels)
          # Here batch size will be None at the beginng at later when we fit the model it will be used
          model.add(Conv2D(hparams[HP_NUM_UNITS_1],kernel_size =(3,3),activation ='relu', padding='same', input
        _shape=(28, 28,1)))
          model.add(Conv2D(hparams[HP_NUM_UNITS_2], kernel_size=(3,3), activation='relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))
          model.add(Dropout(hparams[HP_DROPOUT]))
          #flatted 3d ouput to 1d
          model.add(Flatten())
          model.add(Dense(hparams[HP_NUM_UNITS_3], activation='relu'))
          model.add(Dense(hparams[HP_NUM_UNITS_3], activation='relu'))
          # Output clasess =10 Hence used 10 outputs in Dense layer
          model.add(Dense(10, activation='softmax'))


          model.compile(
              optimizer=hparams[HP_OPTIMIZER],
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'],
          )

          model.fit(x_train, y_train, epochs=3)
          _, accuracy = model.evaluate(x_test, y_test)
          return accuracy
```

```
In [ ]: def run(run_dir, hparams):
          with tf.summary.create_file_writer(run_dir).as_default():
            hp.hparams(hparams)  # record the values used in this trial
            accuracy = train_test_model(hparams)
            tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

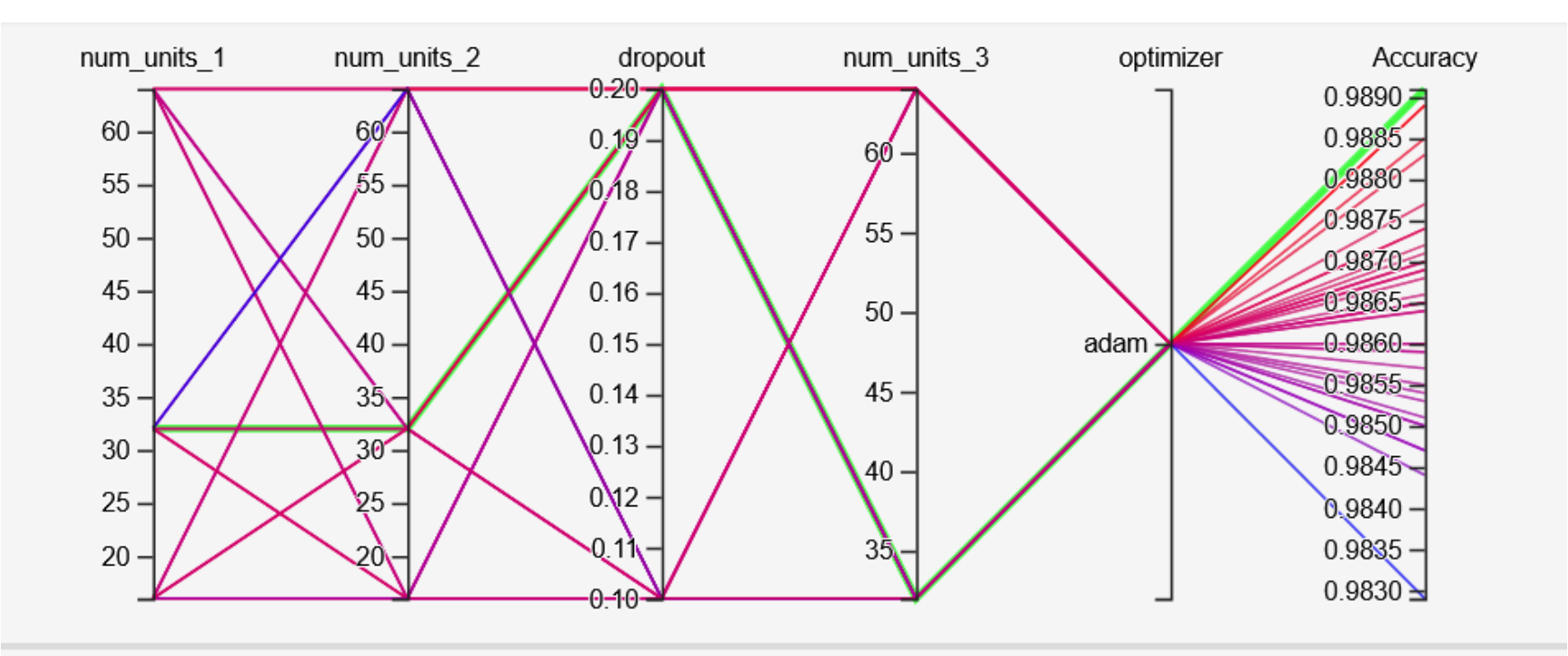
```
In [1]: session_num = 0
        for num_units_1 in HP_NUM_UNITS_1.domain.values:
          for num_units_2 in HP_NUM_UNITS_2.domain.values:
            for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_value):
              for num_units_3 in HP_NUM_UNITS_3.domain.values:
                hparams = {
                   HP_NUM_UNITS_1: num_units_1,
                   HP_NUM_UNITS_2: num_units_2,
                   HP_DROPOUT: dropout_rate,
                   HP_NUM_UNITS_3: num_units_3,
                   HP_OPTIMIZER: 'adam',
                }
                run_name = "run-%d" % session_num
                print('--- Starting trial: %s' % run_name)
                print({h.name: hparams[h] for h in hparams})
                run('logs/hparam_tuning/' + run_name, hparams)
                session_num += 1
```

