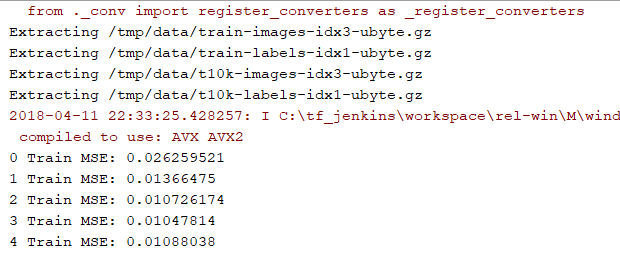
**Deep Learning Assignment – 13**

**Kishan Sarpangala**

**M04307966**

**Output Screenshots**:

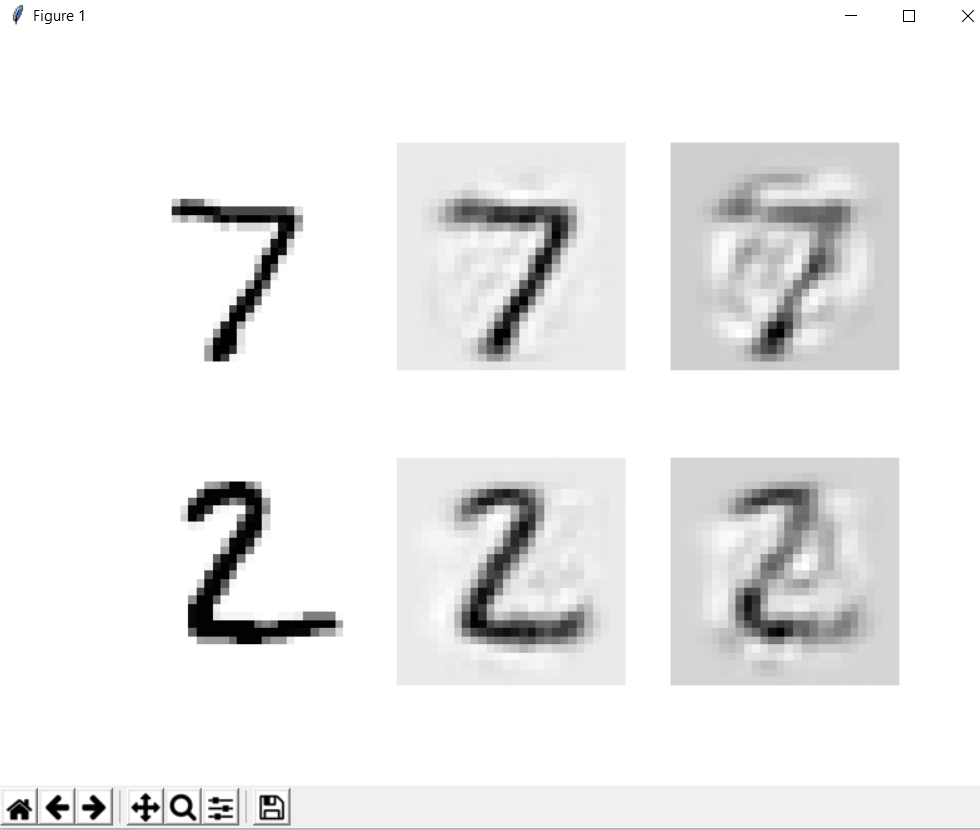
1. Autoencoder1.py:



