```
import tensorflow as tf
import h5py
import os
import numpy as np

# import the data
"""
Helper functions to implement PointNet
"""
MODELNET40_PATH = "modelnet40_ply_hdf5_2048"
h5_filename_train =
["ply_data_train0.h5","ply_data_train1.h5","ply_data_train2.h5","ply_data_train3.h5
","ply_data_train4.h5"]
h5_filename_test = ["ply_data_test0.h5","ply_data_test1.h5"]

def load_h5(h5_filename):
    """
    Data loader function.
    Input: The path of h5 filename
    Output: A tuple of (data,label)
    """
    f = h5py.File(h5_filename)
    data = f['data'][:]
    label = f['label'][:]
    return (data, label)
def get_category_names():
    """
    Function to list out all the categories in MODELNET40
    """
    shape_names_file = os.path.join(MODELNET40_PATH, 'shape_names.txt')
    shape_names = [line.rstrip() for line in open(shape_names_file)]
    return shape_names
def evaluate(true_labels,predicted_labels):
    """
    Function to calculate the total accuracy.
    Input: The ground truth labels and the predicted labels
    Output: The accuracy of the model
    """
    return np.mean(true_labels == predicted_labels)

# read the data
train_data0, train_label0 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_train[0]))
train_data1, train_label1 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_train[1]))
train_data2, train_label2 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_train[2]))
train_data3, train_label3 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_train[3]))
train_data4, train_label4 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_train[4]))
train_data = np.concatenate((train_data0, train_data1), axis=0)
train_data = np.concatenate((train_data, train_data2), axis=0)
train_data = np.concatenate((train_data, train_data3), axis=0)
```

```
train_data = np.concatenate((train_data, train_data4), axis=0)
train_data = train_data[:,:1024,:]
train_label = np.concatenate((train_label0, train_label1), axis=0)
train_label = np.concatenate((train_label, train_label2), axis=0)
train_label = np.concatenate((train_label, train_label3), axis=0)
train_label = np.concatenate((train_label, train_label4), axis=0)
train_label = np.reshape(train_label,[-1])
test_data0, test_label0 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_test[0]))
test_data1, test_label1 = load_h5(os.path.join(MODELNET40_PATH,
h5_filename_test[1]))
test_data = np.concatenate((test_data0, test_data1), axis=0)
test_data = test_data[:,:1024,:]
test_label = np.concatenate((test_label0, test_label1), axis=0)
test_label = np.reshape(test_label,[-1])

# network structure
tf.reset_default_graph()
# learning rate decay
def get_learning_rate(batch):
    learning_rate = tf.train.exponential_decay(
                        0.001,       # Base learning rate.
                        batch * 32,  # Current index into the dataset.
                        200000,      # Decay step.
                        0.7,        # Decay rate.
                        staircase=True)
    learning_rate = tf.maximum(learning_rate, 0.00001) # CLIP THE LEARNING RATE!
