hw3.py


```
#!/usr/bin/env python3

import tensorflow as tf
import tensorflow.contrib.slim as slim
from tensorflow.contrib.learn.python.learn.datasets import mnist
import numpy as np

class Model():
    def __init__(self, sess, pixels, n_batch, n_classes, learning_rate):
        self.sess = sess
        self.learning_rate = learning_rate
        self.pixels = pixels
        self.n_batch = n_batch
        self.n_classes = n_classes
        self.build_model()

    def build_model(self):
        self.inputs = tf.placeholder(tf.float32, [None, self.pixels, self.pixels, 1])
        self.labels = tf.placeholder(tf.float32, [None, self.n_classes])
        with slim.arg_scope([slim.conv2d, slim.fully_connected],
                            activation_fn=tf.nn.relu,
                            weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                            weights_regularizer=slim.l2_regularizer(0.0005)):
            net = self.inputs
            net = slim.repeat(net, 1, slim.conv2d, 32, [5, 5], scope='conv1')
            net = slim.max_pool2d(net, [2, 2], scope='pool1')
            net = slim.repeat(net, 1, slim.conv2d, 64, [5, 5], scope='conv2')
            net = slim.max_pool2d(net, [2, 2], scope='pool2')
            net = slim.flatten(net, scope='flatten2')
            net = slim.fully_connected(net, 1024, scope='fc3')
            net = slim.dropout(net, 0.8, scope='dropout3')
            net = slim.fully_connected(net, self.n_classes, activation_fn=None, normalizer_fn=None, scope='fc4')
        self.predictions = net
        slim.losses.softmax_cross_entropy(self.predictions, self.labels)
        self.loss = slim.losses.get_total_loss(add_regularization_losses=True)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate).minimize(self.loss)

        correct = tf.equal(tf.argmax(self.predictions,1), tf.argmax(self.labels,1))
        self.accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

    def train_minibatch(self, xs, ys):
        self.sess.run(self.optimizer, feed_dict={self.inputs: xs, self.labels: ys})

    def train(self, xs, ys, xvs, yvs, epochs):
        num_minibatches = xs.shape[0] // self.n_batch
        for epoch in range(epochs):
            p = np.random.permutation(xs.shape[0])
            for i in range(num_minibatches):
                # print("Minibatch " + str(i) + '/' + str(num_minibatches))
                start = i * self.n_batch
                end = (i + 1) * self.n_batch
```

```
52                 self.train_minibatch(xs[p][start:end], ys[p][start:end])
53             accuracy = self.validate(xvs, yvs)
54             print("Epoch {} validation accuracy: {}".format(epoch, accuracy))
55
56     def validate(self, xs, ys):
57         return self.sess.run(self.accuracy, feed_dict={self.inputs: xs, self.labels: ys})
58
59     def test(self, xs):
60         return self.sess.run(self.predictions, feed_dict={self.inputs: xs})
61
62 with tf.Session() as sess:
63     data = mnist.read_data_sets("MNIST_data/", dtype=tf.uint8, reshape=False, one_hot=True)
64     m = Model(sess, 28, 50, 10, 0.001)
65     sess.run(tf.global_variables_initializer())
66     m.train(data.train.images, data.train.labels, data.validation.images, data.validation.labels, 2)
67     test_accuracy = m.validate(data.test.images, data.test.labels)
68     print("Test set accuracy: {}".format(test_accuracy))
```