案例五：实时表情识别

1. 案例目的

• 掌握图像多分类；

• 掌握实时识别；

• 了解并应用人脸检测；

• 了解并应用模型训练过程的监控。

2. 案例内容

本实案例练一个表情识别模型，然后基于该模型实现实时的表情识别。

3. 案例知识点

• 卷积神经网络

• 图像分类

• 人脸检测

• 实时识别

• 监控模型训练过程

4. 案例时长

共4学时，具体安排如下：

• 编写人脸检测模块（1学时）

• 实现CNN核心类（0.5学时）

• 实现模型训练过程监控类（0.5学时）

• 训练模型（1学时）

• 实时表情识别（1学时）

5. 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）Tensorflow 2.3.0

3）Keras 2.4.3

**•开发环境与工具：**

1）Spyder 4

6. 案例分析

本案例主要分为以下5部分：

1）做表情识别首先要从图像中获取到所有的人脸图像。首先应该编写人脸检测模块找到人脸位置；

2）编写CNN核心类代码，用于做图像分类；

3）编写训练过程监控类，用于实时查看模型训练过程中的Accuracy值和Loss值；

4）训练模型，评估模型性能，修改超参数以提高性能，最后保存模型；

5）使用预训练模型和人脸检测做实时表情识别。

7. 案例实验过程

实时表情识别，可分为以下5个步骤：

1、编写人脸检测模块；

2、实现CNN核心类；

3、实现模型训练过程监控类；

4、训练模型；

5、实时表情识别。

7.1开发准备

在桌面上新建一个目录，名字叫facial-expression。本案例中所有的数据集和代码都放在该目录下。

7.1.1 打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至facial-expression目录下。

7.1.2 获取数据

本案例需要1个数据集，是人脸各种表情数据，即人脸各种表情的图片。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/6e24fd96e41a4b198067b1452f5de0dd.zip下载数据集，并解压。

解压后有2个文件（xinyuanjieyi.avi和facial\_expression\_fer.hdf5）和1个文件夹（dataset）。

dataset目录存放着数据集。

xinyuanjieyi.avi是一个视频文件，用于检验案例实验效果。

facial\_expression\_fer.hdf5是预训练的表情分类模型。

将dataset目录和xinyuanjieyi.avi复制到刚刚创建的facial-expression目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为facial-expression。使用Spyder新建5个Python文件，这5个Python文件的名字及用途如下：

faceutildlib.py：人脸检测的类，提供人脸检测功能。表情识别为什么和人脸检测扯上关系呢？因为表情识别模型的输入数据是人脸图片，而不是整张图片。所以，需要先通过人脸检测模块从整张图片中检测出人脸，再利用表情识别程序对人脸图片进行表情识别。

minivggnet.py：卷积神经网络的核心类，卷积神经网络用于表情识别，即表情图片分类。

test.py：测试表情识别程序。

train.py：训练表情识别程序。

trainingmonitor.py：训练表情识别模型的辅助类，用于实时画出loss和accuracy曲线图像，方便监督训练过程。

7.1.4 新建模型输出目录

在facial-expression目录下新建一个目录，命名为model。训练好的模型将存储在此目录下。

将【7.1.2 获取数据】小节中提到的模型预训练文件facial\_expression\_fer.hdf5复制到model目录下。

7.1.5 新建图像输出目录

在facial-expression目录下新建一个目录，命名为output。训练过程中生成的loss曲线图像和accuracy曲线图像将存储在此目录下。

至此，本次案例目录结构如图1所示。

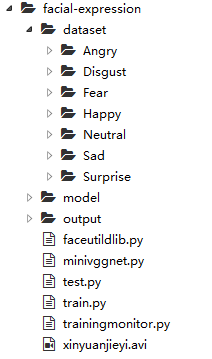


图1 案例目录结构

7.1.6 了解数据

数据源共有7个文件夹，共存有35887张图像。7个文件夹代表7种表情，其中Angry表示生气，Disgust表示厌恶，Fear表示害怕，Happy表示开心，Neutral表示正常表情，Sad表示伤心，Surprise表示吃惊。数据文件如图2、图3所示。

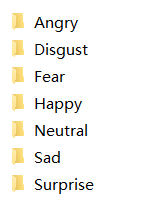


图2 7种表情的目录



图3 微笑表情目录下的图片

7.2编写人脸检测模块

在一幅画面中，首先使用人脸检测程序检测到人脸，然后将人脸部分截取出来再做表情识别。

如果一幅图片中有多张人脸，那么把所有的人脸都检测出来，然后对每一张人脸做表情识别。

打开faceutildlib.py。添加代码。

7.2.1 导入人脸检测库

【代码7-2-1】faceutildlib.py

# -\*- coding: utf-8 -\*-

'''

使用dlib实现人脸检测

'''

#%%

import face\_recognition

import cv2

#%%

执行代码块。

人脸检测有专门的神经网络模型，比如CascadeCNN、HyperFace等。这里不做人脸检测模型的训练，仅仅是调用。

在【搭建深度学习开发环境】案例中，有2个步骤是搭建人脸检测与人脸识别库。目前最主流的库是dlib，准确率达到99%以上。

由于dlib写代码不是特别友好，有人在dlib的基础上进一步封装，开发了face\_recognition库。目前face\_recognition库是人脸检测与人脸识别最简单的库。

补充：

1）dlib官方只支持Linux系统和Mac OS系统，不支持Windows系统。后面同学们会遇到越来越多的库不支持Windows系统。这也是很多人在Linux系统做深度学习的原因。

2）人脸检测与人脸识别不同。人脸检测是从图片中找到所有的人脸。人脸识别是在找到所有的人脸的基础上，再识别出来这张脸是谁。人脸检测与人脸识别的区别如图4所示。



图4 人脸检测vs人脸识别

7.2.2 定义人脸检测类

定义人脸检测的类，方便被其他模块调用。类包含2个属性，分别为检测方法与容忍度。代码如下：

【代码7-2-2-1】faceutildlib.py

#%%

class FaceUtil:

detection\_method = 'hog' # either 'hog' or 'cnn'. default is hog.

tolerance = 0.3

不要执行代码。

detection\_method参数：人脸检测方法，有2个可选参数，分别是hog和cnn。hog意味着使用传统机器学习模型做人脸检测；cnn意味着使用卷积神经网络模型做人脸检测。使用hog速度比较快但准确率较低，使用cnn速度较慢但准确率比较高。

tolerance参数：两张脸之间有多少距离才算匹配。数值越低越严格。0.6是典型的最佳性能。

接下来定义类的方法，该方法返回检测到的所有人脸的坐标。坐标用(left, top, right, bottom)表示。代码如下：

【代码7-2-2-2】faceutildlib.py

# face detection

def get\_face\_location(self, image):

face\_location\_list = []

