案例二：基于CNN的体感游戏手势识别

1.案例目的

• 掌握如何构建卷积神经网络模型；

• 掌握CNN模型训练；

• 掌握CNN模型评估。

2.案例内容

本案例基于自定义的卷积神经网络训练手势识别模型，在测试集上评估模型性能，以及使用训练好的模型对图片识别，并显示图片以及识别结果。

模型可以用在体感游戏中，通过手势控制着游戏的操作。

3.案例知识点

• Python语言编程；

• NumPy和OpenCV库的使用；

• 构建卷积神经网络模型。

4.案例时长

共4学时，具体安排如下：

• 基于自定义卷积神经网络训练手势识别模型（2.5学时）

• 应用训练好的模型测试图片（1.5学时）

5.案例实验环境

**操作系统：**

• Linux Ubuntu 16.04 64位操作系统

**软件环境：**

• Python 3.7.4

• Tensorflow 2.3.0

• Keras 2.4.3

**开发环境与工具：**

• Spyder

6.案例分析

本案例分为以下两个主要部分：

（1）基于自定义卷积神经网络训练手势识别模型

基于自定义的卷积神经网络训练手势识别模型，并在测试集上评估模型性能。

（2）应用训练好的模型测试图片

调用在（2）中训练得到的模型对图片进行测试，并显示图片及测试结果。

7.案例实验过程

该案例可分为以下几个步骤：

1、 使用自定义的CNN训练手势识别模型

1.1 导入库

1.2 设置全局变量

1.3 设置超参数

1.4 构建查找字典

1.5 数据预处理

1.6 拆分数据集

1.7 构建卷积神经网络模型

1.8 编译模型

1.9 训练模型

1.10 评估模型

1.11 保存模型

2、应用训练好的模型测试图片

7.1开发准备

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

7.1.1 获取数据

本案例需要1个数据集，是leapGestRecog目录，该目录下存储了手势图像。

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/065ba178ec8947d3897dce15162263c2.zip下载数据集。下载后得到leapGestRecog.zip文件。将下载后的leapGestRecog.zip文件解压，得到leapGestRecog目录。将leapGestRecog目录复制到刚创建的Hand-Gesture-Recognition目录下。

7.1.2 打开Spyder软件

在桌面上双击Spyder图标打开Spyder，并把工作目录切换至Hand-Gesture-Recognition目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为Hand-Gesture-Recognition。使用Spyder新建两个Python文件，分别命名为hand\_train.py、predict.py。

7.1.4 新建目录

在Hand-Gesture-Recognition目录下新建两个目录，分别命名为model、examples。model目录用于存放训练好的模型，examples用于存放需要被识别的图片。

至此，整个案例的目录结构如图1所示。

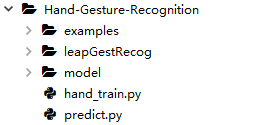


图1 案例目录结构

7.1.6 了解数据

本案例使用的是手势识别数据集，该数据集中的所有图像都是红外图像。数据集中包含10种手势，分别是“palm”，“l”，“fist”，“fist\_moved”，“thumb”，“index”，“ok“，“palm\_moved”，“c”，“down”。分为10个子目录，分别命名为00~09，每个子目录下又包含10个子目录，每个手势类别对应一个子目录，每个子目录中包含200张图像。手势识别数据集中共包含20000张图像。

