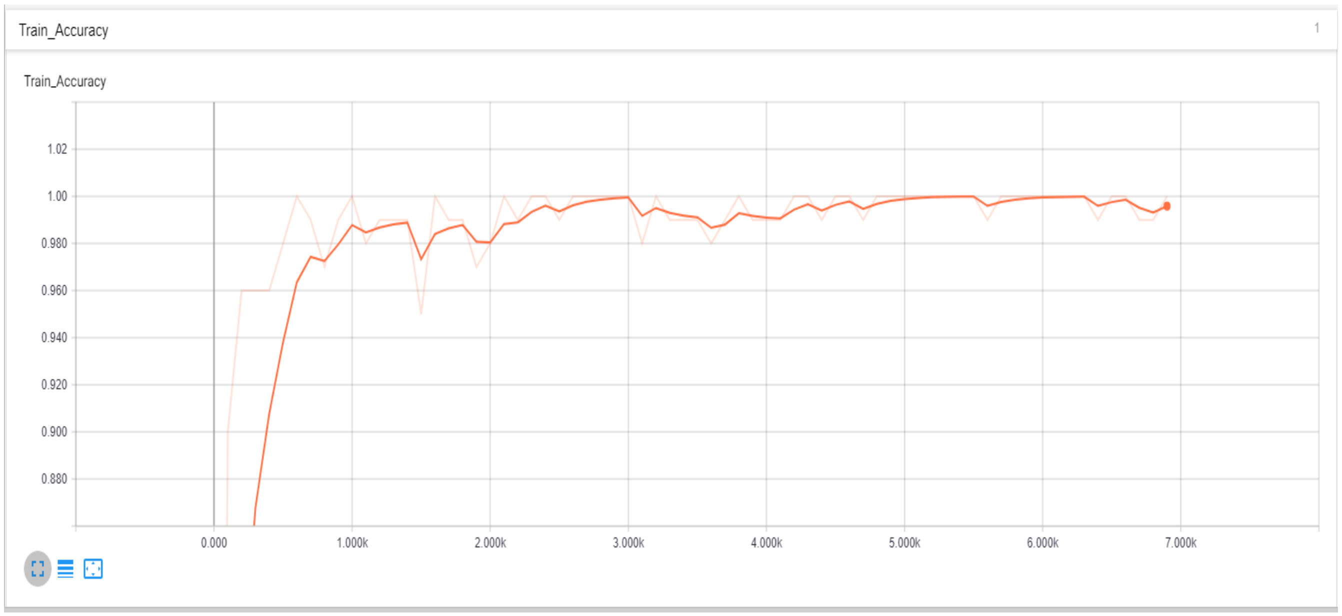
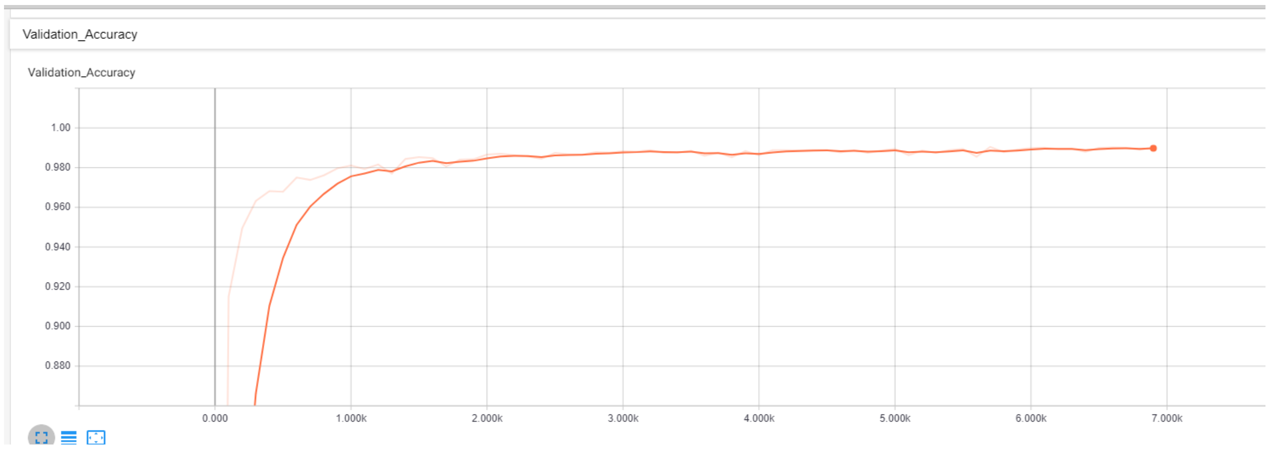
Page 1: (MNIST w TUTORIAL)

1. Accuracy plot (train and val)
2. Accuracies (train, test, val) from the 'best' model with a brief explanation.

Training :



Validation:



Best Model

The Best model I choose is after 6900 batches. The validation accuracy seems almost high and constant (before and after) certain number of fixed batches. It helps me choose the best accuracy for test and training data too after certain number of fixed batches. The validation set also help me select the best hyper parameters for the model.

