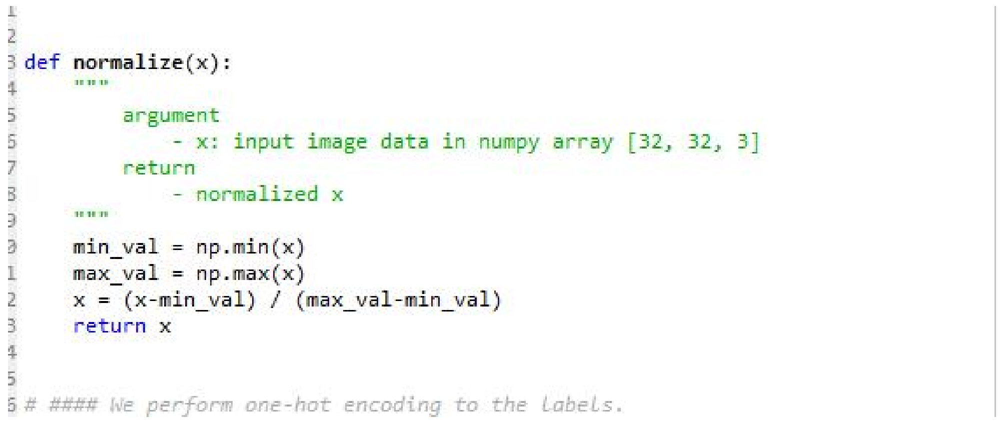
Distributed Data Analytics
Exercise sheet # 7
Arooba Jamil Khokhar
278077

Exercise 1: Normalization Effect (CNN)

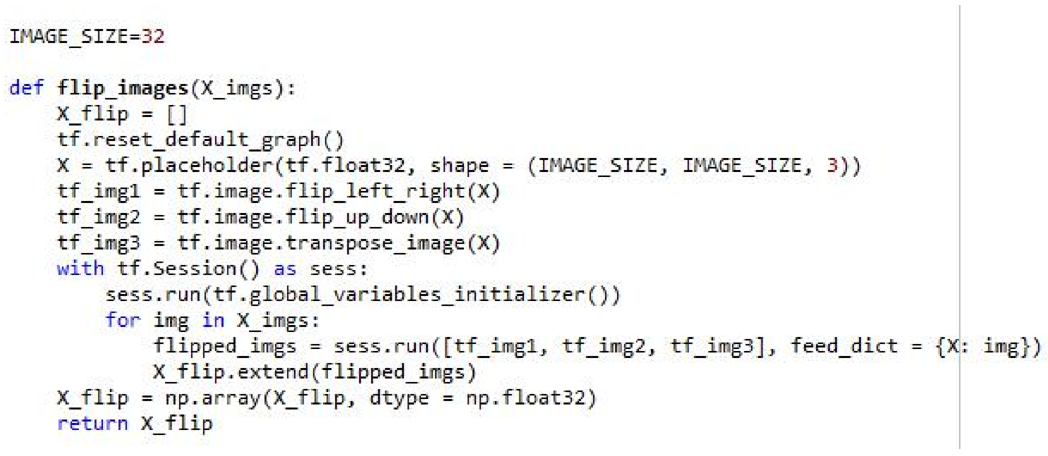
The normalization layer added

Normalize function to normalize the values between 0 and 1



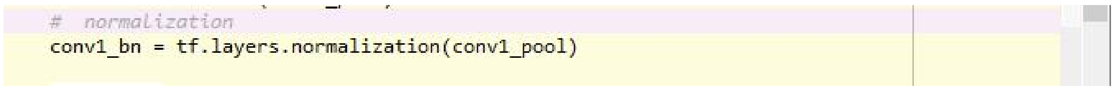
```


def normalize(x):
    """
        argument
            - x: input image data in numpy array [32, 32, 3]
        return
            - normalized x
    """
    min_val = np.min(x)
    max_val = np.max(x)
    x = (x-min_val) / (max_val-min_val)
    return x


# #### We perform one-hot encoding to the labels.
```

data augmentation



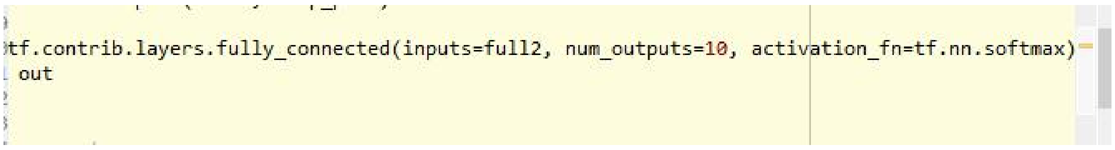
```
IMAGE_SIZE=32

def flip_images(X_imgs):
    X_flip = []
    tf.reset_default_graph()
    X = tf.placeholder(tf.float32, shape = (IMAGE_SIZE, IMAGE_SIZE, 3))
    tf_img1 = tf.image.flip_left_right(X)
    tf_img2 = tf.image.flip_up_down(X)
    tf_img3 = tf.image.transpose_image(X)
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for img in X_imgs:
            flipped_imgs = sess.run([tf_img1, tf_img2, tf_img3], feed_dict = {X: img})
            X_flip.extend(flipped_imgs)
    X_flip = np.array(X_flip, dtype = np.float32)
    return X_flip
```

normalization layer



```
#  normalization
conv1_bn = tf.layers.normalization(conv1_pool)
```

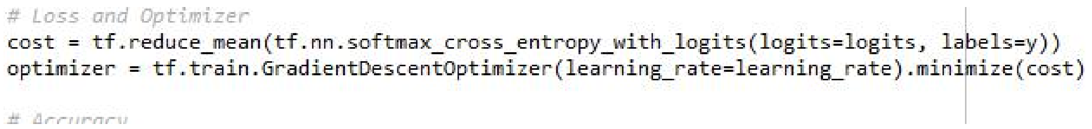
Activation function using softmax



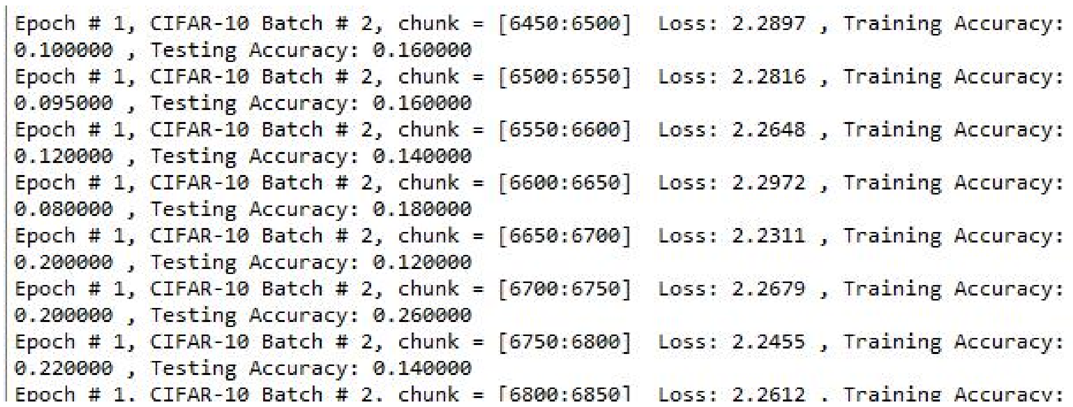
```
tf.contrib.layers.fully_connected(inputs=full2, num_outputs=10, activation_fn=tf.nn.softmax)
 out
```

