案例二：基于CNN的肺炎识别

1.案例目的

• 掌握迁移学习中的参数迁移方法；

• 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

本案例基于自定义的网络结构训练X胸片分类模型。在训练模型过程中，巧妙的使用数据增强技术平衡训练数据，并利用迁移学习技术，使用在Imagenet数据集上预训练模型的部分参数初始化自定义网络的部分参数，最后在测试集上评估分类模型的有效性。

3.案例知识点

• Python语言编程；

• 数据增强；

• 参数迁移；

• 训练模型；

• 评估模型

4.案例时长

共4学时，具体安排如下：

• 建立卷积神经网络（0.5学时）

• 训练模型（3学时）

• 评估模型分类性能（0.5学时）

5.案例实验环境

**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

本案例主要分为以下3部分：

1）设计卷积神经网络结构；

2）训练模型；

模型训练使用迁移学习的方法，使用VGG16前4层卷积层的参数初始化设计的网络的前4层卷积层的参数。

3）评估模型性能。

7.案例实验过程

该案例可分为以下几个步骤：

1. 建立卷积神经网络

2. 训练模型

2.1 导入库

2.2 设置模型超参数

2.3 设置全局变量

2.4 图像预处理

2.5 处理训练集

2.6 处理验证集

2.7 数据增强

2.8 训练数据生成器

2.9 加载网络

2.10 迁移学习

2.11 编译模型

2.12 训练模型

2.13 画出accuracy曲线

2.14 画出loss曲线

3. 评估分类模型

3.1 导入库

3.2 设置全局变量

3.3 图像预处理

3.4 处理测试集

3.5 加载模型

3.6 评估模型分类性能

7.1开发准备

在桌面上新建一个目录，名字叫x\_ray。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

获取数据之前，在x\_ray目录下新建一个目录，命名为dataset。获取到的数据将存放在此目录下。

本案例需要1个数据集，是胸透X片（肺炎）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/c71af9500b124c4bb4905079bf3fbd24.zip下载数据集。将下载后的数据集解压，得到chest-xray-pneumonia目录。

将chest-xray-pneumonia目录直接复制到刚刚创建的x\_ray/dataset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x\_ray目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为x\_ray。使用Spyder新建3个Python文件，分别命名为build\_model.py、train\_model.py和test\_model.py。

7.1.4 新建图像输出目录

在x\_ray目录下新建一个目录，命名为output。在output目录下创建model目录和plots目录，output目录用于存放训练好的模型，plots目录用于存放loss、accuracy等曲线。

7.1.5 预训练模型

在x\_ray目录下新建一个目录，命名为state\_of\_art\_model。本案例需要1个预训练的VGG16网络模型，为 vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5。

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/ e93be86b973a4db7afee05f60727738e.h5下载模型，下载后得到文件vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5。将下载后的vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5复制到x\_ray/state\_of\_art\_model目录下。

至此，整个案例的目录结构如图1所示。

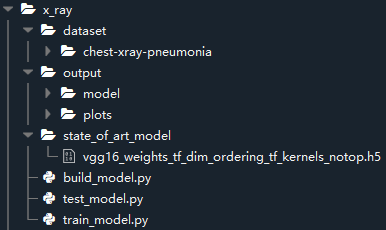


图1 案例目录结构

7.1.6 了解数据

本案例使用的胸透X片（肺炎）数据集分为train（训练集）、val（验证集）、test（测试集）三个目录，train、val、test三个目录下都分别包含NORMAL（正常）、PNEUMONIA（肺炎）两个目录，两个目录分别存放了正常和肺炎两个类别的图片。数据集目录如图2所示。