Train Best accuracy : 99.58 after 6900 batches

Validation best accuracy: 99.02% after 6900 batches

Test accuracy :98.85% after 6900 batches

Page 2: (MNIST w MODIFIED)

1. Description of changes
2. Accuracy plot (train and val)
3. Accuracies (train, test, val) from the 'best' model with a brief explanation.

Changes:

1. Change Optimizer to : AdamOptimizer

   AdamOptimizer: tf.train.AdamOptimizer(learning_rate=0.001)
   Adam optimizer:The main advantage is Appropriate for problems with very noisy/or sparse gradients.

2. Change Learning Rate: We experimented with several learning rate (0.1,0.01,0.001,0.0001)

   At end we choose 0.001

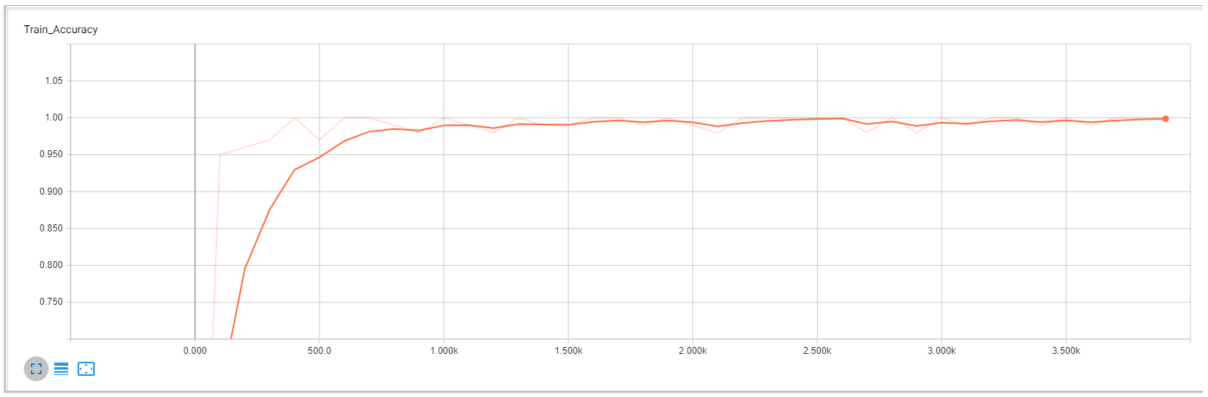
3. Changed Filter/Kernel Size ->We changed filter size of almost every layer with respect to the tutorial architecture ,
   First Conv Layer we chose : 3 x 3
   2nd Conv Layer :5 x 5
   3rd conv Layer: 5 x 5

4. Output channels-We changes output channels of the layers.
   First Conv Layer:64
   2nd Conv Layer :32
   3rd conv Layer: 32

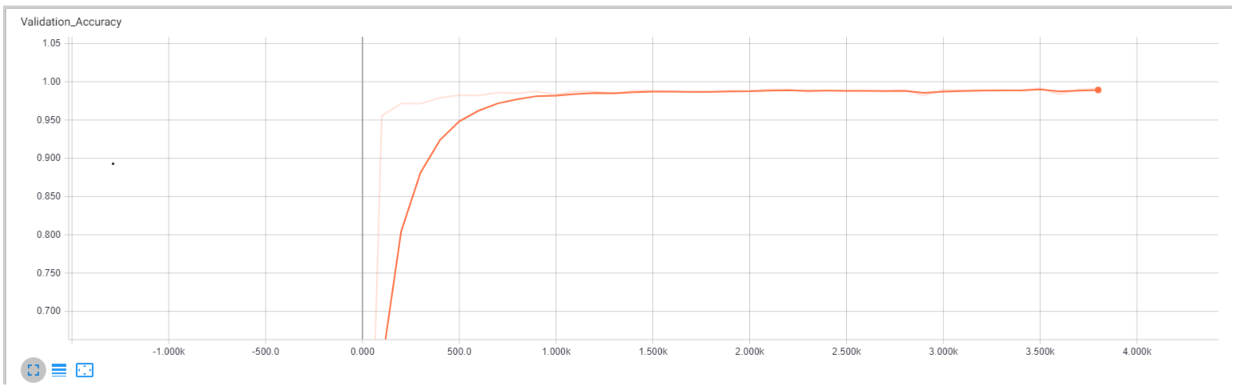
5. Add one extra convolution Layer ->Before the fully connected/dense layers at end we added one extra convolution layers
   conv Layer: 5 x 5 (Filter Size)
   output channel :32
   padding="same"
   activation:Relu

