```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Mar  8 19:03:20 2018

@author: chunyilyu
"""

import numpy as np
import tensorflow as tf
import utils
#import pymesh
#from pyntcloud import PyntCloud
import pandas as pd

#reading train data
train_cloud = []
train_label = []
dir_path = "/datasets/home/55/755/cs291eau/PA3/modelnet40_ply_hdf5_2048/"
category_names = utils.get_category_names()

def read_data(dir_path):
    train_cloud = []
    train_label = []
    for i in range(0,5):
        file_path = dir_path+ 'ply_data_train' + str(i) +'.h5'
        train_data0 = utils.load_h5(file_path)
        train_cloud.extend(np.array(train_data0[0]))
        train_label.extend(np.array(train_data0[1]))

    return np.array(train_cloud),np.array(train_label)

train_cloud,train_label = read_data(dir_path)
#reading test data
def read_data(dir_path):
    train_cloud = []
    train_label = []
    for i in range(0,2):
        file_path = dir_path+ 'ply_data_test' + str(i) +'.h5'
        train_data0 = utils.load_h5(file_path)
        train_cloud.extend(np.array(train_data0[0]))
        train_label.extend(np.array(train_data0[1]))

    return np.array(train_cloud),np.array(train_label)
test_cloud,test_label = read_data(dir_path)
```

```
#rotation and jitter
def rotate(points):
    theta = np.random.uniform() * 2 * np.pi
    #ratation matrix
    rotation_matrix = np.array([[ np.cos(theta), 0, np.sin(theta)],
                                [ 0,             1,             0],
                                [-np.sin(theta), 0, np.cos(theta)]])
    rotated_pt_cloud = []
    for p in points:
         rotated_pt_cloud.append((np.matmul(p,rotation_matrix)))
    return np.array(rotated_pt_cloud)
def jitter(points,mean=0,std=0.02):
    return points+np.random.normal(mean, std, points.shape)
tf.reset_default_graph()
#define input transform
def input_transform_net(point_cloud, is_training, bn_decay=None, K=3):

    num_point = point_cloud.get_shape()[1].value
    pt_points = tf.expand_dims(point_cloud, -1)
    layer_conv1 = tf.layers.conv2d(inputs=pt_points, filters=64, kernel_si
                               activation=tf.nn.relu)
    layer_conv1 = tf.contrib.layers.batch_norm(layer_conv1,is_training = i

    layer_conv2 = tf.layers.conv2d(inputs=layer_conv1, filters=128, kernel
                               activation=tf.nn.relu)
    layer_conv2 = tf.contrib.layers.batch_norm(layer_conv2,is_training = i

    layer_conv3 = tf.layers.conv2d(inputs=layer_conv2, filters=1024, kerne
                               activation=tf.nn.relu)
    layer_conv3 = tf.contrib.layers.batch_norm(layer_conv3,is_training = i

    layer_max1 = tf.nn.max_pool(layer_conv3, ksize=[1,num_point,1,1], stri

    layer_max1 = tf.reshape(layer_max1, [-1,1024])

    layer_fc1 = tf.contrib.layers.fully_connected(inputs=layer_max1,num_ou
    layer_fc1 = tf.contrib.layers.batch_norm(layer_fc1,is_training = is_tr

    layer_fc2 = tf.contrib.layers.fully_connected(inputs=layer_fc1,num_out
    layer_fc2 = tf.contrib.layers.batch_norm(layer_fc2,is_training = is_tr

    output = tf.contrib.layers.fully_connected(inputs=layer_fc2,num_output
                                               biases_initializer=tf.zeros
                                               weights_initializer=tf.cont
    print(np.shape(output))
    transform = tf.reshape(output, [-1, K, K])
    print(np.shape(transform))
    return transform

#define feature transform
def feature_transform_net(inputs, is_training, bn_decay=None, K=64):


    num_point = inputs.get_shape()[1].value

    layer_conv1 = tf.layers.conv2d(inputs=inputs, filters=64, kernel_size=
                                activation=tf.nn.relu)
