```
# DongKyu Kim
# ECE 471 CGML Assignment 3
# Professor Curro

# library imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm
from keras.datasets import mnist

# Parameters
N = 50000 # Number of training images this means 10000 of 60000 will be used for
 validation
batch_size = 64
iterations = 1000
display = 100
numclass = 10

# hyper parameters
learning_rate = 5e-4
filter_size = [64,64]
kernel_size = [3,3]
lambda_ = 1e-5 # regularization constant
dropout = 0.6

def one_hot_encoding(data,numclass):
    targets = np.array(data).reshape(-1)
    return np.eye(numclass)[targets]

# Data Generation
class Data(object):
    def __init__(self,N,numclass): # N is number of training data that will be u
sed
        (self.xtemp,self.ytemp),(self.X_test,self.Y_test) = mnist.load_data()
        mask = np.full(self.xtemp.shape[0],True)
        mask[np.random.choice(self.xtemp.shape[0], self.xtemp.shape[0]-N, replac
e=False)] = False
        self.numclass = numclass
        self.X_train = self.xtemp[mask].astype(’float32’)
        self.Y_train = self.ytemp[mask]
        self.X_val = self.xtemp[~mask].astype(’float32’)
        self.Y_val = self.ytemp[~mask]
        self.X_test = self.X_test.astype(’float32’)
        self.Y_test = self.Y_test
        self.X_train /= 255
        self.X_val /= 255
        self.X_test /= 255
        self.Y_train = one_hot_encoding(self.Y_train,self.numclass)
        self.Y_val = one_hot_encoding(self.Y_val,self.numclass)
        self.Y_test = one_hot_encoding(self.Y_test,self.numclass)

    def get_batch(self,batch_size):
        choices = np.random.choice(self.X_train.shape[0],batch_size)
        return self.X_train[choices],self.Y_train[choices]

class My_Model(object):
    def __init__(self,sess,data,learning_rate,filter_size,kernel_size,lambda_,dr
opout,batch_size,iterations,display):
        self.sess = sess
        self.learning_rate = learning_rate
        self.data = data
        self.filter_size = filter_size
        self.kernel_size = kernel_size
        self.lambda_ = lambda_
        self.dropout = tf.placeholder(tf.float32)
        self.batch_size = batch_size
        self.iterations = iterations
        self.display = display
        self.build_model()

    def conv_setup(self,x,filter_size):
        return tf.layers.conv2d(x,filter_size,self.kernel_size,padding=’same’,ac
tivation=tf.nn.relu,kernel_regularizer=self.regularizer)

    def mp_setup(self,x):
        return tf.layers.max_pooling2d(x,[2,2],2)

    def build_model(self):
        self.regularizer = tf.contrib.layers.l2_regularizer(scale = self.lambda_
)
        self.x = tf.placeholder(tf.float32,[None,28,28])
        self.y = tf.placeholder(tf.float32,[None,self.data.numclass])
        x = tf.reshape(self.x,shape=[-1,28,28,1])
        self.yhat = self.conv_setup(x,self.filter_size[0])
        self.yhat = self.mp_setup(self.yhat)
        # maxpooling with size [2,2] stride 2 causes dimension to half, so image
 is [14,14]
        self.yhat = self.conv_setup(self.yhat,self.filter_size[1])
        self.yhat = self.mp_setup(self.yhat)
        # now the output is [7,7]
        self.yhat = tf.reshape(self.yhat,[-1,7*7*self.filter_size[1]])
        self.yhat = tf.layers.dense(inputs = self.yhat, units = 1024, activation
 = tf.nn.relu,kernel_regularizer = self.regularizer)
        self.yhat = tf.layers.dropout(self.yhat,self.dropout)
        self.yhat = tf.layers.dense(inputs=self.yhat, units=self.data.numclass,k
ernel_regularizer = self.regularizer)
        self.costs = tf.losses.softmax_cross_entropy(self.y, self.yhat)
        self.loss = self.costs + tf.losses.get_regularization_loss()

    def train(self):
        self.optim = tf.train.AdamOptimizer(self.learning_rate).minimize(self.lo
ss)
        self.init = tf.global_variables_initializer()
        self.sess.run(self.init)
        print(’Output for DongKyu Kim\’s 3rd Assignment’)
        print(’Training Starts’)
        for k in tqdm(range(0,self.iterations)):
            batch_x, batch_y = self.data.get_batch(self.batch_size)
            self.sess.run([self.optim],feed_dict={self.x:batch_x,self.y:batch_y,
self.dropout:dropout})
            if k % self.display == 99:
                loss = self.sess.run(self.loss,feed_dict={self.x:batch_x,self.y:
batch_y,self.dropout:1.0})
                print(’The Current Loss at ’+str(k+1)+’th iteration is ’+str(loss))

        print(’Training Completed’)
        self.validation_()

    def validation_(self):
        print(’Validation Starts’)
        self.correct = tf.equal(tf.argmax(self.yhat,1),tf.argmax(self.y,1))
        self.accuracy = tf.reduce_mean(tf.cast(self.correct,tf.float32))
        accuracy = 0
        for k in tqdm(range(0,100)):
            temp = self.sess.run(self.accuracy,feed_dict={self.x:self.data.X_val
[k*100:(k+1)*100],self.y:self.data.Y_val[k*100:(k+1)*100],self.dropout:1.0})
            accuracy += temp
        print("Validation Accuracy: ",accuracy,'%')

    def test(self):
        print('Test Starts')
        self.correct = tf.equal(tf.argmax(self.yhat,1),tf.argmax(self.y,1))
        self.accuracy = tf.reduce_mean(tf.cast(self.correct,tf.float32))
        accuracy = 0
        for k in tqdm(range(0,100)):
            temp = self.sess.run(self.accuracy,feed_dict={self.x:self.data.X_tes
t[k*100:(k+1)*100],self.y:self.data.Y_test[k*100:(k+1)*100],self.dropout:1.0})
            accuracy += temp
        print("Test Accuracy: ",accuracy,'%\nbye bye')

# Test Run
data = Data(N,numclass)
sess_test = tf.Session()
model = My_Model(sess_test,data,learning_rate,filter_size,kernel_size,lambda_,dr
opout,batch_size,iterations,display)
model.train()

# Final
model.test()
```