6. Change the value of Dropout in the Final Layer –We experimented with different drop out value too in final dense layer, so that we could avoid overfitting in the model.
   We chose dropout value:0.5

Training Accuracy:



Validation Accuracy:



The Best model I choose is after 3800 batches. The validation accuracy seems to be high and constant (before and after) certain number of fixed batches .It helps me choose the best accuracy for test and training data too after certain number of fixed batches. The validation set also help me select the best hyper parameters for the model.

TRAIN Best Accuracy:99.88 after 3800 batches

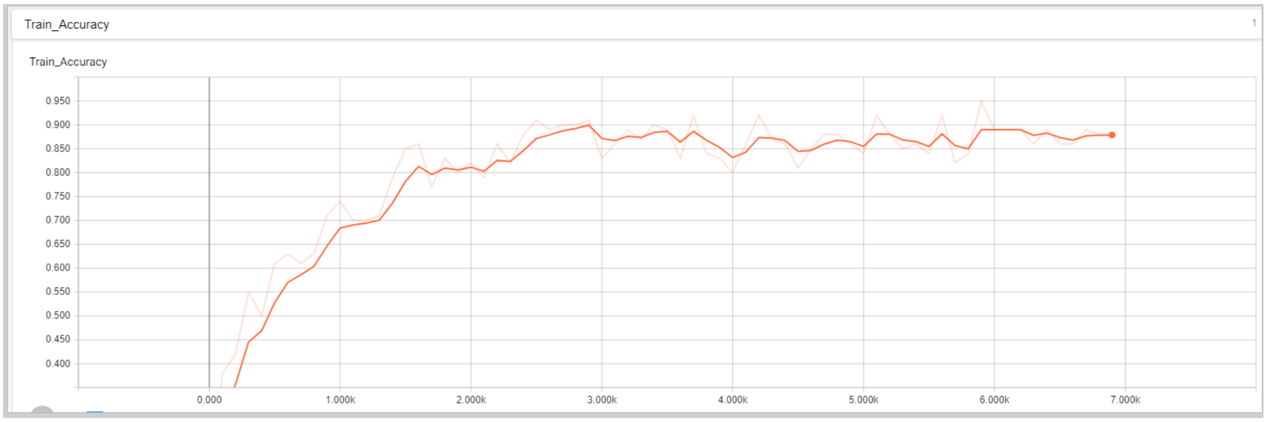
Validation best accuracy:98.93% after 3800 batches

Test Accuracy :98.76% after 3800 batches

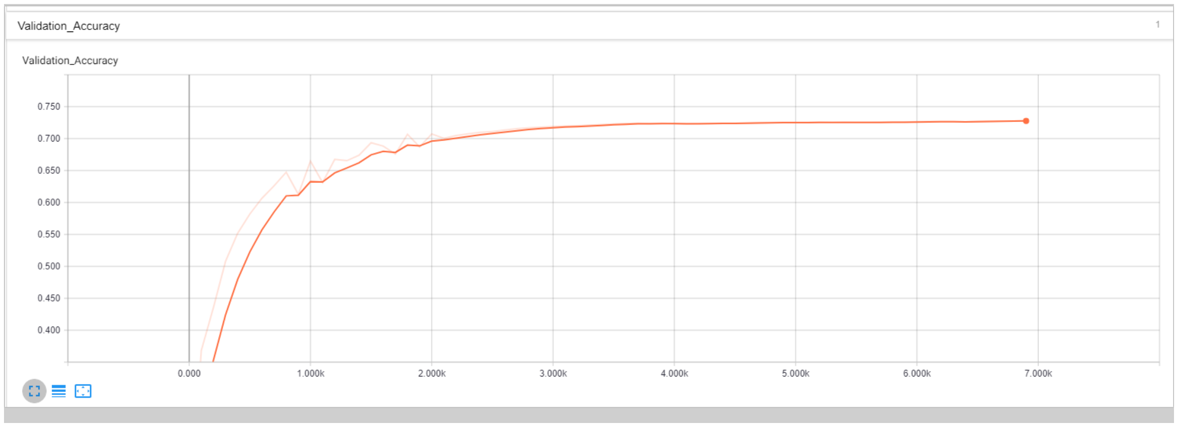
Page 3: (CIFAR w TUTORIAL)

1. Accuracy plot (train and val)
2. Accuracies (train, test, val) from the 'best' model with a brief explanation

Training Accuracy:



Validation Accuracy:



The Best model I choose is after 6900 batches. The validation accuracy seems to be high and constant (before and after) certain number of fixed batches .It helps me choose the best accuracy for test and training data too after certain number of fixed batches. The validation set also help me select the best hyper parameters for the model.