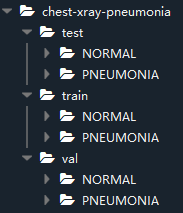
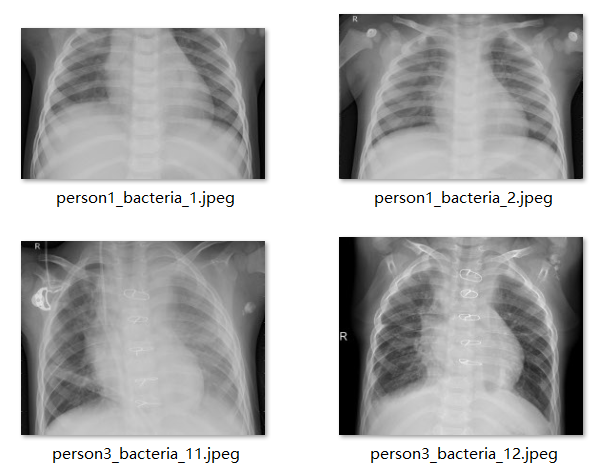
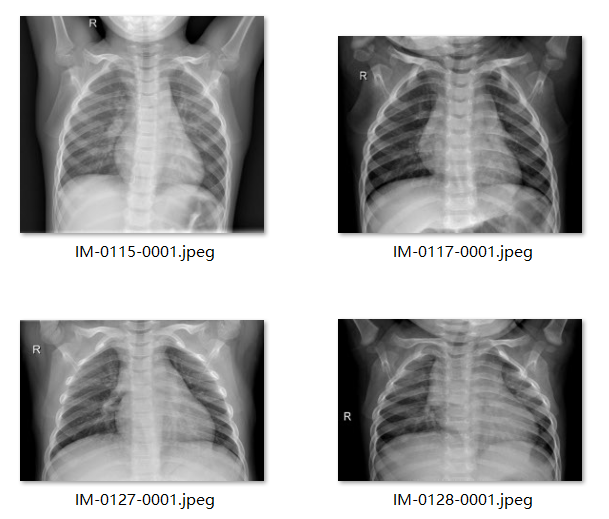


图2 胸透X片（肺炎）数据集

部分胸透X片图像如图3所示。



(a) 有肺炎的X片



(b) 无肺炎的X片

图3 部分胸透X片

7.2建立卷积神经网络

打开build\_model.py文件。正如模块名字所提示的那样，该模块的目的是建立卷积神经网络模型。

SeparableConv2D实现整个深度分离卷积过程，即深度方向的空间卷积 （分别作用于每个输入通道）+ 输出通道混合在一起的逐点卷积。

本案例除前两层外，其他层使用深度可分离卷积，而不使用标准的卷积层（Conv2D），主要目的是降低模型参数量和计算量。

向build\_model.py中添加代码。

【代码7-2】build\_model.py

#导入库

from keras.models import Model

from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input, Flatten, SeparableConv2D

from keras.layers.normalization import BatchNormalization

class CustomNN:

#建立模型

def build\_model():

# 输入层，输入图像大小为224\*224

input\_img = Input(shape=(224,224,3), name='ImageInput')

# 卷积层 => 卷积层 => 池化层

x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1\_1')(input\_img)

x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1\_2')(x)

x = MaxPooling2D((2,2), name='pool1')(x)

# 深度可分离卷积层 => 深度可分离卷积层 => 池化层

x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2\_1')(x)

x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2\_2')(x)

x = MaxPooling2D((2,2), name='pool2')(x)

# 深度可分离卷积层 => BN层（归一化）=> 深度可分离卷积层 => BN层 => 深度可分离卷积层 => 池化层

x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3\_1')(x)

x = BatchNormalization(name='bn1')(x)

x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3\_2')(x)

x = BatchNormalization(name='bn2')(x)

x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3\_3')(x)

x = MaxPooling2D((2,2), name='pool3')(x)

# 深度可分离卷积层 => BN层=> 深度可分离卷积层 => BN层 => 深度可分离卷积层 => 池化层

x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4\_1')(x)

x = BatchNormalization(name='bn3')(x)

x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4\_2')(x)

x = BatchNormalization(name='bn4')(x)

x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4\_3')(x)

x = MaxPooling2D((2,2), name='pool4')(x)

# FC => Dropout => FC => Dropout => FC

x = Flatten(name='flatten')(x)

x = Dense(1024, activation='relu', name='fc1')(x)

x = Dropout(0.7, name='dropout1')(x)

x = Dense(512, activation='relu', name='fc2')(x)

x = Dropout(0.5, name='dropout2')(x)

x = Dense(2, activation='softmax', name='fc3')(x)

#模型输入输出，并返回模型结构

model = Model(inputs=input\_img, outputs=x)

return model

代码不需要执行，模型训练和测试时会调用该模块。

7.2训练模型

接下来向train\_model.py中添加代码。该模块作用是加载数据源利用数据增强和迁移学习等技术训练分类模型，输出训练过程中的accuracy和loss的训练曲线图，并保存模型。

