#import tensorflow as tf

import cv2

import os

import numpy as np

from keras.layers.core import Flatten, Dense, Dropout, Reshape

from keras.models import Model

from keras.layers import Input, ZeroPadding2D, Dropout

from keras import optimizers

from keras.optimizers import SGD

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import EarlyStopping

from keras.applications.vgg16 import VGG16

TRAIN\_DIR = 'train/'

TEST\_DIR = 'test/'

v = 'v/'

BATCH\_SIZE = 32

NUM\_EPOCHS = 5

def crop\_img(img, h, w):

h\_margin = (img.shape[0] - h) // 2 if img.shape[0] > h else 0

w\_margin = (img.shape[1] - w) // 2 if img.shape[1] > w else 0

crop\_img = img[h\_margin:h + h\_margin,w\_margin:w + w\_margin,:]

return crop\_img

def subtract\_gaussian\_blur(img):

return cv2.addWeighted(img, 4, cv2.GaussianBlur(img, (0, 0), 5), -4, 128)

def ReadImages(Path):

LabelList = list()

ImageCV = list()

classes = ["nonPdr", "pdr"]

# Get all subdirectories

FolderList = [f for f in os.listdir(Path) if not f.startswith('.')]

# Loop over each directory

for File in FolderList:

for index, Image in enumerate(os.listdir(os.path.join(Path, File))):

# Convert the path into a file

ImageCV.append(cv2.resize(cv2.imread(os.path.join(Path, File) + os.path.sep + Image), (224,224)))

#ImageCV[index]= np.array(ImageCV[index]) / 255.0

LabelList.append(classes.index(os.path.splitext(File)[0]))

img\_crop = crop\_img(ImageCV[index].copy(), 224, 224)

ImageCV[index] = subtract\_gaussian\_blur(img\_crop.copy())

return ImageCV, LabelList

data, labels = ReadImages(TRAIN\_DIR)

valid, vlabels = ReadImages(TEST\_DIR)

vgg16\_model = VGG16(weights="imagenet", include\_top=True)

# (2) remove the top layer

base\_model = Model(input=vgg16\_model.input,

output=vgg16\_model.get\_layer("block1\_pool").output)

print(base\_model)

# (3) attach a new top layer

base\_out = base\_model.output

base\_out = Reshape(25088,)(base\_out)

top\_fc1 = Dropout(0.5)(base\_out)

# output layer: (None, 5)

top\_preds = Dense(1, activation="sigmoid")(top\_fc1)

# (4) freeze weights until the last but one convolution layer (block4\_pool)

for layer in base\_model.layers[0:4]:

layer.trainable = False

# (5) create new hybrid model

model = Model(input=base\_model.input, output=top\_preds)

# (6) compile and train the model

sgd = SGD(lr=1e-4, momentum=0.9)

model.compile(optimizer=sgd, loss="binary\_crossentropy", metrics=["accuracy"])

data = np.asarray(data)

valid = np.asarray(valid)

data = data.astype('float32')

valid = valid.astype('float32')

data /= 255

valid /= 255

labels = np.array(labels)

datagen = ImageDataGenerator(

featurewise\_center=True,

featurewise\_std\_normalization=True,

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

horizontal\_flip=True)

# compute quantities required for featurewise normalization

# (std, mean, and principal components if ZCA whitening is applied)

datagen.fit(data)

mean = datagen.mean

std = datagen.std

print(mean, "mean")

print(std, "std")

es = EarlyStopping(monitor='val\_loss', verbose=1)

# fits the model on batches with real-time data augmentation:

model.fit\_generator(datagen.flow(data, np.array(labels), batch\_size=32),

steps\_per\_epoch=len(data) / 32, epochs=15,

validation\_data=(valid, np.array(vlabels)),

nb\_val\_samples=72, callbacks=[es])

model.save('model.h5')