TRAINING best Accuracy: 87.89% after 6900 batches

Validation best accuracy:72.83% after 6900 batches

Test Accuracy : 72.67% after 6900 batches

Page 4: (CIFAR w MODIFIED)

1. Description of changes
2. Accuracy plot (train and val)
3. Accuracies (train, test, val) from the 'best' model with a brief explanation

Changes:

1. Change Optimizer to : AdamOptimizer

   AdamOptimizer: tf.train.AdamOptimizer(learning_rate=0.001)

   Adam optimizer:The main advantage is Appropriate for problems with very noisy/or sparse gradients.

2 .Change Learning Rate : : We experimented with several learning rates so the cost functions reaches global minimum. (0.1,0.01,0.001,0.0001)
We used adaptive learning rate ,we reduced it after certain number of steps.

At the end we choose 0.001 till 2000 steps
And after we reduce it to 0.00001

3. Changed Filter/Kernel Size ->We changed filter size of almost every layer with respect to the tutorial architecture .

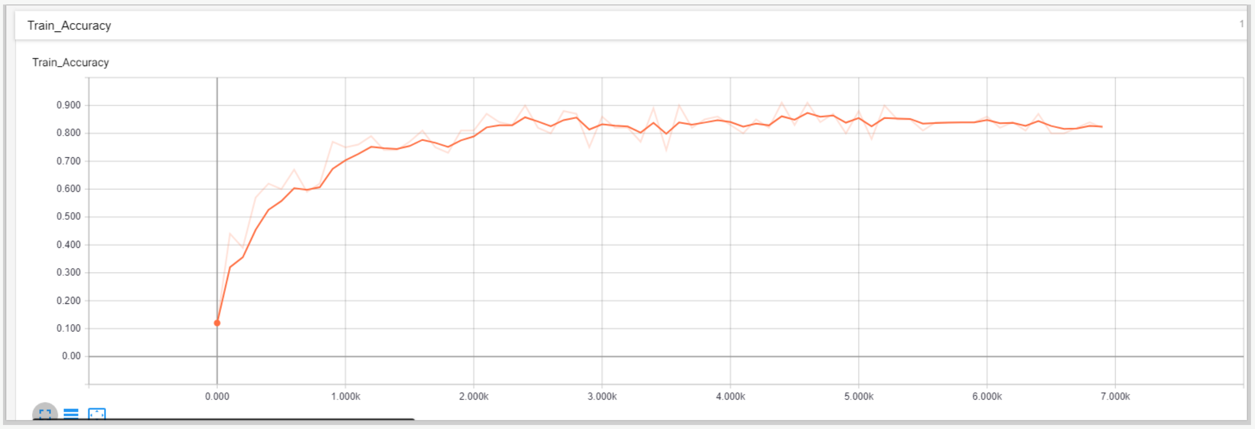
   First Conv Layer- 5 x5
   Pooling layer –Filter size (3 x3) and stride (2 x2)
   2nd Conv Layer :5 x 5
   Pooling layer –Filter size (3 x3) and stride (2 x2)

4. Output channels-We changes output channels of the layers
   First Conv Layer:16
   2nd Conv Layer :32

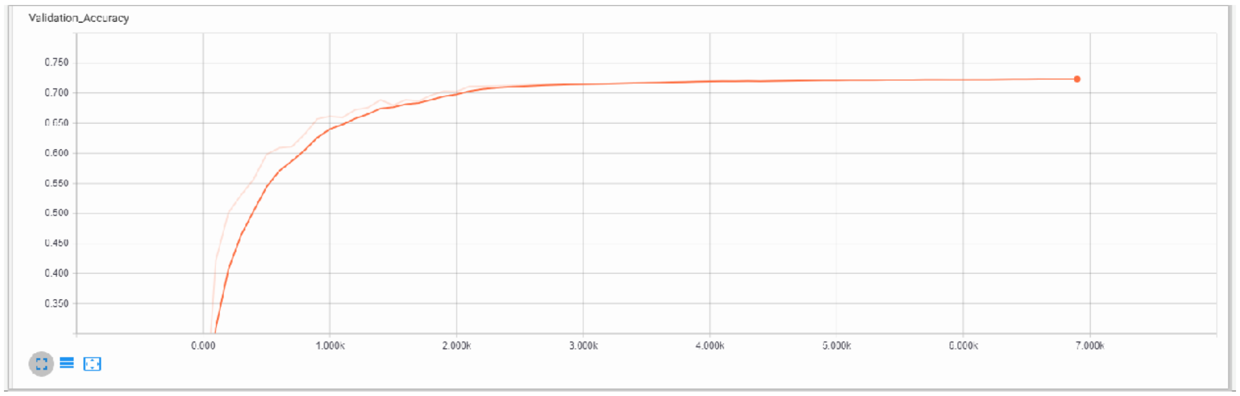
5. Added extra fully Connected Layers- We added extra fully connected layer /dense layer.
   The number of output units of the fully connected layers is 192