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)#转为灰度图像以加快检测速度，因为人脸检测不需要彩色图像

face\_locations = face\_recognition.face\_locations(

gray,number\_of\_times\_to\_upsample=1,

model = self.detection\_method)

# 人脸位置

for (top, right, bottom, left) in face\_locations:

face\_location\_list.append((left, top, right, bottom))

return face\_location\_list

#%%

执行代码块。

7.2.3 查看人脸检测效果

最后在main函数中调用人脸检测类，查看效果。代码如下：

【代码7-2-3】faceutildlib.py

#%%

if \_\_name\_\_ == "\_\_main\_\_":

input\_video = 'xinyuanjieyi.avi' # 视频路径

camera = cv2.VideoCapture(input\_video) # 读取视频

faceutil = FaceUtil() # 实例化人脸检测类

# 遍历视频的每一帧

while True:

# 得到1帧

(grabbed, frame) = camera.read()

# 如果视频结束了，终止程序

if not grabbed:

break

# 得到图片中所有的人脸

face\_location\_list = faceutil.get\_face\_location(frame)

# 在人脸上画矩形

for (left, top, right, bottom) in face\_location\_list:

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# 实时显示处理后的视频

cv2.imshow("Face Detection", frame)

# 按'ESC'键随时退出程序

k = cv2.waitKey(100) & 0xff

if k == 27:

break

# 程序执行完毕后释放资源

camera.release()

cv2.destroyAllWindows()

#%%

执行代码块。

程序执行效果如图5所示。

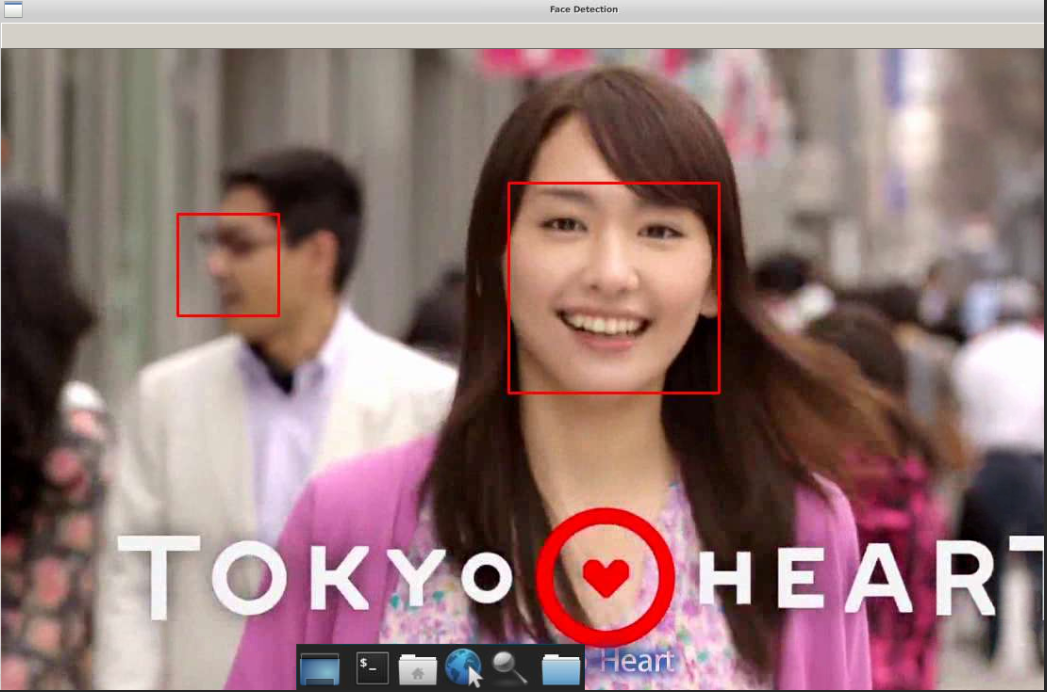


图5 人脸检测效果

鼠标点击视频任意位置，然后按下ESC键退出程序。

同学们可以改变detection\_method参数和tolerance参数，查看效果。

人脸检测模块结束。

7.3 实现CNN核心类

CNN架构使用MiniVGGNet。

打开minivggnet.py。添加代码如下：

【代码7-3】minivggnet.py

# import the necessary packages

from keras.models import Sequential

from keras.layers.normalization import BatchNormalization

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.layers.core import Activation

from keras.layers.core import Flatten

from keras.layers.core import Dropout

from keras.layers.core import Dense

class MiniVGGNet:

@staticmethod

def build(width, height, depth, classes):

# initialize the model along with the input shape to be

# "channels last" and the channels dimension itself

model = Sequential()

inputShape = (height, width, depth)

chanDim = -1

# 输入层 + 第1层 CONV => RELU => CONV => RELU => POOL 层

model.add(Conv2D(32, (3, 3), padding="same",

input\_shape=inputShape))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(32, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# 第2层 CONV => RELU => CONV => RELU => POOL 层

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# 第1层FC => RELU 层

model.add(Flatten())

model.add(Dense(512))

model.add(Activation("relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

# 输出层

model.add(Dense(classes))

model.add(Activation("softmax"))

# 返回模型架构

return model

不要执行代码，别的模块会调用它。

注意：这个类在别的案例中已经编写过多次了，可以直接拿来用。不过调用时一定要看清楚参数的顺序。

7.4实现模型训练过程监控类

在模型的训练中，有必要对模型的loss和accuracy实时监控，这样可以实时了解模型训练的状况。

比如，模型计划训练100个epoch，在训练了20个epoch后，发现training accuracy和validation accuracy差距越来越大，说明模型过拟合，那么应该立即终止模型。

这个类不需要掌握实现，只需要会使用即可。

打开trainingmonitor.py。添加如下代码：

【代码7-4】trainingmonitor.py

# 导入包

from keras.callbacks import BaseLogger

import matplotlib.pyplot as plt

import numpy as np

import json

import os

class TrainingMonitor(BaseLogger):

def \_\_init\_\_(self, figPath, jsonPath=None, startAt=0):