.GradientDescentOptimizer

We initialize our CNN, define the optimizer and loss function and also accuracy

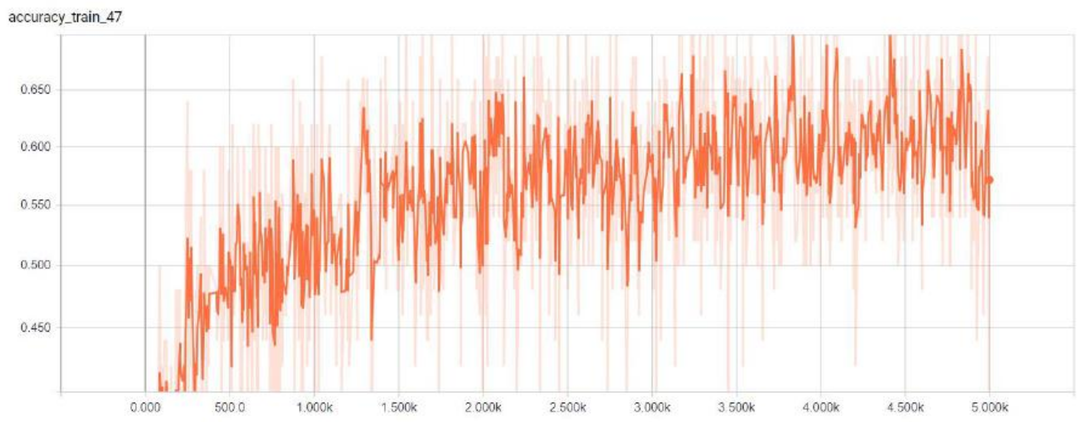


```
# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy
```

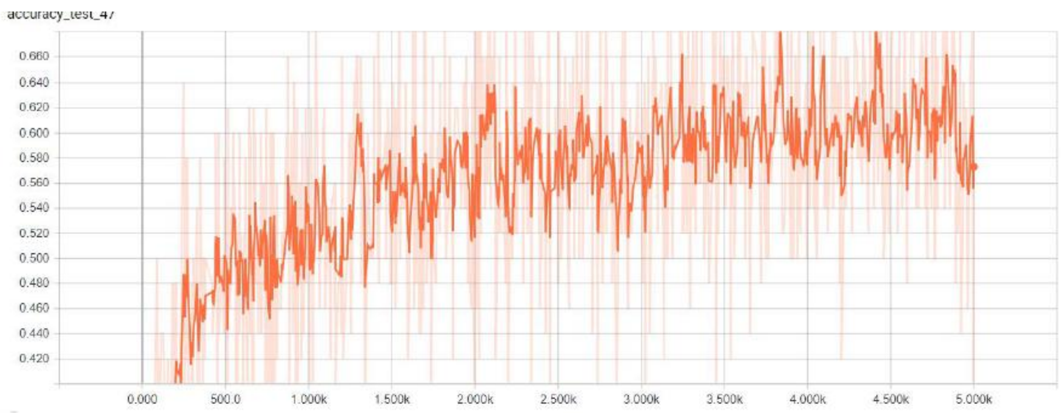


```
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6450:6500]  Loss: 2.2897 , Training Accuracy:
0.100000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6500:6550]  Loss: 2.2816 , Training Accuracy:
0.095000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6550:6600]  Loss: 2.2648 , Training Accuracy:
0.120000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6600:6650]  Loss: 2.2972 , Training Accuracy:
0.080000 , Testing Accuracy: 0.180000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6650:6700]  Loss: 2.2311 , Training Accuracy:
0.200000 , Testing Accuracy: 0.120000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6700:6750]  Loss: 2.2679 , Training Accuracy:
0.200000 , Testing Accuracy: 0.260000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6750:6800]  Loss: 2.2455 , Training Accuracy:
0.220000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6800:6850]  Loss: 2.2612 , Training Accuracy:
```

