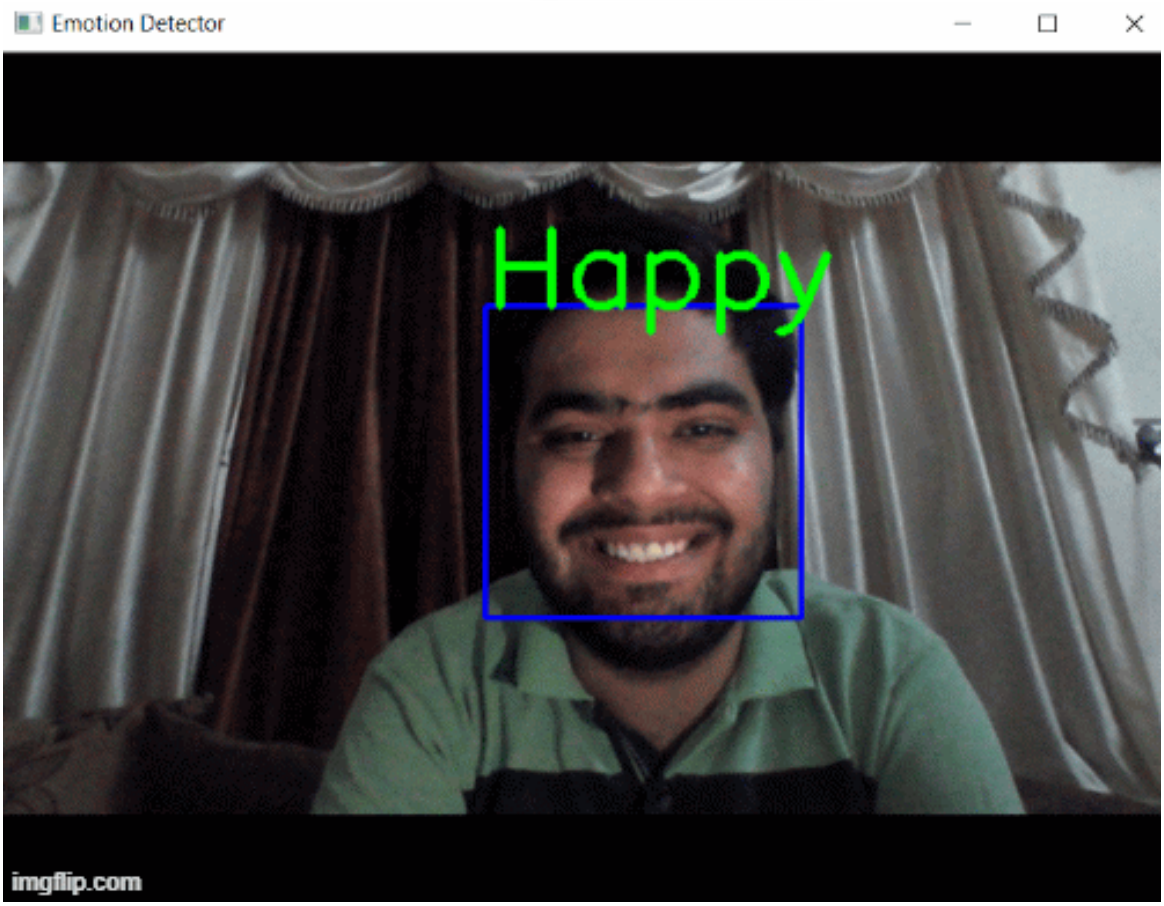
基於OpencvCV的情緒檢測

小白學視覺 2022-01-16 10:05



情緒檢測或表情分類在深度學習領域中有著廣泛的研究。使用相機和一些簡單的代碼我們就可以對情緒進行實時分類，這也是邁向高級人機交互的一步。

前言

本期我們將首先介紹如何使用**Keras** 創建卷積神經網絡模型，再使用攝像頭獲取圖片進行情緒檢測。

為了更好的閱讀體驗，我們最好具備一下知識：

- Python
- OpenCV的
- 卷積神經網絡（CNN）

• numpy

（注意：我們使用的Tensorflow是1.13.1版本、keras是版本2.3.1）

模型的製作

首先，我們將創建模型代碼並解釋其中的含義。代碼的創建總共分為以下5個部分。

任務1：

導入該項目所需的必需模塊。

```
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense,Dropout,Activation,Flatten,BatchNormalization
from keras.layers import Conv2D,MaxPooling2D
import os
```