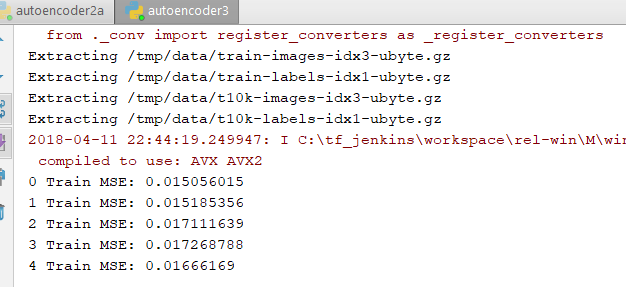


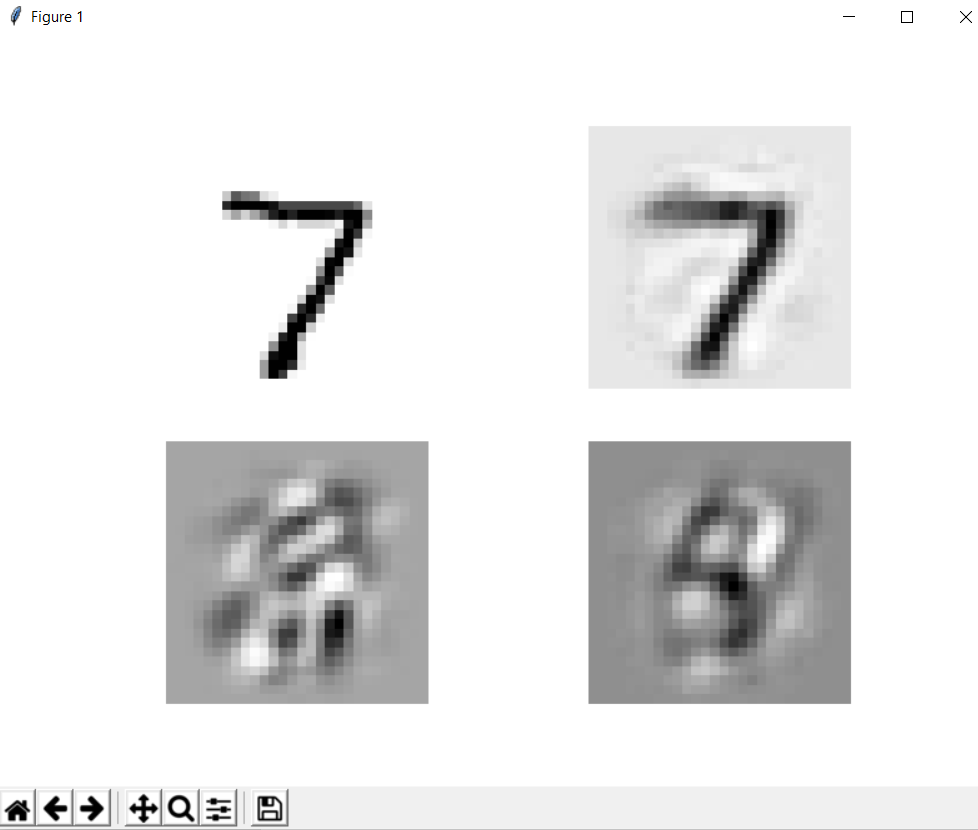
1. Autoencoder2.py:



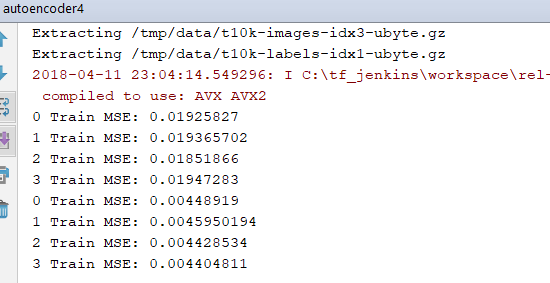


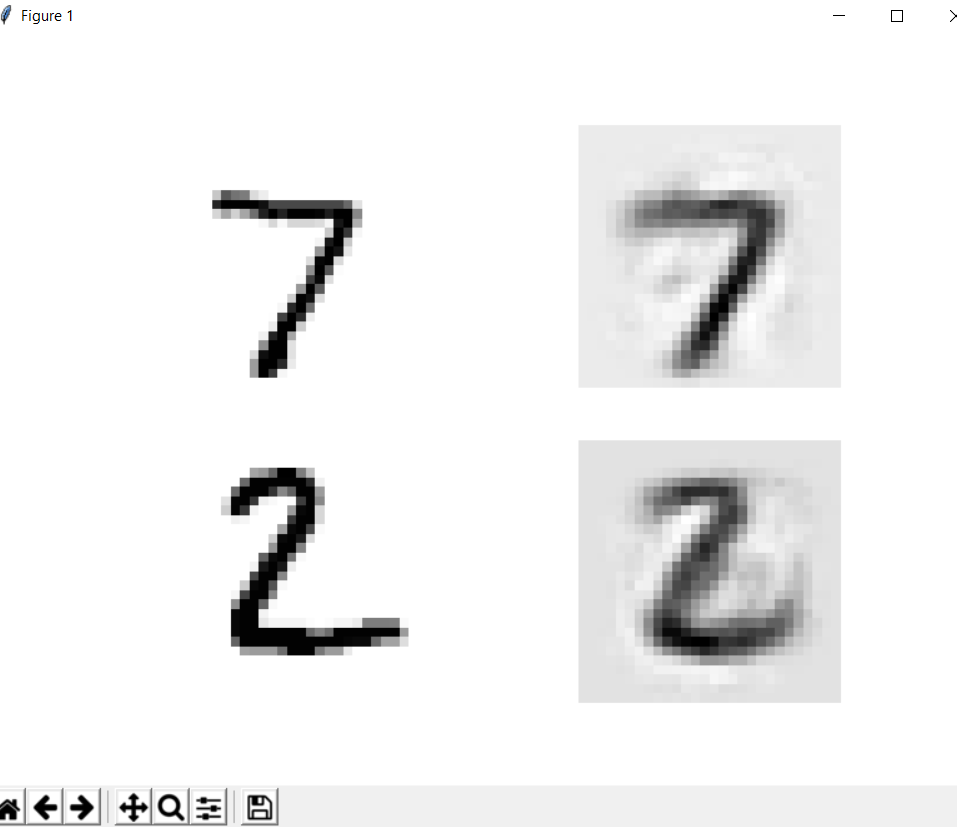
1. Autoencoder3.py:





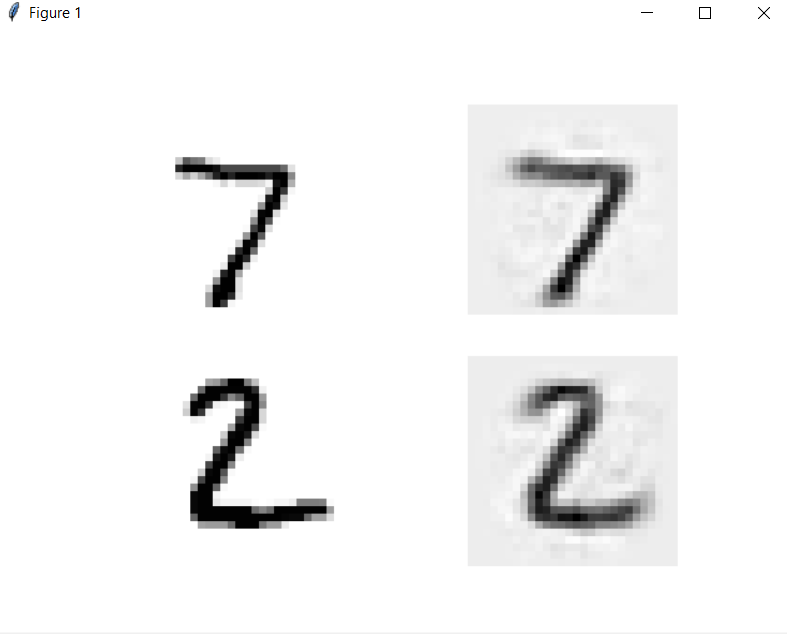
1. autoencoder4.py:





1. Autoencoder5.py:





**Code for autoencoder4.py (to plot images)**

*# Training one autoencoder at a time by Geron  
# need visualization of reconstruction***import** tensorflow **as** tf  
**import** sys  
**import** numpy.random **as** rnd  
**from** functools **import** partial  
  
*#added by Kishan Sarpangala to plot images***import** matplotlib  
**import** matplotlib.pyplot **as** plt  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
mnist = input\_data.read\_data\_sets(**"/tmp/data/"**)  
  
  
**from** functools **import** partial  
  
  
**def** train\_autoencoder(X\_train, n\_neurons, n\_epochs, batch\_size,  
 learning\_rate = 0.01, l2\_reg = 0.0005,  
 hidden\_activation=tf.nn.elu,  
 output\_activation=tf.nn.elu):  
 graph = tf.Graph()  
 **with** graph.as\_default():  
 n\_inputs = X\_train.shape[1]  
  
 X = tf.placeholder(tf.float32, shape=[**None**, n\_inputs])  
   
 my\_dense\_layer = partial(  
 tf.layers.dense,  
 kernel\_initializer=tf.contrib.layers.variance\_scaling\_initializer(),  
 kernel\_regularizer=tf.contrib.layers.l2\_regularizer(l2\_reg))  
  
 hidden = my\_dense\_layer(X, n\_neurons, activation=hidden\_activation, name=**"hidden"**)  
 outputs = my\_dense\_layer(hidden, n\_inputs, activation=output\_activation, name=**"outputs"**)  
  
 reconstruction\_loss = tf.reduce\_mean(tf.square(outputs - X))  
  
 reg\_losses = tf.get\_collection(tf.GraphKeys.REGULARIZATION\_LOSSES)  
 loss = tf.add\_n([reconstruction\_loss] + reg\_losses)  
  
 optimizer = tf.train.AdamOptimizer(learning\_rate)  
 training\_op = optimizer.minimize(loss)  
  
 init = tf.global\_variables\_initializer()  
  