注：本案例在6核8G的CPU上运行一个epoch需要23分钟左右，完成训练需要6小时左右。

7.3.1 导入库

把所有用到的库都导入。

【代码7-3-1】train\_model.py

# 导入库

import h5py

import numpy as np

import matplotlib.pyplot as plt

import imgaug.augmenters as iaa

from keras.optimizers import Adam

from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.utils import to\_categorical

from build\_model import CustomNN

import cv2

from pathlib import Path

执行代码块。

7.3.2 设置模型超参数

这些超参数都是训练模型的一些指定参数，比如一批加载几张图片，学习率是多少。如果模型性能不好或者出现其他问题，要调整这些参数。在案例中，EPOCHS的数量不足，会导致模型欠拟合。

【代码7-3-2】train\_model.py

#设置超参数

BATCH\_SIZE = 32

EPOCHS = 15

LR = 1e-4 #学习率

DECAY = 1e-5 #衰减率

ES\_PATIENCE = 5 #在模型训练过程中，如果连续超过5个epoch，模型性能还没有提升，则终止训练

执行代码块。

7.3.3 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、loss和accuracy曲线保存的路径等。设置为全局变量比较方便使用。

【代码7-3-3】train\_model.py

#设置全局变量

data\_dir = Path('dataset/chest-xray-pneumonia') # 数据集路径

train\_normal\_cases\_dir = data\_dir / 'train/NORMAL' # 训练集中正常样本的子目录

train\_pneumonia\_cases\_dir = data\_dir / 'train/PNEUMONIA' # 训练集中有肺炎病例样本的子目录

val\_normal\_cases\_dir = data\_dir / 'val/NORMAL' # 验证集中正常样本的子目录

val\_pneumonia\_cases\_dir = data\_dir / 'val/PNEUMONIA' # 验证集中正常样本的子目录

vgg16\_path = 'state\_of\_art\_model/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'#vgg16预训练模型路径

saved\_model\_name = 'output/model/best\_model.h5'#保存最佳训练模型路径

output\_plot\_loss\_path = 'output/plots/plot\_loss.png' #保存loss曲线路径

output\_plot\_accuracy\_path = 'output/plots/plot\_accuracy.png' #保存accuracy曲线路径

执行代码块。

7.3.4 图像预处理

该模块的目的是对数据进行预处理，包含读取图片，更改图像尺寸，转换图像通道、图像颜色空间转换、图像标准化等操作。其中，因为该数据集中除三通道图像外，还包含部分单通道的图像，因此需要进行判断，若为单通道图像，要将其转换为3通道；此外，使用cv2.imread读取图像时，读进来的是BGR格式，图像色彩发生了变化，因此要进行图像颜色空间转换这一步。

【代码7-3-4】train\_model.py

# 图像预处理

def image\_preprocess(img\_path):

# 1）读取图片

img = cv2.imread(img\_path)

# 2）将图像尺寸resize到224x224

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

# 3）如果图像通道数为1，将其转换为3个通道的图像

if img.shape[2] ==1:

img = np.dstack([img, img, img])

# 4）图像颜色空间转换

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

# 5）图像标准化

img = img.astype(np.float32)/255.

#返回图像

return img

执行代码块。

7.3.5 处理训练集数据

该模块的主要作用是对训练集中的数据进行处理。

首先，生成x\_train和y\_train两个空列表，以方便后续存储训练集数据的图像和标签信息。

【代码7-3-5-1】train\_model.py

print('[INFO] 处理训练集数据...')

# 生成x\_train和y\_train

x\_train = []

y\_train = []

执行代码块。

从现在开始，建议切换到Spyder的Variable explorer面板，查看变量信息。

生成x\_train和y\_train两个空列表，如图4所示。

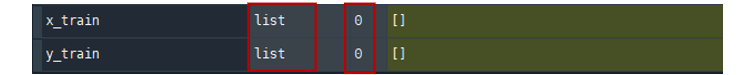


图4 x\_train和y\_train变量

其次，获得所有NORMAL类（无肺炎）和PNEUMONIA类（有肺炎）样本。

【代码7-3-5-2】train\_model.py

# 获得正常（无肺炎）和有肺炎的所有样本

train\_normal\_cases = train\_normal\_cases\_dir.glob('\*.jpeg')

train\_pneumonia\_cases = train\_pneumonia\_cases\_dir.glob('\*.jpeg')

