案例一：根据图片识别11种食物

1. 案例目的

• 掌握微调（Fine-Tuning）的迁移学习方法；

• 掌握数据增强；

• 掌握CNN模型训练；

• 掌握CNN模型评估

2. 案例内容

迁移学习是指将一个场景下学习到的知识迁移到另一种场景去应用。使用深度学习技术解决问题的过程中，最常见的困难在于，因为模型有大量的参数需要训练，因此需要海量训练数据作支撑。但是在解决某一领域的具体问题时，经常无法得到足够的数据进行模型训练。而借助迁移学习，可在一定程度上解决这个问题。

本案例基于在Imagenet数据集上预训练的无头部的VGG16网络模型，在food11食物数据集上训练分类模型，微调部分网络的参数，并在测试集上评估模型性能。

3. 案例知识点

• Python语言编程；

• NumPy和OpenCV库的使用；

• 微调的迁移学习方式；

• VGG16分类模型的使用

4. 案例时长

共4学时，具体安排如下：

• 生成数据集（0.5学时）

• 使用微调的迁移学习方式训练模型（3学时）

• 应用训练好的模型测试单张图片（0.5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

2）Terminal（终端）

6. 案例分析

迁移学习作为机器学习的一个重要分支，侧重于将已经学习过的知识迁移应用于新的问题中。使用迁移学习技术可以使模型得到更高的准确率，也可以缩短模型的训练时间。迁移学习可以应用于图像分类、目标检测、图像分割等多个计算机视觉领域。

本案例分为以下三个主要部分：

**1)生成数据集**

虽然我们已经准备好了数据集，但是在本案例中并不是直接使用这个数据集，需要自己生成训练所需要的数据集格式。

**2)使用微调的迁移学习方式训练模型**

基于预训练的VGG16网络模型使用微调的迁移学习方式训练食物分类模型。

**3)应用训练好的模型测试单张图片**

调用在（2）中训练得到的模型对单张图片进行测试，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、生成数据集

1.1导入库

1.2设置全局变量

1.3类别标签

1.4 生成新的数据集

2、使用微调的迁移学习方式训练模型

2.1 导入库

2.2 设置超参数

2.3 设置全局变量

2.4 类别标签

2.5 定义绘制训练过程中loss和accuracy的曲线的函数

2.6 确定数据量

2.7 数据增强及图像预处理

2.8 数据生成器

2.9 加载VGG16网络

2.10 定义新的网络头部

2.11 将新的头部放在baseModel上

2.12 冻结baseModel的所有层

2.13 编译模型

2.14 训练网络头部

2.15 评估模型分类性能

2.16 解冻baseModel的层

2.17 重新编译模型

2.18 重新训练模型

2.19 评估最终模型性能

2.20 保存模型

3、应用训练好的模型测试单张图片

7.1开发准备

在桌面上新建一个目录，名字叫food\_classification。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11种食物的图片。

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/b4577f9f45fc40839f8780db21c2e0b3.zip。下载数据集并解压，会得到一个命名为Food-11的文件夹，文件夹下存储着10种食物的图片。

将Food-11目录直接复制到刚刚创建的food\_classification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food\_classification目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为food\_classification。使用Spyder新建3个Python文件，分别命名为build\_dataset.py、train.py和predict.py。

7.1.4 新建模型和图像输出目录

在food\_classification目录下新建一个目录，命名为output。用以存放训练好的模型以及loss、accuracy的曲线。

至此，整个案例的目录结构如图1所示。

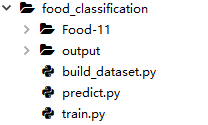


图1 案例目录结构

7.1.5 了解数据

本案例用的食物数据集Food-11共包含11种食物，分别为Bread(1724 images)、Dairy product(721 images)、Dessert(2,500 images)、Egg(1,648 images)、Fried food(1,461images)、Meat(2,206 images)、Noodles/pasta(734 images)、Rice(472 images)、Seafood(1,505 images)、Soup(2,500 images)、Vegetable/fruit(1,172 images)，共16643张图像。该数据集被划分为3个子目录training（9866 images）、validation（3430 images）、evaluation（3347 images）分别训练、验证、测试评估。该数据目录如图2所示。

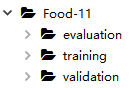


图2 Food-11数据集

部分食物图像如图3所示。



图3 Food-11数据集中的部分图像

7.2生成数据集

虽然我们已经准备好了数据集，但是在本案例中并不是直接使用这个数据集，需要自己生成训练所需要的数据集格式。向build\_dataset.py中添加代码。

7.2.1 导入库

把所有用到的库都导入。

【代码7-2-1】build\_dataset.py

# 导入包

from imutils import paths

import shutil

import os

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、新目录的路径等。设置为全局变量比较方便使用。

【代码7-2-2】build\_dataset.py

#全局变量

ORIG\_INPUT\_DATASET = 'Food-11'# 初始化图像的“原始”输入目录路径

BASE\_PATH = 'dataset'#初始化新目录的路径

TRAIN = 'training'# 定义训练，测试，验证目录的名称

TEST = 'evaluation'