 **with** tf.Session(graph=graph) **as** sess:  
 init.run()  
 **for** epoch **in** range(n\_epochs):  
 n\_batches = len(X\_train) // batch\_size  
 **for** iteration **in** range(n\_batches):  
 print(**"\r{}%"**.format(100 \* iteration // n\_batches), end=**""**)  
 sys.stdout.flush()  
 indices = rnd.permutation(len(X\_train))[:batch\_size]  
 X\_batch = X\_train[indices]  
 sess.run(training\_op, feed\_dict={X: X\_batch})  
 loss\_train = reconstruction\_loss.eval(feed\_dict={X: X\_batch})  
 print(**"\r{}"**.format(epoch), **"Train MSE:"**, loss\_train)  
 params = dict([(var.name, var.eval()) **for** var **in** tf.get\_collection(tf.GraphKeys.TRAINABLE\_VARIABLES)])  
 hidden\_val = hidden.eval(feed\_dict={X: X\_train})  
 **return** hidden\_val, params[**"hidden/kernel:0"**], params[**"hidden/bias:0"**], params[**"outputs/kernel:0"**], params[**"outputs/bias:0"**]  
  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
mnist = input\_data.read\_data\_sets(**"/tmp/data/"**)  
  
hidden\_output, W1, b1, W4, b4 = train\_autoencoder(mnist.train.images, n\_neurons=300, n\_epochs=4, batch\_size=150,  
 output\_activation=**None**)  
\_, W2, b2, W3, b3 = train\_autoencoder(hidden\_output, n\_neurons=150, n\_epochs=4, batch\_size=150)  
  
tf.reset\_default\_graph()  
  
n\_inputs = 28\*28  
  
*#added by Kishan Sarpangala to plot images*X = tf.placeholder(tf.float32, shape=[**None**, n\_inputs])  
hidden1 = tf.nn.elu(tf.matmul(X, W1) + b1)  
hidden2 = tf.nn.elu(tf.matmul(hidden1, W2) + b2)  
hidden3 = tf.nn.elu(tf.matmul(hidden2, W3) + b3)  
outputs = tf.matmul(hidden3, W4) + b4  
  
**def** plot\_image(image, shape=[28, 28]):  
 plt.imshow(image.reshape(shape), cmap=**"Greys"**, interpolation=**"nearest"**)  
 plt.axis(**"off"**)  
  
**def** show\_reconstructed\_digits(X, outputs, model\_path = **None**, n\_test\_digits = 2):  
 **with** tf.Session() **as** sess:  
 X\_test = mnist.test.images[:n\_test\_digits]  
 outputs\_val = outputs.eval(feed\_dict={X: X\_test})  
  
 fig = plt.figure(figsize=(8, 3 \* n\_test\_digits))  
 **for** digit\_index **in** range(n\_test\_digits):  
 plt.subplot(n\_test\_digits, 2, digit\_index \* 2 + 1)  
 plot\_image(X\_test[digit\_index])  
 plt.subplot(n\_test\_digits, 2, digit\_index \* 2 + 2)  
 plot\_image(outputs\_val[digit\_index])  
 plt.show()  
  
*#original code*show\_reconstructed\_digits(X, outputs)

**Code autoencoder5.py (to plot images)**

*# the single-graph approach from Geron  
# need visualization of results***import** tensorflow **as** tf  
*#Added (Kishan Sarpangala)***import** matplotlib  
**import** matplotlib.pyplot **as** plt  
  
n\_inputs = 28 \* 28  
n\_hidden1 = 300  
n\_hidden2 = 150 *# codings*n\_hidden3 = n\_hidden1  
n\_outputs = n\_inputs  
  
learning\_rate = 0.01  
l2\_reg = 0.0001  
  
activation = tf.nn.elu  
regularizer = tf.contrib.layers.l2\_regularizer(l2\_reg)  
initializer = tf.contrib.layers.variance\_scaling\_initializer()  
  
X = tf.placeholder(tf.float32, shape=[**None**, n\_inputs])  
  
weights1\_init = initializer([n\_inputs, n\_hidden1])  
weights2\_init = initializer([n\_hidden1, n\_hidden2])  
weights3\_init = initializer([n\_hidden2, n\_hidden3])  
weights4\_init = initializer([n\_hidden3, n\_outputs])  
  
weights1 = tf.Variable(weights1\_init, dtype=tf.float32, name=**"weights1"**)  
weights2 = tf.Variable(weights2\_init, dtype=tf.float32, name=**"weights2"**)  
weights3 = tf.Variable(weights3\_init, dtype=tf.float32, name=**"weights3"**)  
weights4 = tf.Variable(weights4\_init, dtype=tf.float32, name=**"weights4"**)  
  