该数据目录如图2所示：



图2手势识别数据集目录

部分手势图像如图3所示：

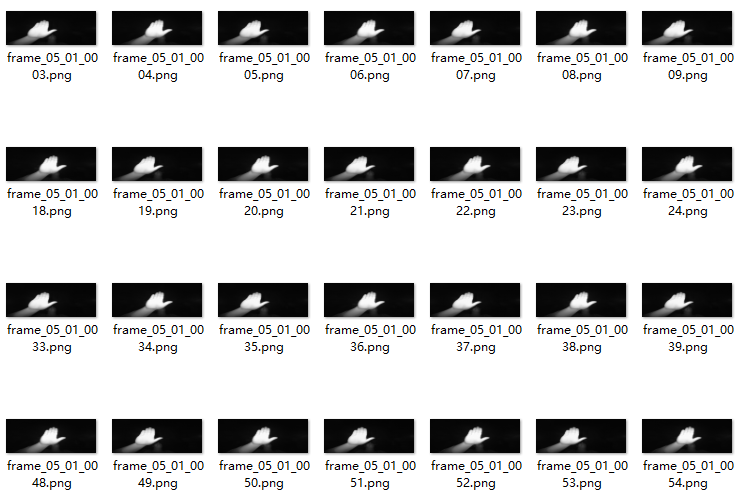


图3 手势识别数据集中的部分图像

7.2使用自定义的CNN训练手势识别模型

该模块的主要作用是基于自定义的CNN训练手势识别模型，在测试集上对模型进行评估，并保存最终模型。

7.2.1 导入库

导入所有相关库。

【代码7-2-1】hand\_train.py

# 导入库

import numpy as np

import os

from PIL import Image

from keras import layers

from keras import models

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

from keras.utils import to\_categorical

执行代码块。

7.2.2 设置全局变量

这些变量用于指定数据源的路径、模型保存的路径等。设置为全局变量比较方便使用。

【代码7-2-2】hand\_train.py

# 全局变量

model\_path = 'model/final.model'

image\_path = 'leapGestRecog'

执行代码块。

7.2.3 设置超参数

这两个超参数是训练模型的指定参数，像一批加载几张图片，训练多少个epoch。如果模型性能不好或者出现其他问题，我们需要调整这两个参数。所以把这两个参数统一放在一起作为常量。

【代码7-2-3】hand\_train.py

# 超参数

EPOCHS = 8

BATCH\_SIZE = 64

执行代码块。

7.2.4 构建查找字典

如7.1.6节中了解数据所述，有00~09共10个目录，在每个目录中，又包含每个手势的子目录。在该部分，将构建一个查找字典，该字典存储10种手势的名称，并为每种手势赋予数字标签。还将构建一个反向查找字典，该字典中是标签与其所关联手势之间的映射。

【代码7-2-4】hand\_train.py

# 构建查找字典

lookup = dict()

reverselookup = dict()

for i in os.listdir(image\_path+'/00/'):