    return learning_rate
# batch-normalization stats estimate update decay
def get_bn_decay(batch):
    bn_momentum = tf.train.exponential_decay(
                      0.5,
                      batch*32,
                      200000,
                      0.5,
                      staircase=True)
    bn_decay = tf.minimum(0.99, 1 - bn_momentum)
    return bn_decay
# fully connected layer
def fully_connected(prev_layer, num_units, batch_norm, batch_norm_decay, scope,
is_training=False):
    with tf.variable_scope(scope) as sc:
        layer = tf.layers.dense(prev_layer, num_units, use_bias=False,
activation=None)
        if batch_norm:
            layer = tf.layers.batch_normalization(layer, momentum=batch_norm_decay,
training=is_training)
        layer = tf.nn.relu(layer)
        return layer
# convolution layer
def conv_layer(prev_layer, layer_depth, kernel_size, batch_norm, batch_norm_decay,
scope, is_training=False):
    with tf.variable_scope(scope) as sc:
        strides = [1,1]
```

```
        conv_layer = tf.layers.conv2d(prev_layer, layer_depth, kernel_size,
strides, use_bias=False, activation=None)
        if batch_norm:
            conv_layer = tf.layers.batch_normalization(conv_layer,
momentum=batch_norm_decay, training=is_training)
        conv_layer = tf.nn.relu(conv_layer)
        return conv_layer
# dropout layer
def dropout_layer(prev_layer, keep_prob, scope, is_training=False):
    with tf.variable_scope(scope) as sc:
        dropout_layer = tf.cond(is_training,
                                lambda: tf.nn.dropout(prev_layer, keep_prob),
                                lambda: prev_layer)
        return dropout_layer

def input_transform_net(point_cloud, is_training, bn_decay=None, K=3):
    batch_size = point_cloud.get_shape()[0].value
    num_point = point_cloud.get_shape()[1].value
    input_image = tf.expand_dims(point_cloud, -1)
    net = conv_layer(input_image, 64, [1,3], scope='t1_conv_layer1',
batch_norm=True, batch_norm_decay=bn_decay, is_training=is_training)
    net = conv_layer(net, 128, [1,1], scope='t1_conv_layer2', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
    net = conv_layer(net, 1024, [1,1], scope='t1_conv_layer3', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
    net = tf.nn.max_pool(net, ksize=[1,num_point,1,1], strides=[1,2,2,1],
padding='VALID')
    net = tf.reshape(net, [-1,1024])
    net = fully_connected(net, 512, scope='t1_fc_layer1', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
    net = fully_connected(net, 256, scope='t1_final_layer', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)

    with tf.variable_scope('transform_XYZ') as sc:
        assert(K==3)
        weights = tf.get_variable('t1_weights', [256, 3*K],
                                  initializer=tf.constant_initializer(0.0),
                                  dtype=tf.float32)
        biases = tf.get_variable('t1_biases', [3*K],
                                 initializer=tf.constant_initializer(0.0),
                                 dtype=tf.float32)
        biases += tf.constant([1,0,0,0,1,0,0,0,1], dtype=tf.float32)
        transform = tf.matmul(net, weights)
        transform = tf.nn.bias_add(transform, biases)

    transform = tf.reshape(transform, [-1, 3, K])
    return transform

def feature_transform_net(inputs, is_training, bn_decay=None, K=64):
    batch_size = inputs.get_shape()[0].value
    num_point = inputs.get_shape()[1].value

    net = conv_layer(inputs, 64, [1,1], scope='t2_conv_layer1', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
```

```
    net = conv_layer(net, 128, [1,1], scope='t2_conv_layer2', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
    net = conv_layer(net, 1024, [1,1], scope='t2_conv_layer3', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
    net = tf.nn.max_pool(net, ksize=[1,num_point,1,1], strides=[1,2,2,1],
padding='VALID')
    net = tf.reshape(net, [-1,1024])
    net = fully_connected(net, 512, scope='t2_fc_layer1', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
    net = fully_connected(net, 256, scope='t2_final_layer', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)

    with tf.variable_scope('transform_feat') as sc:
        weights = tf.get_variable('weights', [256, K*K],