VAL = 'validation'

执行代码块。

7.2.3 类别标签

数据集中共包含11类食物，根据图片的命名规则，以Food-11/training/ 0\_0.jpg为例，“\_”前的数字“0”表示其类别，对应“Bread”。以此类推，1对应“Dairy product”、2对应“Dessert”、3对应“Egg”、4对应“Fried food”、5对应“Meat”、6对应“Noodles/Pasta”、7对应“Rice”、8对应“Seafood”、9对应“Soup”、10对应“Vegetable/Fruit”。因此，按顺序初始化类别标签名称列表。

【代码7-2-3】build\_dataset.py

# 初始化类别标签名称列表

CLASSES = ["Bread", "Dairy product", "Dessert", "Egg", "Fried food",

"Meat", "Noodles/Pasta", "Rice", "Seafood", "Soup",

"Vegetable/Fruit"]

执行代码块。

7.2.4 生成新的数据集

接下来，遍历Food-11数据集，分别处理training、validation、evaluation三个子目录下的图像，遍历每个子目录的图像，从图片名字中提取类别标签，构建新的数据集dataset，并将图像复制到dataset目录下的各对应的子目录下。

【代码7-2-4】build\_dataset.py

# 遍历拆分数据

for split in (TRAIN, TEST, VAL):

#抓取当前拆分中的所有图像路径

print("[INFO] processing '{} split'...".format(split))

p = os.path.sep.join([ORIG\_INPUT\_DATASET, split])

imagePaths = list(paths.list\_images(p))

# 遍历图像路径

for imagePath in imagePaths:

# 从文件名中提取类标签

filename = imagePath.split(os.path.sep)[-1]

label = CLASSES[int(filename.split("\_")[0])]

# 构建输出目录的路径

dirPath = os.path.sep.join([BASE\_PATH, split, label])

# 如果输出路径不存在，则创建它

if not os.path.exists(dirPath):

os.makedirs(dirPath)

# 将图像复制到目标路径

p = os.path.sep.join([dirPath, filename])

shutil.copy2(imagePath, p)

执行代码块。

生成新的数据集dataset如图4所示。training、validation、evaluation三个子目录下都生成了对应11个类别的子目录。

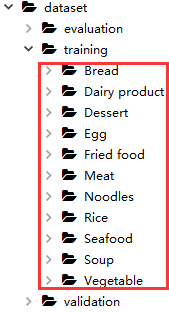


图4 dataset目录

7.3使用微调的迁移学习方式训练模型

该模块的主要作用是基于预训练的VGG16网络模型使用微调的迁移学习方式训练食物分类模型。向train.py中添加代码。

在该部分的模型训练中，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20分钟，训练网络头部50个epoch和微调20个epoch大约需要24小时，由于训练时间过长，以下展示的案例实验结果是调整了WARM\_UP\_EPOCHS和FINE\_TUNING\_EPOCHS的大小得到的。

7.3.1 导入库

把所有用到的库都导入。

【代码7-3-1】train.py

#导入包

import matplotlib

matplotlib.use("Agg")

from keras.preprocessing.image import ImageDataGenerator

from keras.applications import VGG16

from keras.layers.core import Dropout

from keras.layers.core import Flatten

from keras.layers.core import Dense

from keras.layers import Input

from keras.models import Model

from keras.optimizers import SGD

from sklearn.metrics import classification\_report

from imutils import paths

import matplotlib.pyplot as plt

import numpy as np

执行代码块。

7.3.2 设置超参数

这些超参数都是训练模型的一些指定参数，像一批加载几张图片，学习率是多少。如果模型性能不好或者出现其他问题，我们需要调整这些参数。所以把这些参数统一放在一起作为常量。

EPOCHS参数是影响模型训练时间最主要的因素。本案例中（使用CPU训练，不用GPU训练）有两个关于EPOCH的参数，WARM\_UP\_EPOCHS=50是设置训练新加入的FC层50个EPOCH。FINE\_TUNING\_EPOCHS = 20是设置微调网络模型（最后一个卷积池化块和FC层）20个EPOCH。可以更改EPOCHS参数的值，增大或减少，但是在本案例中若训练EPOCH的数量数不足，模型会欠拟合。