# store the output path for the figure, the path to the JSON

# serialized file, and the starting epoch

super(TrainingMonitor, self).\_\_init\_\_()

self.figPath = figPath

self.jsonPath = jsonPath

self.startAt = startAt

# epoch开始时调用

def on\_train\_begin(self, logs={}):

'''

is called once the training process starts:

'''

# initialize the history dictionary

self.H = {}

# if the JSON history path exists, load the training history

if self.jsonPath is not None:

if os.path.exists(self.jsonPath):

self.H = json.loads(open(self.jsonPath).read())

# check to see if a starting epoch was supplied

if self.startAt > 0:

# loop over the entries in the history log and

# trim any entries that are past the starting

# epoch

for k in self.H.keys():

self.H[k] = self.H[k][:self.startAt]

# epoch结束时调用

def on\_epoch\_end(self, epoch, logs={}):

# loop over the logs and update the loss, accuracy, etc.

# for the entire training process

for (k, v) in logs.items():

l = self.H.get(k, [])

l.append(v)

self.H[k] = l

# check to see if the training history should be serialized

# to file

if self.jsonPath is not None:

f = open(self.jsonPath, "w")

f.write(json.dumps(self.H))

f.close()

# ensure at least two epochs have passed before plotting

# (epoch starts at zero)

if len(self.H["loss"]) > 1:

# plot the training loss and accuracy

N = np.arange(1, len(self.H["loss"])+1)

plt.style.use("ggplot")

loss\_plot\_file = os.path.splitext(self.figPath)[0] + '\_loss' + os.path.splitext(self.figPath)[1]

accuracy\_plot\_file = os.path.splitext(self.figPath)[0] + '\_accuracy' + os.path.splitext(self.figPath)[1]

# 画出loss曲线

plt.figure()

plt.plot(N, self.H["loss"], label="train\_loss")

plt.plot(N, self.H["val\_loss"], label="val\_loss")

plt.title("Training Loss [Epoch {}]".format(len(self.H["loss"])))

plt.xlabel("Epoch #")

plt.ylabel("Loss")

plt.legend()

# 将loss曲线保存

plt.savefig(loss\_plot\_file)

plt.close()

# 画出accuracy曲线

plt.figure()

plt.plot(N, self.H["accuracy"], label="train\_acc")

plt.plot(N, self.H["val\_accuracy"], label="val\_acc")

plt.title("Training Accuracy [Epoch {}]".format(len(self.H["loss"])))

plt.xlabel("Epoch #")

plt.ylabel("Accuracy")

plt.legend()

# 将accuracy曲线保存

plt.savefig(accuracy\_plot\_file)

plt.close()

不要执行代码。其他模块会调用它。

7.5训练模型

模型训练时间较长，在Tesla P40 GPU上训练，需要40分钟左右。在CPU上需要训练数个小时。

当在CPU上训练时，可以手动终止程序。如何下载预训练模型已在【7.1.6 下载预训练模型】小节中提到。

由于这是一个新的py文件，请重启Spyder的kernel或者新建一个console。

打开train.py。添加代码。

7.5.1 导入库

【代码7-5-1】train.py

# -\*- coding: utf-8 -\*-

'''

训练表情识别模型

'''

#%%

import numpy as np

import random

import cv2

import os

from imutils import paths

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer

from keras.preprocessing.image import ImageDataGenerator

from keras.preprocessing.image import img\_to\_array

from keras.optimizers import SGD

from minivggnet import MiniVGGNet

from sklearn.metrics import classification\_report

from trainingmonitor import TrainingMonitor

#%%

执行代码块。

7.5.2 定义全局变量

全局变量有3个，分别是数据集路径，模型保存路径，accuracy曲线与loss曲线保存路径。

【代码7-5-2】train.py

#%%

# 全局变量

dataset\_path = 'dataset'

output\_model\_path = 'model/facial\_expression\_fer.hdf5'

output\_accuracy\_loss\_path = 'output/facial\_expression.png'

#%%

执行代码块。

从现在开始，建议切换到Spyder explorer面板，查看变量信息。

7.5.3 定义全局常量

定义全局变量。这些变量都是神经网络模型的超参数。如果模型性能不佳可以调整这些参数进行优化。

把这些参数放在一起便于管理。

【代码7-5-3】train.py

#%%

# 全局常量

LR = 0.01

TARGET\_WIDTH = 48

TARGET\_HEIGHT = 48

BATCH\_SIZE = 32

EPOCHS = 200

DECAY = LR/EPOCHS

MOMENTUM = 0.9

#%%

执行代码块。

7.5.4 获取数据集

定义变量data，用于存储图片像素数据。定义变量labels，用于存储图像类别（生气、开心、惊讶……）。data可以称为自变量，labels可以称为因变量。

【代码7-5-4】train.py

#%%

# 得到数据集

# initialize data and labels

print('[INFO] loading images...')

data = []

labels = []

#%%

执行代码块。

7.5.5 打乱数据集顺序

【代码7-5-5】train.py

#%%

# grab the image paths and randomly shuffle them

image\_paths = sorted(list(paths.list\_images(dataset\_path)))# 存储图片路径

random.seed(42)

random.shuffle(image\_paths) # 打乱顺序

#%%

执行代码块。

7.5.6 加载图片

【代码7-5-6】train.py

#%%

# 加载图片

counter = 1

for image\_path in image\_paths:

if counter % 500 == 0:#每加载500张图片便打印出信息，以免用户等太久着急

print('[INFO] loading %d/%d images.' %(counter, len(image\_paths)))

image = cv2.imread(image\_path)# 读取图片数据

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)#转换为灰度图像，加快训练速度。数据集本身就是灰度图像

gray = cv2.resize(gray, (TARGET\_HEIGHT, TARGET\_WIDTH))#处理成模型需要的尺寸

gray = img\_to\_array(gray)#像素数据由unit8类型转换为float32类型

data.append(gray)