現在讓我們定義一些變量，這些變量將節省手動輸入的時間。

```
num_classes=5
img_rows,img_cols=48,48
batch_size=32
```

以上變量的說明如下：

• num_classses = 5：訓練模型時要處理的類即情感的種類數。

• img_rows=48，img_cols = 48：饋送到神經網絡中的圖像陣列大小。

• batch_size = 32：更新模型之前处理的样本数量。epochs 是完整通过训练数据集的次数。batch_size必须大于等于1并且小于或等于训练数据集中的样本数。

任务2：

现在让我们开始加载模型，这里使用的数据集是**fer2013**，该数据集是由kaggle托管的开源数据集。数据集共包含7类，分别是愤怒、厌恶、恐惧、快乐、悲伤、惊奇、无表情，训练集共有28,709个示例。该数据集已从网站上删除，但我们在以下链接中可以找到相关代码和数据集。https://github.com/karansjc1/emotion-detection

数据集的存储库中

我们将数据储存在特定文件夹中。例如，“愤怒”文件夹包含带有愤怒面孔等的图片。在这里，我们使用5类，包括“愤怒”，“快乐”，“悲伤”，“惊奇”和“无表情”。使用24256张图像作为训练数据，3006张图像作为检测数据。

现在让我们将数据加载到一些变量中。

```
train_data_dir='fer2013/train'
validation_data_dir='fer2013/validation'
```

以上两行导入了检测和训练数据。该模型是在训练数据集上进行训练的；在检测数据集上检测该模型性能，检测数据集是原始数据集的一部分，从原始数据集上分离开来的。

任务3：

现在，我们对这些数据集进行图像增强。图像数据增强可以扩展训练数据集大小，改善图像质量。Keras深度学习神经网络库中的ImageDataGenerator类通过图像增强来拟合模型。

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    shear_range=0.3,
    zoom_range=0.3,
    width_shift_range=0.4,
    height_shift_range=0.4,
    horizontal_flip=True,
    fill_mode='nearest')
validation_datagen = ImageDataGenerator(rescale=1./255)
```

train_datagen变量以下方法人为地扩展数据集：

• rotation_range：随机旋转，在这里我们使用30度。

• shear_range：剪切强度（逆时针方向的剪切角，以度为单位）。在这里我们使用0.3作为剪切范围。

• zoom_range：随机缩放的范围，这里我们使用0.3作为缩放范围。

• width_shift_range：在图像的整个宽度上移动一个值。

• height_shift_range：这会在整个图像高度上移动一个值。

• horizontal_flip：水平翻转图像。

• fill_mode：通过上述使用的方法更改图像的方向后填充像素，使用“最近”作为填充模式，即用附近的像素填充图像中丢失的像素。

在这里，我只是重新保存验证数据，而没有执行任何其他扩充操作，因为我想使用与训练模型中数据不同的原始数据来检查模型。

```
train_generator = train_datagen.flow_from_directory(
 train_data_dir,
 color_mode='grayscale',
 target_size=(img_rows,img_cols),
 batch_size=batch_size,
 class_mode='categorical',
 shuffle=True)
validation_generator = validation_datagen.flow_from_directory(
 validation_data_dir,
 color_mode='grayscale',
 target_size=(img_rows,img_cols),
 batch_size=batch_size,
 class_mode='categorical',
 shuffle=True)
```

上面代码的输出将是：