Part of training process

```
--- Starting trial: run-34
{'num_units_1': 64, 'num_units_2': 64, 'dropout': 0.2, 'num_units_3': 32, 'optimizer': 'adam'}
Epoch 1/3
1875/1875 [==============================] - 166s 88ms/step - loss: 0.3052 - accuracy: 0.9016
Epoch 2/3
1875/1875 [==============================] - 165s 88ms/step - loss: 0.0511 - accuracy: 0.9838
Epoch 3/3
1875/1875 [==============================] - 165s 88ms/step - loss: 0.0339 - accuracy: 0.9891
313/313 [==============================] - 7s 21ms/step - loss: 0.0408 - accuracy: 0.9870
--- Starting trial: run-35
{'num_units_1': 64, 'num_units_2': 64, 'dropout': 0.2, 'num_units_3': 64, 'optimizer': 'adam'}
Epoch 1/3
1875/1875 [==============================] - 169s 89ms/step - loss: 0.2786 - accuracy: 0.9126
Epoch 2/3
1875/1875 [==============================] - 168s 89ms/step - loss: 0.0462 - accuracy: 0.9853
Epoch 3/3
1875/1875 [==============================] - 167s 89ms/step - loss: 0.0306 - accuracy: 0.9898
313/313 [==============================] - 7s 22ms/step - loss: 0.0376 - accuracy: 0.9885
```

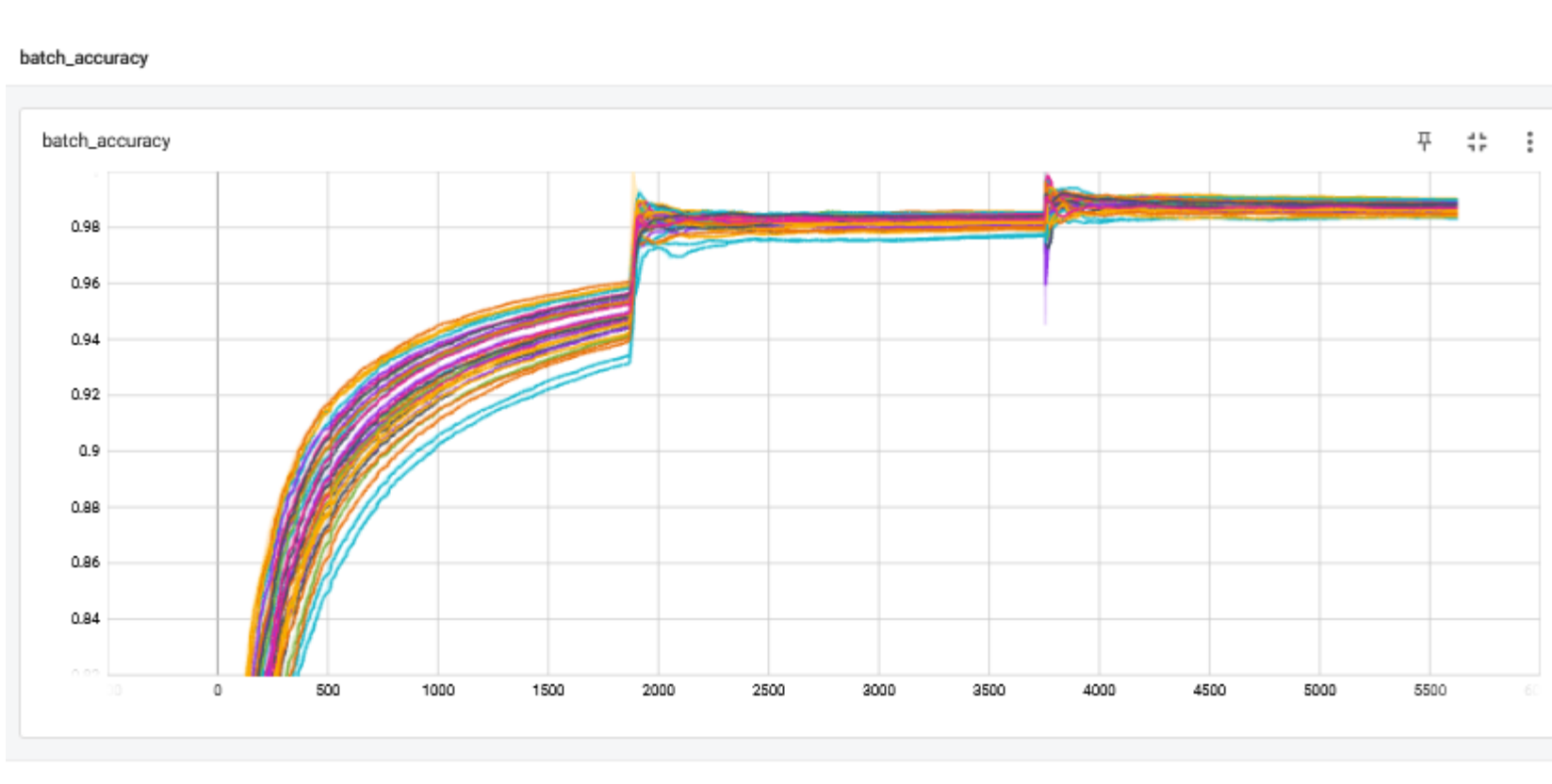
```
In [2]: %tensorboard --logdir logs/hparam_tuning
```

