# hwalsuklee\_tensorflow-mnist-cnnmnist\_cnn\_test

# Some code was borrowed from https://github.com/petewarden/tensorflow\_makefile/blob/master/tensorflow/models/image/mnist/convolutional.py  
  
from \_\_future\_\_ import absolute\_import  
from \_\_future\_\_ import division  
from \_\_future\_\_ import print\_function  
  
import numpy  
import os  
import tensorflow as tf  
import tensorflow.contrib.slim as slim  
from tensorflow.examples.tutorials.mnist import input\_data  
  
import mnist\_data  
import cnn\_model  
  
# user input  
from argparse import ArgumentParser  
  
# refernce argument values  
MODEL\_DIRECTORY = "model"  
TEST\_BATCH\_SIZE = 5000  
ENSEMBLE = True  
  
# build parser  
def build\_parser():  
 parser = ArgumentParser()  
  
 parser.add\_argument('--model-dir',  
 dest='model\_directory', help='directory where model to be tested is stored',  
 metavar='MODEL\_DIRECTORY', required=True)  
 parser.add\_argument('--batch-size', type=int,  
 dest='batch\_size', help='batch size for test',  
 metavar='TEST\_BATCH\_SIZE', required=True)  
 parser.add\_argument('--use-ensemble',  
 dest='ensemble', help='boolean for usage of ensemble',  
 metavar='ENSEMBLE', required=True)  
 return parser  
  
# test with test data given by mnist\_data.py  
def test(model\_directory, batch\_size):  
 # Import data  
 PIXEL\_DEPTH = mnist\_data.PIXEL\_DEPTH  
 mnist = input\_data.read\_data\_sets('data/', one\_hot=True)  
  
 is\_training = tf.placeholder(tf.bool, name='MODE')  
  
 # tf Graph input  
 x = tf.placeholder(tf.float32, [None, 784])  
 y\_ = tf.placeholder(tf.float32, [None, 10]) # answer  
 y = cnn\_model.CNN(x, is\_training=is\_training)  
  
 # Add ops to save and restore all the variables  
 sess = tf.InteractiveSession()  
 sess.run(tf.global\_variables\_initializer(), feed\_dict={is\_training: True})  
  
 # Restore variables from disk  
 saver = tf.train.Saver()  
  
 # Calculate accuracy for all mnist test images  
 test\_size = mnist.test.num\_examples  
 total\_batch = int(test\_size / batch\_size)  
  
 saver.restore(sess, model\_directory)  
  
 acc\_buffer = []  
 # Loop over all batches  
 for i in range(total\_batch):  
  
 batch = mnist.test.next\_batch(batch\_size)  
 batch\_xs = (batch[0] - (PIXEL\_DEPTH / 2.0) / PIXEL\_DEPTH) # make zero-centered distribution as in mnist\_data.extract\_data()  
 batch\_ys = batch[1]  
  
 y\_final = sess.run(y, feed\_dict={x: batch\_xs, y\_: batch\_ys, is\_training: False})  
  
 correct\_prediction = numpy.equal(numpy.argmax(y\_final, 1), numpy.argmax(batch\_ys, 1))  
  
 acc\_buffer.append(numpy.sum(correct\_prediction) / batch\_size)  
  
 print("test accuracy for the stored model: %g" % numpy.mean(acc\_buffer))  
  
# test with test data given by mnist\_data.py  
def test\_org(model\_directory, batch\_size):  
 # Import data  
 PIXEL\_DEPTH = mnist\_data.PIXEL\_DEPTH  
 train\_total\_data, train\_size, validation\_data, validation\_labels, test\_data, test\_labels = mnist\_data.prepare\_MNIST\_data(  
 False)  
  
 is\_training = tf.placeholder(tf.bool, name='MODE')  
  
 # tf Graph input  
 x = tf.placeholder(tf.float32, [None, 784])  
 y\_ = tf.placeholder(tf.float32, [None, 10]) # answer  
 y = cnn\_model.CNN(x, is\_training=is\_training)  
  
 # Add ops to save and restore all the variables  
 sess = tf.InteractiveSession()  
 sess.run(tf.global\_variables\_initializer(), feed\_dict={is\_training: True})  
  