6. Added Batch Normalization –We added batch normalization between the layers
7. We added L2 Regularization to first two Convolution Layers with scale= 1e-4

Training Accuracy:



Validation Accuracy:



The Best model I choose is after 6900 batches. The validation accuracy seems to be high and constant (before and after) certain number of fixed batches .It helps me choose the best accuracy for test and training data too after certain number of fixed batches. The validation set also help me select the best hyper parameters for the model.

#Train Accuracy:82.39 after 6900 batches

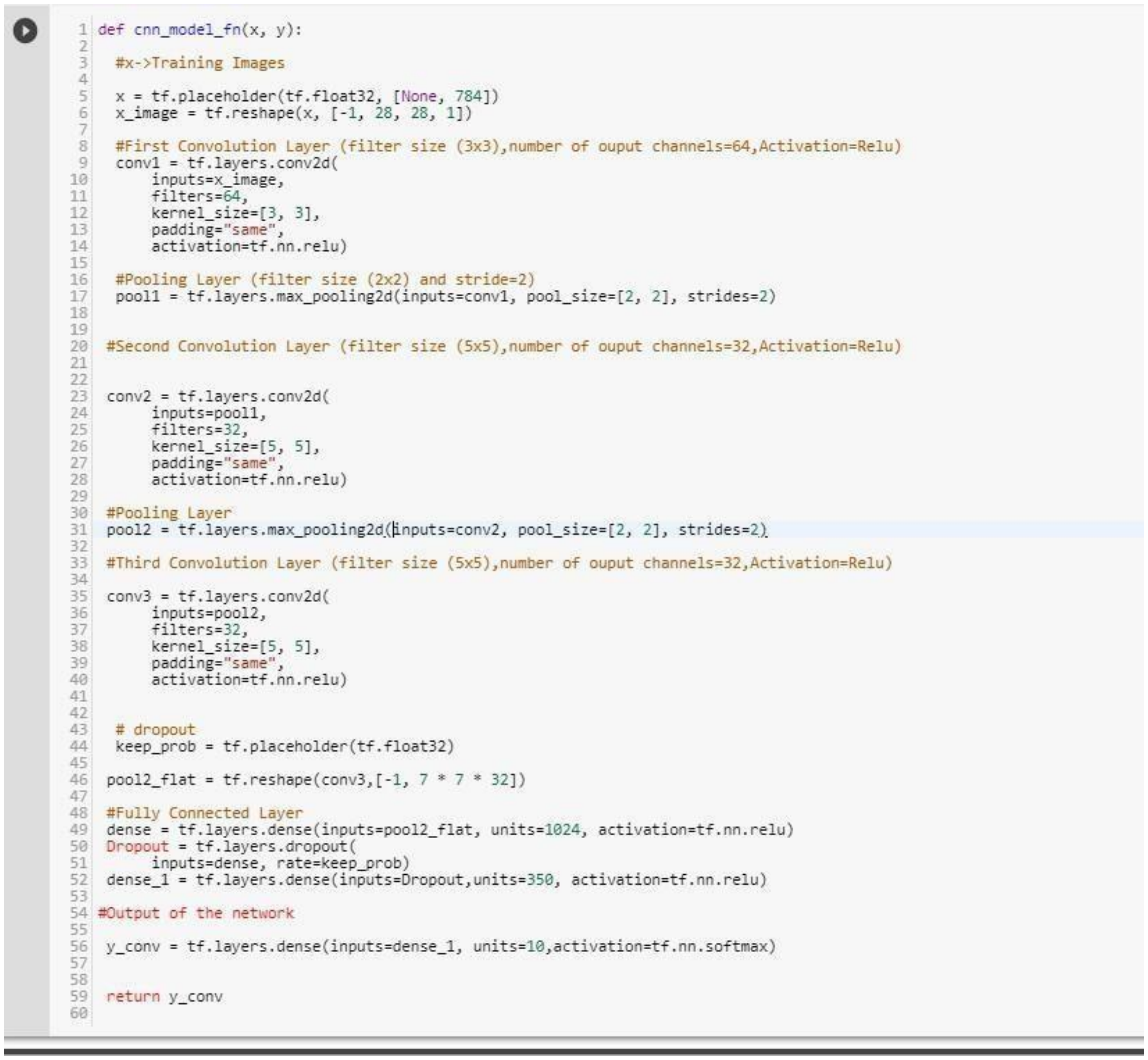
# Validation best accuracy:72.32% after 6900 batches