# 提取图片的类别

label = image\_path.split(os.path.sep)[-2]

labels.append(label)

counter += 1

#%%

执行代码块。

加载图片的效果如图6所示。

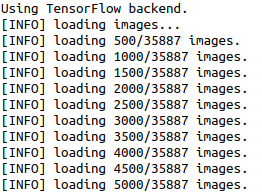


图6 加载图片

图片加载完毕后，data变量存储着图片像素数据，labels变量存储着图片标签数据。data变量的值如图7所示，labels变量的值如图8所示。

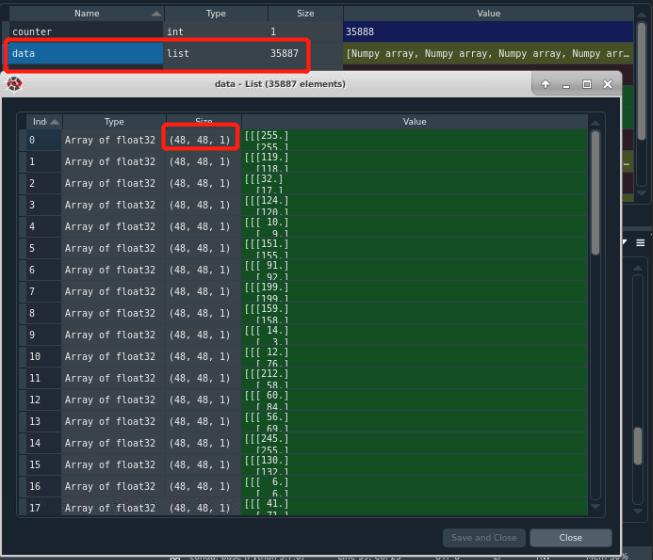


图7 data变量存储着图片像素数据

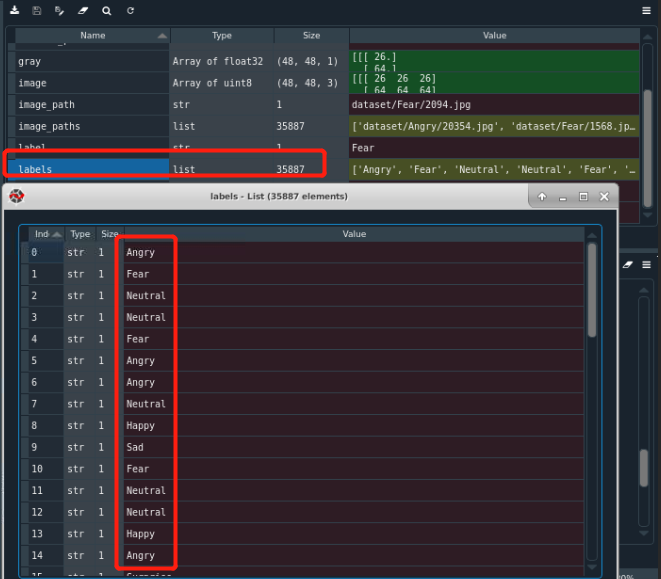


图8 labels变量存储着标签信息

7.5.7 数据标准化

【代码7-5-7】train.py

#%%

# scale the raw pixel intensities to the range [0,1]

data = np.array(data, dtype = 'float')/255.0

#%%

执行代码块。

此时data变量的值在[0, 1]之间。

上面提到，data相当于自变量。处理完自变量，下一步处理因变量。

7.5.8 处理因变量

由于CNN网络的输出层使用的是softmax激活函数，所以因变量的列数应该和类别数相同。即：因变量应该有7列。

处理因变量之前，labels变量只有1列，如上述图8所示。

处理因变量的代码如下：

【代码7-5-8-1】train.py

#%%

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

#%%

执行代码块。

此时，labels变量有7列，它的值如图9所示。

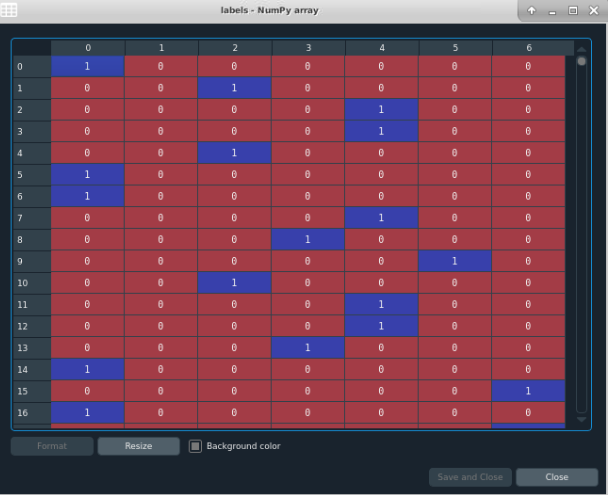


图9 labels变量有7列

标签编码后，必须知道标签和编码的对应关系。代码如下：

【代码7-5-8-2】train.py

#%%

print(lb.classes\_)

#%%

执行代码块。

输出结果如图10所示。



图10 标签和编码的对应关系

由上图可知，Angry对应0，Disgust对应1，Fear对应2，Happy对应3，Neutral对应4，Sad对应5，Surprise对应6。

7.5.9 拆分数据集

将数据集拆分成2部分，75%是训练集，25%是测试集。代码如下：

【代码7-5-9】train.py

#%%

# partition the data into training and testing splits using 75% of the data for training and the remaining 25% for testing

(trainX, testX, trainy, testy) = train\_test\_split(data, labels, test\_size = 0.25, random\_state = 42)

#%%

执行代码块。

训练集有26915条，测试集有8972条。

7.5.10 数据增强

这是一个非常重要的步骤。这里并不是人为的处理图片，而是在模型训练过程中程序自动对图片做适当的处理，如垂直翻转、旋转、缩放等操作。即使图片拍得有点歪，也没关系，训练过程中会自动旋转。整个过程叫数据增强(data augmentation)。数据增强的代码如下：

【代码7-5-10】train.py

#%%

# data augmentation

aug = ImageDataGenerator(rotation\_range=30, width\_shift\_range=0,

height\_shift\_range=0, shear\_range=0, zoom\_range=0.2,

horizontal\_flip=True, fill\_mode="nearest")

#%%

执行代码块。

请仔细观察数据增强的参数。其中width\_shift\_range和height\_shift\_range的值都是0。这是因为人的表情本来区别就很小，一张高兴的图片拉伸一下，可能就是生气的表情了。

7.5.11 编译模型

编译模型，优化算法选择SGD，loss值选择二元交叉熵，评估指标选择accuracy。

优化算法选择SGD，一般需要手动指定learning rate、decay、momentum、nesterov的值，甚至需要不断调整这些超参数。在全局常量中，本案例为这些超参数设置的值是不断调整之后得到的，并不是凭运气一次就能设置准确。

模型编译的代码如下：

【代码7-5-11】train.py

#%%

# 编译模型

print("[INFO] compiling model...")

opt = SGD(lr=LR, decay=DECAY, momentum = MOMENTUM)

model = MiniVGGNet.build(width=TARGET\_WIDTH, height=TARGET\_HEIGHT, depth=1,

classes=7)

model.compile(loss="categorical\_crossentropy", optimizer=opt,

metrics=["accuracy"])

