#!/usr/bin/env python

from \_\_future\_\_ import print\_function

import argparse

import numpy as np

from keras.callbacks import CSVLogger

from keras.callbacks import TensorBoard

from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

from keras.models import load\_model

from unet\_model\_building import \*

from dataset import LabeledImageDataset

from tensorboardX import SummaryWriter

import os

def \_batch\_generator(batch\_size, dset):

while True:

x = []

y = []

for i in range(dset.\_\_len\_\_()):

print(dset.\_pairs[i][0])

xy = dset.get\_myexample(i)

x.append(xy[0].transpose([1, 2, 0]))

y.append(np.expand\_dims(xy[1], axis=0).transpose([1, 2, 0]))

if len(y) >= int(args.batchsize):

yield np.array(x),np.array(y)

x = []

y = []

def train\_model(args):

assert (args.tcrop % 16 == 0) and (args.vcrop % 16 == 0), "tcrop and vcrop must be divisible by 16."

print('GPU: {}'.format(args.gpu))

print('# Minibatch-size: {}'.format(args.batchsize))

print('# Crop-size: {}'.format(args.tcrop))

print('# epoch: {}'.format(args.epoch))

print('')

this\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

models\_dir = os.path.normpath(os.path.join(this\_dir, "../../models"))

log\_dir = os.path.join(models\_dir, args.out)

writer = SummaryWriter(log\_dir=log\_dir)

# Set up a neural network to train

# Classifier reports softmax cross entropy loss and accuracy at every

# iteration, which will be used by the PrintReport extension below.

model = unet\_model(im\_sz=400)

model = load\_model(models\_dir + '/weights.049-0.098.hdf5')

model.compile(optimizer = Adam(lr = 1e-4), loss = 'binary\_crossentropy', metrics = ['accuracy'])

model\_checkpoint = ModelCheckpoint(models\_dir + '/weights.{epoch:03d}-{val\_loss:.3f}.hdf5', monitor='val\_loss', save\_best\_only=True)

csv\_logger = CSVLogger('log\_unet.csv', append=True, separator=';')

tensorboard = TensorBoard(log\_dir='./tensorboard\_unet/', write\_graph=True, write\_images=True)

model.summary()

# Load mean image

mean = np.load(os.path.join(args.dataset, "mean.npy"))

# Load the dataset # Trained and tested on same dataset

train = LabeledImageDataset(os.path.join(args.dataset, "mydata.txt"), args.images, args.labels,

mean=mean, crop\_size=args.tcrop, test=False, distort=False)

print(train.get\_example(1)[0].shape)

print(train.get\_example(1)[1].shape)

test = LabeledImageDataset (os.path.join(args.dataset, "mydata.txt"), args.images, args.labels,

mean=mean, crop\_size=args.vcrop, test=True, distort=False)

print("Training")

history = model.fit\_generator(generator = \_batch\_generator(args.batchsize, train), steps\_per\_epoch = train.\_\_len\_\_()/int(args.batchsize), epochs = args.epoch, validation\_data=\_batch\_generator(args.batchsize,test), validation\_steps=test.\_\_len\_\_()/int(args.batchsize) ,callbacks=[csv\_logger,model\_checkpoint])

model.save("weights\_last.hdf5")

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument('dataset', help='Path to directory containing train.txt, val.txt, and mean.npy')

parser.add\_argument('images', help='Root directory of input images')

parser.add\_argument('labels', help='Root directory of label images')

parser.add\_argument('--batchsize', '-b', type=int, default=1,

help='Number of images in each mini-batch')

parser.add\_argument('--test-batchsize', '-B', type=int, default=4,

help='Number of images in each test mini-batch')

parser.add\_argument('--epoch', '-e', type=int, default=50,

help='Number of sweeps over the dataset to train')

parser.add\_argument('--frequency', '-f', type=int, default=1,

help='Frequency of taking a snapshot')

parser.add\_argument('--gpu', '-g', type=int, default=-1,

help='GPU ID (negative value indicates CPU)')

parser.add\_argument('--out', '-o', default='logs',

help='Directory to output the result under "models" directory')

parser.add\_argument('--resume', '-r', default='',

help='Resume the training from snapshot')

parser.add\_argument('--noplot', dest='plot', action='store\_false',

help='Disable PlotReport extension')

parser.add\_argument('--tcrop', type=int, default=400,

help='Crop size for train-set images')

parser.add\_argument('--vcrop', type=int, default=400,

help='Crop size for validation-set images')

args = parser.parse\_args()

train\_model(args)