3. Justification for Hyperparameter selection

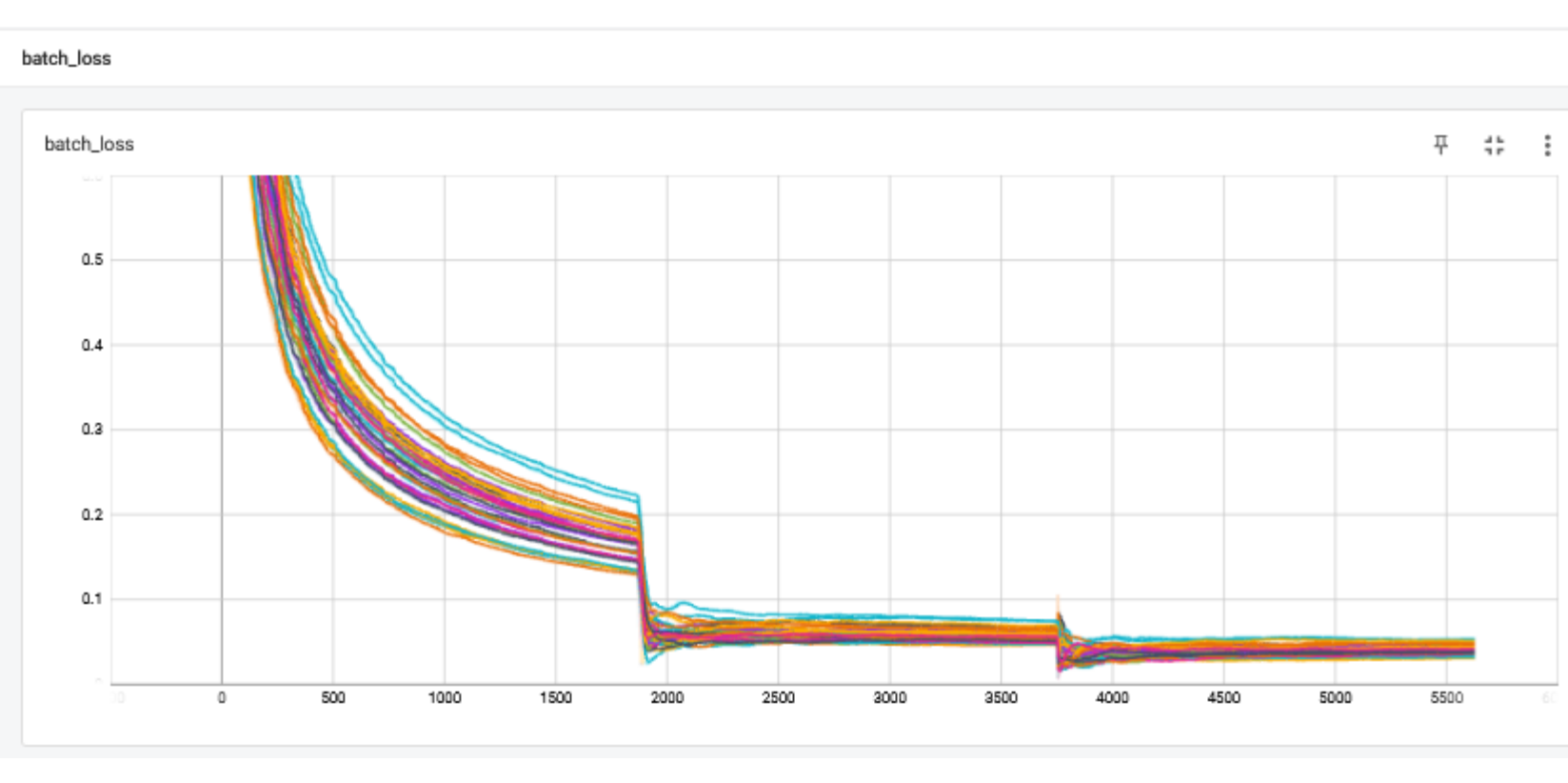


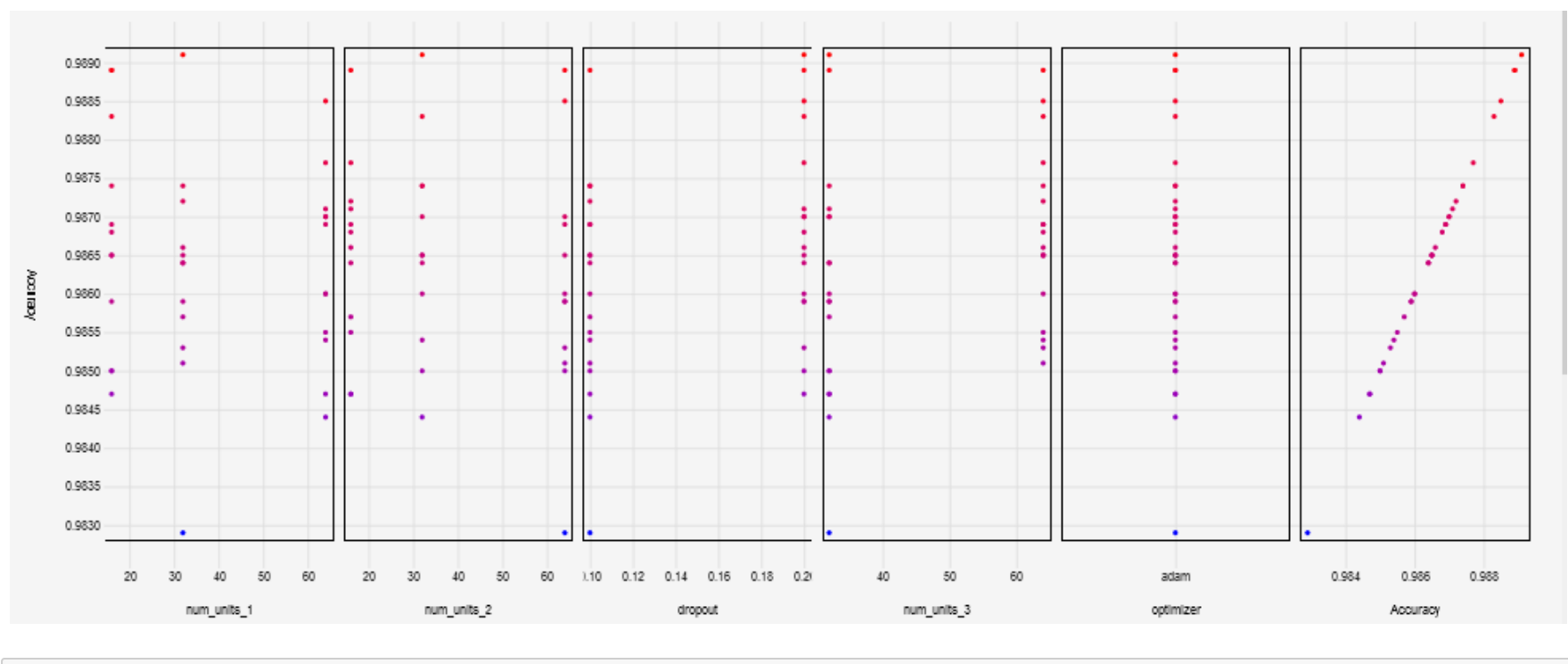
Trial ID	num_units_1	num_units_2	dropout	num_units_3	optimizer	Accuracy
0f00e6d5ca2a4...	32.000	32.000	0.20000	32.000	adam	0.98910

batch_accuracy



batch_loss





```
In [ ]: # number of possible label values
        model = Sequential()
        #input_shape = (height,width,channels)
        # Here batch size will be None at the beginng at later when we fit the model it will be used
        model.add(Conv2D(32,kernel_size =(3,3),activation ='relu', padding='same', input_shape=(28, 28,1)))
        model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))
        #flatted 3d ouput to 1d
        model.add(Flatten())
        model.add(Dense(32, activation='relu'))
        model.add(Dense(32, activation='relu'))
        # Output clasess =10 Hence used 10 outputs in Dense layer
        model.add(Dense(10, activation='softmax'))
        model.summary()
```

```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_72 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 26, 26, 32)        9248
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_36 (Dropout)         (None, 13, 13, 32)        0
_________________________________________________________________
flatten_36 (Flatten)         (None, 5408)              0
_________________________________________________________________
dense_108 (Dense)            (None, 32)                173088
_________________________________________________________________
dense_109 (Dense)            (None, 32)                1056
_________________________________________________________________
dense_110 (Dense)            (None, 10)                330
=================================================================
Total params: 184,042
Trainable params: 184,042
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: model.compile(optimizer= 'adam',loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
        #tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Model Training

```
In [ ]: log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
        tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

        model.fit(x_train,
                  y_train,
                  validation_split=0.2,
                  epochs=10,
                  callbacks=[tensorboard_callback])
```