index = int(i.split('\_')[0])-1

lookup[i] = index

reverselookup[index] = i

执行代码块。

此时建议切换到Spyder的Variable explorer面板，查看变量信息。

运行代码后，查找字典如图4所示：

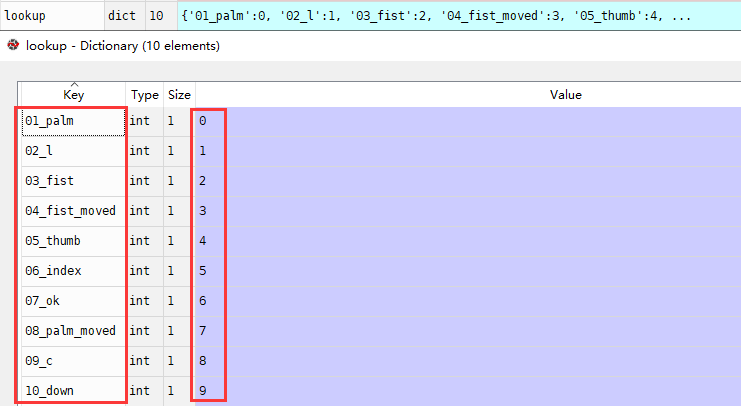


图4 查找字典

反向查找字典如图5所示：

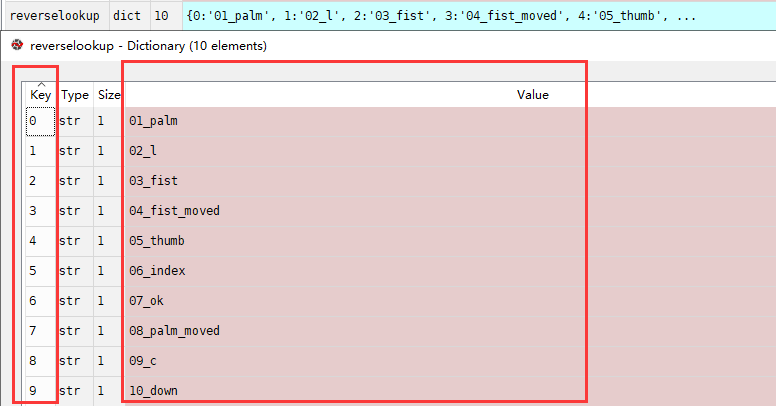


图5 反向查找字典

7.2.5 数据预处理

在正式开始训练模型之前，需要对数据进行预处理，为建立模型做准备。

首先，定义空列表，用于下面存储数据以及标签。

【代码7-2-5-1】hand\_train.py

x\_data = []# 定义空列表x\_data，用于存储数据

y\_data = []# 定义空列表y\_data，用于存储标签

datacount = 0 # 初始化变量，用来统计数据集中图片的数量

执行代码块。

其次，遍历所有图像，进行调整图像尺寸、转换数据类型等操作后，将图像数据加入x\_data列表中。根据查找字典为图像添加标签，并将标签添加到y\_data列表中。

【代码7-2-5-2】hand\_train.py

for i in range(0, 10): # 遍历10个文件夹

for j in os.listdir(image\_path+'/0' + str(i) + '/'):

if not j.startswith('.'):

count = 0 # 统计给定手势的图像

for k in os.listdir(image\_path+'/0' +

str(i) + '/' + j + '/'):