```
Found 24256 images belonging to 5 classes.
Found 3006 images belonging to 5 classes.
```

在上面的代码中，我正在使用flow_from_directory（）方法从目录中加载我们的数据集，该目录已扩充并存储在train_generator和validation_generator变量中。flow_from_directory（）采用目录的路径并生成一批扩充数据。因此，在这里，我们为该方法提供了一些选项，以自动更改尺寸并将其划分为类，以便更轻松地输入模型。

给出的选项是：

• directory：数据集的目录。

• color_mode：在这里，我将图像转换为灰度，因为我对图像的颜色不感兴趣，而仅对表达式感兴趣。

• target_size：将图像转换为统一大小。

• batch_size：制作大量数据以进行训练。

• class_mode：在这里，我将“类别”用作类模式，因为我将图像分为5类。

• shuffle：随机播放数据集以进行更好的训练。

任务4：

数据集的修改已完成，现在是该模型的大脑即CNN网络。

因此，首先，我将定义将要使用的模型的类型。在这里，我使用的是Sequential模型，该模型定义网络中的所有层将依次相继并将其存储在变量模型中。

```
model = Sequential()
```

该网络由7个块组成：（**后面我们将逐层解释**）

```
#Block-1
model.add(Conv2D(32,(3,3),padding='same',kernel_initializer='he_normal',
                 input_shape=(img_rows,img_cols,1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(32,(3,3),padding='same',kernel_initializer='he_normal',
                 input_shape=(img_rows,img_cols,1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
#Block-2
model.add(Conv2D(64,(3,3),padding='same',kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),padding='same',kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
#Block-3
model.add(Conv2D(128,(3,3),padding='same',kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(128,(3,3),padding='same',kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
#Block-4
model.add(Conv2D(256,(3,3),padding='same',kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),padding='same',kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
#Block-5
model.add(Flatten())
model.add(Dense(64,kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
#Block-6
model.add(Dense(64,kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
#Block-7
model.add(Dense(num_classes,kernel_initializer='he_normal'))
model.add(Activation('softmax'))
```

运行以上代码，如果使用的是旧版本的tensorflow，则会收到一些警告。

在这里，我使用了存在于keras.layers中的7种类型的层。

这些层是：

•Conv2D(
filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
dilation_rate=(1, 1), activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None, **kwargs)

• Activation(activation_type)

• BatchNormalization()

• MaxPooling2D(pool_size, strides,padding, data_format, **kwargs)

• Dropout(dropout_value)

• Flatten()

• Dense(
units,
activation=None,
use_bias=True,
kernel_initializer="glorot_uniform",
bias_initializer="zeros",
kernel_regularizer=None,
bias_regularizer=None,
activity_regularizer=None,
kernel_constraint=None,
bias_constraint=None,
**kwargs)

Block-1层的出现顺序如下：

• Conv2D层-此层为网络创建卷积层。我们创建的该层包含32个大小为（3,3）滤波器，其中使用padding ='same'填充图像并使用内核初始化程序he_normal。添加了2个卷积层，每个层都有一个激活层和批处理归一化层。

• 激活层-使用elu激活。

• BatchNormalization（批处理归一化）-归一化每一层的激活，即将平均激活值保持在接近0并将激活标准偏差保持在接近1。

• MaxPooling2D层-通过沿pool_size定义的沿特征轴的每个尺寸的窗口上的最大值，对输入表示进行下采样。在此， pool_size大小为（2,2）。

• Dropout：是一种在训练过程中忽略随机选择的神经元的技术。在这里，我将dropout设为0.5，这意味着它将忽略一半的神经元。

Block–2层的出现顺序如下：

• 与block-1相同的层，但是卷积层具有64个滤波器。

Block–3层的出现顺序如下：

• 与block-1相同的层，但是卷积层具有128个滤波器。

Block–4层的出现顺序如下：

• 与block-1相同的层，但是卷积层具有256个滤波器。

Block–5层的出现顺序如下：

• 展平层-将前一层的输出展平，即转换为矢量形式。

• 密集层-该层中每个神经元都与其他每个神经元相连。在这里，我使用带有内核的程序初始化64个单元或64个神经元-he_normal。

• 这些层之后使用elu激活，批处理归一化，最后以dropout为50％选择忽略。

块6层的出现顺序如下：

• 与模块5相同的层，但没有展平层，因为该模块的输入已展平。

块7层的出现顺序如下：

• 密集层-网络的最后一个块中，我使用num_classes创建一个密集层，该层具有he_normal初始值设定项，其unit =类数。

• 激活层-在这里，我使用softmax，该层多用于分类。

现在检查模型的整体结构：