【代码7-3-2】train.py

# 模型超参数

BATCH\_SIZE = 32

WARM\_UP\_EPOCHS = 50

FINE\_TUNING\_EPOCHS = 20

LR = 1e-4

MOMENTUM = 0.9

执行代码块。

7.3.3 全局变量

这些变量用于指定数据源的路径、loss和accuracy曲线保存的路径。设置为全局变量比较方便使用。

【代码7-3-3】train.py

#设置全局变量

BASE\_PATH = 'dataset'#新数据集的路径

trainPath = BASE\_PATH+'/training'# 训练集路径

valPath = BASE\_PATH+'/validation'#验证集路径

testPath = BASE\_PATH+'/evaluation'#测试集路径

MODEL\_PATH = 'output/food11.model'#设置训练后模型保存的路径

WARMUP\_PLOT\_PATH = 'output/warmup.png'#保存只训练FC层模型的loss/accuracy曲线的路径

FINE\_TUNING\_PLOT\_PATH = 'output/fine\_tuning.png'# 保存微调后模型的loss/accuracy曲线的路径

执行代码块。

7.3.4 类别标签

数据集中共包含以下11类食物。

【代码7-3-4】train.py

# 初始化类别标签名称列表

CLASSES = ["Bread", "Dairy product", "Dessert", "Egg", "Fried food",

"Meat", "Noodles/Pasta", "Rice", "Seafood", "Soup",

"Vegetable/Fruit"]

执行代码块。

7.3.5 定义绘制训练过程loss和accuracy的曲线的函数

定义绘制训练过程loss和accuracy的曲线的函数plot\_training，包括三个参数：H表示模型训练的信息，N表示训练的EPOCH数量，plotPath表示曲线保存的路径。

【代码7-3-5】train.py

#构建绘制训练过程loss和accuracy的曲线的函数

def plot\_training(H, N, plotPath):

plt.style.use("ggplot")

plt.figure()

plt.plot(np.arange(0, N), H.history["loss"], label="train\_loss")

plt.plot(np.arange(0, N), H.history["val\_loss"], label="val\_loss")

plt.plot(np.arange(0, N), H.history["accuracy"], label="train\_accuracy")

plt.plot(np.arange(0, N), H.history["val\_accuracy"], label="val\_accuracy")

plt.title("Training Loss and Accuracy")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig(plotPath)

执行代码块。

7.3.6 确定数据量

确定训练集、验证集、测试集中的图像数量。

【代码7-3-6】train.py

# 确定训练集、验证集、测试集的数据量

totalTrain = len(list(paths.list\_images(trainPath)))

totalVal = len(list(paths.list\_images(valPath)))

totalTest = len(list(paths.list\_images(testPath)))

执行代码块。

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

训练集、验证集、测试集中的图像数量如图5所示。

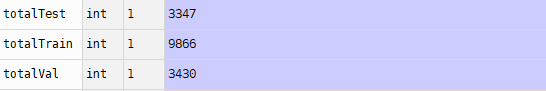


图5 训练集、验证集、测试集中的图像数量

7.3.7 数据增强及图像预处理

数据增强是一个非常重要的步骤。这里并不是人为的处理图片，也不会将增强后的图像添加到数据集中，而是在模型训练过程中程序自动对图片做即时处理，如垂直翻转、旋转、缩放等操作。这里我们只对训练数据集进行图像增强处理。

【代码7-3-7】train.py

# 训练数据增强

trainAug = ImageDataGenerator(

rotation\_range=30,

zoom\_range=0.15,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.15,

horizontal\_flip=True,

fill\_mode="nearest")

执行代码块。

下面初始化验证/测试数据扩充对象，虽然与训练数据增强中使用同样的类ImageDataGenerator，但这里不添加任何参数，也就是说不会对验证和测试数据进行如训练数据一样的数据增强处理，仅用于进行均值减法的图像预处理操作。

【代码7-3-7】train.py

#初始化验证/测试数据扩充对象（我们将在其中添加均值减法）

valAug = ImageDataGenerator()

执行代码块。

接下来，设置ImageNet的平均减法值，所有的数据都将执行逐像素减法的预处理操作。

【代码7-3-7】train.py

# 定义ImageNet均值减法（以RGB顺序），并为每个数据增强对象设置均值减法值

mean = np.array([123.68, 116.779, 103.939], dtype="float32")

trainAug.mean = mean

valAug.mean = mean

执行代码块。

7.3.8 数据生成器

定义生成器将从各自的训练、验证和测试数据中加载成批图像。使用这些生成器可以通过尝试一次加载所有数据来确保我们的机器不会用完RAM。

使用flow\_from\_directory方法，构建训练数据生成器。各参数含义如下：

trainPath：目标文件夹路径，对于每一个类，该文件夹都要包含一个子文件夹，本案例训练集中有11种类别的食物，因此包含11对应的子文件夹。

class\_mode：该参数决定了返回的标签数组的形式，默认为“categorical”，会返回2维的one-hot编码标签。

target\_size：图像将被resize成224×224。

color\_mode：颜色模式,为“grayscale”,“rgb”之一,默认为“rgb”.代表这些图片是否会被转换为单通道或三通道的图片。

shuffle：是否打乱数据,默认为True。

batch\_size：batch数据的大小。

【代码7-3-8】train.py

# 初始化训练数据生成器

trainGen = trainAug.flow\_from\_directory(

trainPath,

class\_mode="categorical",

target\_size=(224, 224),

color\_mode="rgb",

shuffle=True,

batch\_size=BATCH\_SIZE)