    layer_conv1 = tf.contrib.layers.batch_norm(layer_conv1,is_training = i

    layer_conv2 = tf.layers.conv2d(inputs=layer_conv1, filters=128, kernel
                                activation=tf.nn.relu)
    layer_conv2 = tf.contrib.layers.batch_norm(layer_conv2,is_training = i

    layer_conv3 = tf.layers.conv2d(inputs=layer_conv2, filters=1024, kerne
                                activation=tf.nn.relu)
    layer_conv3 = tf.contrib.layers.batch_norm(layer_conv3,is_training = i

    layer_max1 = tf.nn.max_pool(layer_conv3, ksize=[1,num_point,1,1], stri

    layer_max1 = tf.reshape(layer_max1, [-1,1024])

    layer_fc1 = tf.contrib.layers.fully_connected(inputs=layer_max1,num_ou
    layer_fc1 = tf.contrib.layers.batch_norm(layer_fc1,is_training = is_tr

    layer_fc2 = tf.contrib.layers.fully_connected(inputs=layer_fc1,num_out
    layer_fc2 = tf.contrib.layers.batch_norm(layer_fc2,is_training = is_tr


    output = tf.contrib.layers.fully_connected(inputs=layer_fc2,num_output
                                               biases_initializer=tf.zeros
                                               weights_initializer=tf.cont
    print(np.shape(output))
    transform = tf.reshape(output, [-1, K, K])
    print(np.shape(transform))
    return transform

#define the model
tf.reset_default_graph()
#define some placeholder
points = tf.placeholder(tf.float32,[None,2048,3])
pt_points = tf.expand_dims(points, -1)
```

```
num_point = points.get_shape()[1].value

is_training = tf.placeholder(tf.bool)
batch = tf.Variable(0)
labels = tf.placeholder(tf.int32,[None])      #need to make things more ge

drop_rate = tf.placeholder(tf.float32)
learning_rate = tf.placeholder(tf.float32)

#start defining the model
layer_conv1 = tf.layers.conv2d(inputs=pt_points, filters=64, kernel_size=[
                               activation=tf.nn.relu)
layer_conv1 = tf.contrib.layers.batch_norm(layer_conv1,is_training = is_tr

layer_conv2 = tf.layers.conv2d(inputs=layer_conv1, filters=64, kernel_size
                                       activation=tf.nn.relu)
layer_conv2 = tf.contrib.layers.batch_norm(layer_conv2,is_training = is_tr

layer_conv3 =  tf.layers.conv2d(inputs=layer_conv2, filters=64, kernel_siz
                                       activation=tf.nn.relu)
layer_conv3 = tf.contrib.layers.batch_norm(layer_conv3,is_training = is_tr

layer_conv4 = tf.layers.conv2d(inputs=layer_conv3, filters=128, kernel_siz
                                       activation=tf.nn.relu)
layer_conv4 = tf.contrib.layers.batch_norm(layer_conv4,is_training = is_tr

layer_conv5 = tf.layers.conv2d(inputs=layer_conv4, filters=1024, kernel_si
                                       activation=tf.nn.relu)
layer_conv5 = tf.contrib.layers.batch_norm(layer_conv5,is_training = is_tr

layer_max = tf.nn.max_pool(layer_conv5, ksize=[1,num_point,1,1], strides=[
layer_global = tf.reshape(layer_max,[-1,1024])

layer_fnn1 = tf.contrib.layers.fully_connected(inputs=layer_global,num_out
layer_fnn2 = tf.contrib.layers.fully_connected(inputs=layer_fnn1,num_outpu

layer_fnn3 = tf.layers.dropout(layer_fnn2, rate=drop_rate,training = is_tr
output = tf.contrib.layers.fully_connected(inputs=layer_fnn3,num_outputs=4

#change labels as onehot vector to feed into loss function
labels_onehot = tf.one_hot(labels,depth=40,dtype=tf.int32)

#define loss function by calling softmax
loss = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=label
loss = tf.reduce_mean(loss)
```

```
#get predict results
predict = tf.cast(tf.argmax(output,1),tf.int32)
correct_prediction = tf.equal(predict, labels)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#define optimazer
optim = tf.train.AdamOptimizer(learning_rate=learning_rate)