                                  initializer=tf.constant_initializer(0.0),
                                  dtype=tf.float32)
        biases = tf.get_variable('biases', [K*K],
                                 initializer=tf.constant_initializer(0.0),
                                 dtype=tf.float32)
        biases += tf.constant(np.eye(K).flatten(), dtype=tf.float32)
        transform = tf.matmul(net, weights)
        transform = tf.nn.bias_add(transform, biases)

    transform = tf.reshape(transform, [-1, K, K])
    return transform


batch = tf.Variable(0)
end_points = {}

X = tf.placeholder(tf.float32, [None, 1024, 3])
label = tf.placeholder(tf.int32, [None])
is_training = tf.placeholder(tf.bool)

batch_size = X.get_shape()[0].value
num_point = X.get_shape()[1].value

bn_decay = get_bn_decay(batch)
tf.summary.scalar('bn_decay', bn_decay)

#input_x = tf.expand_dims(X, -1)
# input transform layer
transform = input_transform_net(X, is_training, bn_decay, K=3)
X_new = tf.matmul(X, transform)
input_x = tf.expand_dims(X_new, -1)

# MLP implemented as conv2d
conv_layer1 = conv_layer(input_x, 64, [1,3], scope='conv_layer1', batch_norm=True,
batch_norm_decay=bn_decay, is_training=is_training)
conv_layer2 = conv_layer(conv_layer1, 64, [1,1], scope='conv_layer2',
batch_norm=True, batch_norm_decay=bn_decay, is_training=is_training)

#net_transformed = conv_layer2
# feature transform layer
```

```
with tf.variable_scope('transform_net2') as sc:
    transform = feature_transform_net(conv_layer2, is_training, bn_decay, K=64)
end_points['transform'] = transform
net_transformed = tf.matmul(tf.squeeze(conv_layer2, axis=[2]), transform)
net_transformed = tf.expand_dims(net_transformed, [2])

# MLP implemented as conv2d
conv_layer3 = conv_layer(net_transformed, 64, [1,1], scope='conv_layer3',
batch_norm=True, batch_norm_decay=bn_decay, is_training=is_training)
conv_layer4 = conv_layer(conv_layer3, 128, [1,1], scope='conv_layer4',
batch_norm=True, batch_norm_decay=bn_decay, is_training=is_training)
conv_layer5 = conv_layer(conv_layer4, 1024, [1,1], scope='conv_layer5',
batch_norm=True, batch_norm_decay=bn_decay, is_training=is_training)
# Maxpooling
maxpool_layer = tf.nn.max_pool(conv_layer5, ksize=[1,num_point,1,1],
strides=[1,2,2,1], padding='VALID')
global_layer = tf.reshape(maxpool_layer, [-1,1024])
# MLP implemented as fully-connected
fc_layer1 = fully_connected(global_layer, 512, batch_norm=True, scope='fc_layer1',
batch_norm_decay=bn_decay, is_training=is_training)
fc_layer2 = fully_connected(fc_layer1, 256, batch_norm=True, scope='fc_layer2',
batch_norm_decay=bn_decay, is_training=is_training)
# Dropout
dropout_layer = dropout_layer(fc_layer2, 0.7, scope = 'dropout',
is_training=is_training,)
output = fully_connected(dropout_layer, 40, batch_norm=True, scope = 'fc_output',
batch_norm_decay=bn_decay, is_training=is_training)

# loss
loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=output, labels=label)
loss = tf.reduce_mean(loss)

# feature transform regularization loss
transform = end_points['transform'] # BxKxK
K = transform.get_shape()[1].value
mat_diff = tf.matmul(transform, tf.transpose(transform, perm=[0,2,1]))
mat_diff -= tf.constant(np.eye(K), dtype=tf.float32)
mat_diff_loss = tf.nn.l2_loss(mat_diff)
loss = loss + mat_diff_loss * 0.001

tf.summary.scalar('loss', loss)

# accuracy
predict = tf.cast(tf.argmax(output,1),tf.int32)
correct_prediction = tf.equal(predict,label)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# optimize
learning_rate = get_learning_rate(batch)
tf.summary.scalar('learning_rate', learning_rate)
optimizer = tf.train.AdamOptimizer(learning_rate)
with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
    model_train = optimizer.minimize(loss, global_step=batch)
```

```
# data augment
def rotation_pointcloud(pc):
    pc_new = np.copy(pc)
    for b in range(pc.shape[0]):
        angle_x = np.random.uniform() * 2 * np.pi
        angle_y = np.random.uniform() * 2 * np.pi
        angle_z = np.random.uniform() * 2 * np.pi
        matrix_x = np.array([[1,0,0],[0,np.cos(angle_x),-np.sin(angle_x)],
[0,np.sin(angle_x),np.cos(angle_x)]])
        matrix_y = np.array([[np.cos(angle_y),0,np.sin(angle_y)],[0,1,0],[-