执行代码块。

同样使用flow\_from\_directory方法，构建验证数据生成器。

其中，shuffle=False，表示不打乱数据顺序。

【代码7-3-8】train.py

# 初始化验证数据生成器

valGen = valAug.flow\_from\_directory(

valPath,

class\_mode="categorical",

target\_size=(224, 224),

color\_mode="rgb",

shuffle=False,

batch\_size=BATCH\_SIZE)

执行代码块。

同样使用flow\_from\_directory方法，构建测试数据生成器。

【代码7-3-8】train.py

# 初始化测试数据生成器

testGen = valAug.flow\_from\_directory(

testPath,

class\_mode="categorical",

target\_size=(224, 224),

color\_mode="rgb",

shuffle=False,

batch\_size=BATCH\_SIZE)

执行代码块。

7.3.9 加载VGG16网络

本案例使用的是在Imagenet数据集上训练好的VGG16网络模型，且不包含3层全连接层，如果本地不存在该模型，那么会自动从网络上下载。

【代码7-3-9】train.py

# 加载VGG16网络，不包含FC层

baseModel = VGG16(weights="imagenet", include\_top=False,

input\_tensor=Input(shape=(224, 224, 3)))

执行代码块。

7.3.10 定义新的网络头部

由于我们使用迁移学习的模型，是不包含全连接层的VGG16网络模型，因此我们需要建立新的网络头部。

【代码7-3-10】train.py

# 构建新的头部

headModel = baseModel.output

headModel = Flatten(name="flatten")(headModel)

headModel = Dense(512, activation="relu")(headModel)

headModel = Dropout(0.5)(headModel)

headModel = Dense(len(CLASSES), activation="softmax")(headModel)

执行代码块。

7.3.11 将新的头部放在baseModel上

该部分是将新建立的网络头部放到网络主体上面，以组成完整的网络结构。

【代码7-3-11】train.py

#将新的头部放在baseModel上

model = Model(inputs=baseModel.input, outputs=headModel)

执行代码块。

7.3.12 冻结baseModel的所有层

baseModel部分是已经在Imagenet数据集上训练好的，但FC层是新建立的，因此需要首先训练新建立的网络头部，baseModel部分暂时冻结。

【代码7-3-12】train.py

#循环遍历baseModel的所有层，并冻结他们

for layer in baseModel.layers:

layer.trainable = False

执行代码块。

7.3.13 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和momentum的值，甚至需要不断调整这些参数的值。

【代码7-3-13】train.py

# 编译模型

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

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

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

metrics=["accuracy"])

执行代码块。

7.3.14 训练网络头部

该部分的目的是训练新建立的网络头部，baseModel部分已经冻结。根据代码中设置的超参数WARM\_UP\_EPOCHS，我们需要训练50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。

【代码7-3-14】train.py

# 训练网络的头部（其它部分被冻结了）

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

H = model.fit(trainGen,

steps\_per\_epoch=totalTrain // BATCH\_SIZE,

validation\_data=valGen,

validation\_steps=totalVal // BATCH\_SIZE,

epochs=WARM\_UP\_EPOCHS)

执行代码块。

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20分钟，50个epoch大约需要16个小时。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7.3.2节中的超参数WARM\_UP\_EPOCHS设置为2**，仅用于查看案例效果即可。

模型训练过程如图6所示。（图6中展示的结果WARM\_UP\_EPOCHS设置为10）

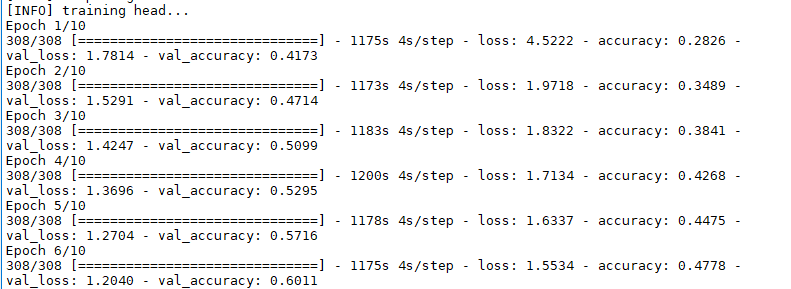


图6 训练网络头部过程

7.3.15 评估模型分类性能

头部训练完成后，我们在测试集上评估模型性能。并且调用plot\_training函数绘制训练过程中的loss和accuracy曲线。

【代码7-3-15】train.py

#网络头部训练结束后，重置测试集生成器，评估模型

print("[INFO] evaluating after fine-tuning network head...")

testGen.reset()

predIdxs = model.predict(testGen,

steps=(totalTest // BATCH\_SIZE) + 1)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification\_report(testGen.classes, predIdxs,

target\_names=testGen.class\_indices.keys()))

plot\_training(H, WARM\_UP\_EPOCHS, WARMUP\_PLOT\_PATH)

