Гайворонский Сергей Игоревич, группа 2.1

Лабораторная работа №2

**Задание**

Поиск шумовых объектов на изображении

**Цель**

Научиться работать со сверточными нейронными сетями и реализовать алгоритм с их помощью

**Код программы**

import os

import cv2

import numpy as np

import keras

from keras import backend as K

from keras.layers import Activation, Dropout, Flatten, Dense

from keras import optimizers

from keras.models import Sequential, load\_model

from keras.layers import Dense, Dropout, Conv2D,MaxPooling2D,Flatten

from keras.utils import plot\_model

from google.colab.patches import cv2\_imshow

def preprocess(img):

    img = cv2.normalize(img, None, alpha=0, beta=1, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_32F)

    R = img[:,:,0]

    G = img[:,:,1]

    B = img[:,:,2]

    RGBshifted = np.dstack( (

    np.roll(R, 10, axis=0),

    np.roll(G, 10, axis=1),

    np.roll(B, -10, axis=0)

    ))

    mean = 0.0

    std = 1.0

    noisy\_img = img + np.random.normal(mean, std, img.shape)

    noisy\_img\_clipped = np.clip(noisy\_img, 0, 255)

    return noisy\_img\_clipped

def loadData():

    labelsTrain = []

    ptrnsTrain = []

    path = "drive/MyDrive/photo/train/bad"

    for fileName in os.listdir(path):

        img = cv2.imread(path + '/' + fileName)

        img = preprocess(img)

        ptrnsTrain.append(img)

        labelsTrain.append(0)

    path = "drive/MyDrive/photo/train/good"

    for fileName in os.listdir(path):

        img = cv2.imread(path + '/' + fileName)

        img = preprocess(img)

        ptrnsTrain.append(img)

        labelsTrain.append(1)

    ptrnsTrain = np.array(ptrnsTrain)

    labelsTrain = np.array(labelsTrain)

    rndState = np.random.get\_state()

    np.random.set\_state(rndState)

    np.random.shuffle(ptrnsTrain)

    np.random.set\_state(rndState)

    np.random.shuffle(labelsTrain)

    return (ptrnsTrain, labelsTrain)

def createModel(wsize):

    if K.image\_data\_format() == 'channels\_first':

        input\_shape = (3, wsize, wsize)

    else:

        input\_shape = (wsize, wsize, 3)

    model = Sequential()

    model.add(Conv2D(16, kernel\_size=5, activation='relu', input\_shape=input\_shape))

    model.add(MaxPooling2D())

    model.add(Conv2D(8, kernel\_size=5, activation='relu'))

    model.add(MaxPooling2D())

    model.add(Dropout(0.5))

    model.add(Conv2D(8, kernel\_size=5, activation='relu'))

    model.add(MaxPooling2D())

    model.add(Flatten())

    model.add(Dense(2, activation='softmax'))

    model.compile(optimizer='adam', loss='mean\_squared\_error', metrics=['accuracy'])

    model.summary()

    return model

def trainClassifier(modelName, pathTrain, pathTest):

    (ptrnsTrain, labelsTrain) = loadData()

    ptrnsTrain = ptrnsTrain.reshape(ptrnsTrain.shape[0], wsize, wsize, 3)

    labelsTrain = keras.utils.to\_categorical(labelsTrain, numclasses)

    history = model.fit(ptrnsTrain, labelsTrain, batch\_size=40, epochs=25)

    plot\_model(model, to\_file='model.png', show\_shapes=True, show\_layer\_names=True)

    model.save(modelName)

    (ptrnsTest, labelsTest) = loadData()

    ptrnsTest  = ptrnsTest.reshape(ptrnsTest.shape[0], wsize, wsize, 3)

    labelsTest = keras.utils.to\_categorical(labelsTest, numclasses)

    evaluateRes = model.evaluate(ptrnsTest, labelsTest, verbose=0)

    print('Test loss:\t', evaluateRes[0])

    print('Test accuracy:\t', evaluateRes[1])

def testClassifier(modelName, imgName):

    model = load\_model(modelName)

    img = cv2.imread(imgName)

    print(img)

    #assert not isinstance(img,type(None))

    width = img.shape[1]

    height = img.shape[0]

    blocksW = int(width/wsize)

    blocksH = int(height/wsize)

    for x in range(blocksW):

        for y in range(blocksH):

            imgRoi = img[y\*wsize:y\*wsize+wsize, x\*wsize:x\*wsize+wsize]

            imgRoi = preprocess(imgRoi)

            ptrnTestX = []

            ptrnTestX.append(imgRoi)

            ptrnTestX = np.array(ptrnTestX)

            ptrnTestX = ptrnTestX.reshape(ptrnTestX.shape[0], wsize, wsize, 3)

            prediction = model.predict(ptrnTestX)

            #print('prediction ', prediction)

            if prediction[0,0] > prediction[0,1]:

                cv2.rectangle(img, (x\*wsize+1, y\*wsize+1), ((x+1)\*wsize-1, (y+1)\*wsize-1), color=(0,255,255), thickness=2)

    print('Marked')

    cv2\_imshow(img)

numclasses = 2

wsize = 50

pathTrain = "sample\_data/train"

pathTest  = "sample\_data/test"

modelName = "sample\_data/classifier.h5"

model = createModel(wsize)

trainClassifier(modelName, pathTrain, pathTest)

testClassifier(modelName, 'sample\_data/00076418.jpg')

**Результаты выполнения**

****

****

**Вывод**

В ходе выполнения данной лабораторной работы я научился работать со сверточной нейронной сетью и реализовал алгоритм нахождения шумовых объектов на изображении.