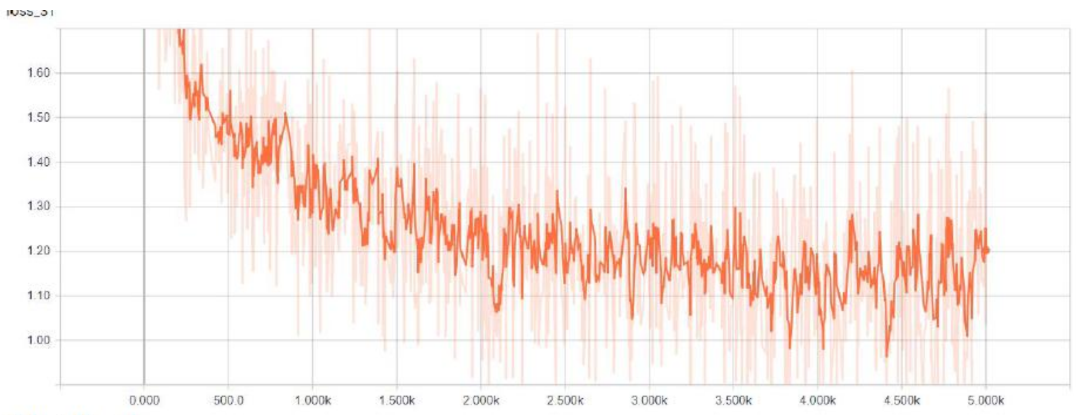
Accuracy_Train



Accuracy_Test

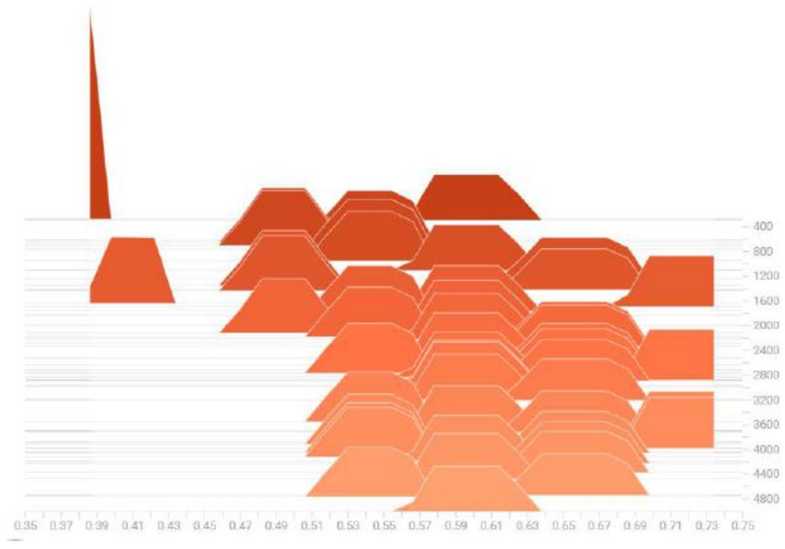


Accuracy_Loss

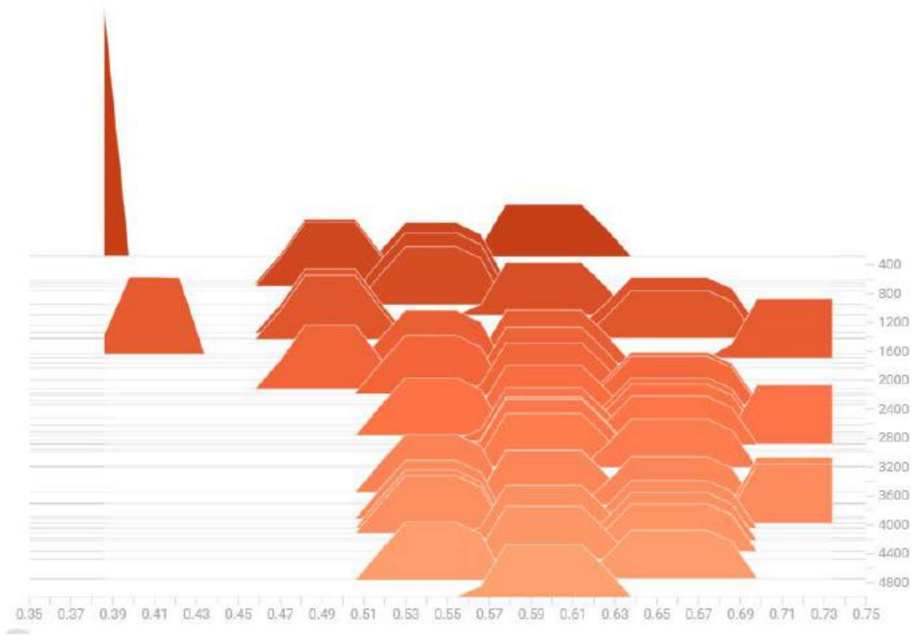


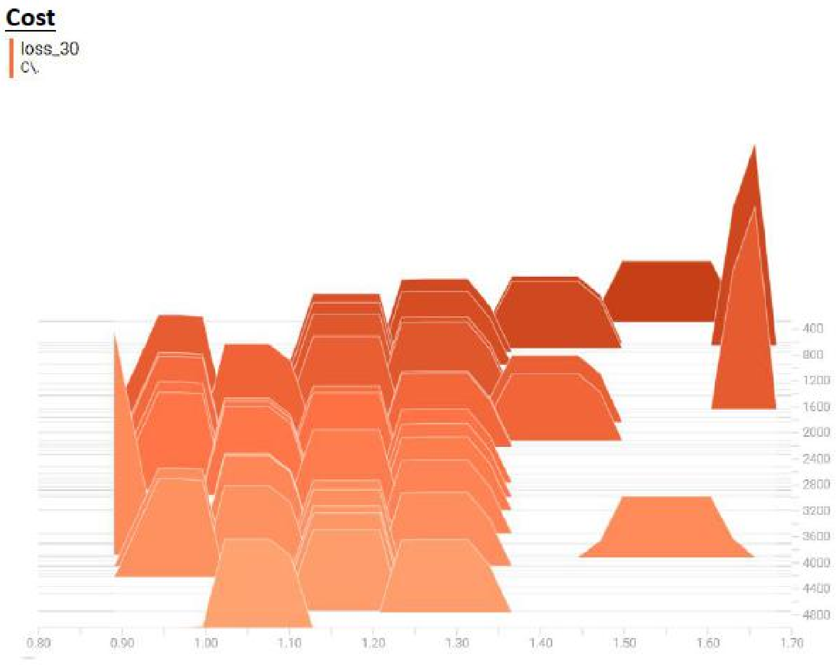
Histogram

Training_Accuracy

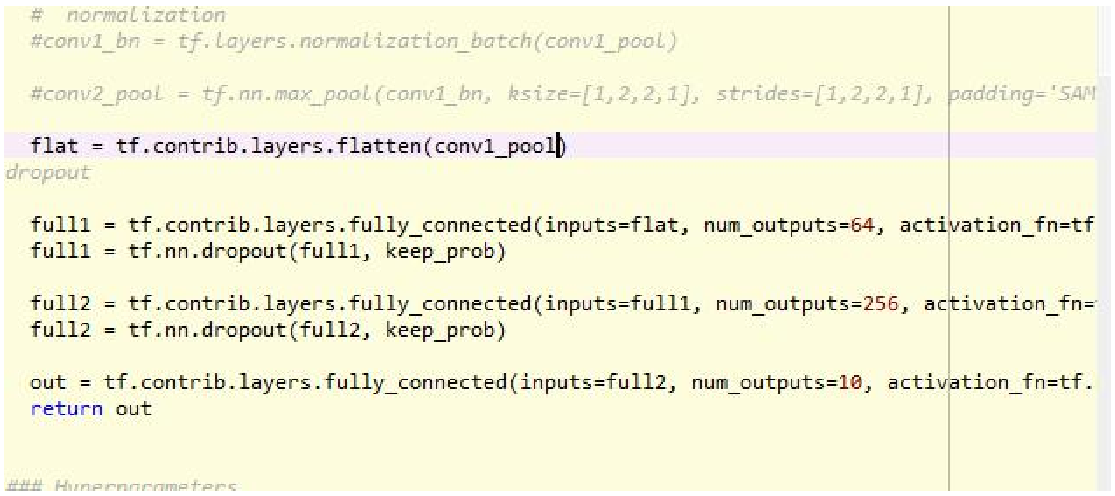


Test_Accuracy

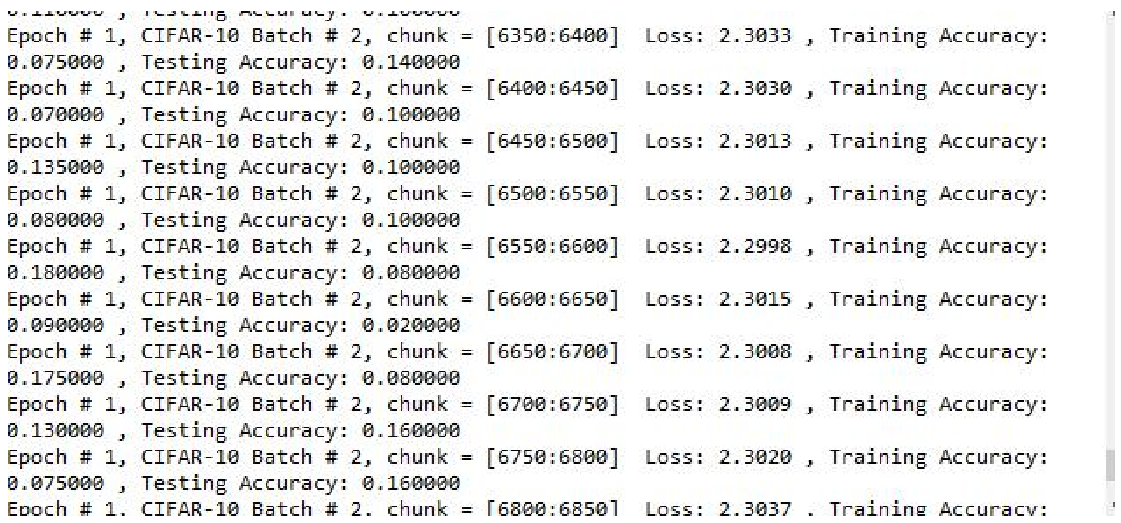




without normalization and report the accuracy.

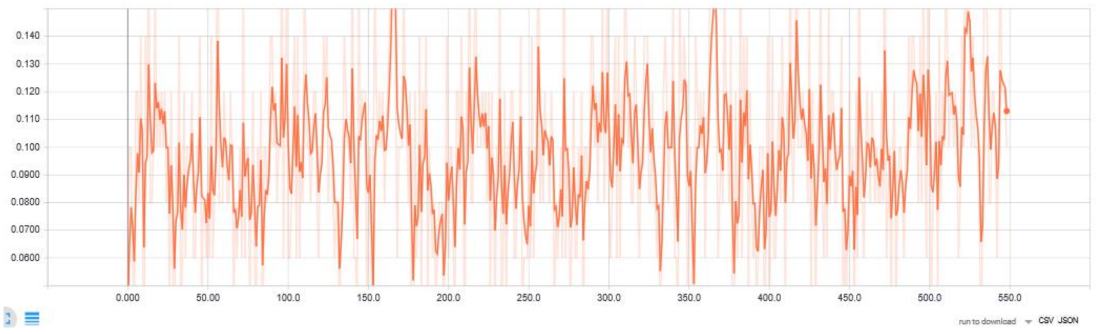


```
  #  normalization
  #conv1_bn = tf.layers.normalization_batch(conv1_pool)

  #conv2_pool = tf.nn.max_pool(conv1_bn, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAM

  flat = tf.contrib.layers.flatten(conv1_pool)
dropout

  full1 = tf.contrib.layers.fully_connected(inputs=flat, num_outputs=64, activation_fn=tf
  full1 = tf.nn.dropout(full1, keep_prob)

  full2 = tf.contrib.layers.fully_connected(inputs=full1, num_outputs=256, activation_fn=
  full2 = tf.nn.dropout(full2, keep_prob)

  out = tf.contrib.layers.fully_connected(inputs=full2, num_outputs=10, activation_fn=tf.
  return out


### Hyperparameters
```