#%%

执行代码块。

7.5.12 设置回调函数

回调函数的作用是，每当执行完1个epoch，便做1个操作。常见的操作包括保存模型、输出loss/accuracy曲线等。

【代码7-5-12】train.py

#%%

# callbacks，用于监控模型训练状态

callbacks = [TrainingMonitor(output\_accuracy\_loss\_path)]

#%%

执行代码块。

由代码可见，callbacks的值的列表类型，所以可以添加多个回调。比如每当执行完1个epoch，便保存1次模型，同时打印出loss/accuracy曲线。

在本案例中，每当1个epoch结束，都会调用一次TrainingMonitor类的一些方法。

7.5.13 训练模型

本案例在8核CPU上训练需要30个小时，在Tesla P40 GPU上训练需要40分钟左右。

根据代码中设置的全局常量的值，模型训练200个epoch。且训练过程中使用了验证集，因此可以实时评估模型性能。模型训练信息存储到H变量中。

模型训练时调用aug.flow()方法，即使用数据增强技术。所以虽然trainX有26915条数据，但是真正训练模型时的训练集数据远远大于26915条。

训练模型的代码如下：

【代码7-5-13】train.py

#%%

# 训练模型

print("[INFO] training network...")

H = model.fit\_generator(aug.flow(trainX, trainy, batch\_size=BATCH\_SIZE),

validation\_data=(testX, testy), steps\_per\_epoch=len(trainX) // BATCH\_SIZE,

epochs=EPOCHS, callbacks = callbacks, verbose=1)

#%%

执行代码块。

模型训练信息如图11所示。

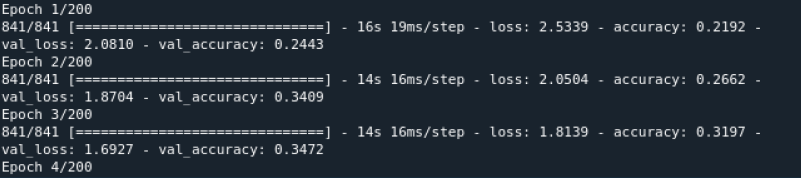


图11 训练模型

特别注意：请打开output目录下的2张图片（当1个epoch结束后才生成此图片），查看训练过程的实时accuracy和loss。正如上面提到的，如果train\_accuracy和test\_accuracy差距越来越大，说明过拟合发生，应该终止程序，调整超参数。

训练第10个epoch时，output目录下的2张图片（分别叫facial\_expression\_accuracy.png和facial\_expression\_loss.png）的效果如图12、图13所示。

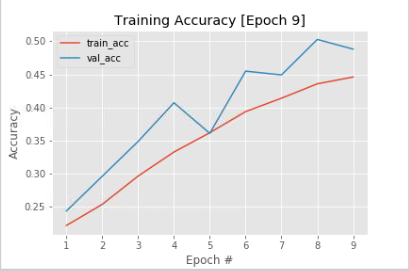


图12 训练第10个epoch时accuracy效果图



图13 训练第10个epoch时loss效果图

注意：由于神经网络初始会设置随机权重，所以同学们的训练结果不会跟本案例完全一致。

模型训练完毕后，output目录下的2张图片的效果如图14、图15所示。

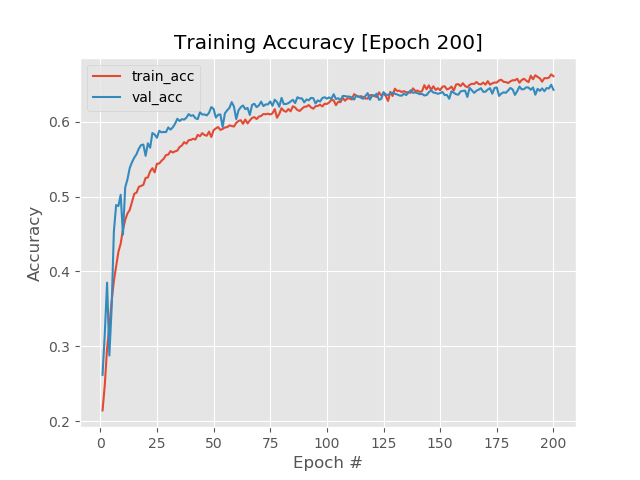


图14 训练完毕accuracy曲线

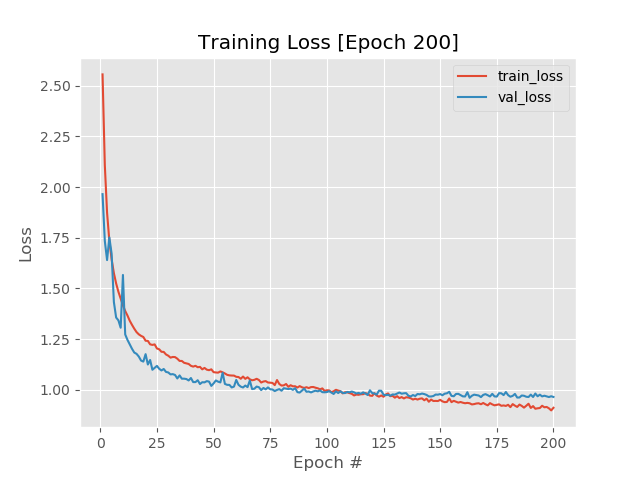


图15 训练完毕loss曲线

通过图14和图15发现，曲线最终收敛，说明模型训练完毕；训练集和验证集高度重合，说明模型没有过拟合。

模型训练完毕，不需要重新训练。接下来评估模型性能。

7.5.14 评估模型性能

通过accuracy曲线可以看出，整体准确率在65%左右。

现需要打印出每个分类的准确率，代码如下：

【代码7-5-14】train.py

#%%

# 评估模型

print("[INFO] evaluating network...")

predictions = model.predict(testX, batch\_size=32)

print(classification\_report(testy.argmax(axis=1),

predictions.argmax(axis=1), target\_names=list(lb.classes\_)))