```
Epoch 1/10
1500/1500 [==============================] - 58s 38ms/step - loss: 0.4340 - accuracy: 0.8680 - val_loss: 0.081
7 - val_accuracy: 0.9755
Epoch 2/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0770 - accuracy: 0.9770 - val_loss: 0.070
5 - val_accuracy: 0.9801
Epoch 3/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0515 - accuracy: 0.9845 - val_loss: 0.048
6 - val_accuracy: 0.9854
Epoch 4/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0377 - accuracy: 0.9878 - val_loss: 0.051
7 - val_accuracy: 0.9852
Epoch 5/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0274 - accuracy: 0.9917 - val_loss: 0.047
2 - val_accuracy: 0.9859
Epoch 6/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0260 - accuracy: 0.9910 - val_loss: 0.043
6 - val_accuracy: 0.9881
Epoch 7/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0196 - accuracy: 0.9938 - val_loss: 0.049
4 - val_accuracy: 0.9880
Epoch 8/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0182 - accuracy: 0.9941 - val_loss: 0.041
9 - val_accuracy: 0.9893
Epoch 9/10
1500/1500 [==============================] - 56s 37ms/step - loss: 0.0168 - accuracy: 0.9938 - val_loss: 0.052
1 - val_accuracy: 0.9871
Epoch 10/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.0139 - accuracy: 0.9952 - val_loss: 0.060
3 - val_accuracy: 0.9862
```

Model Evaluate

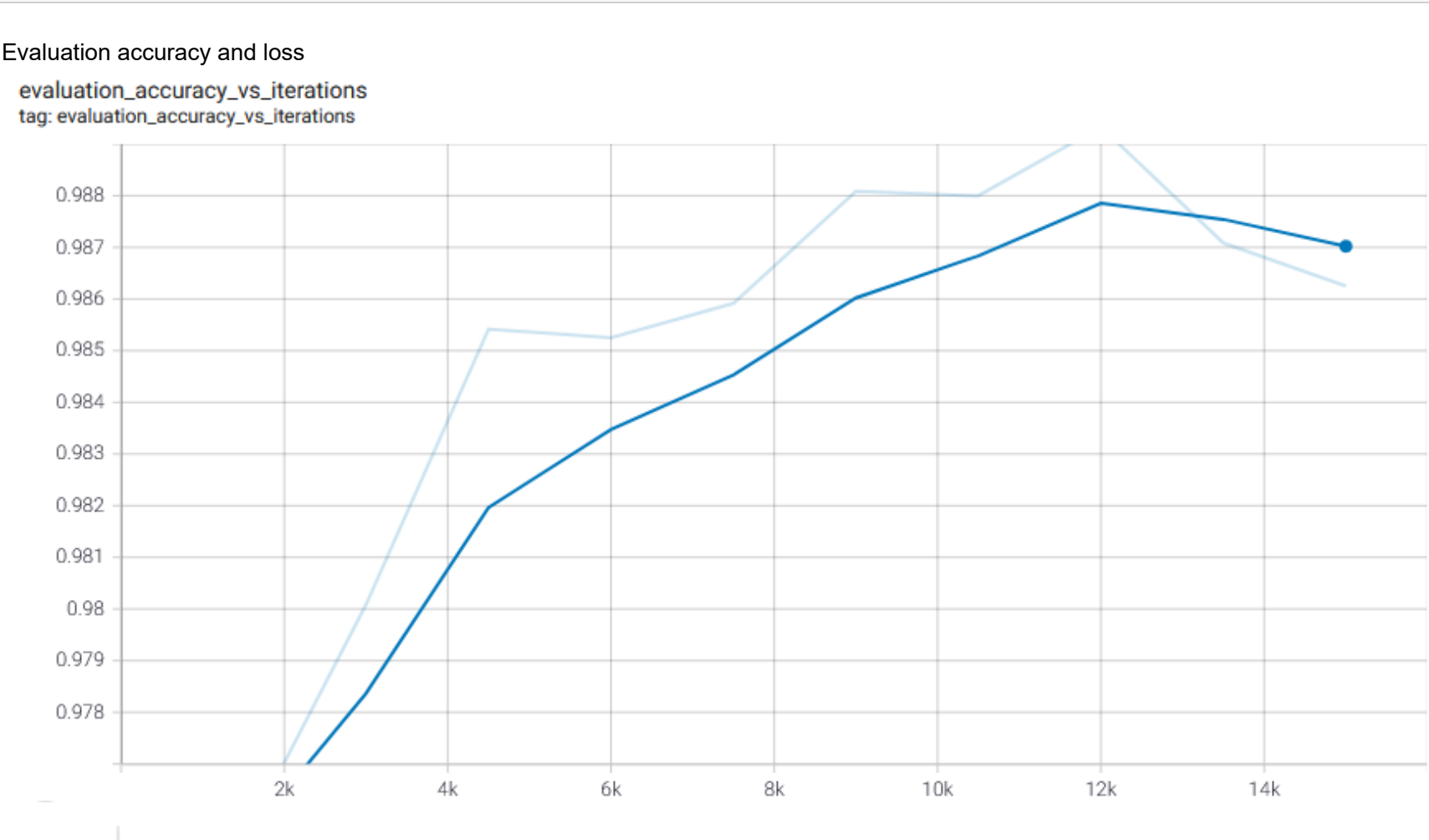
```
In [ ]: test_loss, test_acc = model.evaluate(x_test, y_test)

