LAMPIRAN

Lampiran 1 Script Tensorflow Ichenli Train_Mnist_Image

```
############################################################
#! /usr/bin/python2.7
# -*- coding = utf-8 -*-
# Author: Ali
# Time: 20171119
############################################################

import os
import cv2
import sys
import argparse
import tensorflow as tf

############################################################
# Set parameter
train_tfrecords_dir = 'train.tfrecords'
test_tfrecords_dir = 'test.tfrecords'
num_classes = [str(i) for i in range(0,10)]
epoch = 10000
batch_size = 2000

# current work dir
cwd = os.getcwd()

# data to int64List
def _int64_feature(value):
    return tf.train.Feature(int64_list = tf.train.Int64List(value=[value]))
# data to floatlist
def _float_feature(value):
    return tf.train.Feature(float_list = tf.train.FloatList(value=[value]))
# data to byteslist
def _bytes_feature(value):
    return tf.train.Feature(bytes_list = tf.train.BytesList(value=[value]))
```

```
# convert image data to tfrecords
def generate_tfrecords(data_dir,filepath):
    # gen a tfrecords object write
    write = tf.python_io.TFRecordWriter(filepath)
    #print cwd
    for index,name in enumerate(num_classes):
        #print class_path
        file_dir = data_dir + name + '/'
        for img_name in os.listdir(file_dir):
            img_path = file_dir + img_name
            print img_path
            img = cv2.imread(img_path)
            img_raw = img.tobytes()
            example = tf.train.Example(features=tf.train.Features(feature={
                                        'label':_int64_feature(index),
                                        'img_raw':_bytes_feature(img_raw)}))
            # convert example to binary string
            write.write(example.SerializeToString())
    write.close()

# read and decode tfrecord
def read_and_decode_tfrecord(filename):
    # produce file deque
    filename_deque = tf.train.string_input_producer([filename])
    # gen reader object
    reader = tf.TFRecordReader()
    # read data form filename_deque
    _, serialized_example = reader.read(filename_deque)
    # decode into identy formate
    features = tf.parse_single_example(serialized_example,features={
                                        'label':tf.FixedLenFeature([],tf.int64),
                                        'img_raw':tf.FixedLenFeature([],tf.string)})

    label = tf.cast(features['label'],tf.int32)
    img = tf.decode_raw(features['img_raw'],tf.uint8)
    img = tf.reshape(img,[28,28,3])
    img = tf.cast(img,tf.float32)/255.-0.5
    return label,img
# create network
class network(object):
```

```
# define parameters w and b
   def __init__(self):
        with tf.variable_scope("Weight"):
          self.weights={

        'conv1':tf.get_variable('conv1',[5,5,3,32],initializer=tf.contrib.layers.xavier_ini
tializer_conv2d()),

        'conv2':tf.get_variable('conv2',[5,5,32,64],initializer=tf.contrib.layers.xavier_i
nitializer_conv2d()),
                  'fc1'  :tf.get_variable('fc1',
[7*7*64,1024],initializer=tf.contrib.layers.xavier_initializer()),
                  'fc2'  :tf.get_variable('fc2',  [1024,10],
initializer=tf.contrib.layers.xavier_initializer()),}
        with tf.variable_scope("biases"):
           self.biases={

        'conv1':tf.get_variable('conv1',[32,],initializer=tf.constant_initializer(value=0.
0,dtype=tf.float32)),

        'conv2':tf.get_variable('conv2',[64,],initializer=tf.constant_initializer(value=0.
0,dtype=tf.float32)),
                  'fc1'  :tf.get_variable('fc1',
[1024,],initializer=tf.constant_initializer(value=0.0,dtype=tf.float32)),
                  'fc2'  :tf.get_variable('fc2',  [10,]
,initializer=tf.constant_initializer(value=0.0,dtype=tf.float32)),}

   # define model
   def model(self,img):
        conv1 =
tf.nn.bias_add(tf.nn.conv2d(img,self.weights['conv1'],strides=[1,1,1,1],padding='SA
ME'),self.biases['conv1'])
        relu1 = tf.nn.relu(conv1)
        pool1 =
tf.nn.max_pool(relu1,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

        conv2 =
tf.nn.bias_add(tf.nn.conv2d(pool1,self.weights['conv2'],strides=[1,1,1,1],padding='S
AME'),self.biases['conv2'])
```

```
relu2 = tf.nn.relu(conv2)
        pool2 =
tf.nn.max_pool(relu2,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

        flatten = tf.reshape(pool2,[-1,self.weights['fc1'].get_shape().as_list()[0]])

        drop1 = tf.nn.dropout(flatten,0.8)
        fc1   = tf.matmul(drop1,self.weights['fc1']) + self.biases['fc1']
        fc_relu1 = tf.nn.relu(fc1)
        fc2   = tf.matmul(fc_relu1,self.weights['fc2'])+self.biases['fc2']

        return fc2

   # define model test
   def test(self,img):
        img = tf.reshape(img,shape=[-1,28,28,3])

        conv1 =
tf.nn.bias_add(tf.nn.conv2d(img,self.weights['conv1'],strides=[1,1,1,1],padding='SA
ME'),self.biases['conv1'])
        relu1 = tf.nn.relu(conv1)
        pool1 =
tf.nn.max_pool(relu1,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

        conv2 =