#Test Accuracy :71.46% after 6900 batches

Page 5-6: (CODE SNIPPETS - MNIST MODIFIED)

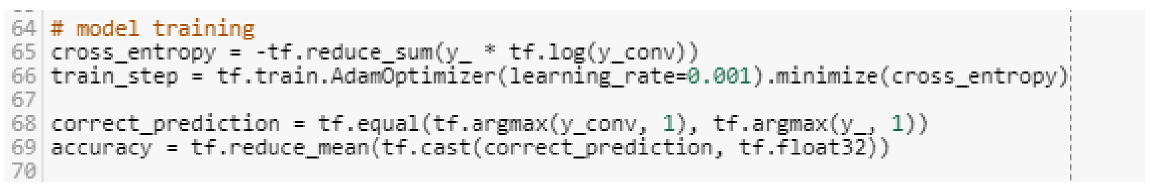
1. Creating the model/graph: Corresponds to the cnn_model_fn in the mnist tutorial.
2. Defining loss & optimizer: Part where you set the loss (Corresponds to line 100 in the mnist tutorial) and optimizer.
3. Training: Part where you either manually defined a train loop or where you used a built-in train function to train the model.
4. Any other snippet you found relevant

1. Model Architecture



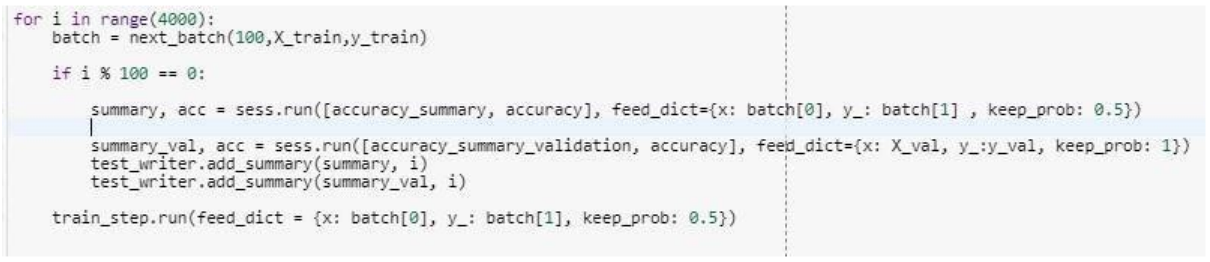
```
def cnn_model_fn(x, y):

  #x->Training Images

  x = tf.placeholder(tf.float32, [None, 784])
  x_image = tf.reshape(x, [-1, 28, 28, 1])

  #First Convolution Layer (filter size (3x3),number of ouput channels=64,Activation=Relu)
  conv1 = tf.layers.conv2d(
      inputs=x_image,
      filters=64,
      kernel_size=[3, 3],
      padding="same",
      activation=tf.nn.relu)

  #Pooling Layer (filter size (2x2) and stride=2)
  pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)


 #Second Convolution Layer (filter size (5x5),number of ouput channels=32,Activation=Relu)


 conv2 = tf.layers.conv2d(
      inputs=pool1,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)

 #Pooling Layer
 pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

 #Third Convolution Layer (filter size (5x5),number of ouput channels=32,Activation=Relu)

 conv3 = tf.layers.conv2d(
      inputs=pool2,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)


  # dropout
  keep_prob = tf.placeholder(tf.float32)

 pool2_flat = tf.reshape(conv3,[-1, 7 * 7 * 32])

 #Fully Connected Layer
 dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.relu)
 Dropout = tf.layers.dropout(
      inputs=dense, rate=keep_prob)
 dense_1 = tf.layers.dense(inputs=Dropout,units=350, activation=tf.nn.relu)

#Output of the network

 y_conv = tf.layers.dense(inputs=dense_1, units=10,activation=tf.nn.softmax)


 return y_conv
```