with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
    model_train = optim.minimize(loss,global_step=batch)

#%% running the model
# Create session
import os
batch_size = 32
current_epoch = 0
num_iter = 100

LOG_DIR = 'log_'
train_loss = []
train_acc = []
test_loss = []
test_acc = []

num_train = train_label.shape[0]
num_test = test_label.shape[0]
#learning_rate = 0.001
#rate = tf.Variable(0.001)
rate = 0.001

with tf.Session() as sess:

    #read the model if we train it before
    if current_epoch != 0:
        saver = tf.train.import_meta_graph("models/PointNet_beta-%i.meta"
        saver.restore(sess, tf.train.latest_checkpoint("./models"))
    else:
        saver = tf.train.Saver()
    sess.run(tf.global_variables_initializer())


    for epoch in range(num_iter):
        #half the epoch every 20 seconds
        if epoch %20 == 0:
            rate /= 2

        loss_train_all = 0
```

```
acc_train_all = 0

#shuffle the data
idx = np.arange(num_train)
np.random.shuffle(idx)
train_cloud = train_cloud[idx, ...]
train_label = train_label[idx]
batch_num = np.ceil(num_train/batch_size).astype(int)
batch_num_test = np.ceil(num_test/batch_size).astype(int)

#start iterating in each batch
for batch_idx in range(batch_num):
    start_idx = batch_idx*batch_size
    end_idx = np.min([(batch_idx+1)*batch_size,num_train-1])
    batch_img = train_cloud[start_idx:end_idx,...]
    batch_y = train_label[start_idx:end_idx,...]

    batch_y = np.reshape(batch_y,[-1])

    #argumented data
    rotation_data = rotate(batch_img)
    augment_data = jitter(rotation_data)

    sess.run([model_train],feed_dict={points: augment_data, labels
                                      learning_rate:rate,drop_rate
    loss_train,acc_train = sess.run([loss,accuracy], \
                                feed_dict={points: augment_data, l
                                          learning_rate:rate,drop_
    loss_train_all += loss_train*(end_idx-start_idx)
    acc_train_all += acc_train * (end_idx-start_idx)

#get average loss
loss_train_all = loss_train_all / num_train
acc_train_all = acc_train_all / num_train
train_loss.append(loss_train_all)
train_acc.append(acc_train_all)
print('TRAIN: epoch: ', epoch+1, '\tloss: %.4f'%loss_train_all, '\
if (epoch+1) % 20 == 0:
    save_path = saver.save(sess,"models/PointNet_beta", global_ste
    print('TRAIN: epoch: ', epoch+1, '\tloss: %.4f'%loss_train_all
loss_test_all = 0
acc_test_all = 0

#for test dataset, do the same thing
for batch_idx in range(batch_num_test):
    start_idx = batch_idx*batch_size
```

```
            end_idx = np.min([(batch_idx+1)*batch_size,num_train-1])
            batch_img = test_cloud[start_idx:end_idx,...]
            batch_y = test_label[start_idx:end_idx]
            batch_y = np.reshape(batch_y,[-1])
            rotation_data = rotate(batch_img)

            #change drop rate as 1 during test
            loss_test,acc_test = sess.run([loss,accuracy], \
                                        feed_dict={points: rotation_data,
                                                  learning_rate:rate,drop_
            loss_test_all = loss_test_all + loss_test*(end_idx-start_idx)
            acc_test_all = acc_test_all + acc_test*(end_idx-start_idx)

        loss_test_all = loss_test_all / num_test
        acc_test_all = acc_test_all / num_test
        test_loss.append(loss_test_all)
        test_acc.append(acc_test_all)
        print('Test: epoch: ', epoch+1, '\tloss: %.4f'%loss_test_all, '\ta
        if (epoch+1) % 20 == 0:
            #save_path = saver.save(sess,"models/PointNet_Vanilla", global
            print('TEST: epoch: ', epoch+1, '\tloss: %.4f'%loss_test_all,
```