CNN - CODE

```
import tensorflow as tf
import numpy as np
import input_data
import os

class parameters():

    def __init__(self):
        self.BATCH_SIZE = 128
        self.TEST_SIZE = 256

def load_data():
    # Load the data
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    trainX, trainY, testX, testY = mnist.train.images, mnist.train.labels, mnist.test.images, mnist.test.labels
    trainX = trainX.reshape(-1, 28, 28, 1)
    testX = testX.reshape(-1, 28, 28, 1)
    return trainX, trainY, testX, testY
```

Reshape to image dimensions

```
def model(X, w, w2, w3, w4, w_o, dropout_value_conv, dropout_value_hidden):
    l1a = tf.nn.relu(tf.nn.conv2d(X, w, strides=[1,1,1,1], padding='SAME'))
    l1 = tf.nn.max_pool(l1a, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    l1 = tf.nn.dropout(l1, dropout_value_conv)

    l2a = tf.nn.relu(tf.nn.conv2d(l1, w2, strides=[1,1,1,1], padding='SAME'))
    l2 = tf.nn.max_pool(l2a, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    l2 = tf.nn.dropout(l2, dropout_value_conv)

    l3a = tf.nn.relu(tf.nn.conv2d(l2, w3, strides=[1,1,1,1], padding='SAME'))
    l3 = tf.nn.max_pool(l3a, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
    l3 = tf.reshape(l3, [-1, w4.get_shape().as_list()[0]]) # flatten to shape(?, 2048)
    l3 = tf.nn.dropout(l3, dropout_value_conv)

    l4 = tf.nn.relu(tf.matmul(l3, w4))
    l4 = tf.nn.dropout(l4, dropout_value_hidden)

    return tf.matmul(l4, w_o)
```

CONV & ReLU

POOL

Flatten

FC

CNN - CODE

```
def init_weights(shape):
    return tf.Variable(tf.random_normal(shape, stddev=0.01))

def train():
    # Placeholders
    X = tf.placeholder("float", [None, 28, 28, 1])
    y = tf.placeholder("float", [None, 10])
    dropout_value_conv = tf.placeholder("float")
    dropout_value_hidden = tf.placeholder("float")

    # Initalize weights
    w = init_weights([3, 3, 1, 32])       # 3x3x1 conv, 32 outputs
    w2 = init_weights([3, 3, 32, 64])     # 3x3x32 conv, 64 outputs
    w3 = init_weights([3, 3, 64, 128])    # 3x3x32 conv, 128 outputs
    w4 = init_weights([128 * 4 * 4, 625]) # FC 128 * 4 * 4 = 2048 inputs, 625 outputs
    w_o = init_weights([625, 10])         # FC 625 inputs, 10 outputs (labels)

    logits = model(X, w, w2, w3, w4, w_o, dropout_value_conv, dropout_value_hidden)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits, y))
    optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
    predictions = tf.argmax(logits, 1)

    return dict(X=X, y=y, dropout_value_conv=dropout_value_conv, dropout_value_hidden=dropout_value_hidden,
                cost=cost, optimizer=optimizer, predictions=predictions)
```

Initialize weights

CNN - CODE

```
def main(config, g, trainX, trainY, testX, testY):
    # Variables for saving the model along the training procedure
    ckpt_dir = "./CNN_ckpt_dir"
    if not os.path.exists(ckpt_dir):
        os.makedirs(ckpt_dir)
    global_step = tf.Variable(0, name='global_step', trainable=False)
    # Call this after declaring all tf.Variables.
    saver = tf.train.Saver()

    with tf.Session() as sess:
        tf.initialize_all_variables().run()
        print('All Variables Initialized')

        # Load previous session from checkpoint if available
        ckpt = tf.train.get_checkpoint_state(ckpt_dir)
        if ckpt and ckpt.model_checkpoint_path:
            print(ckpt.model_checkpoint_path)
            saver.restore(sess, ckpt.model_checkpoint_path) # restore all variables
        start = global_step.eval() # get last global_step
        print("Start from:", start)

        for i in range(100):
            # Save the model at each epoch
            if i % 1 == 0:
                global_step.assign(i).eval() # set and update(eval) global_step with index, i
                saver.save(sess, ckpt_dir + "/model.ckpt ", global_step=global_step)

            # Test batch: we dont want to test on entire test set for each minibatch
            test_indices = np.arange(len(testX))
            np.random.shuffle(test_indices)
            test_indices = test_indices[:config.TEST_SIZE]

            # Train
            for start, end in zip(range(0, len(trainX), config.BATCH_SIZE), range(config.BATCH_SIZE, len(trainX), config.BATCH_SIZE)):
                sess.run(g['optimizer'], feed_dict={g['X']:trainX[start:end], g['y']:trainY[start:end], g['dropout_value_conv']:0.8,
g['dropout_value_hidden']:0.5})
            print("Epoch: %.2f, Test Accuracy: %.3f" % (i + start/float(trainX.shape[0]),
                                        np.mean(np.argmax(testY[test_indices], axis=1) == sess.run(g['predictions'],
                                        feed_dict={g['X']: testX[test_indices], g['y']:testY[test_indices],
                                        g['dropout_value_conv']: 1.0, g['dropout_value_hidden']: 1.0}))))

if __name__ == '__main__':
    config = parameters()
    trainX, trainY, testX, testY = load_data()
    g = train()
    main(config, g, trainX, trainY, testX, testY)
```

directory to save model state

load previous checkpoint if available

save model