执行代码块。

性能评估结果如图7所示。

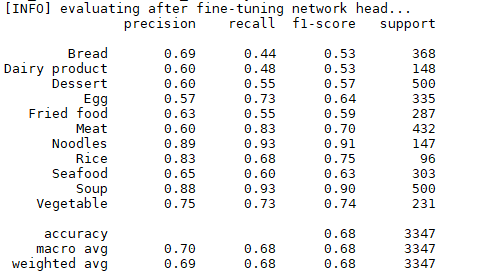


图7 测试评估结果

从图7结果可以看出，在测试集的3347张图像上，accuracy、macro avg（加权平均）、weighted avg（加权平均）几乎都低于0.70左右，说明模型还有进一步提升的空间。

注意：由于模型的新建立的FC层初始会设置随即权重，所以同学们的训练结果不会和本案例一模一样。

网络头部训练过程中，loss和accuracy曲线图如图8所示。当随着epoch的递增，accuracy和loss曲线趋于稳定，说明模型已经训练完成了，不再需要额外的epoch。当train\_acc和val\_acc几乎完全重合，说明模型没有过拟合。上图满足这2个条件。模型的准确率为68%，说明模型还有进一步提升的空间。



图8 网络头部训练过程中的loss和accuracy曲线

7.3.16 解冻baseModel的层

首先，重置训练和验证数据生成器。方便下一步训练模型使用。

【代码7-3-16】train.py

# 重置训练数据和验证数据生成器

trainGen.reset()

valGen.reset()

执行代码块。

由于在卷积神经网络中，浅层网络（前几层）捕获了图像的一般细节，例如边缘信息、颜色信息等，且在该分类任务中，前4个‘卷积池化块‘提取的特征已经足够，深层网络会捕获图像的更深层次的语义信息，因此，我们解冻baseModel的最后1个‘卷积池化层块’（最后三层卷积层和最后的池化层），在接下来的训练中微调其参数。

【代码7-3-16】train.py

# 解冻最后一个卷积层

for layer in baseModel.layers[15:]:

layer.trainable = True

执行代码块。

为了更加直观的了解哪些层是可训练的，哪些层是被冻结的，在控制台打印出baseModel各层的信息以供查看。

【代码7-3-16】train.py

# 显示哪些层是可训练的，哪些是冻结的

for layer in baseModel.layers:

print("{}: {}".format(layer, layer.trainable))

执行代码块。

baseModel各层的信息如图9所示。可以看到只有最后1个‘卷积池化层块’（最后三层卷积层和最后的池化层）是可以被训练的。

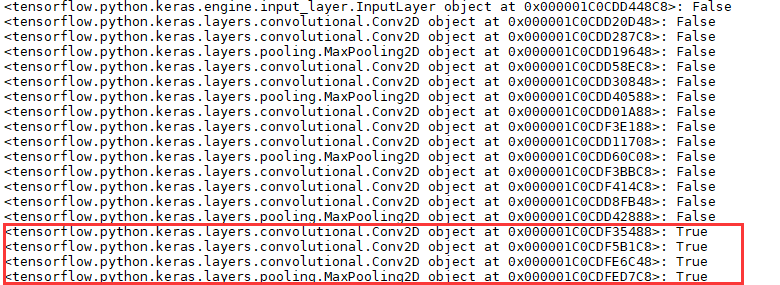


图9 baseModel各层的信息

7.3.17 重新编译模型

为了微调最后一个卷积池化块和网络头部的参数，需要重新编译模型，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和momentum的值，甚至需要不断调整这些参数的值。

【代码7-3-17】train.py

# 重新编译模型

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

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

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

metrics=["accuracy"])

执行代码块。

7.3.18 重新训练模型

该部分的目的是重新训练模型，微调最后一个卷积池化块和网络头部的参数。根据代码中设置的超参数FINE\_TUNING\_EPOCHS，我们需要训练20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。

【代码7-3-18】train.py

# 再次训练模型，微调网络头部和最后的conv block

H = model.fit(

trainGen,

steps\_per\_epoch=totalTrain // BATCH\_SIZE,

validation\_data=valGen,

validation\_steps=totalVal // BATCH\_SIZE,

epochs=FINE\_TUNING\_EPOCHS)

执行代码块。

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20分钟，20个epoch大约需要7个小时。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7.3.2节中的超参数FINE\_TUNING\_EPOCHS设置为2**，仅用于查看案例效果即可。