```
print(model.summary())
```

输出将是：

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 48, 48, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 48, 48, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 24, 24, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 24, 24, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 64)        0
```

```
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
activation_5 (Activation)    (None, 12, 12, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 128)       147584
_________________________________________________________________
activation_6 (Activation)    (None, 12, 12, 128)       0
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 256)         295168
_________________________________________________________________
activation_7 (Activation)    (None, 6, 6, 256)         0
_________________________________________________________________
batch_normalization_7 (Batch (None, 6, 6, 256)         1024
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 6, 6, 256)         590080
_________________________________________________________________
activation_8 (Activation)    (None, 6, 6, 256)         0
_________________________________________________________________
batch_normalization_8 (Batch (None, 6, 6, 256)         1024
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 3, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                147520
_________________________________________________________________
activation_9 (Activation)    (None, 64)                0
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256
```

```
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
activation_10 (Activation)   (None, 64)                0
_________________________________________________________________
batch_normalization_10 (Batc (None, 64)                256
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 325
_________________________________________________________________
activation_11 (Activation)   (None, 5)                 0
=================================================================
Total params: 1,328,037
Trainable params: 1,325,861
Non-trainable params: 2,176
_________________________________________________________________
None
```

上面的输出显示了该网络中使用的所有层。这是一个大型网络，包含**1,328,037个 参数**。

任务5：

最后一步：编译和训练

现在剩下的事情就是编译和训练模型。但是首先让我们导入更多的依赖。

```
from keras.optimizers import RMSprop,SGD,Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
```

在编译之前，我将使用keras.callbacks类创建以下3个东西：

Checkpoint（函数— ModelCheckpoint（ ））

它将监视验证损失，并使用mode ='min'属性尝试将损失降至最低。到达检查点时，它将保存训练有素的最佳大小。Verbose = 1仅用于代码创建检查点时的可视化。这里我使用以下参数：