biases1 = tf.Variable(tf.zeros(n\_hidden1), name=**"biases1"**)  
biases2 = tf.Variable(tf.zeros(n\_hidden2), name=**"biases2"**)  
biases3 = tf.Variable(tf.zeros(n\_hidden3), name=**"biases3"**)  
biases4 = tf.Variable(tf.zeros(n\_outputs), name=**"biases4"**)  
  
hidden1 = activation(tf.matmul(X, weights1) + biases1)  
hidden2 = activation(tf.matmul(hidden1, weights2) + biases2)  
hidden3 = activation(tf.matmul(hidden2, weights3) + biases3)  
outputs = tf.matmul(hidden3, weights4) + biases4  
  
reconstruction\_loss = tf.reduce\_mean(tf.square(outputs - X))  
  
optimizer = tf.train.AdamOptimizer(learning\_rate)  
  
**with** tf.name\_scope(**"phase1"**):  
 phase1\_outputs = tf.matmul(hidden1, weights4) + biases4 *# bypass hidden2 and hidden3* phase1\_reconstruction\_loss = tf.reduce\_mean(tf.square(phase1\_outputs - X))  
 phase1\_reg\_loss = regularizer(weights1) + regularizer(weights4)  
 phase1\_loss = phase1\_reconstruction\_loss + phase1\_reg\_loss  
 phase1\_training\_op = optimizer.minimize(phase1\_loss)  
  
**with** tf.name\_scope(**"phase2"**):  
 phase2\_reconstruction\_loss = tf.reduce\_mean(tf.square(hidden3 - hidden1))  
 phase2\_reg\_loss = regularizer(weights2) + regularizer(weights3)  
 phase2\_loss = phase2\_reconstruction\_loss + phase2\_reg\_loss  
 train\_vars = [weights2, biases2, weights3, biases3]  
 phase2\_training\_op = optimizer.minimize(phase2\_loss, var\_list=train\_vars) *# freeze hidden1*init = tf.global\_variables\_initializer()  
saver = tf.train.Saver()  
  
training\_ops = [phase1\_training\_op, phase2\_training\_op]  
reconstruction\_losses = [phase1\_reconstruction\_loss, phase2\_reconstruction\_loss]  
n\_epochs = [4, 4]  
batch\_sizes = [150, 150]  
  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
mnist = input\_data.read\_data\_sets(**"/tmp/data/"**)  
  
**import** sys  
  
**with** tf.Session() **as** sess:  
 init.run()  
 **for** phase **in** range(2):  
 print(**"Training phase #{}"**.format(phase + 1))  
 **for** epoch **in** range(n\_epochs[phase]):  
 n\_batches = mnist.train.num\_examples // batch\_sizes[phase]  
 **for** iteration **in** range(n\_batches):  
 print(**"\r{}%"**.format(100 \* iteration // n\_batches), end=**""**)  
 sys.stdout.flush()  
 X\_batch, y\_batch = mnist.train.next\_batch(batch\_sizes[phase])  
 sess.run(training\_ops[phase], feed\_dict={X: X\_batch})  
 loss\_train = reconstruction\_losses[phase].eval(feed\_dict={X: X\_batch})  
 print(**"\r{}"**.format(epoch), **"Train MSE:"**, loss\_train)  
 saver.save(sess, **"./my\_model\_one\_at\_a\_time.ckpt"**)  
 loss\_test = reconstruction\_loss.eval(feed\_dict={X: mnist.test.images})  
 print(**"Test MSE:"**, loss\_test)  
  
*#Added by Kishan Sarpangala to plot images*n\_test\_digits = 2  
X\_test = mnist.test.images[:n\_test\_digits]  
  
**with** tf.Session() **as** sess:  
 saver.restore(sess, **"./my\_model\_one\_at\_a\_time.ckpt"**) *# not shown in the book* outputs\_val = outputs.eval(feed\_dict={X: X\_test})  
  
**def** plot\_image(image, shape=[28, 28]):  
 plt.imshow(image.reshape(shape), cmap=**"Greys"**, interpolation=**"nearest"**)  
 plt.axis(**"off"**)  
  
**for** digit\_index **in** range(n\_test\_digits):  
 plt.subplot(n\_test\_digits, 2, digit\_index \* 2 + 1)  
 plot\_image(X\_test[digit\_index])  
 plt.subplot(n\_test\_digits, 2, digit\_index \* 2 + 2)  
 plot\_image(outputs\_val[digit\_index])  
plt.show()