 # Restore variables from disk  
 saver = tf.train.Saver()  
  
 # Calculate accuracy for all mnist test images  
 test\_size = test\_labels.shape[0]  
 total\_batch = int(test\_size / batch\_size)  
  
 saver.restore(sess, model\_directory)  
  
 acc\_buffer = []  
  
 # Loop over all batches  
 for i in range(total\_batch):  
 # Compute the offset of the current minibatch in the data.  
 offset = (i \* batch\_size) % (test\_size)  
 batch\_xs = test\_data[offset:(offset + batch\_size), :]  
 batch\_ys = test\_labels[offset:(offset + batch\_size), :]  
  
 y\_final = sess.run(y, feed\_dict={x: batch\_xs, y\_: batch\_ys, is\_training: False})  
  
 correct\_prediction = numpy.equal(numpy.argmax(y\_final, 1), numpy.argmax(batch\_ys, 1))  
  
 acc\_buffer.append(numpy.sum(correct\_prediction) / batch\_size)  
  
 print("test accuracy for the stored model: %g" % numpy.mean(acc\_buffer))  
  
# For a given matrix, each row is converted into a one-hot row vector  
def one\_hot\_matrix(a):  
 a\_ = numpy.zeros\_like(a)  
 for i, j in zip(numpy.arange(a.shape[0]), numpy.argmax(a, 1)): a\_[i, j] = 1  
 return a\_  
  
# test with test data given by mnist\_data.py  
def test\_ensemble(model\_directory\_list, batch\_size):  
 # Import data  
 PIXEL\_DEPTH = mnist\_data.PIXEL\_DEPTH  
 mnist = input\_data.read\_data\_sets('data/', one\_hot=True)  
  
 is\_training = tf.placeholder(tf.bool, name='MODE')  
  
 # tf Graph input  
 x = tf.placeholder(tf.float32, [None, 784])  
 y\_ = tf.placeholder(tf.float32, [None, 10]) # answer  
 y = cnn\_model.CNN(x, is\_training=is\_training)  
  
 # Add ops to save and restore all the variables  
 sess = tf.InteractiveSession()  
 sess.run(tf.global\_variables\_initializer(), feed\_dict={is\_training: True})  
  
 # Restore variables from disk  
 saver = tf.train.Saver()  
  
 # Calculate accuracy for all mnist test images  
 test\_size = mnist.test.num\_examples  
 total\_batch = int(test\_size / batch\_size)  
  
 acc\_buffer = []  
 # Loop over all batches  
 for i in range(total\_batch):  
  
 batch = mnist.test.next\_batch(batch\_size)  
 batch\_xs = (batch[0] - (PIXEL\_DEPTH / 2.0) / PIXEL\_DEPTH) # make zero-centered distribution as in mnist\_data.extract\_data()  
 batch\_ys = batch[1]  
  
 y\_final = numpy.zeros\_like(batch\_ys)  
  
 for dir in model\_directory\_list:  
 saver.restore(sess, dir+'/model.ckpt')  
 pred = sess.run(y, feed\_dict={x: batch\_xs, y\_: batch\_ys, is\_training: False})  
 y\_final += one\_hot\_matrix(pred) # take a majority vote as an answer  
  
 correct\_prediction = numpy.equal(numpy.argmax(y\_final, 1), numpy.argmax(batch\_ys, 1))  
  
 acc\_buffer.append(numpy.sum(correct\_prediction) / batch\_size)  
  
 print("test accuracy for the stored model: %g" % numpy.mean(acc\_buffer))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Parse argument  
 parser = build\_parser()  
 options = parser.parse\_args()  
 ensemble = options.ensemble  
 model\_directory = options.model\_directory  
 batch\_size = options.batch\_size  
  
 # Select ensemble test or a single model test  
 if ensemble=='True': # use ensemble model  
 model\_directory\_list = [x[0] for x in os.walk(model\_directory)]  
 test\_ensemble(model\_directory\_list[1:], batch\_size)  
 else: # test a single model  
 # test\_org(model\_directory, batch\_size) #test with test data given by mnist\_data.py  
 test(model\_directory+'/model.ckpt',  
 batch\_size) # test with test data given by tensorflow.examples.tutorials.mnist.input\_data()