img = Image.open(image\_path+'/0' +

str(i) + '/' + j + '/' + k).convert('L')# 读取图像，并转换图像为灰度图像

img = img.resize((320, 120))# 调整成模型需要的尺寸

arr = np.array(img)# 转换为Numpy格式

x\_data.append(arr)# 将数据加入x\_data列表

count = count + 1

y\_values = np.full((count, 1), lookup[j]) # 根据查找字典添加标签

y\_data.append(y\_values)# 将所有数据的标签加入y\_data列表

datacount = datacount + count

执行代码块。

所有图像的像素信息保存在x\_data变量中，如图6所示。

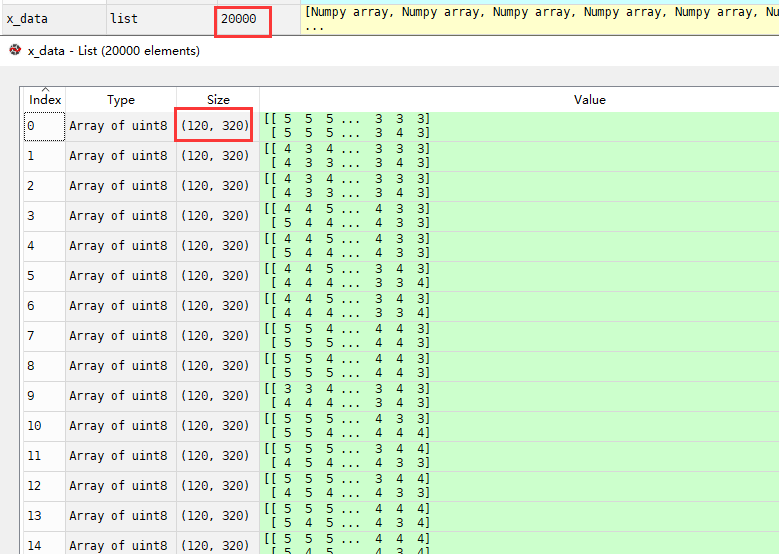


图6 所有图像的信息保存在x\_data变量中

所有图像的标签信息保存在y\_data变量中，如图7所示。

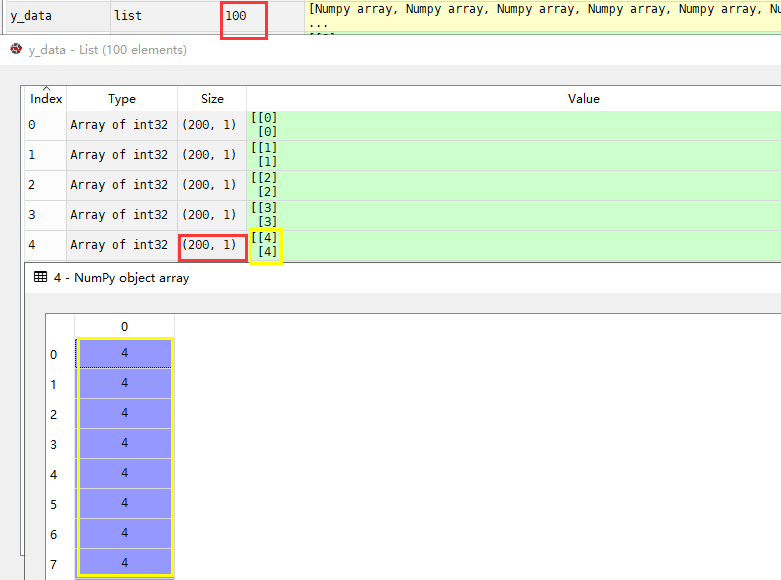


图7 所有图像的标签信息保存在y\_data变量中

接下来，将x\_data转换为float32类型，将y\_data转换为int32类型。

【代码7-2-5-3】hand\_train.py

x\_data = np.array(x\_data, dtype = 'float32') # 转换为float32类型

y\_data = np.array(y\_data)# 转换为int32类型

y\_data = y\_data.reshape(datacount, 1)

执行代码块。

代码运行后，x\_data变量如图8所示。



图8 x\_data变量

y\_data变量如图9所示。

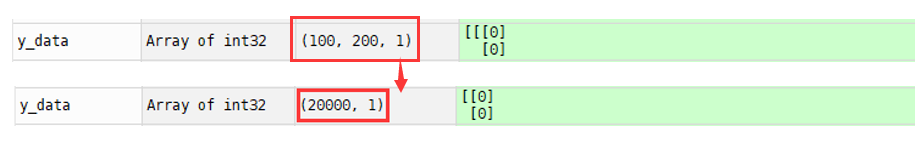


图9 y\_data变量

因为CNN输出层的激活函数用的是softmax，这里需要将标签编码转化为独热编码。

【代码7-2-5-4】hand\_train.py

# 独热编码

y\_data = to\_categorical(y\_data)

执行代码块。

进行独热编码后，有几个类别就有几列，y\_data变量如图10所示。

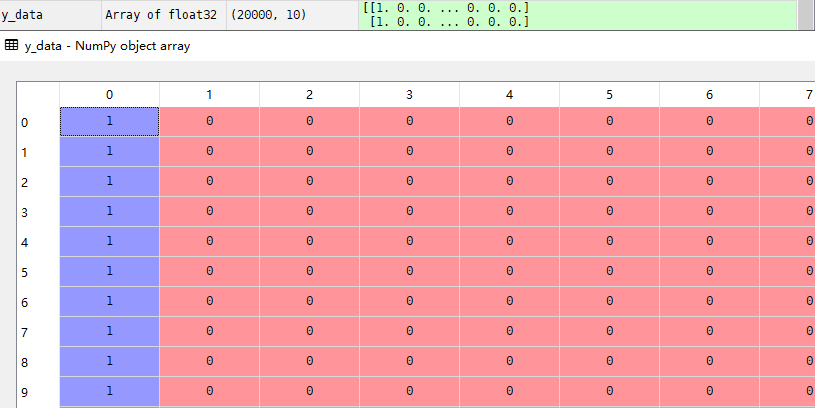


图10 进行独热编码后，y\_data变量

由于卷积需要的数据输入格式是一个Tensor（张量），具有[batch, in\_height, in\_width, in\_channels]这样的shape，具体含义是[训练时一个batch的图片数量, 图片高度, 图片宽度, 图像通道数]，因此，这里我们先将数据转化为4维。注意，这里（datacount）20000是所有的图像数量，在训练的时候每次按BATCH\_SIZE输入数据。

然后将数据标准化，像素值将在0~1之间。

【代码7-2-5-5】hand\_train.py

# 数据标准化

x\_data = x\_data.reshape((datacount, 120, 320, 1))

x\_data /= 255

执行代码块。

维度转化以及数据标准化后，效果如图11所示。



图11 数据标准化

7.2.6 拆分数据集

该模块的作用是划分数据集，首先将数据的80%划分为训练集，10%划分为验证集，10%划分为测试集。

【代码7-2-6】hand\_train.py

#划分训练集、验证集、测试集

x\_train,x\_further,y\_train,y\_further = train\_test\_split(x\_data,y\_data,test\_size = 0.2)

x\_validate,x\_test,y\_validate,y\_test = train\_test\_split(x\_further,y\_further,test\_size = 0.5)