执行代码块。

然后，调用7.3.4中定义的图像预处理函数image\_preprocess()对NORMAL类（无肺炎）图像进行预处理，并加载到x\_train中。

因为CNN输出层的激活函数用的是softmax，所以需要把因变量处理成哑变量。并将结果加载到y\_train中。

【代码7-3-5-3】train\_model.py

# 把NORMAL类（无肺炎）图片加载到x\_train和y\_train中

for img\_path in train\_normal\_cases:

x\_train.append(image\_preprocess(str(img\_path)))

y\_train.append(to\_categorical(0, num\_classes=2))

执行代码块。

执行完毕后，1191张NORMAL类（无肺炎）图像的像素数据存储在x\_train变量中，且经过图像预处理，图像大小都是224x224，图片像素值在[0, 1]之间，效果如图5所示。

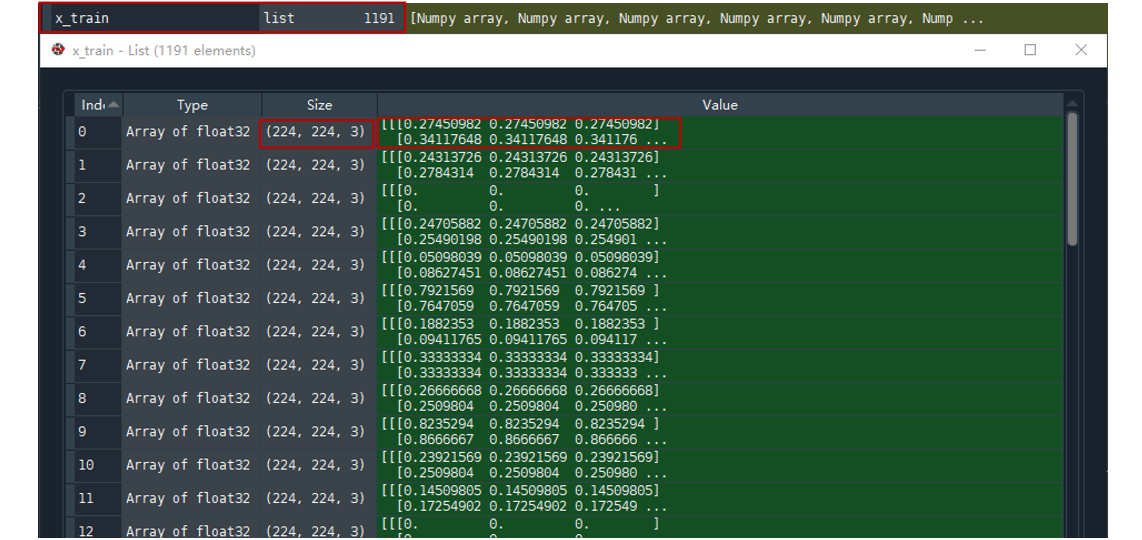


图5 1191张NORMAL类（无肺炎）图像的像素数据存储在x\_train变量中

同理，1191张NORMAL类（无肺炎）图像的类别数据存储在y\_train变量中。效果如图4所示。

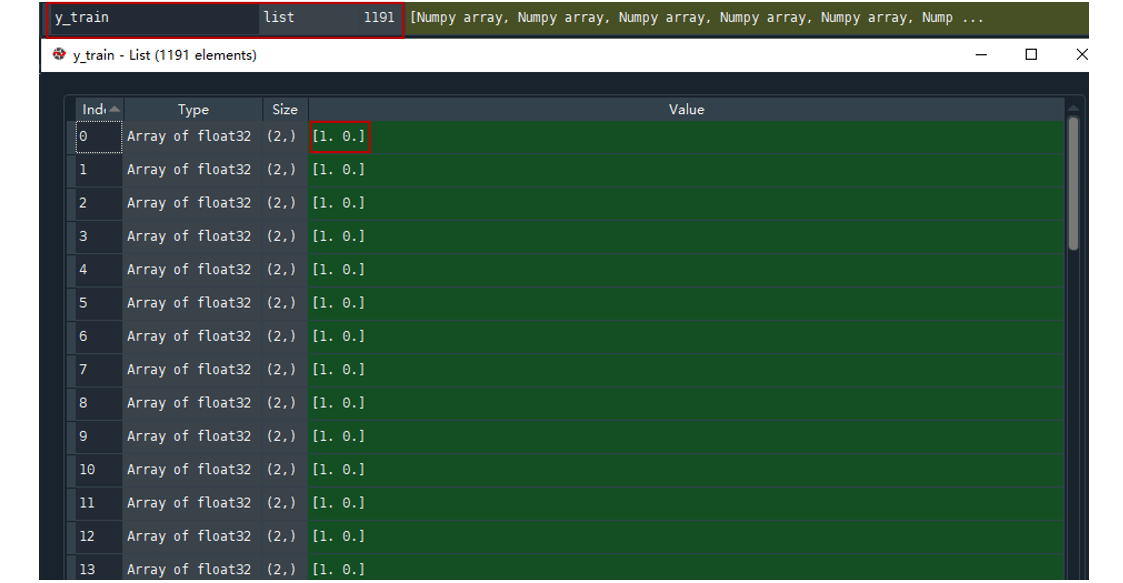


图6 1191张NORMAL类（无肺炎）图像的类别数据存储在y\_train变量中