        print('Test Accuracy',test_acc)
        print('\nTest loss:', test_loss)
```

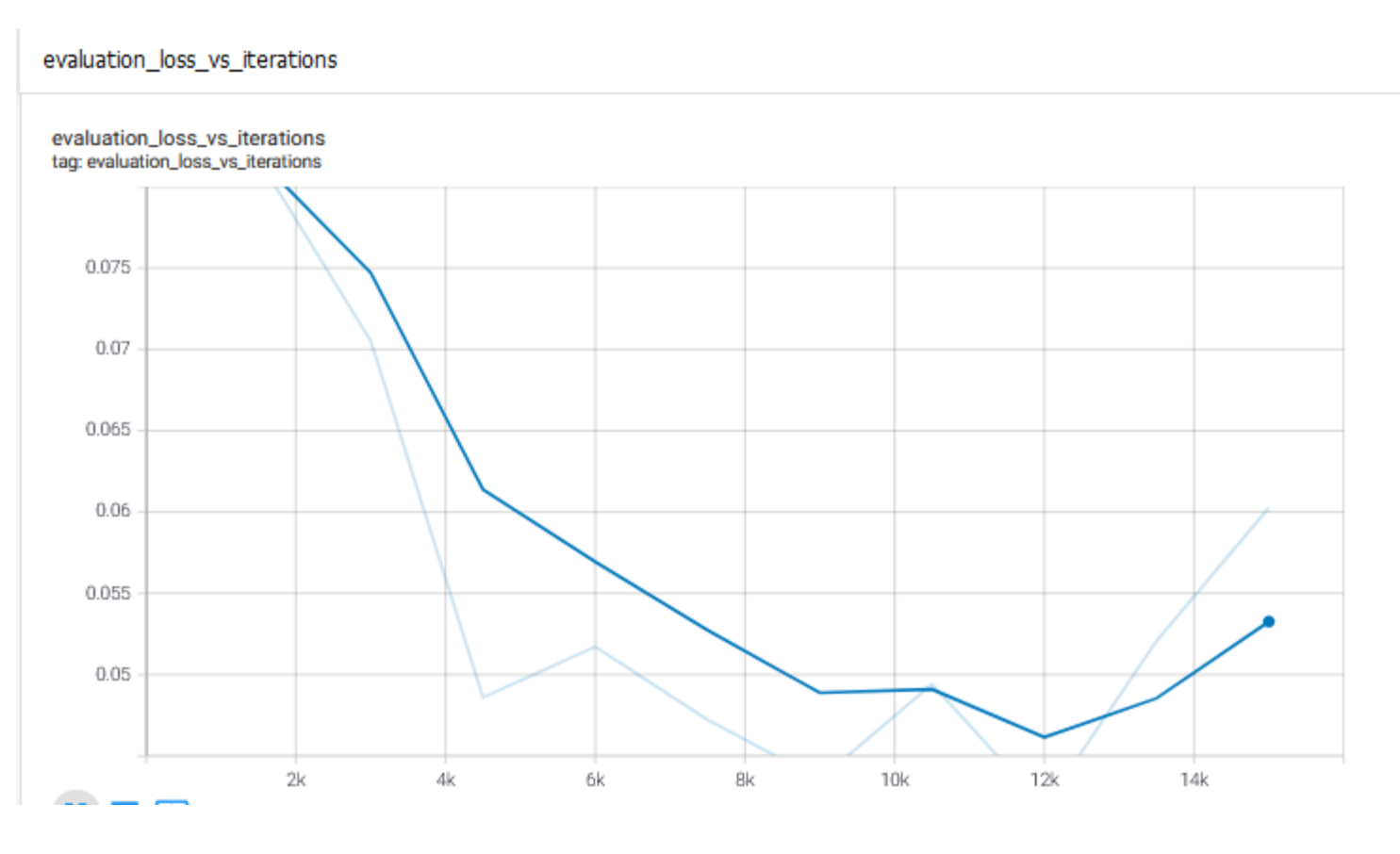
```
313/313 [==============================] - 3s 10ms/step - loss: 0.0478 - accuracy: 0.9872
Test Accuracy 0.9872000217437744