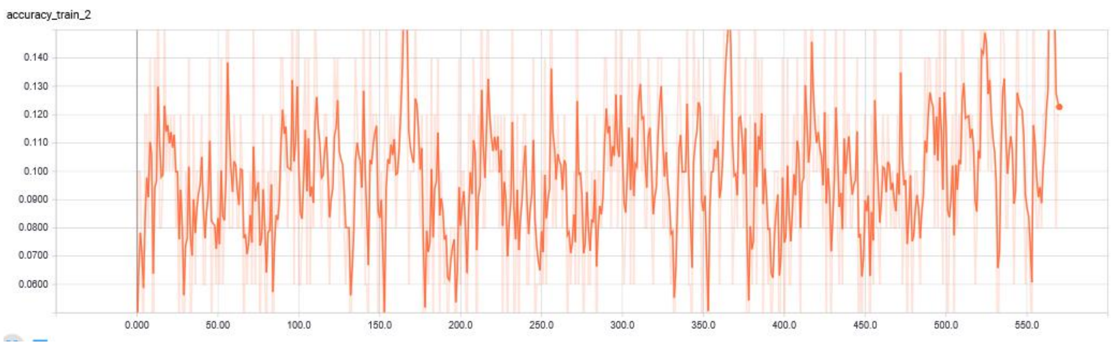


```
0.110000 , Testing Accuracy: 0.100000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6350:6400]  Loss: 2.3033 , Training Accuracy:
0.075000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6400:6450]  Loss: 2.3030 , Training Accuracy:
0.070000 , Testing Accuracy: 0.100000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6450:6500]  Loss: 2.3013 , Training Accuracy:
0.135000 , Testing Accuracy: 0.100000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6500:6550]  Loss: 2.3010 , Training Accuracy:
0.080000 , Testing Accuracy: 0.100000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6550:6600]  Loss: 2.2998 , Training Accuracy:
0.180000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6600:6650]  Loss: 2.3015 , Training Accuracy:
0.090000 , Testing Accuracy: 0.020000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6650:6700]  Loss: 2.3008 , Training Accuracy:
0.175000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6700:6750]  Loss: 2.3009 , Training Accuracy:
0.130000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6750:6800]  Loss: 2.3020 , Training Accuracy:
0.075000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6800:6850]  Loss: 2.3037 , Training Accuracy:
```

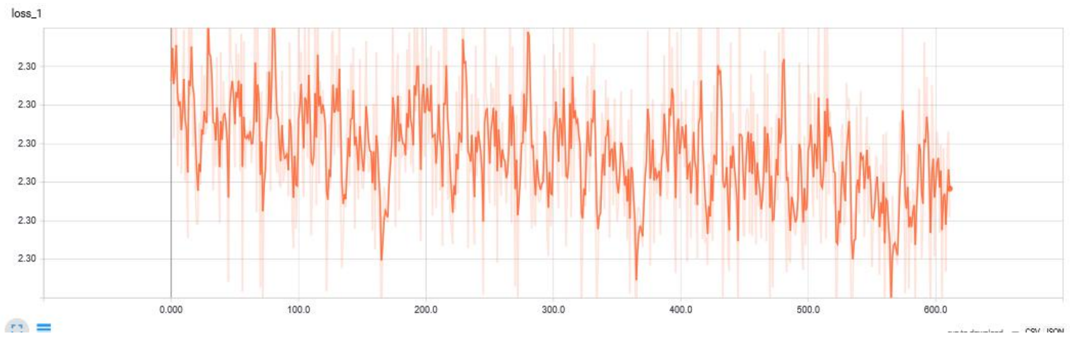
<u>Accuracy_Test</u>



Accuracy_Train



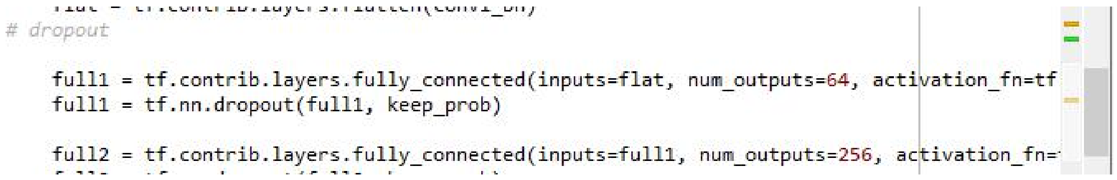
<u>Accuracy_Loss</u>



Exercise 2: Network Regularization

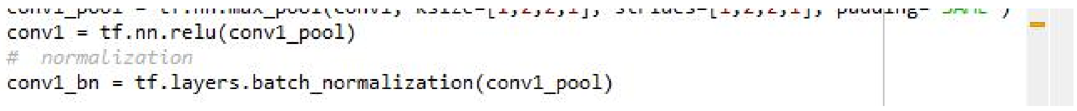
You have to compare both the solutions with and without dropout regularization

dropout regularization.



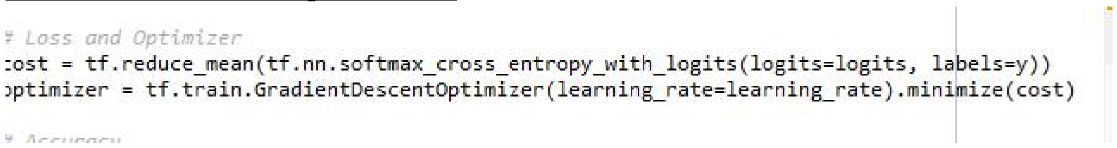
```
    flat = tf.contrib.layers.flatten(conv1_bn)
# dropout

    full1 = tf.contrib.layers.fully_connected(inputs=flat, num_outputs=64, activation_fn=tf
    full1 = tf.nn.dropout(full1, keep_prob)

    full2 = tf.contrib.layers.fully_connected(inputs=full1, num_outputs=256, activation_fn=
```

batch normalization layers



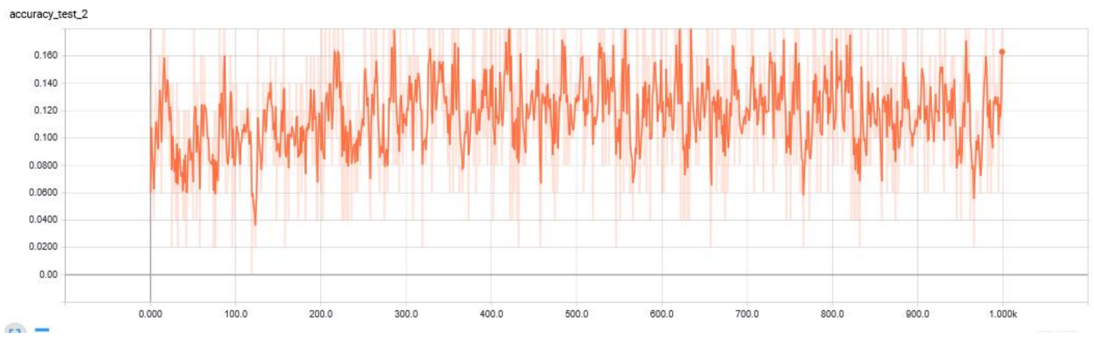
```
    conv1_pool = tf.nn.max_pool(conv1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    conv1 = tf.nn.relu(conv1_pool)
    #  normalization
    conv1_bn = tf.layers.batch_normalization(conv1_pool)
```

GradientDescentOptimizer

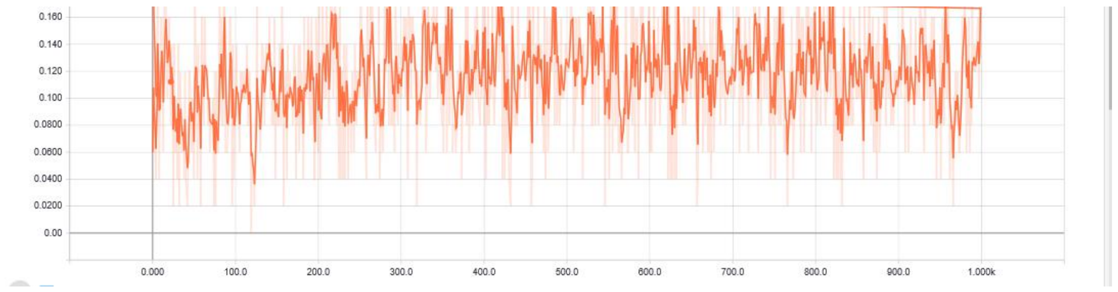


```
# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)
```