• file-path：保存模型文件的路径，这里我保存的模型文件名为EmotionDetectionModel.h5

• monitor：要监视的数量。在这里，我正在监视验证损失。

• mode：{自动，最小，最大}之一。如果save_best_only = True，则基于监视数量的最大化或最小化来决定覆盖当前保存文件。

• save_best_only：如果save_best_only = True，则根据监视数量的最新最佳模型将不会被覆盖。

• verbose：1：更新数据，0：不变。

提前停止（功能— EarlyStopping（））

通过检查以下属性，以提前结束运行。

• monitor：要监视的数量。在这里，我正在监视验证损失。

• min_delta：被监视的数量的最小变化有资格作为改进，即绝对变化小于min_delta将被视为没有任何改进。在这里我给了0。

• patience：没有改善的时期数，此后将停止训练。我在这里给了它3。

• restore_best_weights：是否从时期以受监视数量的最佳值恢复模型权重。如果为False，则使用在训练的最后一步获得的模型权重。

• verbose：1：更新数据，0：不变。

降低学习率（函数— ReduceLROnPlateau（））

一旦学习停滞，模型通常会受益于将学习率降低2-10倍。回调监视数量，并且如果没有发现patience的改善，则学习率会降低，为此使用了以下属性。

• monitor：监视特定损失。在这里，我正在监视验证损失。

• factor：降低学习率的因素。**new_lr = lr *因子**。在这里我使用0.2作为系数。

• patience：没有改善的时期数，之后学习率将降低。我在这里使用3。

• min_delta：测量新的最佳阈值，仅关注重大变化。

• verbose：1：更新数据，0：不变。

现在是时候到最后使用编译模型model.compile（）和适合训练数据集的模型model.fit_generator（）

model.compile（）

具有以下参数：

• loss：此值将确定要在代码中使用的损失函数的类型。在这里，我们有5个类别或类别的分类数据，因此使用了“ categorical_crossentropy”损失。

• optimizer：此值将确定要在代码中使用的优化器功能的类型。这里我使用的学习率是0.001的Adam优化器，因为它是分类数据的最佳优化器。

• metrics：metrics参数应该是一个列表，模型可以有任意数量的metrics。它是模型在训练和测试过程中要评估的metrics列表。这里我们使用了精度作为度量标准。

model.fit_generator（）

使模型适合Python逐批生成的数据。

它具有以下参数：
• generator：我们之前创建的train_generator对象。
• steps_per_epochs：在一个纪元内接受训练数据的步骤。
• epoch：一次通过整个数据集。
• callbacks：包含我们之前创建的所有回调的列表。
• validation_data：我们之前创建的validation_generator对象。
• validation_steps：在一个时期内采取验证数据的步骤。

```
model.compile(loss='categorical_crossentropy',
 optimizer = Adam(lr=0.001),
 metrics=['accuracy'])
nb_train_samples = 24176
nb_validation_samples = 3006
epochs=25
history=model.fit_generator(
 train_generator,
 steps_per_epoch=nb_train_samples//batch_size,
 epochs=epochs,
 callbacks=callbacks,
```

```
    validation_data=validation_generator,
    validation_steps=nb_validation_samples//batch_size)
```

驱动程式码

现在，我们将使用在上一节中创建的模型来说明用于情感检测的代码。

首先，让我们再次导入一些运行代码所需的模块。

```
from keras.models import load_model
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import cv2
import numpy as np
```

现在，让我们加载模型，并加载我用来检测摄像头前方人脸的分类器。使用haarcascade_frontalface_default分类器。**Haar Cascade**是一种机器学习对象检测算法，用于识别图像或视频中的对象，并基于Paul Viola和Michael Jones在其论文《使用简单特征的增强**级联**进行快速对象检测》中提出的特征概念。2001。haarcascade_frontalface_default分类器可检测图像或连续视频源中人的正面。

```
face_classifier=cv2.CascadeClassifier('/haarcascade_frontalface_default.xml')
classifier = load_model('/EmotionDetectionModel.h5')
```

现在，我将定义一个变量class_labels来存储类的名称或我们要预测的情绪类型，还定义一个变量cap来存储cv2.VideoCapture方法返回的值。在此，VideoCapture中的值0用于指示该方法使用便携式计算机的主要网络摄像头。

```
class_labels=['Angry','Happy','Neutral','Sad','Surprise']
cap=cv2.VideoCapture(0)
```

结论

因此，在这里我已经解释了使用OpenCV和Keras创建情绪检测的过程。通过以下链接可以查看完整的代码以及数据集。

https://github.com/karansjc1/emotion-detection

实验结果如下：

欢迎加入公众号读者群一起和同行交流，目前有**SLAM、三维视觉、传感器、自动驾驶、计算摄影**、检测、分割、识别、**医学影像、GAN、算法竞赛**等微信群（以后会逐渐细分），请扫描下面微信号加群，备注：”昵称+学校/公司+研究方向“，例如：”张三 + 上海交大 + 视觉SLAM“。**请按照格式备注，否则不予通过。**添加成功后会根据研究方向邀请进入相关微信群。**请勿**在群内发送**广告**，否则会请出群，谢谢理解~

新機器視覺