模型训练过程如图10所示。（图10中展示的结果FINE\_TUNING\_EPOCHS设置为5）

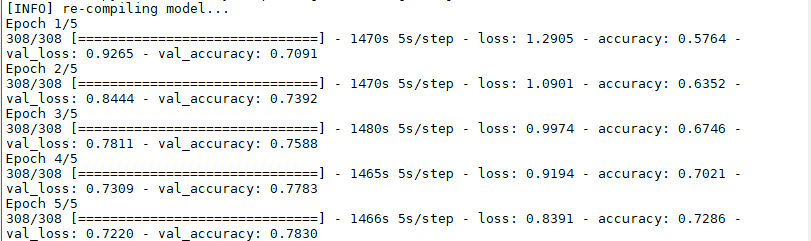


图10 模型训练过程

7.3.19 评估最终模型性能

最后1个‘卷积池化层块’（最后三层卷积层和最后的池化层）和网络头部微调完成后，我们在测试集上评估模型性能。并且调用plot\_training函数绘制训练过程中的loss和accuracy曲线。

【代码7-3-19】train.py

#重置测试集生成器，评估模型性能

print("[INFO] evaluating after fine-tuning network...")

testGen.reset()

predIdxs = model.predict(testGen,

steps=(totalTest // BATCH\_SIZE) + 1)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification\_report(testGen.classes, predIdxs,

target\_names=testGen.class\_indices.keys()))

plot\_training(H, FINE\_TUNING\_EPOCHS, FINE\_TUNING\_PLOT\_PATH)

执行代码块。

性能评估结果，如图11所示。

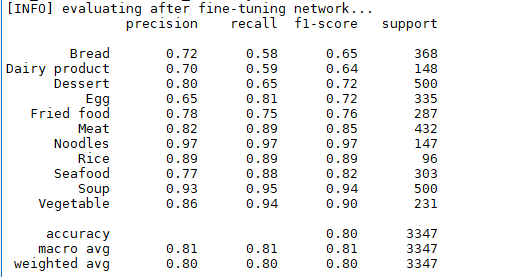


图11 测试评估结果

从图7结果可以看出，在测试集的3347张图像上，accuracy、macro avg（加权平均）、weighted avg（加权平均）几乎都高于0.80左右，说明模型性能已经很不错了。

网络微调过程中，loss和accuracy曲线图如图12所示。当随着epoch的递增，accuracy和loss曲线还未趋于稳定，说明模型还未收敛，仍然需要训练。但由于训练时间过长，仅展示了FINE\_TUNING\_EPOCHS设置为5的训练曲线，模型的准确率已经达到了80%，说明模型分类性能还是很不错的。

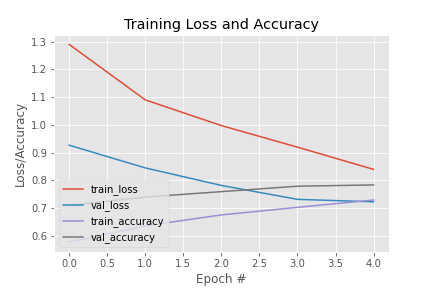


图12 微调过程中的loss和accuracy曲线

7.3.20 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。

对于花了很长时间才训练出来的模型，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7-3-20】train.py

# 保存模型

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

model.save(MODEL\_PATH)

执行代码块。

模型被保存到output目录下。

7.4应用训练好的模型测试单张图片

接下来向predict.py中添加代码。代码比较简单，直接提供完整代码。

【代码7-4】predict.py

'''

应用训练好的模型测试单张图片，并显示结果

'''

# 导入包

from keras.models import load\_model

import numpy as np

import argparse

import imutils

import cv2

# 构造参数解析器，并解析参数

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--image", type=str, required=True,

help="path to our input image")

args = vars(ap.parse\_args())

#全局变量

MODEL\_PATH ='output/food11.model'

# 初始化类别标签名称列表

CLASSES = ["Bread", "Dairy product", "Dessert", "Egg", "Fried food",

"Meat", "Noodles/Pasta", "Rice", "Seafood", "Soup",

"Vegetable/Fruit"]

# 加载图片，将其复制，并更改输出图片尺寸

image = cv2.imread(args["image"])

output = image.copy()

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

#转换图像颜色通道，更改图像尺寸，以输入模型

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

image = cv2.resize(image, (224, 224))

# 转换图像格式，并执行平均减法

image = image.astype("float32")

mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")

image -= mean

# 加载训练的分类模型

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

model = load\_model(MODEL\_PATH)

# 将图片输入模型，进行预测

preds = model.predict(np.expand\_dims(image, axis=0))[0]

i = np.argmax(preds)

label = CLASSES[i]

# 在输出图像上绘制预测结果

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

cv2.putText(output, text, (3, 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

(0, 255, 0), 2)

# 展示分类结果

cv2.imshow("Output", output)

cv2.waitKey(0)

由于代码相对简单，所以没有逐块运行。

在测试图片时需要调用训练好的模型，若同学们没有完成模型的训练，请按以下步骤将训练好的模型复制到output目录下。

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/6a88986783d9477b9c09f42cb8f58c69.zip。解压后会得到一个命名为food11.model的文件夹。将food11.model文件夹复制到output目录下。

运行该文件请切换至Terminal（终端），输入以下命令：

$ cd ~/anaconda3/bin

$ source activate tensorflow

$ cd 代码存放路径/food\_classification (举例：代码存放在桌面上，那么命令是：$ cd ~/Desktop/food\_classification)