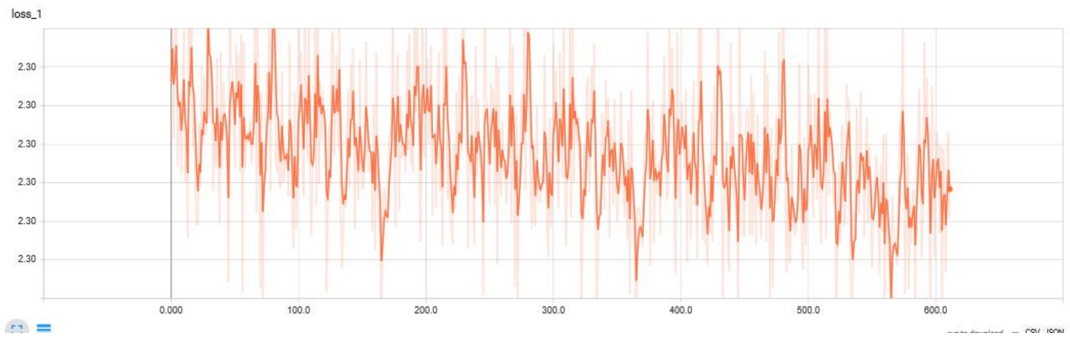
Accuracy_Test

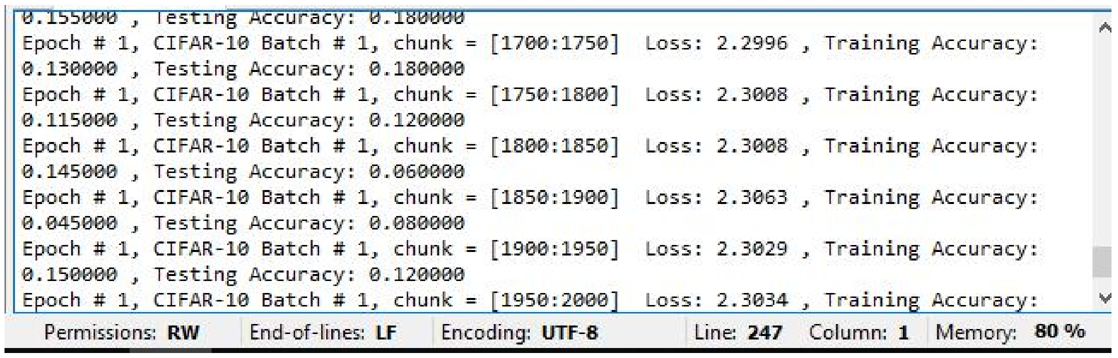


Accuracy_Training



Accuracy_Loss

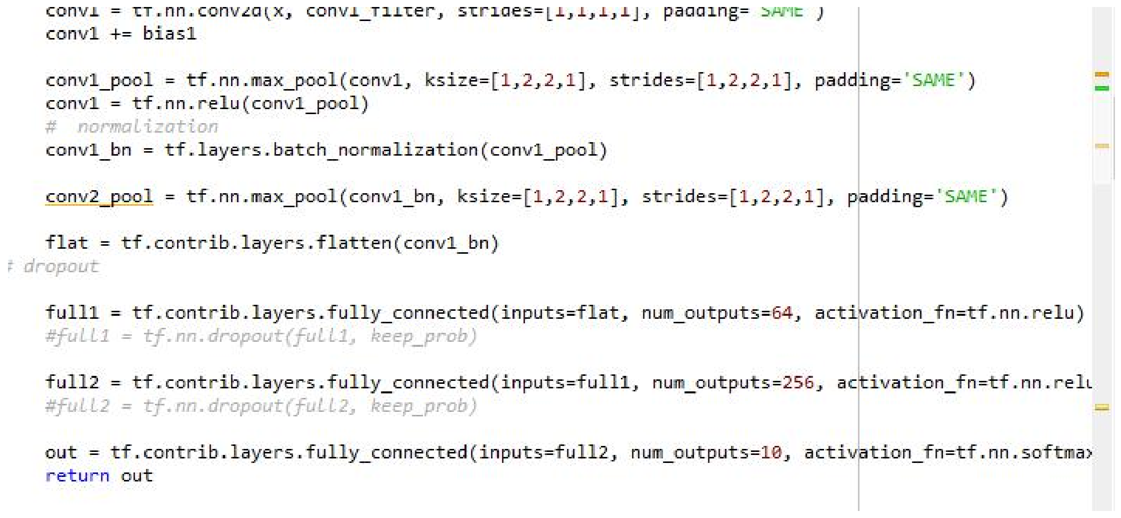




```
0.155000 , Testing Accuracy: 0.180000
Epoch # 1, CIFAR-10 Batch # 1, chunk = [1700:1750]  Loss: 2.2996 , Training Accuracy:
0.130000 , Testing Accuracy: 0.180000
Epoch # 1, CIFAR-10 Batch # 1, chunk = [1750:1800]  Loss: 2.3008 , Training Accuracy:
0.115000 , Testing Accuracy: 0.120000
Epoch # 1, CIFAR-10 Batch # 1, chunk = [1800:1850]  Loss: 2.3008 , Training Accuracy:
0.145000 , Testing Accuracy: 0.060000
Epoch # 1, CIFAR-10 Batch # 1, chunk = [1850:1900]  Loss: 2.3063 , Training Accuracy:
0.045000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 1, chunk = [1900:1950]  Loss: 2.3029 , Training Accuracy:
0.150000 , Testing Accuracy: 0.120000
Epoch # 1, CIFAR-10 Batch # 1, chunk = [1950:2000]  Loss: 2.3034 , Training Accuracy:
```

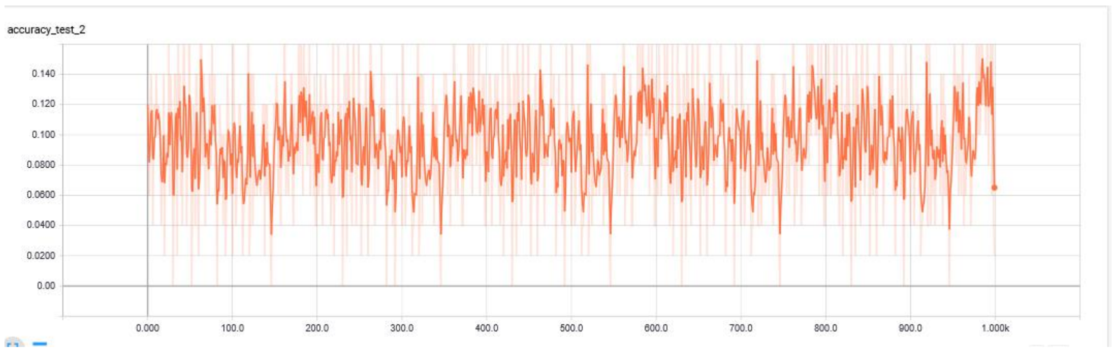
Permissions: **RW** | End-of-lines: **LF** | Encoding: **UTF-8** | Line: **247** | Column: **1** | Memory: **80 %**

without dropout regularization.

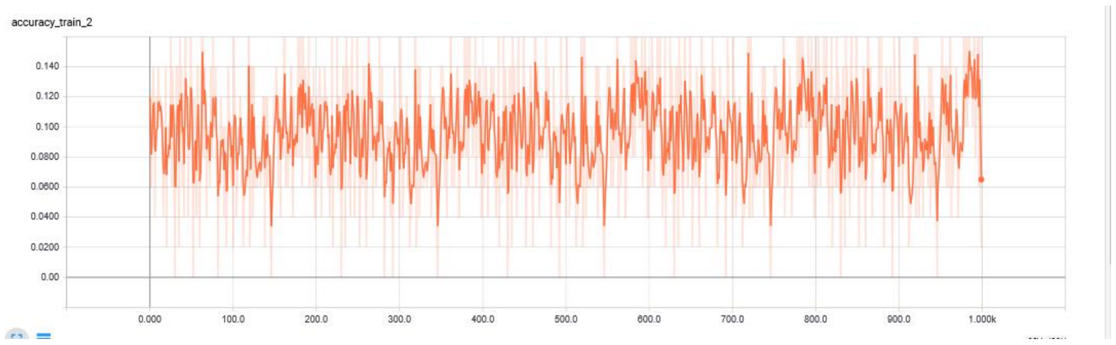


```
    conv1 = tf.nn.conv2d(x, conv1_filter, strides=[1,1,1,1], padding='SAME')
    conv1 += bias1

    conv1_pool = tf.nn.max_pool(conv1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    conv1 = tf.nn.relu(conv1_pool)
    #  normalization
    conv1_bn = tf.layers.batch_normalization(conv1_pool)

    conv2_pool = tf.nn.max_pool(conv1_bn, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

    flat = tf.contrib.layers.flatten(conv1_bn)
# dropout

    full1 = tf.contrib.layers.fully_connected(inputs=flat, num_outputs=64, activation_fn=tf.nn.relu)
    #full1 = tf.nn.dropout(full1, keep_prob)

    full2 = tf.contrib.layers.fully_connected(inputs=full1, num_outputs=256, activation_fn=tf.nn.relu
    #full2 = tf.nn.dropout(full2, keep_prob)

    out = tf.contrib.layers.fully_connected(inputs=full2, num_outputs=10, activation_fn=tf.nn.softmax
    return out
```