#%%

执行代码块。

每个类别的准确率如图16所示。

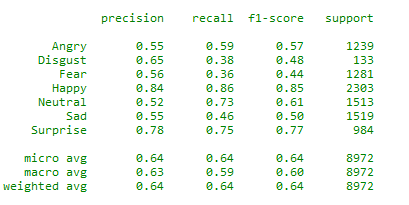


图16 模型准确率报告

通过上图发现，Happy和Surprise这2个类别的准确率较高（主要关注f1-score指标），f1-score分别是0.85和0.77。

有的同学可能觉得准确率还是太低，想继续提升准确率。请查阅相关的资料进行优化。

7.5.15 保存模型

【代码7-5-15】train.py

#%%

# 保存模型

print("[INFO] serializing network...")

model.save(output\_model\_path)

#%%

执行代码块。

模型保存到model/facial\_expression\_fer.hdf5中。

模型训练到此结束。

7.6实时表情识别

下面需要验证一下模型的性能。

由于这是一个新的py文件，请重启Spyder的kernel或者新建一个console。

打开test.py。添加代码。

7.6.1 导入库

【代码7-6-1】test.py

# -\*- coding: utf-8 -\*-

'''

测试表情识别模型

'''

#%%

# import the necessary packages

from keras.preprocessing.image import img\_to\_array

from keras.models import load\_model

from faceutildlib import FaceUtil

import numpy as np

import imutils

import cv2

import time

#%%

执行代码块。

7.6.2 设置全局变量

全局变量有2个，分别是训练好的模型路径和测试视频路径。

如果不指定视频文件，那么程序会打开摄像头采集视频。由于同学们用的是虚拟机，无法打开摄像头，所以input\_video变量必须赋值。

【代码7-6-2】test.py

#%%

# 全局变量

model\_path = 'model/facial\_expression\_fer.hdf5'#训练的模型

input\_video = 'xinyuanjieyi.avi'#视频文件

#%%

执行代码块。

7.6.3 设置全局常量

全局常量有3个，分别为模型需要的图片尺寸（宽度和高度）、编码与表情的对应关系。（在训练模型时，将因变量进行了字符编码，接着转换为了哑变量）。代码如下：

【代码7-6-3】test.py

#%%

# 全局常量

FACIAL\_EXPRESSION\_TARGET\_WIDTH = 48

FACIAL\_EXPRESSION\_TARGET\_HEIGHT = 48

FACIAL\_EXPRESSION\_MAP = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Neutral', 5:'Sad', 6:'Surprise'}

#%%

执行代码块。

7.6.4 加载模型

加载训练好的模型，代码如下：

【代码7-6-4】test.py

#%%

# load the face detector cascade and smile detector CNN

model = load\_model(model\_path)

#%%

执行代码块。

7.6.5 初始化视频类

当input\_video变量的值是空时，便打开摄像头捕捉视频。当input\_video变量的值不是空时，便读取视频文件的内容。代码如下：

【代码7-6-5】test.py

#%%

# if a video path was not supplied, grab the reference to the webcam

if not input\_video:

camera = cv2.VideoCapture(0)

time.sleep(2)

else:

camera = cv2.VideoCapture(input\_video)

#%%

执行代码块。

7.6.6 实例化人脸检测类

【代码7-6-6】test.py

#%%

faceutil = FaceUtil()

#%%

执行代码块。

7.6.7 实时表情识别

对视频的每一帧进行表情识别。

首先检测人脸，接下来将人脸数据传递给表情识别模型进行识别。代码如下：

【代码7-6-7】test.py

#%%

# 遍历视频的每一帧

while True:

# 得到当前帧

(grabbed, frame) = camera.read()

# 如果视频文件播放完了

if input\_video and not grabbed:

break

# 如果通过摄像头捕捉视频，那么把画面左右翻转

if not input\_video:

frame = cv2.flip(frame, 1)

# 设置视频的尺寸（一般是减小尺寸，以提高识别速度）

frame = imutils.resize(frame, width=600)

# 调用人脸检测类，识别人脸

face\_location\_list = faceutil.get\_face\_location(frame)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)#将彩色帧转换为灰度图像

# 遍历每一张人脸的坐标

for (left, top, right, bottom) in face\_location\_list:

roi = gray[top:bottom, left:right] # 从图像中截取出人脸区域

roi = cv2.resize(roi, (FACIAL\_EXPRESSION\_TARGET\_WIDTH,

FACIAL\_EXPRESSION\_TARGET\_HEIGHT))#将人脸区域的尺寸设置成模型需要的

roi = roi.astype("float") / 255.0#数据标准化

roi = img\_to\_array(roi)#元素类型由uint8转换为float32

roi = np.expand\_dims(roi, axis=0)#扩展成4维数据，模型需要的参数就是4维的

# 得到预测的表情

label = FACIAL\_EXPRESSION\_MAP[model.predict(roi)[0].argmax(axis=0)]

# 把人脸框出来，并且把表情写在框的上面

cv2.putText(frame, label, (left, top - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 255), 2)

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# 实时显示处理后的视频

cv2.imshow("Facial Expression Detect", frame)

# 按ESC键随时退出视频

k = cv2.waitKey(100) & 0xff

if k == 27:

break

# 释放资源

camera.release()

cv2.destroyAllWindows()

#%%

执行代码块。

运行效果如图17所示。

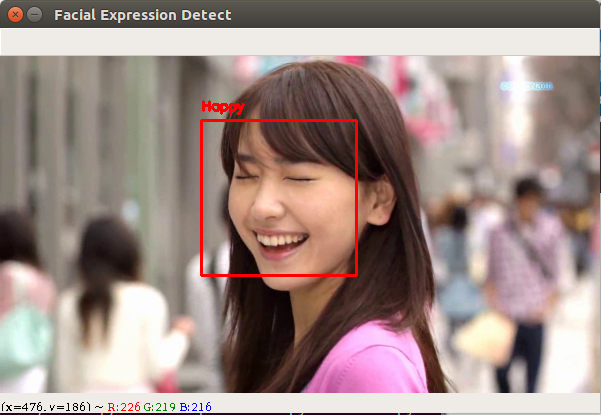


图17 程序运行效果

鼠标点击视频任意位置，然后按下ESC键退出程序。

注意：人脸检测类中的detection\_method属性的值默认是hog。如果同学们改为cnn，那么视频资源要加载到GPU显存中，可能造成GPU显存不足的情况。报错信息如图18所示。