2. Cross Entropy loss and AdamOptimizer



```
# model training
cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv))
train_step = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Training of the Network and Logging Accuracy on TensorBoard:

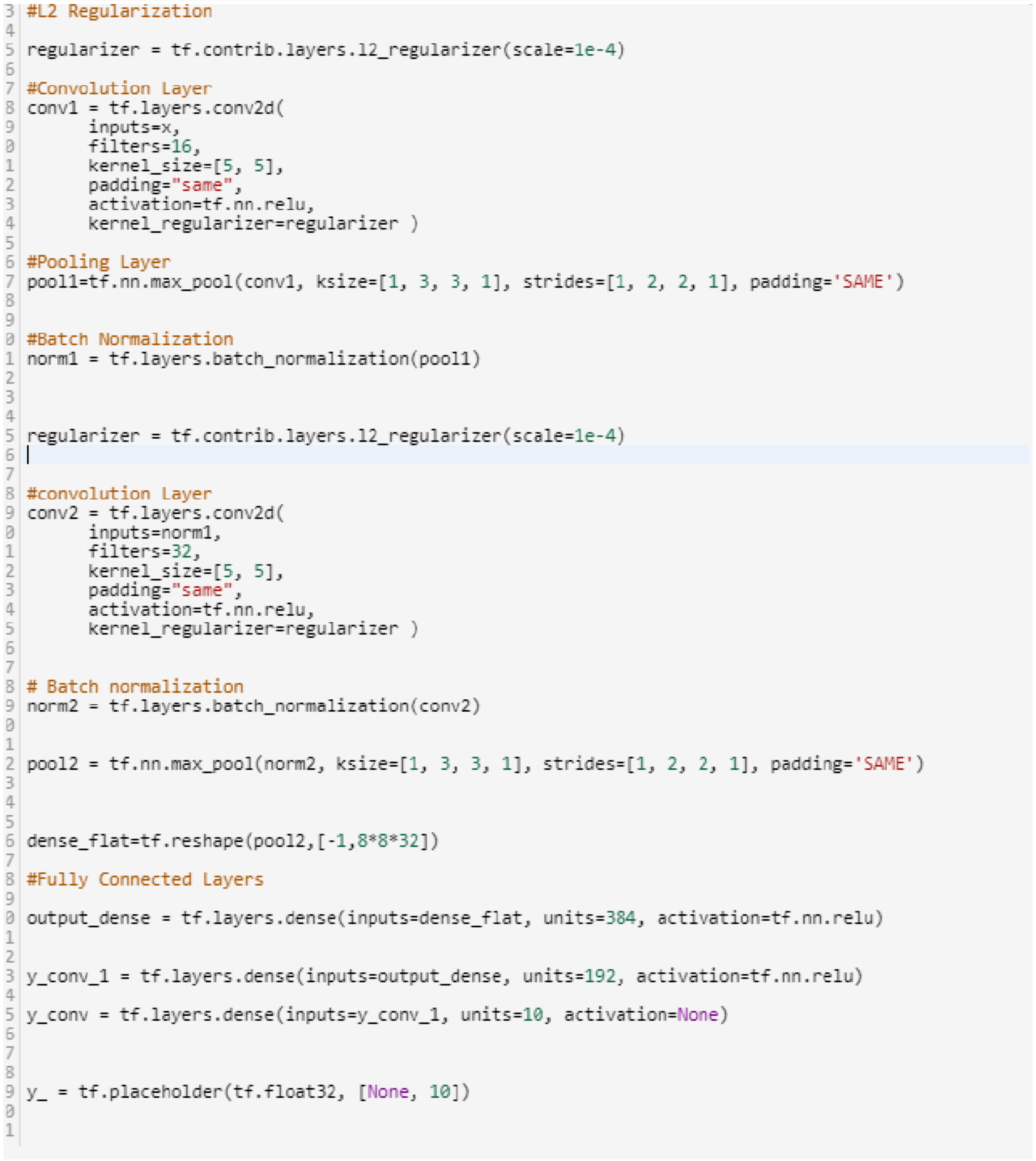


```
for i in range(4000):
    batch = next_batch(100,X_train,y_train)

    if i % 100 == 0:

        summary, acc = sess.run([accuracy_summary, accuracy], feed_dict={x: batch[0], y_: batch[1] , keep_prob: 0.5})

        summary_val, acc = sess.run([accuracy_summary_validation, accuracy], feed_dict={x: X_val, y_:y_val, keep_prob: 1})
        test_writer.add_summary(summary, i)
        test_writer.add_summary(summary_val, i)