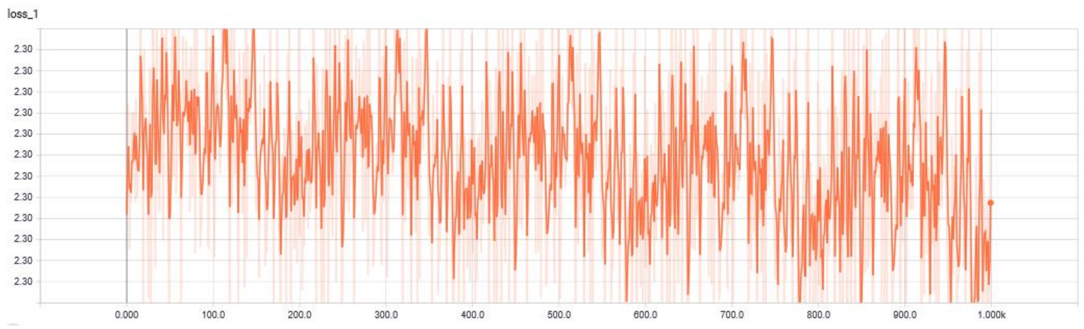
<u>Accuracy_Test</u>

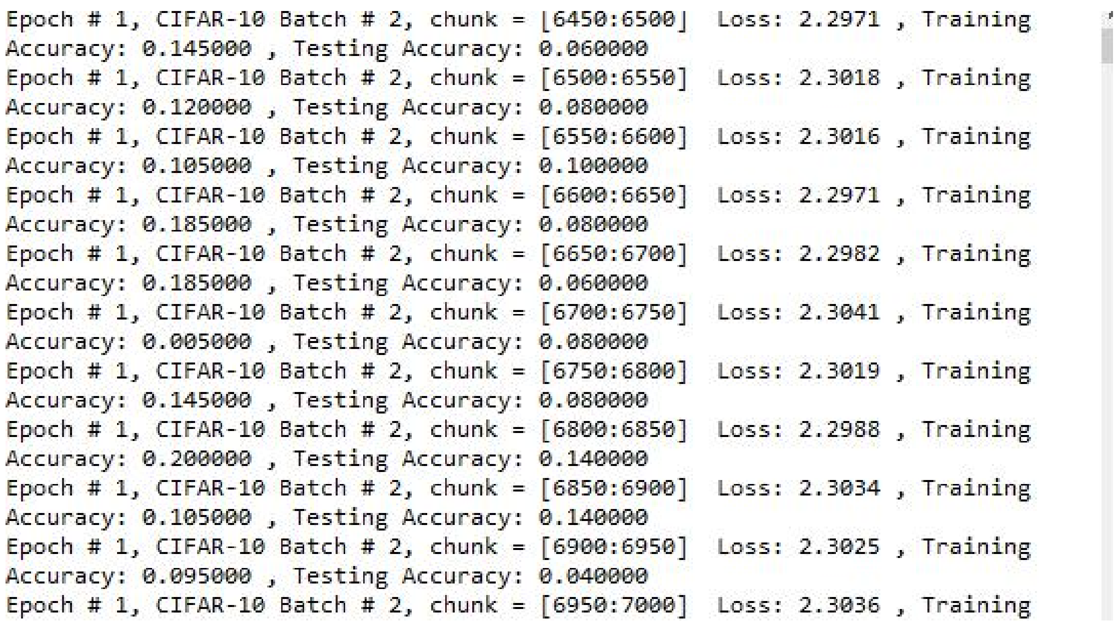


<u>Accuracy_Training</u>



Accuracy_Loss



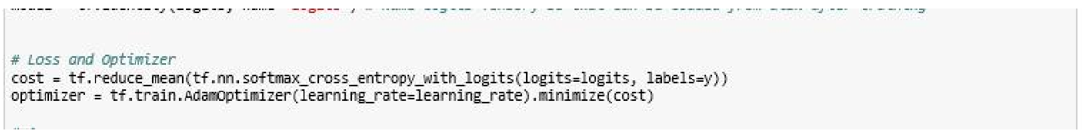


```
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6450:6500]  Loss: 2.2971 , Training
Accuracy: 0.145000 , Testing Accuracy: 0.060000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6500:6550]  Loss: 2.3018 , Training
Accuracy: 0.120000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6550:6600]  Loss: 2.3016 , Training
Accuracy: 0.105000 , Testing Accuracy: 0.100000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6600:6650]  Loss: 2.2971 , Training
Accuracy: 0.185000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6650:6700]  Loss: 2.2982 , Training
Accuracy: 0.185000 , Testing Accuracy: 0.060000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6700:6750]  Loss: 2.3041 , Training
Accuracy: 0.005000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6750:6800]  Loss: 2.3019 , Training
Accuracy: 0.145000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6800:6850]  Loss: 2.2988 , Training
Accuracy: 0.200000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6850:6900]  Loss: 2.3034 , Training
Accuracy: 0.105000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6900:6950]  Loss: 2.3025 , Training
Accuracy: 0.095000 , Testing Accuracy: 0.040000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [6950:7000]  Loss: 2.3036 , Training
```

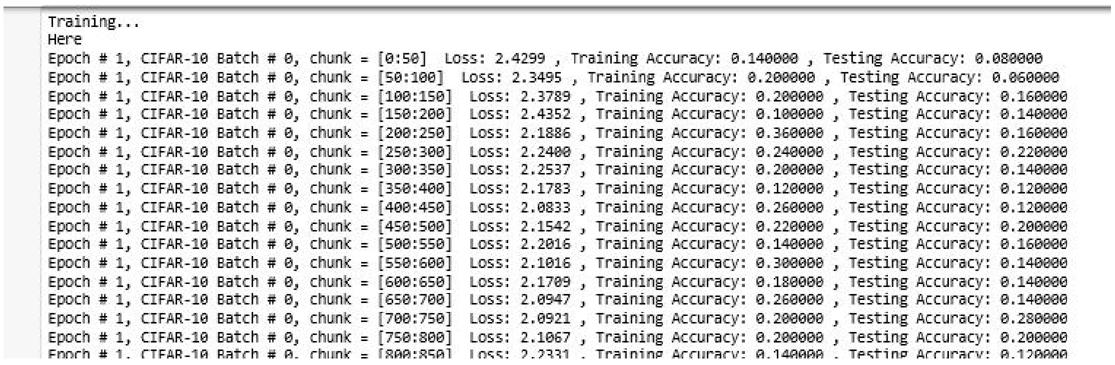
Exercise 3: Optimizers (CNN)

you will use RMSPropOptimizer AdamOptimizer for training

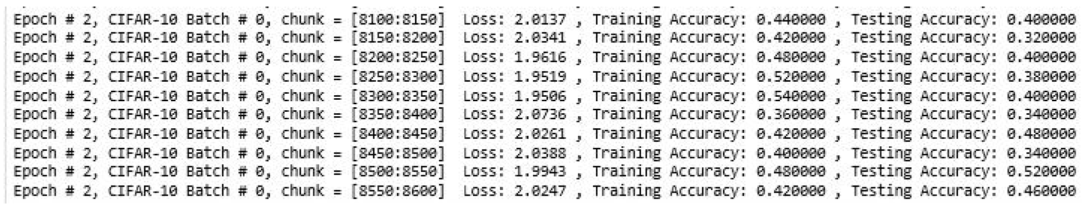
AdamOptimizer



```
# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

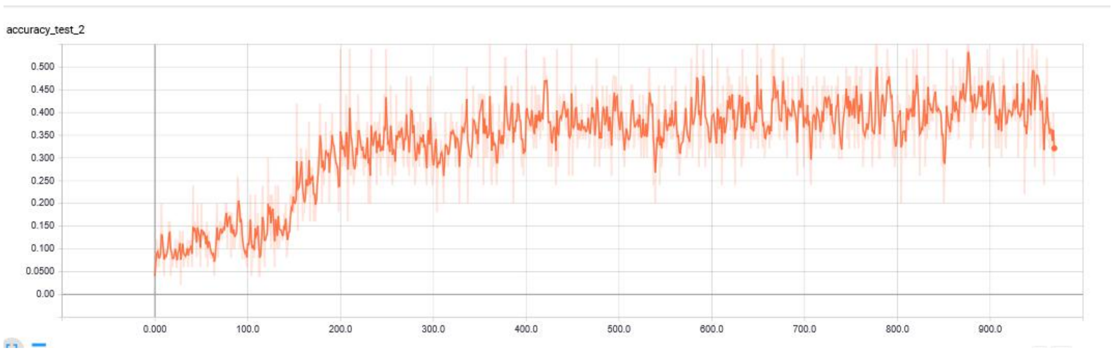


