import os  
#os.environ["CUDA\_VISIBLE\_DEVICES"] = "0"  
import numpy as np  
import tensorflow  
  
from keras.models import \*  
from keras.layers import Input, concatenate, Conv2D, MaxPooling2D, UpSampling2D, Dropout, Cropping2D  
from keras.optimizers import \*  
from keras.callbacks import ModelCheckpoint, LearningRateScheduler  
from keras import backend as keras  
from data import \*  
  
  
# import data  
class myUnet(object):  
  
 def \_\_init\_\_(self, img\_rows=512, img\_cols=512):  
 self.img\_rows = img\_rows  
 self.img\_cols = img\_cols  
  
 def load\_train\_data(self,file\_path):  
 mydata = dataProcess(self.img\_rows, self.img\_cols)  
 imgs\_train, imgs\_mask\_train = mydata.load\_train\_data(file\_path)  
 return imgs\_train, imgs\_mask\_train  
  
 def get\_unet(self):  
 inputs = Input((self.img\_rows, self.img\_cols, 1))  
  
 conv1 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(inputs)  
 print("conv1 shape:", conv1.shape)  
 conv1 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv1)  
 print("conv1 shape:", conv1.shape)  
 pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)  
 print("pool1 shape:", pool1.shape)  
  
 conv2 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool1)  
 print("conv2 shape:", conv2.shape)  
 conv2 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv2)  
 print("conv2 shape:", conv2.shape)  
 pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)  
 print("pool2 shape:", pool2.shape)  
  
 conv3 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool2)  
 print("conv3 shape:", conv3.shape)  
 conv3 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv3)  
 print("conv3 shape:", conv3.shape)  
 pool3 = MaxPooling2D(pool\_size=(2, 2))(conv3)  
 print("pool3 shape:", pool3.shape)  
  
 conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool3)  
 conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv4)  
 drop4 = Dropout(0.5)(conv4)  
 pool4 = MaxPooling2D(pool\_size=(2, 2))(drop4)  
  
 conv5 = Conv2D(1024, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool4)  
 conv5 = Conv2D(1024, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv5)  
 drop5 = Dropout(0.5)(conv5)  
  
 up6 = Conv2D(512, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(  
 UpSampling2D(size=(2, 2))(drop5))  
 merge6 = concatenate([drop4, up6], axis=3)  
 conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge6)  
 conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv6)  
  
 up7 = Conv2D(256, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(  
 UpSampling2D(size=(2, 2))(conv6))  
 merge7 = concatenate([conv3, up7], axis=3)  
 conv7 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge7)  
 conv7 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv7)  
  
 up8 = Conv2D(128, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(  
 UpSampling2D(size=(2, 2))(conv7))  
 merge8 = concatenate([conv2, up8], axis=3)  
 conv8 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge8)  
 conv8 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv8)  
  
 up9 = Conv2D(64, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(  
 UpSampling2D(size=(2, 2))(conv8))  
 merge9 = concatenate([conv1, up9], axis=3)  
 conv9 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge9)  
 conv9 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv9)  
 conv9 = Conv2D(2, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv9)  
 conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)  
  
 model = Model(input=inputs, output=conv10)  
  
 model.compile(optimizer=Adam(lr=1e-4), loss='binary\_crossentropy', metrics=['accuracy'])  
  
 return model  
  
 def train(self,file\_path):  
 print("loading data")  
 imgs\_train, imgs\_mask\_train = self.load\_train\_data(file\_path)  
 print("loading data done")  
  
 #model = load\_model("unet.h5")  
  
 model = self.get\_unet()  
 print("got unet")  
  
 model\_checkpoint = ModelCheckpoint('unet.hdf5', monitor='loss', verbose=1, save\_best\_only=True)  
 print('Fitting model...')  
 model.fit(imgs\_train, imgs\_mask\_train, batch\_size=4, nb\_epoch=5, verbose=1, validation\_split=0.2, shuffle=True,  
 callbacks=[model\_checkpoint])  
  
 model.save("unet.h5")  
 print("fitting done ")  
  
  
 def prediect(self,file\_path):  
 mydata = dataProcess(self.img\_rows, self.img\_cols)  
 imgs\_test= mydata.load\_test\_data(file\_path)  
 print('predict test data')  
 imgs\_mask\_test = model.predict(imgs\_test, batch\_size=1, verbose=1)  
 np.save('imgs\_mask\_test.npy', imgs\_mask\_test)  
  
 def save\_img(self):  
 print("array to image")  
 imgs = np.load('imgs\_mask\_test.npy')  
 for i in range(imgs.shape[0]):  
 img = imgs[i]  
 img = array\_to\_img(img)  
 img.save("results/%d.jpg" % (i))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 files = "train/10"  
 myunet = myUnet()  
 file\_path=""  
 for i in range(9,10):  
 if i<10:  
 file\_path=files+"0"+str(i)  
 else:  
 if i==75:  
 i=i+1  
 file\_path=files+str(i)  
  
 myunet.train(file\_path)  
 #myunet.predict(test\_path)  
 #myunet.save\_img()