tf.nn.bias_add(tf.nn.conv2d(pool1,self.weights['conv2'],strides=[1,1,1,1],padding='S
AME'),self.biases['conv2'])
        relu2 = tf.nn.relu(conv2)
        pool2 =
tf.nn.max_pool(relu2,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

        flatten = tf.reshape(pool2,[-1,self.weights['fc1'].get_shape().as_list()[0]])

        drop1 = tf.nn.dropout(flatten,1)
        fc1   = tf.matmul(drop1,self.weights['fc1']) + self.biases['fc1']
        fc_relu1 = tf.nn.relu(fc1)
        fc2   = tf.matmul(fc_relu1,self.weights['fc2'])+self.biases['fc2']

        return fc2
```

```
#loss
    def softmax_loss(self,predicts,labels):
        predicts = tf.nn.softmax(predicts)
        labels   = tf.one_hot(labels,len(num_classes))
        loss = -tf.reduce_mean(labels*tf.log(predicts))
        self.cost = loss
        return self.cost

    # optimizer
    def optimizer(self,loss,lr=0.001):
        train_optimizer = tf.train.GradientDescentOptimizer(lr).minimize(loss)
        return train_optimizer

def generate_data():
    train_data_path    = cwd + '/mnist_train/'
    test_data_path = cwd + '/mnist_test/'
    generate_tfrecords(train_data_path,train_tfrecords_dir)
    generate_tfrecords(test_data_path,test_tfrecords_dir)

def evaluate(model_path,test_img):
    img = tf.placeholder(tf.float32,shape=(28,28,3))
    net = network()
    predict = net.test(img)
    predict = tf.nn.softmax(predict)
    label = tf.argmax(predict,1)
    init = tf.global_variables_initializer()
    with tf.Session() as sess:
        sess.run(init)
        tf.train.Saver(max_to_keep=None).restore(sess,model_path)
        pred,lab = sess.run([predict,label],feed_dict={img:test_img})
      print pred,lab

def train():
    label, img = read_and_decode_tfrecord(train_tfrecords_dir)
    img_batch,label_batch =
tf.train.shuffle_batch([img,label],num_threads=16,batch_size=batch_size,capacity=5
0000,min_after_dequeue=49000)

    net = network()
```

```
predicts = net.model(img_batch)
    loss = net.softmax_loss(predicts,label_batch)
    opti = net.optimizer(loss)
    # add trace
    tf.summary.scalar('cost fuction',loss)
    merged_summary_op = tf.summary.merge_all()


    train_correct = tf.equal(tf.cast(tf.argmax(predicts,1),tf.int32),label_batch)
    train_accuracy = tf.reduce_mean(tf.cast(train_correct,tf.float32))

    #evaluate
    test_label,test_img = read_and_decode_tfrecord(test_tfrecords_dir)
    test_img_batch,test_label_batch =
tf.train.shuffle_batch([test_img,test_label],num_threads=16,batch_size=batch_size,ca
pacity=50000,min_after_dequeue=40000)
    test_out = net.test(test_img_batch)

    test_correct = tf.equal(tf.cast(tf.argmax(test_out,1),tf.int32),test_label_batch)
    test_accuracy = tf.reduce_mean(tf.cast(test_correct,tf.float32))

    # init varibels
    init = tf.global_variables_initializer()
    with tf.Session() as sess:
            sess.run(init)
            # manerge different threads
            coord = tf.train.Coordinator()
          summary_writer = tf.summary.FileWriter('log',sess.graph)
            # run deque
            threads = tf.train.start_queue_runners(sess=sess,coord=coord)
            path =cwd+'/'+'model/model.ckpt'
            #if os.path.exists(path):
            try:
                print "try to reload model ....."
                tf.train.Saver(max_to_keep=None).restore(sess,path)
                print 'reload successful ......'
            except:
                print 'reload model failed ......'
            finally:
                print 'training .......'
```

```
for i in range(1,epoch+1):
            #val,l= sess.run([img_batch,label_batch])
            if i%50 ==0:
               loss_np,_,label_np,img_np,predict_np =
sess.run([loss,opti,label_batch,img_batch,predicts])
                  tr_accuracy_np = sess.run([train_accuracy])
                  print i,' epoch loss :',loss_np,'    train accuracy: ', tr_accuracy_np
            if i%200==0:
                  summary_str,_l,_o = sess.run([merged_summary_op,loss,opti])
                  summary_writer.add_summary(summary_str,i)
                  te_accuracy = sess.run([test_accuracy])
                  print 'test accuracy: ', te_accuracy
            if i%1000==0:

tf.train.Saver(max_to_keep=None).save(sess,os.path.join('model','model.ckpt'))
         # somethind happend that the thread should stop
         coord.request_stop()
         # wait for all threads should stop and then stop
         coord.join(threads)

def main():
   parser = argparse.ArgumentParser()
   parser.add_argument('--mode',type=str,default='train',help='train,evaluate')
   args = parser.parse_args()
   if args.mode=='data':
         generate_data()
         print "data done"
   elif args.mode=="train":
         print 'train'
         train()
   elif args.mode=='evaluate':
         print "evaluate"
         image = './19.bmp'
         model_path = os.getcwd() +'/'+'model/model.ckpt'
      img = cv2.imread(image)
         evaluate(model_path,img)
   else:
         print "uknown mode"
         exit()
```

```
if __name__ == "__main__":
    main()
```