执行代码块。

划分结果如图12所示，红色框内为训练数据，共16000条；黄色框内为验证数据，共2000条数据；蓝色框内为测试数据，共2000条。

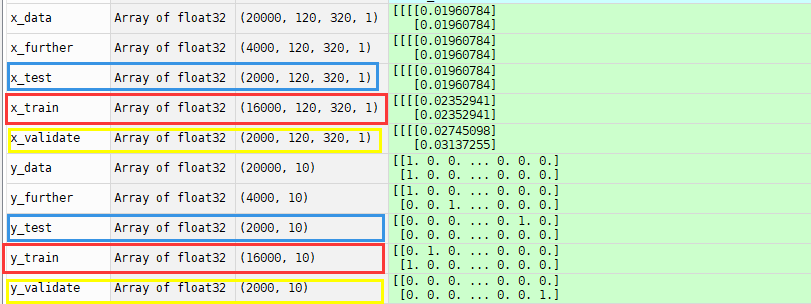


图12 拆分数据集

7.2.7 构建卷积神经网络模型

此部分中，没有使用成熟的卷积神经网络模型，手势识别任务比较简单，这里定义了一个较为简单的卷积神经网络模型，同学们也可以尝试己定义卷积神经网络模型用于此案例。

【代码7-2-7】hand\_train.py

#构建卷积神经网络模型

model=models.Sequential()

model.add(layers.Conv2D(32, (5, 5), strides=(2, 2), activation='relu', input\_shape=(120, 320,1)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

执行代码块。

7.2.8 编译模型

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

【代码7-2-8】hand\_train.py

#编译模型

model.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

执行代码块。

7.2.9 训练模型

该部分的目的是训练模型。根据代码中设置的超参数EPOCHS，我们需要训练8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。

在Intel(R) Core(TM) i7-8750H [CPU@2.20GHz](mailto:CPU@2.20GHz)CPU，8GBRAM的计算机上，每个epoch的运行时间大约为112s，8个epoch大约需要15分钟。由于机器性能不同，运行时间有所不同。若训练速度过慢，建议同学们将减小EPOCHS参数的设置，仅用于查看案例效果即可。

【代码7-2-9】hand\_train.py

#训练模型

model.fit(x\_train, y\_train, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1, validation\_data=(x\_validate, y\_validate))

执行代码块。

训练过程如图13所示。训练完成后，在验证集上的accuracy为99.90%左右，模型性能是非常不错的。

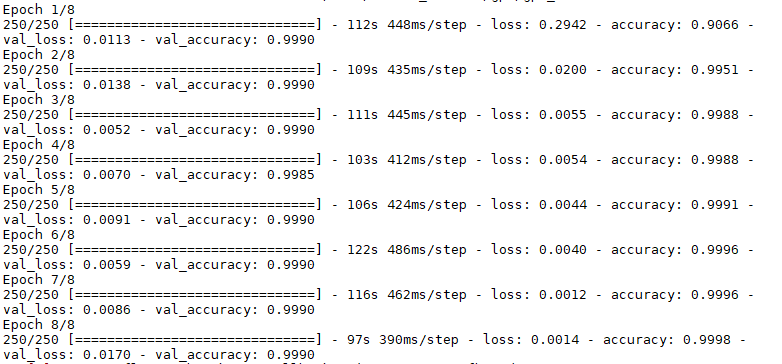


图13 模型训练过程

7.2.10 评估模型

模型训练完成后，在测试集上评估模型性能。

【代码7-2-10】hand\_train.py

# 评估模型

[loss, acc] = model.evaluate(x\_test,y\_test,verbose=1)

print("Accuracy:" + str(acc))

执行代码块。

测试评估结果如图14所示，accuracy约为99.94%，证明训练的模型对手势识别是非常准确的。



图14 测试评估结果

7.2.11 保存模型

如果同学们没有完成模型的训练，请不要执行这个步骤。

保存模型是非常重要的一个步骤。下次使用该模型时，只需用1个函数加载进来即可。

【代码7-2-11】hand\_train.py

# 保存模型

print("[INFO] Saving Models...")

model.save(model\_path)