$ python predict.py --image dataset/evaluation/Seafood/8\_186.jpg

测试结果如图13所示。可以看到能够正确识别牡蛎为海鲜，且准确率为100%。



图13 正确识别牡蛎为海鲜

同学们若要尝试测试其他图片，只需要更改图片路径即可。如：

$ python predict.py --image （添加图片路径）

8.案例代码

【案例代码】build\_dataset.py

'''

生成训练、验证、测试模型的数据

'''

# 导入包

from imutils import paths

import shutil

import os

#全局变量

ORIG\_INPUT\_DATASET = 'Food-11'# 初始化图像的“原始”输入目录路径

BASE\_PATH = 'dataset'#初始化“新的”新目录的路径

TRAIN = 'training'# 定义训练，测试，验证目录的名称

TEST = 'evaluation'

VAL = 'validation'

# 初始化类别标签名称列表

CLASSES = ["Bread", "Dairy product", "Dessert", "Egg", "Fried food",

"Meat", "Noodles/Pasta", "Rice", "Seafood", "Soup",

"Vegetable/Fruit"]

# 遍历拆分数据

for split in (TRAIN, TEST, VAL):

#抓取当前拆分中的所有图像路径

print("[INFO] processing '{} split'...".format(split))

p = os.path.sep.join([ORIG\_INPUT\_DATASET, split])

imagePaths = list(paths.list\_images(p))

# 遍历图像路径

for imagePath in imagePaths:

# 从文件名中提取类标签

filename = imagePath.split(os.path.sep)[-1]

label = CLASSES[int(filename.split("\_")[0])]

# 构建输出目录的路径

dirPath = os.path.sep.join([BASE\_PATH, split, label])

# 如果输出路径不存在，则创建它

if not os.path.exists(dirPath):

os.makedirs(dirPath)

# 将图像复制到目标路径

p = os.path.sep.join([dirPath, filename])

shutil.copy2(imagePath, p)

【案例代码】train.py

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

'''

使用微调的迁移学习方式训练模型

'''

#导入包

import matplotlib

matplotlib.use("Agg")

from keras.preprocessing.image import ImageDataGenerator

from keras.applications import VGG16

from keras.layers.core import Dropout

from keras.layers.core import Flatten

from keras.layers.core import Dense

from keras.layers import Input

from keras.models import Model

from keras.optimizers import SGD

from sklearn.metrics import classification\_report

from imutils import paths

import matplotlib.pyplot as plt

import numpy as np

# 模型超参数

BATCH\_SIZE = 32

WARM\_UP\_EPOCHS = 50

FINE\_TUNING\_EPOCHS = 20

LR = 1e-4

MOMENTUM = 0.9

#设置全局变量

BASE\_PATH = 'dataset'#新数据集的路径

trainPath = BASE\_PATH+'/training'# 训练集路径

valPath = BASE\_PATH+'/validation'#验证集路径

testPath = BASE\_PATH+'/evaluation'#测试集路径

MODEL\_PATH = 'output/food11.model'#设置训练后模型保存的路径

WARMUP\_PLOT\_PATH = 'output/warmup.png'#保存只训练FC层模型的loss/accuracy曲线的路径

FINE\_TUNING\_PLOT\_PATH = 'output/fine\_tuning.png'# 保存微调后模型的loss/accuracy曲线的路径

# 初始化类别标签名称列表

CLASSES = ["Bread", "Dairy product", "Dessert", "Egg", "Fried food",

"Meat", "Noodles/Pasta", "Rice", "Seafood", "Soup",

"Vegetable/Fruit"]

#构建绘制训练过程loss和accuracy的曲线的函数

def plot\_training(H, N, plotPath):

plt.style.use("ggplot")

plt.figure()

plt.plot(np.arange(0, N), H.history["loss"], label="train\_loss")

plt.plot(np.arange(0, N), H.history["val\_loss"], label="val\_loss")

plt.plot(np.arange(0, N), H.history["accuracy"], label="train\_accuracy")

plt.plot(np.arange(0, N), H.history["val\_accuracy"], label="val\_accuracy")

plt.title("Training Loss and Accuracy")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig(plotPath)

# 确定训练集、验证集、测试集的数据量

totalTrain = len(list(paths.list\_images(trainPath)))

totalVal = len(list(paths.list\_images(valPath)))

totalTest = len(list(paths.list\_images(testPath)))

# 训练数据增强

trainAug = ImageDataGenerator(

rotation\_range=30,

zoom\_range=0.15,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.15,

horizontal\_flip=True,

fill\_mode="nearest")

#初始化验证/测试数据扩充对象（我们将在其中添加均值减法）

valAug = ImageDataGenerator()