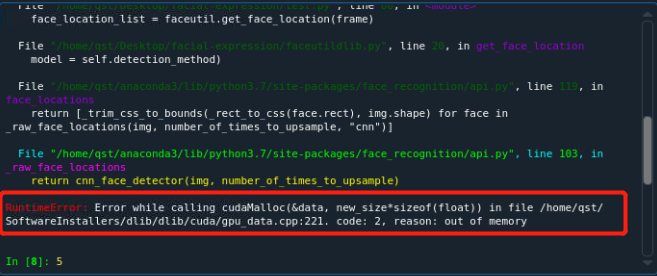


图18 GPU显存不足

结论：人脸检测+表情分类实现了实时表情检测功能。

目前有些机构把此功能部署在教室里，实时检测学生的表情，以分析学生的学习效果。这个做法有利有弊。总之图像分类和人脸检测在日常生活中发挥了重要的作用。

8. 案例代码

【案例代码】faceutildlib.py

# -\*- coding: utf-8 -\*-

'''

使用dlib实现人脸检测

'''

#%%

import face\_recognition

import cv2

#%%

#%%

class FaceUtil:

detection\_method = 'hog' # either 'hog' or 'cnn'. default is hog.

tolerance = 0.3

# face detection

def get\_face\_location(self, image):

face\_location\_list = []

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)#转为灰度图像以加快检测速度，因为人脸检测不需要彩色图像

face\_locations = face\_recognition.face\_locations(

gray,number\_of\_times\_to\_upsample=1,

model = self.detection\_method)

# 人脸位置

for (top, right, bottom, left) in face\_locations:

face\_location\_list.append((left, top, right, bottom))

return face\_location\_list

#%%

#%%

if \_\_name\_\_ == "\_\_main\_\_":

input\_video = 'xinyuanjieyi.avi' # 视频路径

camera = cv2.VideoCapture(input\_video) # 读取视频

faceutil = FaceUtil() # 实例化人脸检测类

# 遍历视频的每一帧

while True:

# 得到1帧

(grabbed, frame) = camera.read()

# 如果视频结束了，终止程序

if not grabbed:

break

# 得到图片中所有的人脸

face\_location\_list = faceutil.get\_face\_location(frame)

# 在人脸上画矩形

for (left, top, right, bottom) in face\_location\_list:

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# 实时显示处理后的视频

cv2.imshow("Face Detection", frame)

# 按'ESC'键随时退出程序

k = cv2.waitKey(100) & 0xff

if k == 27:

break

# 程序执行完毕后释放资源

camera.release()

cv2.destroyAllWindows()

#%%

【案例代码】minivggnet.py

# import the necessary packages

from keras.models import Sequential

from keras.layers.normalization import BatchNormalization

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.layers.core import Activation

from keras.layers.core import Flatten

from keras.layers.core import Dropout

from keras.layers.core import Dense

class MiniVGGNet:

@staticmethod

def build(width, height, depth, classes):

# initialize the model along with the input shape to be

# "channels last" and the channels dimension itself

model = Sequential()

inputShape = (height, width, depth)

chanDim = -1

# 输入层 + 第1层 CONV => RELU => CONV => RELU => POOL 层

model.add(Conv2D(32, (3, 3), padding="same",

input\_shape=inputShape))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(32, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# 第2层 CONV => RELU => CONV => RELU => POOL 层

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# 第1层FC => RELU 层

model.add(Flatten())

model.add(Dense(512))

model.add(Activation("relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

# 输出层

model.add(Dense(classes))

model.add(Activation("softmax"))

# 返回模型架构

return model

【案例代码】trainingmonitor.py

# 导入包

from keras.callbacks import BaseLogger

import matplotlib.pyplot as plt

import numpy as np

import json

import os

class TrainingMonitor(BaseLogger):

def \_\_init\_\_(self, figPath, jsonPath=None, startAt=0):

# store the output path for the figure, the path to the JSON

# serialized file, and the starting epoch

super(TrainingMonitor, self).\_\_init\_\_()

self.figPath = figPath

self.jsonPath = jsonPath

self.startAt = startAt

# epoch开始时调用

def on\_train\_begin(self, logs={}):

'''

is called once the training process starts:

'''

# initialize the history dictionary

self.H = {}

# if the JSON history path exists, load the training history

if self.jsonPath is not None:

if os.path.exists(self.jsonPath):

self.H = json.loads(open(self.jsonPath).read())

# check to see if a starting epoch was supplied

if self.startAt > 0:

# loop over the entries in the history log and

# trim any entries that are past the starting

# epoch

for k in self.H.keys():

self.H[k] = self.H[k][:self.startAt]

# epoch结束时调用

def on\_epoch\_end(self, epoch, logs={}):

# loop over the logs and update the loss, accuracy, etc.

# for the entire training process

for (k, v) in logs.items():

l = self.H.get(k, [])

l.append(v)

self.H[k] = l

# check to see if the training history should be serialized

# to file

if self.jsonPath is not None:

f = open(self.jsonPath, "w")

f.write(json.dumps(self.H))

f.close()

# ensure at least two epochs have passed before plotting

# (epoch starts at zero)

if len(self.H["loss"]) > 1:

# plot the training loss and accuracy

N = np.arange(1, len(self.H["loss"])+1)

plt.style.use("ggplot")

loss\_plot\_file = os.path.splitext(self.figPath)[0] + '\_loss' + os.path.splitext(self.figPath)[1]

accuracy\_plot\_file = os.path.splitext(self.figPath)[0] + '\_accuracy' + os.path.splitext(self.figPath)[1]

# 画出loss曲线

plt.figure()

plt.plot(N, self.H["loss"], label="train\_loss")

plt.plot(N, self.H["val\_loss"], label="val\_loss")

plt.title("Training Loss [Epoch {}]".format(len(self.H["loss"])))

plt.xlabel("Epoch #")

plt.ylabel("Loss")

plt.legend()

# 将loss曲线保存

plt.savefig(loss\_plot\_file)

plt.close()

# 画出accuracy曲线

plt.figure()

plt.plot(N, self.H["accuracy"], label="train\_acc")

plt.plot(N, self.H["val\_accuracy"], label="val\_acc")

plt.title("Training Accuracy [Epoch {}]".format(len(self.H["loss"])))

plt.xlabel("Epoch #")

plt.ylabel("Accuracy")

plt.legend()

# 将accuracy曲线保存

plt.savefig(accuracy\_plot\_file)

plt.close()

【案例代码】train.py

# -\*- coding: utf-8 -\*-

'''

训练表情识别模型

'''

#%%

import numpy as np

import random

import cv2

import os

from imutils import paths

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer

from keras.preprocessing.image import ImageDataGenerator

from keras.preprocessing.image import img\_to\_array

from keras.optimizers import SGD

from minivggnet import MiniVGGNet

from sklearn.metrics import classification\_report

from trainingmonitor import TrainingMonitor

#%%

#%%

# 全局变量

dataset\_path = 'dataset'

output\_model\_path = 'model/facial\_expression\_fer.hdf5'

output\_accuracy\_loss\_path = 'output/facial\_expression.png'