np.sin(angle_y),0,np.cos(angle_y)]])
        matrix_z = np.array([[np.cos(angle_z),-np.sin(angle_z),0],
[np.sin(angle_z),np.cos(angle_z),0],[0,0,1]])
        #pc[b] = np.dot(np.dot(np.dot(np.reshape(pc[b],[-
1,3]),matrix_x),matrix_y),matrix_z)
        #pc[b] = np.dot(np.dot(np.reshape(pc[b],[-1,3]),matrix_x),matrix_y)
        pc_new[b] = np.dot(np.reshape(pc[b],[-1,3]),matrix_y)
    return pc_new

def jittering_pointcloud(pc,sigma = 0.01,clip=0.05):
    pc_new = np.copy(pc)
    B, P, C = pc.shape
    jitter = np.clip(sigma * np.random.randn(B, P, C), -1*clip, clip)
    pc_new = pc + jitter
    return pc_new

# train and validation(test)
# notice we use test data for validation(test)
batch_size = 32
num_point = 1024
max_epoch = 250
num_train = train_label.shape[0]
num_test = test_label.shape[0]
LOG_DIR = 'log_2'
with tf.Session() as sess:
    saver = tf.train.Saver()
    merged = tf.summary.merge_all()
    train_writer = tf.summary.FileWriter(os.path.join(LOG_DIR, 'train'),sess.graph)
    test_writer = tf.summary.FileWriter(os.path.join(LOG_DIR, 'test'))
    sess.run(tf.global_variables_initializer())
    batch_num = np.ceil(num_train/batch_size).astype(int)
    batch_num_test = np.ceil(num_test/batch_size).astype(int)
    for epoch in range(max_epoch):
        # shuffle the data for each epoch
        idx = np.arange(num_train)
        np.random.shuffle(idx)
        train_data = train_data[idx, ...]
        train_label = train_label[idx]
        loss_train_all = 0
        acc_train_all = 0
        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])
```

```
            feed_data = train_data[start_idx:end_idx,...]
            rotation_data = rotation_pointcloud(feed_data)
            augment_data = jittering_pointcloud(rotation_data)
            sess.run([model_train],{X: augment_data, \
                                    label: train_label[start_idx:end_idx],
is_training: True})
            summary,loss_train,acc_train,log_step =
sess.run([merged,loss,accuracy,batch], \
                                         {X: augment_data, \
                                          label: train_label[start_idx:end_idx],
is_training: False})
            train_writer.add_summary(summary, log_step)
            loss_train_all = loss_train_all + loss_train*(end_idx-start_idx)
            acc_train_all = acc_train_all + acc_train*(end_idx-start_idx)
        loss_train_all = loss_train_all / num_train
        acc_train_all = acc_train_all / num_train
        print('TRAIN: epoch: ', epoch+1, '\tloss: %.4f'%loss_train_all,
'\taccuracy: %.4f'%acc_train_all)
        if (epoch+1) % 50 == 0:
            save_path = saver.save(sess, os.path.join(LOG_DIR,
"model_"+str(epoch+1)+".ckpt"))
            print("Model saved in file: %s" % save_path)

        # validation
        loss_val_all = 0
        acc_val_all = 0
        total_seen_class = np.zeros(40)
        total_correct_class = np.zeros(40)
        for batch_idx in range(batch_num_test):
            start_idx = batch_idx*batch_size
            end_idx = np.min([(batch_idx+1)*batch_size,num_test-1])
            feed_data = test_data[start_idx:end_idx,...]
            rotation_data = rotation_pointcloud(feed_data)
            summary,pred,loss_val,acc_val,log_step =
sess.run([merged,predict,loss,accuracy,batch], \
                                         {X: rotation_data, \
                                          label: test_label[start_idx:end_idx],
is_training: False})
            test_writer.add_summary(summary, log_step)
            loss_val_all = loss_val_all + loss_val*(end_idx-start_idx)
            acc_val_all = acc_val_all + acc_val*(end_idx-start_idx)
            for c_index in range(start_idx, end_idx):
                current_class = test_label[c_index]
                total_seen_class[current_class] += 1
                total_correct_class[current_class] += (pred[c_index-
start_idx]==current_class)
        loss_val_all = loss_val_all / num_test
        acc_val_all = acc_val_all / num_test
        acc_avg_class = np.mean(total_correct_class/total_seen_class)
        print('VALID: epoch: ', epoch+1, '\tloss: %.4f'%loss_val_all, \
              '\ttotal accuracy: %.4f'%acc_val_all, \
              '\t per class accuracy: %.4f'%acc_avg_class)
```