```
Training...
Here
Epoch # 1, CIFAR-10 Batch # 0, chunk = [0:50]  Loss: 2.4299 , Training Accuracy: 0.140000 , Testing Accuracy: 0.080000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [50:100]  Loss: 2.3495 , Training Accuracy: 0.200000 , Testing Accuracy: 0.060000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [100:150]  Loss: 2.3789 , Training Accuracy: 0.200000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [150:200]  Loss: 2.4352 , Training Accuracy: 0.100000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [200:250]  Loss: 2.1886 , Training Accuracy: 0.360000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [250:300]  Loss: 2.2400 , Training Accuracy: 0.240000 , Testing Accuracy: 0.220000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [300:350]  Loss: 2.2537 , Training Accuracy: 0.200000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [350:400]  Loss: 2.1783 , Training Accuracy: 0.120000 , Testing Accuracy: 0.120000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [400:450]  Loss: 2.0833 , Training Accuracy: 0.260000 , Testing Accuracy: 0.120000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [450:500]  Loss: 2.1542 , Training Accuracy: 0.220000 , Testing Accuracy: 0.200000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [500:550]  Loss: 2.2016 , Training Accuracy: 0.140000 , Testing Accuracy: 0.160000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [550:600]  Loss: 2.1016 , Training Accuracy: 0.300000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [600:650]  Loss: 2.1709 , Training Accuracy: 0.180000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [650:700]  Loss: 2.0947 , Training Accuracy: 0.260000 , Testing Accuracy: 0.140000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [700:750]  Loss: 2.0921 , Training Accuracy: 0.200000 , Testing Accuracy: 0.280000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [750:800]  Loss: 2.1067 , Training Accuracy: 0.200000 , Testing Accuracy: 0.200000
Epoch # 1, CIFAR-10 Batch # 0, chunk = [800:850]  Loss: 2.2331 , Training Accuracy: 0.140000 , Testing Accuracy: 0.120000
```

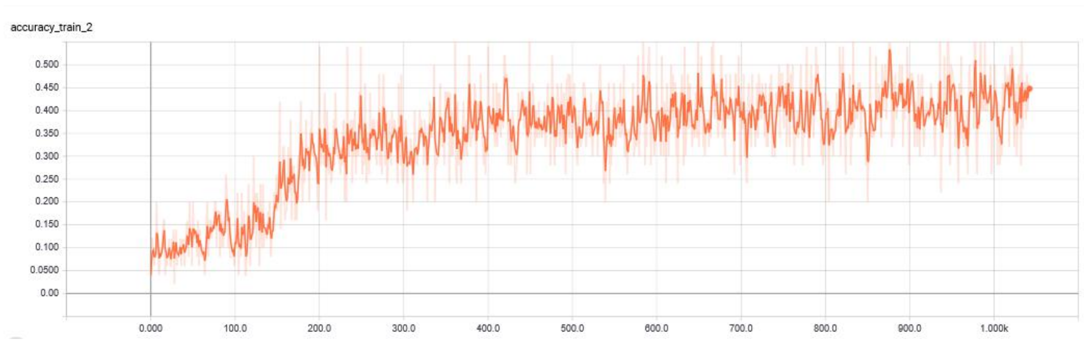


```
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8100:8150]  Loss: 2.0137 , Training Accuracy: 0.440000 , Testing Accuracy: 0.400000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8150:8200]  Loss: 2.0341 , Training Accuracy: 0.420000 , Testing Accuracy: 0.320000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8200:8250]  Loss: 1.9616 , Training Accuracy: 0.480000 , Testing Accuracy: 0.400000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8250:8300]  Loss: 1.9519 , Training Accuracy: 0.520000 , Testing Accuracy: 0.380000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8300:8350]  Loss: 1.9506 , Training Accuracy: 0.540000 , Testing Accuracy: 0.400000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8350:8400]  Loss: 2.0736 , Training Accuracy: 0.360000 , Testing Accuracy: 0.340000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8400:8450]  Loss: 2.0261 , Training Accuracy: 0.420000 , Testing Accuracy: 0.480000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8450:8500]  Loss: 2.0388 , Training Accuracy: 0.400000 , Testing Accuracy: 0.340000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8500:8550]  Loss: 1.9943 , Training Accuracy: 0.480000 , Testing Accuracy: 0.520000
Epoch # 2, CIFAR-10 Batch # 0, chunk = [8550:8600]  Loss: 2.0247 , Training Accuracy: 0.420000 , Testing Accuracy: 0.460000
```

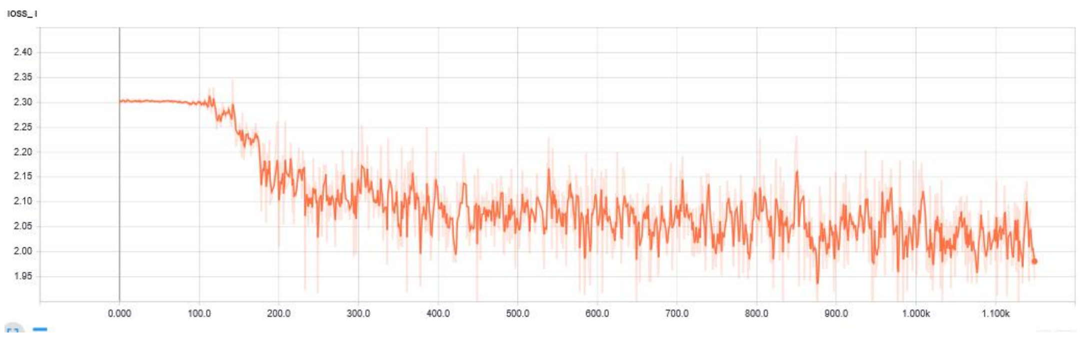
<u>Accuracy_Test</u>



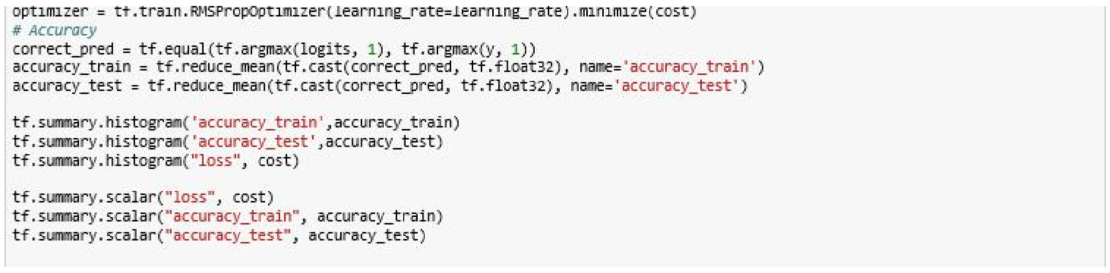
<u>Accuracy_Training</u>



<u>Accuracy_Loss</u>

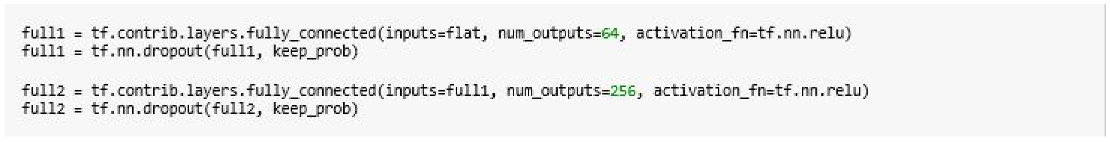


<u>RMSPropOptimizer</u>



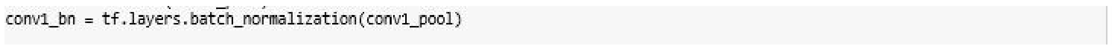
```
optimizer = tf.train.RMSPropOptimizer(learning_rate=learning_rate).minimize(cost)
# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy_train = tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy_train')
accuracy_test = tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy_test')