#%%

#%%

# 全局常量

LR = 0.01

TARGET\_WIDTH = 48

TARGET\_HEIGHT = 48

BATCH\_SIZE = 32

EPOCHS = 200

DECAY = LR/EPOCHS

MOMENTUM = 0.9

#%%

#%%

# 得到数据集

# initialize data and labels

print('[INFO] loading images...')

data = []

labels = []

#%%

#%%

# grab the image paths and randomly shuffle them

image\_paths = sorted(list(paths.list\_images(dataset\_path)))# 存储图片路径

random.seed(42)

random.shuffle(image\_paths) # 打乱顺序

#%%

#%%

# 加载图片

counter = 1

for image\_path in image\_paths:

if counter % 500 == 0:#每加载500张图片便打印出信息，以免用户等太久着急

print('[INFO] loading %d/%d images.' %(counter, len(image\_paths)))

image = cv2.imread(image\_path)# 读取图片数据

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)#转换为灰度图像，加快训练速度。数据集本身就是灰度图像

gray = cv2.resize(gray, (TARGET\_HEIGHT, TARGET\_WIDTH))#处理成模型需要的尺寸

gray = img\_to\_array(gray)#像素数据由unit8类型转换为float32类型

data.append(gray)

# 提取图片的类别

label = image\_path.split(os.path.sep)[-2]

labels.append(label)

counter += 1

#%%

#%%

# scale the raw pixel intensities to the range [0,1]

data = np.array(data, dtype = 'float')/255.0

#%%

#%%

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

#%%

#%%

print(lb.classes\_)

#%%

#%%

# partition the data into training and testing splits using 75% of the data for training and the remaining 25% for testing

(trainX, testX, trainy, testy) = train\_test\_split(data, labels, test\_size = 0.25, random\_state = 42)

#%%

#%%

# data augmentation

aug = ImageDataGenerator(rotation\_range=30, width\_shift\_range=0,

height\_shift\_range=0, shear\_range=0, zoom\_range=0.2,

horizontal\_flip=True, fill\_mode="nearest")

#%%

#%%

# 编译模型

print("[INFO] compiling model...")

opt = SGD(lr=LR, decay=DECAY, momentum = MOMENTUM)

model = MiniVGGNet.build(width=TARGET\_WIDTH, height=TARGET\_HEIGHT, depth=1,

classes=7)

model.compile(loss="categorical\_crossentropy", optimizer=opt,

metrics=["accuracy"])

#%%

#%%

# callbacks，用于监控模型训练状态

callbacks = [TrainingMonitor(output\_accuracy\_loss\_path)]

#%%

#%%

# 训练模型

print("[INFO] training network...")

H = model.fit\_generator(aug.flow(trainX, trainy, batch\_size=BATCH\_SIZE),

validation\_data=(testX, testy), steps\_per\_epoch=len(trainX) // BATCH\_SIZE,

epochs=EPOCHS, callbacks = callbacks, verbose=1)

#%%

#%%

# 评估模型

print("[INFO] evaluating network...")

predictions = model.predict(testX, batch\_size=32)

print(classification\_report(testy.argmax(axis=1),

predictions.argmax(axis=1), target\_names=list(lb.classes\_)))

#%%

#%%

# 保存模型

print("[INFO] serializing network...")

model.save(output\_model\_path)

#%%

【案例代码】test.py

# -\*- coding: utf-8 -\*-

'''

测试表情识别模型

'''

#%%

# import the necessary packages

from keras.preprocessing.image import img\_to\_array

from keras.models import load\_model

from faceutildlib import FaceUtil

import numpy as np

import imutils

import cv2

import time

#%%

#%%

# 全局变量

model\_path = 'model/facial\_expression\_fer.hdf5'#训练的模型

input\_video = 'xinyuanjieyi.avi'#视频文件

#%%

#%%

# 全局常量

FACIAL\_EXPRESSION\_TARGET\_WIDTH = 48

FACIAL\_EXPRESSION\_TARGET\_HEIGHT = 48

FACIAL\_EXPRESSION\_MAP = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Neutral', 5:'Sad', 6:'Surprise'}

#%%

#%%

# load the face detector cascade and smile detector CNN

model = load\_model(model\_path)

#%%

#%%

# if a video path was not supplied, grab the reference to the webcam

if not input\_video:

camera = cv2.VideoCapture(0)

time.sleep(2)

else:

camera = cv2.VideoCapture(input\_video)

#%%

#%%

faceutil = FaceUtil()

#%%

#%%

# 遍历视频的每一帧

while True:

# 得到当前帧

(grabbed, frame) = camera.read()

# 如果视频文件播放完了

if input\_video and not grabbed:

break

# 如果通过摄像头捕捉视频，那么把画面左右翻转

if not input\_video:

frame = cv2.flip(frame, 1)

# 设置视频的尺寸（一般是减小尺寸，以提高识别速度）

frame = imutils.resize(frame, width=600)

# 调用人脸检测类，识别人脸

face\_location\_list = faceutil.get\_face\_location(frame)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)#将彩色帧转换为灰度图像

# 遍历每一张人脸的坐标

for (left, top, right, bottom) in face\_location\_list:

roi = gray[top:bottom, left:right] # 从图像中截取出人脸区域

roi = cv2.resize(roi, (FACIAL\_EXPRESSION\_TARGET\_WIDTH,

FACIAL\_EXPRESSION\_TARGET\_HEIGHT))#将人脸区域的尺寸设置成模型需要的

roi = roi.astype("float") / 255.0#数据标准化

roi = img\_to\_array(roi)#元素类型由uint8转换为float32

roi = np.expand\_dims(roi, axis=0)#扩展成4维数据，模型需要的参数就是4维的

# 得到预测的表情

label = FACIAL\_EXPRESSION\_MAP[model.predict(roi)[0].argmax(axis=0)]

# 把人脸框出来，并且把表情写在框的上面

cv2.putText(frame, label, (left, top - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 255), 2)

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# 实时显示处理后的视频

cv2.imshow("Facial Expression Detect", frame)

# 按ESC键随时退出视频

k = cv2.waitKey(100) & 0xff

if k == 27:

break

# 释放资源

camera.release()

cv2.destroyAllWindows()

#%%