Test loss: 0.04780405014753342
```

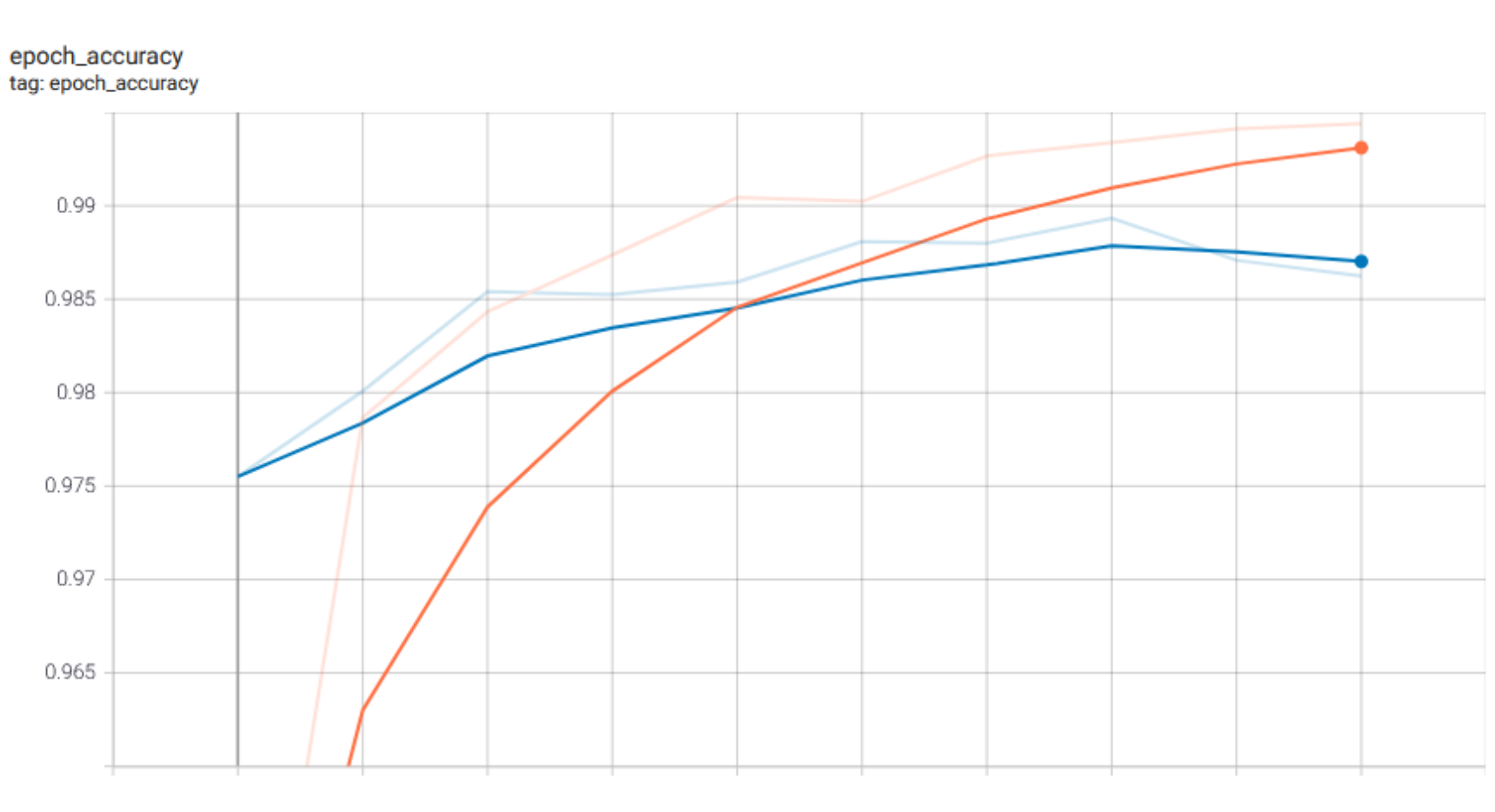
4. Train and Test Results

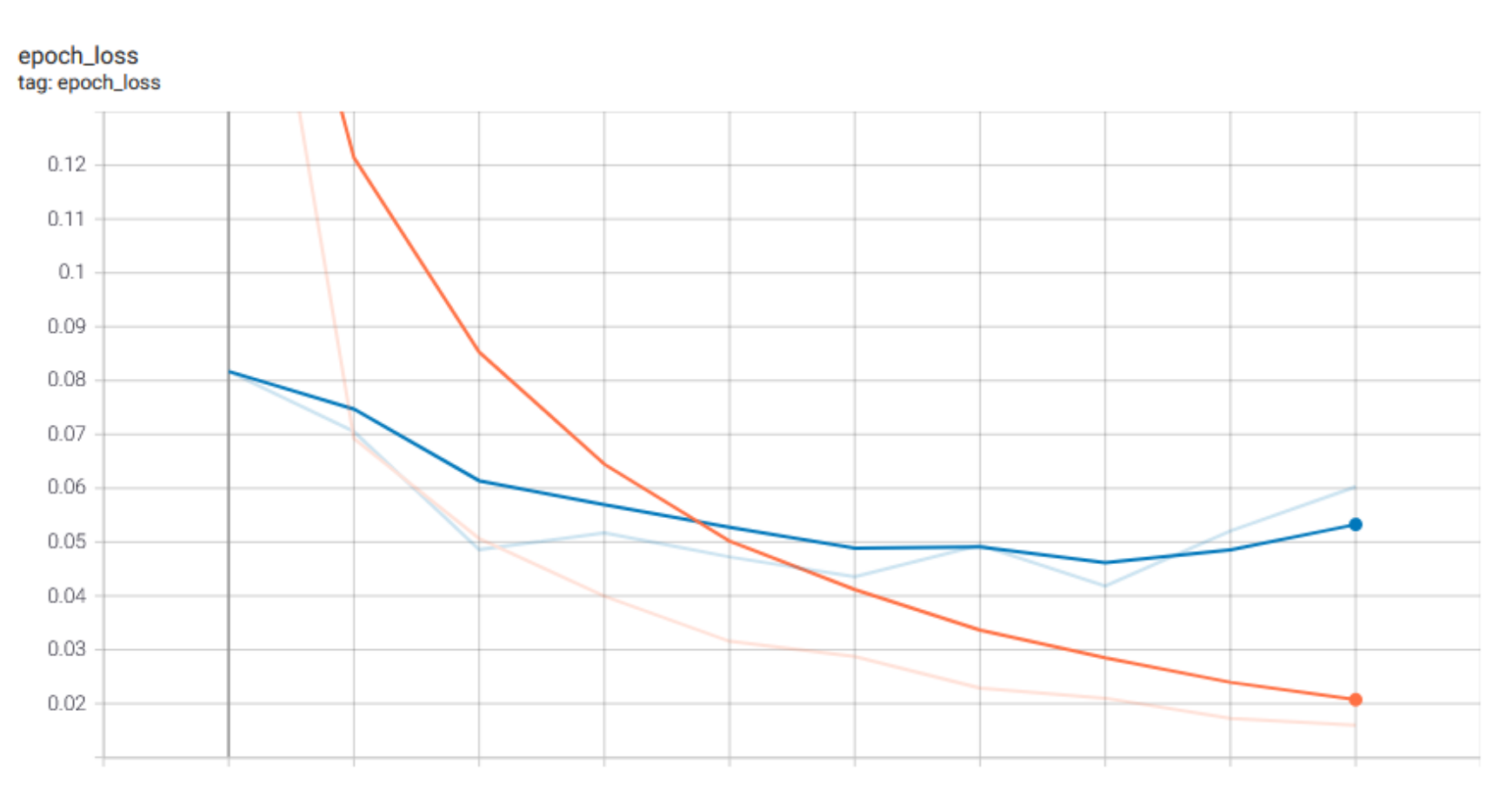