再调用7.3.4中定义的图像预处理函数对PNEUMONIA类（有肺炎）的图像进行预处理，并加载到x\_train中。使用to\_categorical方法处理因变量，并将结果加载到y\_train中。

【代码7-3-5-4】train\_model.py

# 把肺炎图片加载到x\_train和y\_train中

for img\_path in train\_pneumonia\_cases:

x\_train.append(image\_preprocess(str(img\_path)))

y\_train.append(to\_categorical(1, num\_classes=2))

执行代码块。

执行完毕后，x\_train和y\_train的数据数量都变成了4916，说明PNEUMONIA类（有肺炎）图像的像素信息和类别信息都加载到了x\_train和y\_train变量中，有3725张（4916-1191）。效果如图7所示。

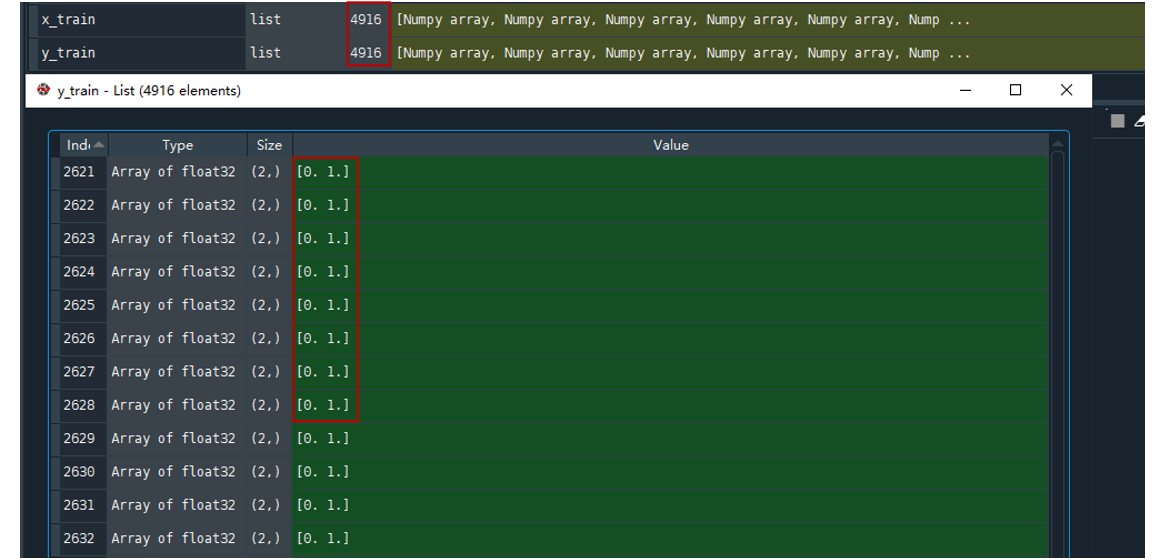


图7 加入PNEUMONIA类（有肺炎）图像像素信息和类别信息的x\_train和y\_train变量

最后，将list转换为Numpy类型。

【代码7-3-5-5】train\_model.py

# list转换为Numpy类型

x\_train = np.array(x\_train)

y\_train = np.array(y\_train)