    train_step.run(feed_dict = {x: batch[0], y_: batch[1], keep_prob: 0.5})
```

Page 7-8: (CODE SNIPPETS - CIFAR MODIFIED)

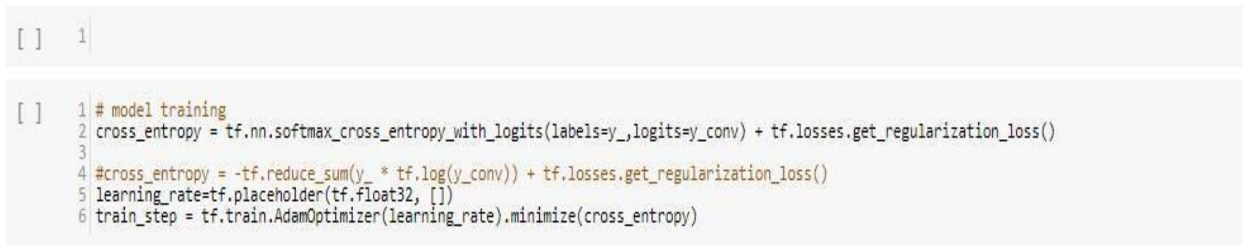
1. Creating the model/graph: Corresponds to the cnn_model_fn in the mnist tutorial.
2. Defining loss & optimizer: Part where you set the loss (Corresponds to line 100 in the mnist tutorial) and optimizer.
3. Training: Part where you either manually defined a train loop or where you used a built-in train function to train the model.
4. Any other snippet you found relevant

1. Model Architecture:



```
#L2 Regularization

regularizer = tf.contrib.layers.l2_regularizer(scale=1e-4)

#Convolution Layer
conv1 = tf.layers.conv2d(
      inputs=x,
      filters=16,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu,
      kernel_regularizer=regularizer )

#Pooling Layer
pool1=tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')


#Batch Normalization
norm1 = tf.layers.batch_normalization(pool1)



regularizer = tf.contrib.layers.l2_regularizer(scale=1e-4)


#convolution Layer
conv2 = tf.layers.conv2d(
      inputs=norm1,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu,
      kernel_regularizer=regularizer )


# Batch normalization
norm2 = tf.layers.batch_normalization(conv2)


pool2 = tf.nn.max_pool(norm2, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')



dense_flat=tf.reshape(pool2,[-1,8*8*32])

#Fully Connected Layers

output_dense = tf.layers.dense(inputs=dense_flat, units=384, activation=tf.nn.relu)


y_conv_1 = tf.layers.dense(inputs=output_dense, units=192, activation=tf.nn.relu)

y_conv = tf.layers.dense(inputs=y_conv_1, units=10, activation=None)



y_ = tf.placeholder(tf.float32, [None, 10])
```

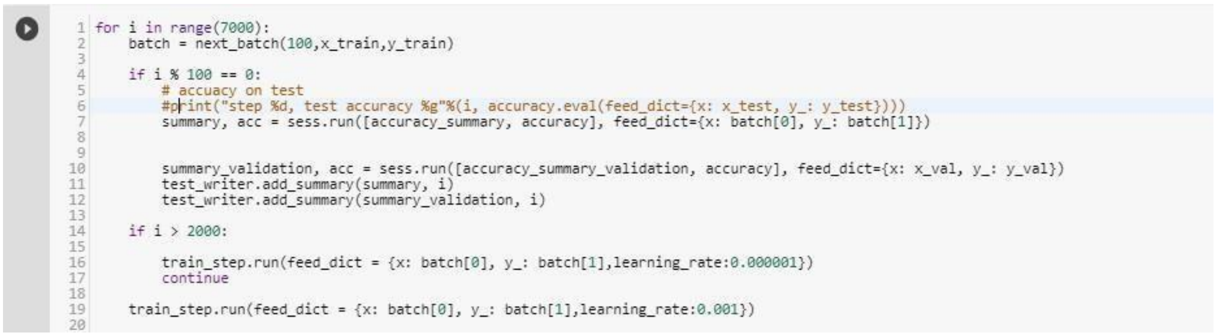
2. Cross Entropy Loss and AdamOptimizer



```
[ ]
```

```
[ ]  # model training
     cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,logits=y_conv) + tf.losses.get_regularization_loss()

     #cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv)) + tf.losses.get_regularization_loss()
     learning_rate=tf.placeholder(tf.float32, [])
     train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)
```

3. Training of the Network and Logging Accuracy on TensorBoard



```
for i in range(7000):
    batch = next_batch(100,x_train,y_train)

    if i % 100 == 0:
        # accuacy on test
        #print("step %d, test accuracy %g"%(i, accuracy.eval(feed_dict={x: x_test, y_: y_test})))
        summary, acc = sess.run([accuracy_summary, accuracy], feed_dict={x: batch[0], y_: batch[1]})


        summary_validation, acc = sess.run([accuracy_summary_validation, accuracy], feed_dict={x: x_val, y_: y_val})
        test_writer.add_summary(summary, i)
        test_writer.add_summary(summary_validation, i)

    if i > 2000:

        train_step.run(feed_dict = {x: batch[0], y_: batch[1],learning_rate:0.000001})
        continue

    train_step.run(feed_dict = {x: batch[0], y_: batch[1],learning_rate:0.001})

```