```
In [ ]: %tensorboard --logdir logs/fit
```

Evaluation accuracy and loss









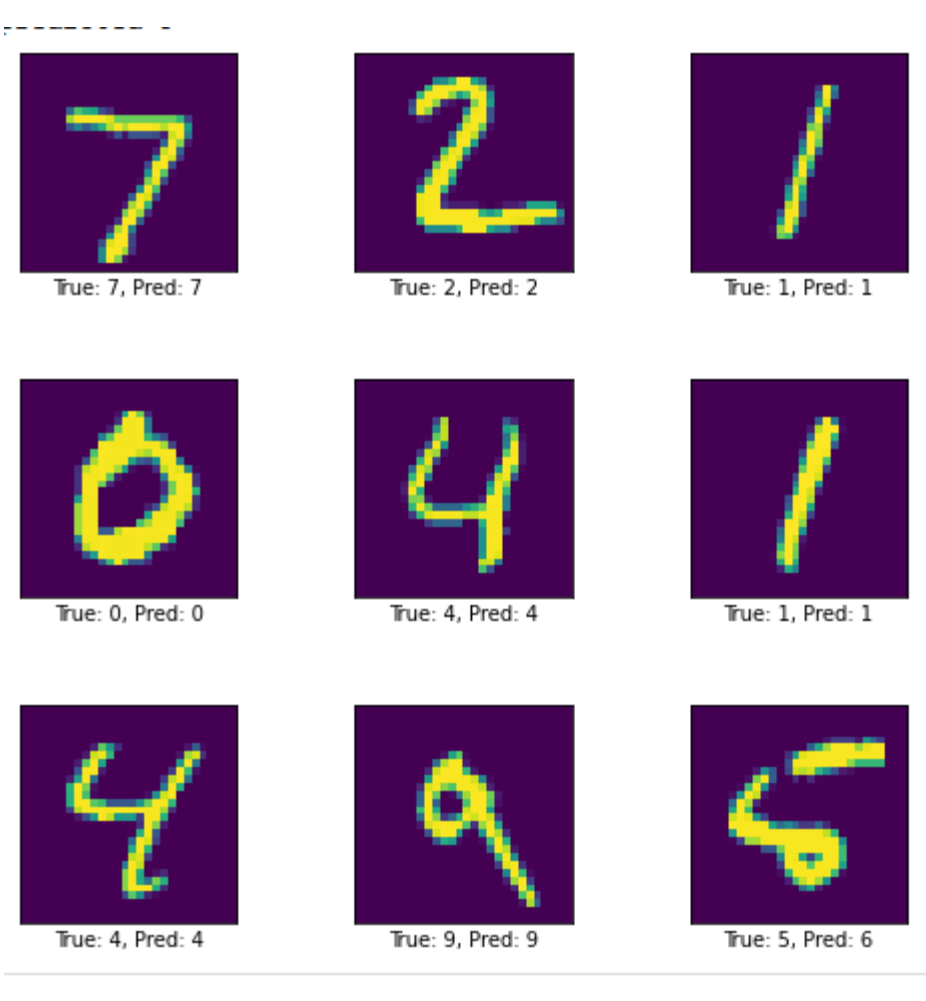
Prediction