# 定义ImageNet均值减法（以RGB顺序），并为每个数据增强对象设置均值减法值

mean = np.array([123.68, 116.779, 103.939], dtype="float32")

trainAug.mean = mean

valAug.mean = mean

# 初始化训练数据生成器

trainGen = trainAug.flow\_from\_directory(

trainPath,

class\_mode="categorical",

target\_size=(224, 224),

color\_mode="rgb",

shuffle=True,

batch\_size=BATCH\_SIZE)

# 初始化验证数据生成器

valGen = valAug.flow\_from\_directory(

valPath,

class\_mode="categorical",

target\_size=(224, 224),

color\_mode="rgb",

shuffle=False,

batch\_size=BATCH\_SIZE)

# 初始化测试数据生成器

testGen = valAug.flow\_from\_directory(

testPath,

class\_mode="categorical",

target\_size=(224, 224),

color\_mode="rgb",

shuffle=False,

batch\_size=BATCH\_SIZE)

# 加载VGG16网络，不包含FC层

baseModel = VGG16(weights="imagenet", include\_top=False,

input\_tensor=Input(shape=(224, 224, 3)))

# 构建新的头部

headModel = baseModel.output

headModel = Flatten(name="flatten")(headModel)

headModel = Dense(512, activation="relu")(headModel)

headModel = Dropout(0.5)(headModel)

headModel = Dense(len(CLASSES), activation="softmax")(headModel)

#将新的头部放在baseModel上

model = Model(inputs=baseModel.input, outputs=headModel)

#循环遍历baseModel的所有层，并冻结他们

for layer in baseModel.layers:

layer.trainable = False

# 编译模型

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

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

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

metrics=["accuracy"])

# 训练网络的头部（其它部分被冻结了）

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

H = model.fit(trainGen,

steps\_per\_epoch=totalTrain // BATCH\_SIZE,

validation\_data=valGen,

validation\_steps=totalVal // BATCH\_SIZE,

epochs=WARM\_UP\_EPOCHS)

#网络头部训练结束后，重置测试集生成器，评估模型

print("[INFO] evaluating after fine-tuning network head...")

testGen.reset()

predIdxs = model.predict(testGen,

steps=(totalTest // BATCH\_SIZE) + 1)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification\_report(testGen.classes, predIdxs,

target\_names=testGen.class\_indices.keys()))

plot\_training(H, WARM\_UP\_EPOCHS, WARMUP\_PLOT\_PATH)

# 重置训练数据和验证数据生成器

trainGen.reset()

valGen.reset()

# 解冻最后一个卷积层

for layer in baseModel.layers[15:]:

layer.trainable = True

# 显示哪些层是可训练的，哪些是冻结的

for layer in baseModel.layers:

print("{}: {}".format(layer, layer.trainable))

# 重新编译模型

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

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

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

metrics=["accuracy"])

# 再次训练模型，微调网络头部和最后的conv block

H = model.fit(

trainGen,

steps\_per\_epoch=totalTrain // BATCH\_SIZE,

validation\_data=valGen,

validation\_steps=totalVal // BATCH\_SIZE,

epochs=FINE\_TUNING\_EPOCHS)

#重置测试集生成器，评估模型性能

print("[INFO] evaluating after fine-tuning network...")

testGen.reset()

predIdxs = model.predict(testGen,

steps=(totalTest // BATCH\_SIZE) + 1)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification\_report(testGen.classes, predIdxs,

target\_names=testGen.class\_indices.keys()))

plot\_training(H, FINE\_TUNING\_EPOCHS, FINE\_TUNING\_PLOT\_PATH)

# 保存模型

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

model.save(MODEL\_PATH)

【案例代码】predict.py

'''

应用训练好的模型测试单张图片，并显示结果

'''

# 导入包

from keras.models import load\_model

import numpy as np

import argparse

import imutils

import cv2

# 构造参数解析器，并解析参数

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--image", type=str, required=True,

help="path to our input image")

args = vars(ap.parse\_args())

#全局变量

MODEL\_PATH ='output/food11.model'

# 初始化类别标签名称列表

CLASSES = ["Bread", "Dairy product", "Dessert", "Egg", "Fried food",

"Meat", "Noodles/Pasta", "Rice", "Seafood", "Soup",

"Vegetable/Fruit"]

# 加载图片，将其复制，并更改输出图片尺寸

image = cv2.imread(args["image"])

output = image.copy()

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

#转换图像颜色通道，更改图像尺寸，以输入模型

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

image = cv2.resize(image, (224, 224))

# 转换图像格式，并执行平均减法

image = image.astype("float32")

mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")

image -= mean

# 加载训练的分类模型

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

model = load\_model(MODEL\_PATH)

# 将图片输入模型，进行预测

preds = model.predict(np.expand\_dims(image, axis=0))[0]

i = np.argmax(preds)

label = CLASSES[i]

# 在输出图像上绘制预测结果

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

cv2.putText(output, text, (3, 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

(0, 255, 0), 2)

# 展示分类结果

cv2.imshow("Output", output)

cv2.waitKey(0)