执行代码块。

x\_train和y\_train由list类型转换为Numpy类型，效果如图8所示。

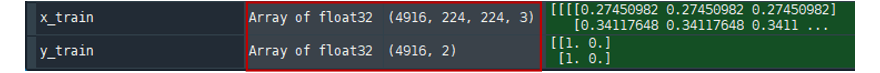


图8 Numpy类型的x\_train和y\_train变量

7.3.6 处理验证集

该模块的作用是对验证集中的图像进行预处理。处理过程和7.3.5节中对训练集数据处理的过程相同。

在代码执行过程中请注意查看x\_val和y\_val变量的变化，这里不再一一展示。

首先，生成x\_val和y\_val两个空列表，以存储验证集的图像和标签信息。

【代码7-3-6-1】train\_model.py

print('[INFO] 处理验证集数据...')

# 生成x\_val和y\_val

x\_val = []

y\_val = []

执行代码块。

其次，获得正常和有肺炎病例样本的所有图像，其中NORMAL类（无肺炎）图像和PNEUMONIA类（有肺炎）图像各158张，共316张。

【代码7-3-6-2】train\_model.py

# 获得NORMAL类（无肺炎）和有肺炎的所有图像

val\_normal\_cases = val\_normal\_cases\_dir.glob('\*.jpeg')

val\_pneumonia\_cases = val\_pneumonia\_cases\_dir.glob('\*.jpeg')

执行代码块。

然后，调用7.3.4中定义的图像预处理函数对NORMAL类（无肺炎）图像进行预处理，并加载到x\_val中。使用to\_categorical方法将因变量处理为哑变量，将结果加载到y\_val中。

【代码7-3-6-3】train\_model.py

# 把正常图片加载到x\_val和y\_val中

for img\_path in val\_normal\_cases:

x\_val.append(image\_preprocess(str(img\_path)))

y\_val.append(to\_categorical(0, num\_classes=2))

执行代码块。

再调用7.3.4中定义的图像预处理函数对PNEUMONIA类（有肺炎）的图像进行预处理，并加载到x\_val中。使用to\_categorical方法将因变量处理为哑变量，将结果加载到y\_val中。

【代码7-3-6-4】train\_model.py

# 把肺炎图片加载到x\_val和y\_val中

for img\_path in val\_pneumonia\_cases:

x\_val.append(image\_preprocess(str(img\_path)))

y\_val.append(to\_categorical(1, num\_classes=2))

执行代码块。

最后，将list转换为Numpy类型。

【代码7-3-6-5】train\_model.py

# list转换为Numpy类型

x\_val = np.array(x\_val)

y\_val = np.array(y\_val)

执行代码块。

7.3.7 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。通过数据增强有助于提高模型的鲁棒性。而且在数据样本不平衡的情况下，可以尝试用以平衡样本分布。

【代码7-3-7】train\_model.py

print('[INFO] 图像增强...')

# 对训练集做数据增强-指定数据增强的3种方法

seq = iaa.OneOf([

iaa.Fliplr(), # 水平翻转

iaa.Affine(rotate=20), # 旋转

iaa.Multiply((1.2, 1.5))]) # 随机亮度

执行代码块。

7.3.8 训练数据生成器

因为训练集中图像数量较多，因此义一个图像生成器以方便对训练数据进行处理。且因为训练集中PNEUMONIA类（有肺炎）是NORMAL类（无肺炎）样本数量的3倍左右，因此对NORMAL类（无肺炎）样本进行数据增强，以达到平衡数据样本的目的。

首先，定义函数，名为data\_generator，传入x\_train, y\_train, batch\_size三个参数。并获得数据的样本总数，以及一共需要处理多少批数据。

【代码7-3-8-1】train\_model.py

# 对训练集做数据增强-训练数据生成器

# 有肺炎的样本是无肺炎样本数量的3倍，这里增强无肺炎样本

def data\_generator(x\_train, y\_train, batch\_size):

# 获取数据的样本总数

x\_train\_len = len(x\_train)

# 得到总共多少批

steps = x\_train\_len//batch\_size

执行代码块。

定义两个numpy数组以包含批处理数据和标签，作为最终返回值。

【代码7-3-8-2】train\_model.py

# 定义两个numpy数组以包含批处理数据和标签，作为最终返回值

batch\_data = np.zeros((batch\_size, 224, 224, 3), dtype=np.float32)

batch\_labels = np.zeros((batch\_size,2), dtype=np.float32)