执行代码块。

模型会保存在model目录下。

7.3应用训练好的模型测试图片

该部分的主要作用是加载训练好的模型，对examples目录中的图像进行手势识别。向predict.py文件中添加代码。代码比较简单，直接提供完整代码。

【代码7-3】predict.py

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

'''

应用训练好的模型测试图片

'''

# 导入库

from keras.models import load\_model

import numpy as np

from imutils import paths

import imutils

import cv2

from PIL import Image

# 全局变量

model\_path = 'model/final.model'

test\_path = 'examples'

# 加载模型

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

model = load\_model(model\_path)

# 定义标签列表

CLASSES = ["palm", "l", "fist", "fist\_moved", "thumb",

"index", "ok", "palm\_moved", "c", "down"]

# 加载图片

image\_paths = sorted(list(paths.list\_images(test\_path)))

for image\_path in image\_paths:

image = Image.open(image\_path)

orig = cv2.imread(image\_path)# orig是原始图片，用于最后显示用。image不断被改变

image = image.resize((320, 120))# 调整成模型需要的尺寸

image = np.array(image) # 将image数组中的元素类型由uint8转换为float32

image = image/255 # 图像标准化

image = np.expand\_dims(image, axis = 0)

image = np.expand\_dims(image, axis = -1)

# 预测图象

pred = model.predict(image)[0]

i = np.argmax(pred)

label = CLASSES[i]# 标签

# 在输出图像上绘制预测结果

text = "{}: {:.2f}%".format(label, pred[i] \* 100)

output = imutils.resize(orig, width = 400)

cv2.putText(output, text, (10,25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0),2)

# 显示图片

cv2.imshow('Output', output)

cv2.waitKey(0)

cv2.destroyAllWindows()

由于代码相对简单，所以没有逐块运行。请同学们注意，被识别的图片存放在examples目录下。

由于这是一个新的py文件，请重启Spyder的kernel。

全部选中代码，运行predict.py，查看运行结果。

按空格键切换下一张图片。

在examples目录提供的图片中，没有识别错误的。准确率还是挺不错的，如图15所示

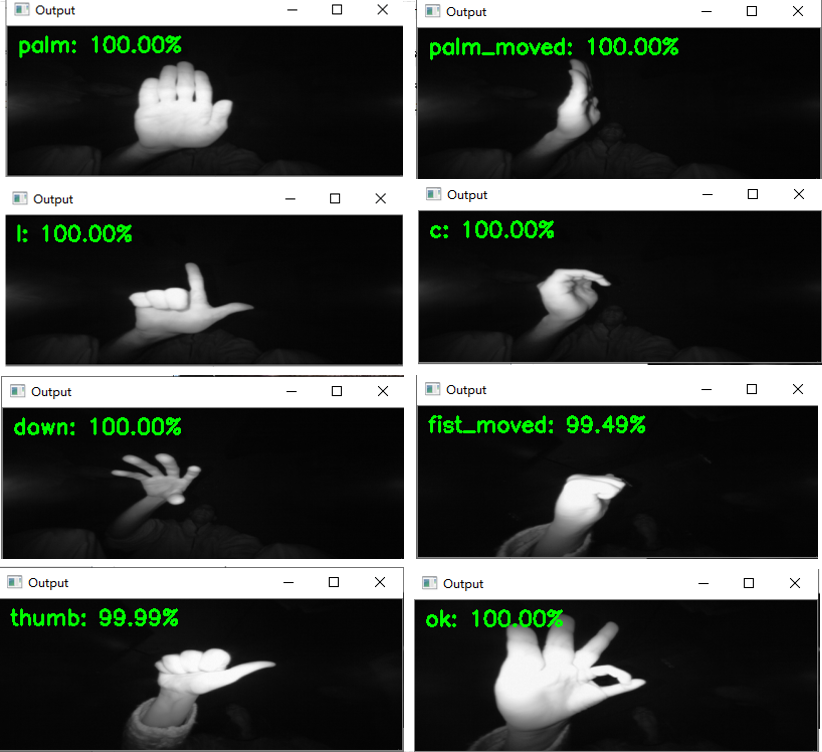


图15 手势识别结果

8.案例代码

【案例代码】build\_dataset.py

'''

训练手势识别模型

'''

