```
In [1]: import tensorflow as tf
        import input_data

In [2]: mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Helper Functions

```
In [3]: #INIT WEIGHTS
        def init_weights(shape):
            init_random_dist = tf.truncated_normal(shape, stddev = 0.1)
            return tf.Variable(init_random_dist)

In [4]: #INIT BIAS
        def init_bias(shape):
            init_bias_vals = tf.constant(0.1, shape = shape)
            return tf.Variable(init_bias_vals)

In [5]: #CONV2D
        def conv2d(x, W):
            #x -> [batch, H, W, Channels]
            #W -> [filterH, filterW, ChannelsIn, ChannelsOut]
            return tf.nn.conv2d(x, W, strides = [1,1,1,1], padding = 'SAME')

In [6]: #POOLING
        def max_pool_2by2(x):
            #x -> [batch, H, W, Channels]
            return tf.nn.max_pool(x, ksize = [1,2,2,1], strides = [1,2,2,1], padding = 'SAME')

In [7]: #CONVOLUTIONAL LAYER
        def convolutional_layer(input_x, shape):
            W = init_weights(shape)
            b = init_bias([shape[3]])
            return tf.nn.relu(conv2d(input_x, W) + b)

In [8]: #NORMAL (FULLY CONNECTED)
        def normal_full_layer(input_layer, size):
            input_size = int(input_layer.get_shape()[1])
            W = init_weights([input_size, size])
            b = init_bias([size])
            return tf.matmul(input_layer, W) + b
```

Placeholders

```
In [9]: x = tf.placeholder(tf.float32, shape = [None, 784])
```

```
In [10]: y_true = tf.placeholder(tf.float32, shape = [None, 10])
```

Create Layers

```
In [11]: x_image = tf.reshape(x, [-1,28,28,1])

In [12]: #32 features for every 5 x 5 patch with 1(grayscale)
         convo_1 = convolutional_layer(x_image, shape = [5,5,1,32])
         convo_1_pooling = max_pool_2by2(convo_1)

In [13]: convo_2 = convolutional_layer(convo_1_pooling, shape = [5,5,32,64])
         convo_2_pooling = max_pool_2by2(convo_2)

In [14]: convo_2_flat = tf.reshape(convo_2_pooling, [-1,7*7*64])
         full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat, 1024))

In [15]: #DROPOUT
         hold_prob = tf.placeholder(tf.float32)
         full_one_dropout = tf.nn.dropout(full_layer_one, keep_prob= hold_prob)

In [16]: y_pred = normal_full_layer(full_one_dropout, 10)

In [17]: #LOSS FUNCTION
         cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_true,

In [18]: #OPTIMIZER
         optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
         train = optimizer.minimize(cross_entropy)

In [19]: init = tf.global_variables_initializer()

In [20]: steps = 1000

         with tf.Session() as sess:
             sess.run(init)

             for i in range(steps):
                 batch_x, batch_y = mnist.train.next_batch(50)
                 sess.run(train, feed_dict = {x:batch_x, y_true:batch_y, hold_prob:0.5})
                 if i%100 == 0:
                     print("ON STEP: {}".format(i))
                     print("ACCURACY: ")
                     match = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y_true, 1))
                     acc = tf.reduce_mean(tf.cast(match, tf.float32))
                     print(sess.run(acc, feed_dict = {x:mnist.test.images, y_true:mnist.test.lab
                     print('\n')
```

```
ON STEP: 0
ACCURACY:
0.101
```

```
ON STEP: 100
ACCURACY:
0.94


ON STEP: 200
ACCURACY:
0.9603


ON STEP: 300
ACCURACY:
0.971


ON STEP: 400
ACCURACY:
0.9717


ON STEP: 500
ACCURACY:
0.9759


ON STEP: 600
ACCURACY:
0.9772


ON STEP: 700
ACCURACY:
0.9811


ON STEP: 800
ACCURACY:
0.9793


ON STEP: 900
ACCURACY:
0.973
```