执行代码块。

使用np.arange()函数获取所有输入数据的索引，并初始化批的计数器。

【代码7-3-8-3】train\_model.py

# 获取所有输入数据索引的numpy数组

indices = np.arange(x\_train\_len)

i =0 # 初始化批的计数器

执行代码块。

下面打乱图像顺序，并获取一批数据。

【代码7-3-8-4】train\_model.py

while True:

# 随机打乱数据顺序

np.random.shuffle(indices)

# 获取一批数据

count = 0

next\_batch = indices[(i\*batch\_size):(i+1)\*batch\_size] # 一批数据的索引

执行代码块。

下面遍历一批数据，对该批数据中的每一张图像进行处理，对NORMAL类（无肺炎）样本进行数据增强，而PNEUMONIA类（有肺炎）样本不进行数据增强处理。

【代码7-3-8-5】train\_model.py

# 遍历一批数据，同时获得索引和值

for j, idx in enumerate(next\_batch):

img = x\_train[idx] # 预处理后的数据

encoded\_label = y\_train[idx] # 独热编码后的标签

# 将数据及标签保存在两个数组中

batch\_data[count] = img

batch\_labels[count] = encoded\_label

#正常样本（无肺炎）量较少，采用数据增强

if encoded\_label[0]==1 and count < batch\_size-2: # encoded\_label[0]==1 相当于label==0

aug\_img1 = seq.augment\_image(img) # 图像增强

aug\_img2 = seq.augment\_image(img)

# 将增强后的样本保存至数组

batch\_data[count+1] = aug\_img1

batch\_labels[count+1] = encoded\_label

batch\_data[count+2] = aug\_img2

batch\_labels[count+2] = encoded\_label

count +=2

# 肺炎样本不需要数据增强

else:

count+=1

#处理完当前批，跳出该循环

if count==batch\_size-1:

break

#处理完一批数据，批计数器加1

i+=1

#返回图像数组batch\_data和标签数组batch\_labels

yield batch\_data, batch\_labels

#当所有批的数据处理完，计数器重置为0

if i>=steps:

i=0

执行代码块。

7.3.9 加载网络

该模块用于加载网络结构。

【代码7-3-9】train\_model.py

#加载网络

model = CustomNN.build\_model()

执行代码块。

7.3.10 迁移学习

迁移学习可以有助于更快更好的训练模型，这里将进行部分迁移学习，其余模型将从头开始训练。

首先，通过h5py读取vgg16预训练模型各层的参数；然后，选取需要的卷积层的权值w和偏置b；最后，将得到的参数w和b交给新网络的对应层。

因为CNN的浅层网络捕获了图像的一般细节，例如边缘信息、颜色信息等，所以这里选取前两层卷积层和第三四层的深度可分离卷积层使用预训练的vgg16的前4层卷积层的参数进行初始化，训练过程中，只需要微调这几层的参数，其余网络部分进行随机初始化。

【代码7-3-10】train\_model.py

#模型迁移（参数迁移）。

f = h5py.File(vgg16\_path, 'r')

#将VGG16的13层卷积层分为5个block

#block1

#得到预训练模型的第一层的权重和偏置，block1\_conv1

w,b = f['block1\_conv1']['block1\_conv1\_W\_1:0'], f['block1\_conv1']['block1\_conv1\_b\_1:0']

#将权重和偏置分别赋给新网络的第一层，卷积层

model.layers[1].set\_weights = [w,b]

#得到预训练模型的第二层的权重和偏置，block1\_conv2

w,b = f['block1\_conv2']['block1\_conv2\_W\_1:0'], f['block1\_conv2']['block1\_conv2\_b\_1:0']

#将权重和偏置分别赋给新网络的第二层，卷积层

model.layers[2].set\_weights = [w,b]

#block2

#得到预训练模型的第三层的权重和偏置，block2\_conv1

w,b = f['block2\_conv1']['block2\_conv1\_W\_1:0'], f['block2\_conv1']['block2\_conv1\_b\_1:0']

#将权重和偏置分别赋给新网络的第三层，深度可分离卷积

model.layers[4].set\_weights = [w,b]