tf.summary.histogram('accuracy_train',accuracy_train)
tf.summary.histogram('accuracy_test',accuracy_test)
tf.summary.histogram("loss", cost)

tf.summary.scalar("loss", cost)
tf.summary.scalar("accuracy_train", accuracy_train)
tf.summary.scalar("accuracy_test", accuracy_test)
```

<u>used dropout Technique</u>

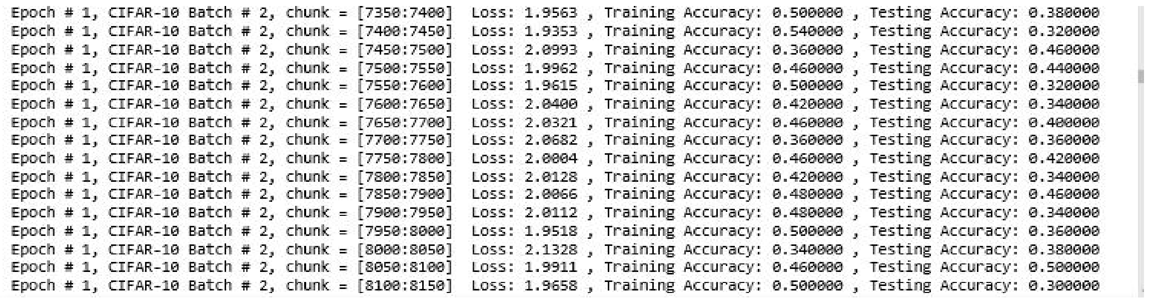


```
full1 = tf.contrib.layers.fully_connected(inputs=flat, num_outputs=64, activation_fn=tf.nn.relu)
full1 = tf.nn.dropout(full1, keep_prob)

full2 = tf.contrib.layers.fully_connected(inputs=full1, num_outputs=256, activation_fn=tf.nn.relu)
full2 = tf.nn.dropout(full2, keep_prob)
```

<u>batch normalization layer</u>

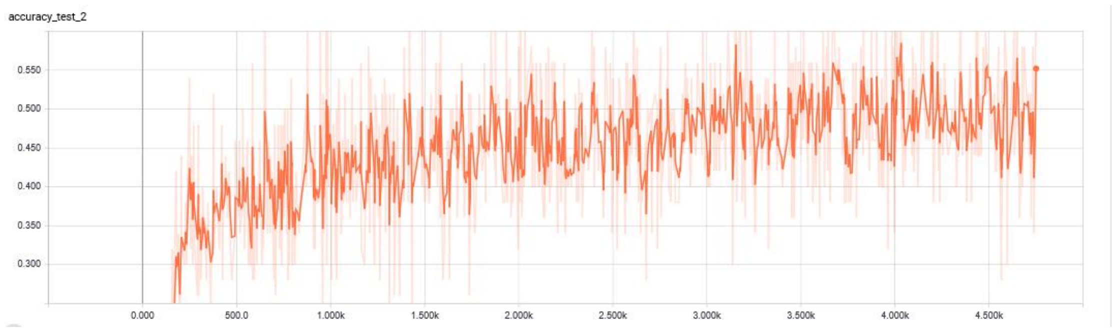


```
conv1_bn = tf.layers.batch_normalization(conv1_pool)
```

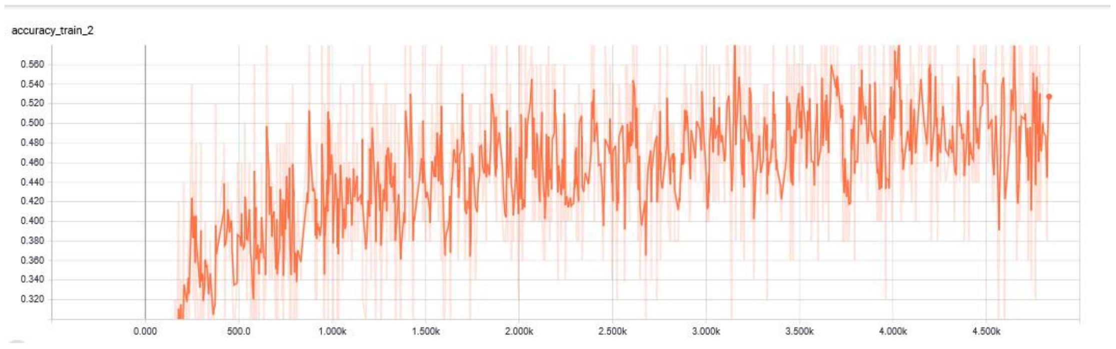


```
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7350:7400]  Loss: 1.9563 , Training Accuracy: 0.500000 , Testing Accuracy: 0.380000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7400:7450]  Loss: 1.9353 , Training Accuracy: 0.540000 , Testing Accuracy: 0.320000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7450:7500]  Loss: 2.0993 , Training Accuracy: 0.360000 , Testing Accuracy: 0.460000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7500:7550]  Loss: 1.9962 , Training Accuracy: 0.460000 , Testing Accuracy: 0.440000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7550:7600]  Loss: 1.9615 , Training Accuracy: 0.500000 , Testing Accuracy: 0.320000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7600:7650]  Loss: 2.0400 , Training Accuracy: 0.420000 , Testing Accuracy: 0.340000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7650:7700]  Loss: 2.0321 , Training Accuracy: 0.460000 , Testing Accuracy: 0.400000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7700:7750]  Loss: 2.0682 , Training Accuracy: 0.360000 , Testing Accuracy: 0.360000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7750:7800]  Loss: 2.0004 , Training Accuracy: 0.460000 , Testing Accuracy: 0.420000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7800:7850]  Loss: 2.0128 , Training Accuracy: 0.420000 , Testing Accuracy: 0.340000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7850:7900]  Loss: 2.0066 , Training Accuracy: 0.480000 , Testing Accuracy: 0.460000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7900:7950]  Loss: 2.0112 , Training Accuracy: 0.480000 , Testing Accuracy: 0.340000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [7950:8000]  Loss: 1.9518 , Training Accuracy: 0.500000 , Testing Accuracy: 0.360000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [8000:8050]  Loss: 2.1328 , Training Accuracy: 0.340000 , Testing Accuracy: 0.380000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [8050:8100]  Loss: 1.9911 , Training Accuracy: 0.460000 , Testing Accuracy: 0.500000
Epoch # 1, CIFAR-10 Batch # 2, chunk = [8100:8150]  Loss: 1.9658 , Training Accuracy: 0.500000 , Testing Accuracy: 0.300000
```

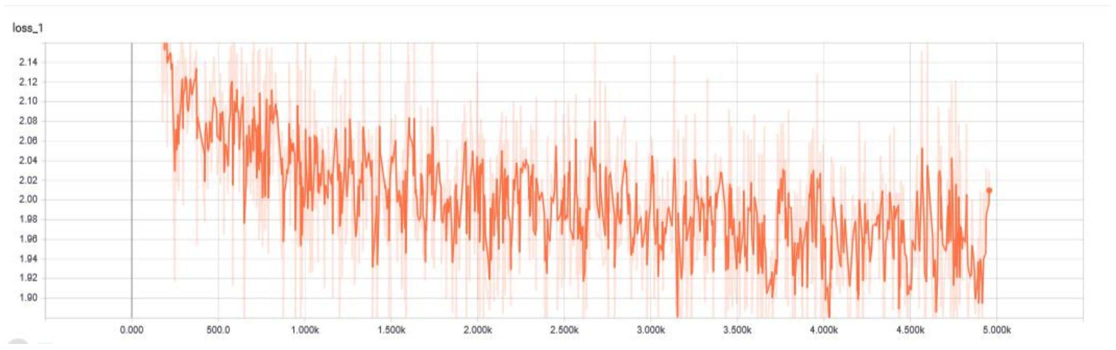
Accuracy_Test



Accuracy_Training



Accuracy_Loss


loss_1
1.000k
1.500k
2.000k
2.500k
3.000k
3.500k
4.000k
4.500k
5.000k