```
In [ ]: # predicting on the test data
        y_pred = model.predict(x_test)
```

5. Actual and Predicted Results with image

```
In [ ]: i=20
        print(y_pred[i])
        print(y_test[i])
        #t=y_test[i]
        #print('t', t)
        #print(t[0])
        print("tested", y_test[i])
        print(np.argmax(y_pred[i]))
        print("predicted" , np.argmax(y_pred[i]))

        fig, axes = plt.subplots(3, 3, figsize=(8,8))
        fig.subplots_adjust(hspace=0.5, wspace=0.5)

        for i, ax in enumerate(axes.flat):
          ax.imshow(x_test[i].reshape(28,28))
          xlabel = "True: {0}, Pred: {1}".format(y_test[i],np.argmax(y_pred[i]))
          ax.set_xlabel(xlabel)
          ax.set_xticks([])
          ax.set_yticks([])

        plt.show()
```

```
[4.5562766e-17 2.6714964e-10 5.9752610e-13 1.0281449e-08 1.4678750e-05
 9.0842045e-09 6.0565212e-17 1.5388780e-08 4.1803919e-08 9.9998534e-01]
9
tested 9
9
predicted 9
```



6. Discussion

Hyperparameters selected as 32 units for two convolutional layers and 32 units as for dense layers (fully connected layers). Drop out value selected as 0.2. Also it's interesting to find out when units are form same value model tend to give good accuracy instead of combining with different units for each layer.

After 10 epcohs model gave 99.38% accuracy and 98.62% validation acuracy. Test set gave 98.72% accuracy which is pretty good value. Considering the training loss 0.04 and test loss 0.06, we can say it's a generalization gap and it's acceptable. Considering the actual and predicted results model had hard time recongizing images that difficult to clearly seperate for even for human . Example -: In section 5, Digit represent 5 but predicted 6.

Limitations and challenges

Orginaly planned architecture was difficult to run due to limitation of GPU and new captcha promting when notebook was inactive for time period in Google colab. Due to all these reason this architecture for CNN used and number of hyperparameters had to reduce for same reasons (long run time and more than 200 expermental trail needed for wider hyperparameter selection). Only used 3 epaochs for hyper parameter tuning.

Other Images

conv2d_72

conv2d_72

conv2d_73

conv2d_73

dense_109