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

# 导入库

import numpy as np

import os

from PIL import Image

from keras import layers

from keras import models

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

from keras.utils import to\_categorical

# 全局变量

model\_path = 'model/final.model'

image\_path = 'leapGestRecog'

# 超参数

EPOCHS = 8

BATCH\_SIZE = 64

# 构建查找字典

lookup = dict()

reverselookup = dict()

for i in os.listdir(image\_path+'/00/'):

index = int(i.split('\_')[0])-1

lookup[i] = index

reverselookup[index] = i

x\_data = []# 定义空列表x\_data，用于存储数据

y\_data = []# 定义空列表y\_data，用于存储标签

datacount = 0 # 初始化变量，用来统计数据集中图片的数量

for i in range(0, 10): # 遍历10个文件夹

for j in os.listdir(image\_path+'/0' + str(i) + '/'):

if not j.startswith('.'):

count = 0 # 统计给定手势的图像

for k in os.listdir(image\_path+'/0' +

str(i) + '/' + j + '/'):

img = Image.open(image\_path+'/0' +

str(i) + '/' + j + '/' + k).convert('L')# 读取图像，并转换图像为灰度图像

img = img.resize((320, 120))# 调整成模型需要的尺寸

arr = np.array(img)# 转换为Numpy格式

x\_data.append(arr)# 将数据加入x\_data列表

count = count + 1

y\_values = np.full((count, 1), lookup[j]) # 根据查找字典添加标签

y\_data.append(y\_values)# 将所有数据的标签加入y\_data列表

datacount = datacount + count

x\_data = np.array(x\_data, dtype = 'float32') # 转换为float32类型

y\_data = np.array(y\_data)# 转换为int32类型

y\_data = y\_data.reshape(datacount, 1)

# 独热编码

y\_data = to\_categorical(y\_data)

# 数据标准化

x\_data = x\_data.reshape((datacount, 120, 320, 1))

x\_data /= 255

#划分训练集、验证集、测试集

x\_train,x\_further,y\_train,y\_further = train\_test\_split(x\_data,y\_data,test\_size = 0.2)

x\_validate,x\_test,y\_validate,y\_test = train\_test\_split(x\_further,y\_further,test\_size = 0.5)

#构建卷积神经网络模型

model=models.Sequential()

model.add(layers.Conv2D(32, (5, 5), strides=(2, 2), activation='relu', input\_shape=(120, 320,1)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

#编译模型

model.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

#训练模型

model.fit(x\_train, y\_train, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1, validation\_data=(x\_validate, y\_validate))

# 评估模型

[loss, acc] = model.evaluate(x\_test,y\_test,verbose=1)

print("Accuracy:" + str(acc))

# 保存模型

print("[INFO] Saving Models...")

model.save(model\_path)

【案例代码】predict.py

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

'''

应用训练好的模型测试图片

'''

# 导入库

from keras.models import load\_model

import numpy as np

from imutils import paths

import imutils

import cv2

from PIL import Image

# 全局变量

model\_path = 'model/final.model'

test\_path = 'examples'

# 加载模型

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

model = load\_model(model\_path)

# 定义标签列表

CLASSES = ["palm", "l", "fist", "fist\_moved", "thumb",

"index", "ok", "palm\_moved", "c", "down"]

# 加载图片

image\_paths = sorted(list(paths.list\_images(test\_path)))

for image\_path in image\_paths:

image = Image.open(image\_path)

orig = cv2.imread(image\_path)# orig是原始图片，用于最后显示用。image不断被改变

image = image.resize((320, 120))# 调整成模型需要的尺寸

image = np.array(image) # 将image数组中的元素类型由uint8转换为float32

image = image/255 # 图像标准化

image = np.expand\_dims(image, axis = 0)

image = np.expand\_dims(image, axis = -1)

# 预测图象

pred = model.predict(image)[0]

i = np.argmax(pred)

label = CLASSES[i]# 标签

# 在输出图像上绘制预测结果

text = "{}: {:.2f}%".format(label, pred[i] \* 100)

output = imutils.resize(orig, width = 400)

cv2.putText(output, text, (10,25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0),2)

# 显示图片

cv2.imshow('Output', output)

cv2.waitKey(0)

cv2.destroyAllWindows()