#得到预训练模型的第四层的权重和偏置，block2\_conv2

w,b = f['block2\_conv2']['block2\_conv2\_W\_1:0'], f['block2\_conv2']['block2\_conv2\_b\_1:0']

#将权重和偏置分别赋给新网络的第四层，深度可分离卷积

model.layers[5].set\_weights = [w,b]

f.close()

执行代码块。

7.3.11 编译模型

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

优化算法选择Adam，一般需要手动指定learning rate、decay、的值，甚至需要不断调整这些超参数。在7.3.2小节中已经设置了模型超参数，这些超参数是不断调试之后得到的，不是随意指定的。

另外，这里使用了EarlyStopping，以val\_loss为标准，连续patience个EPOCHS后val\_loss没有下降，则提前停止训练。此处只保存最佳模型，且使用save\_weights\_only只保存模型的参数。

模型编译的代码如下：

【代码7-3-11】train\_model.py

#编译模型

opt = Adam(lr=LR, decay=DECAY)

es = EarlyStopping(monitor='val\_loss', patience = ES\_PATIENCE)

chkpt = ModelCheckpoint(filepath=saved\_model\_name, save\_best\_only=True, save\_weights\_only=True)

model.compile(loss='binary\_crossentropy', metrics=['accuracy'],optimizer=opt)

执行代码块。

7.3.12 训练模型

该模块的作用是训练模型，根据代码中设置的全局常量的值，模型训练15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。

模型训练时调用图像生成器，即对正常样本使用了图像增强技术。

训练模型代码如下：

【代码7-3-12】train\_model.py

# 训练模型

print('[INFO] 训练模型...')

H = model.fit(data\_generator(x\_train, y\_train, BATCH\_SIZE),

validation\_data=(x\_val, y\_val),

steps\_per\_epoch=len(x\_train)//BATCH\_SIZE,

epochs=EPOCHS,

callbacks=[es, chkpt],

verbose = 1)

执行代码块。

本案例在6核8G的CPU上运行一个epoch需要23分钟左右，完成训练需要6小时左右。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EPOCHS参数设置为1，仅用于查看实验效果即可。

模型训练过程如图9所示：

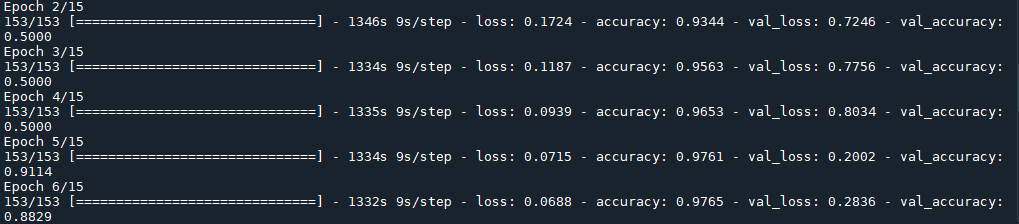


图9 模型训练过程

7.3.13 画出训练过程中的accuracy曲线

7.3.12节已经提到模型训练信息存储在H变量中，该模块根据H变量画出accuracy曲线并保存图片。

【代码7-3-13】train\_model.py

# 画出训练过程中的准确率

N = len(H.history['loss'])

plt.figure()

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

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

plt.title('Training & Validation accuracy on X-ray images')

plt.xlabel("Epoch #")

plt.ylabel("Accuracy")

plt.legend()

plt.savefig(output\_plot\_accuracy\_path)

plt.close()

执行代码块。

生成的accuracy曲线如图10所示。

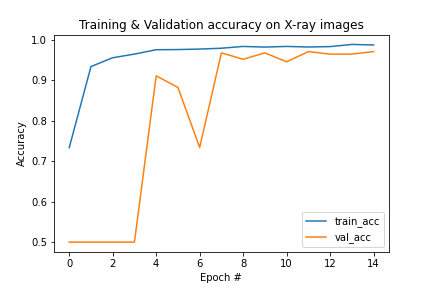


图10 accuracy曲线图

由图10可以看到，当随着epoch的递增，train\_acc和val\_acc曲线都趋于稳定，说明模型已经训练完成了，不再需要额外的epoch。且val\_acc达到了与97%，说明模型性能不错。

7.3.14 画出训练过程中的loss曲线

7.3.12节已经提到模型训练信息存储在H变量中，该模块根据H变量画出loss曲线并保存图片。

【代码7-3-14】train\_model.py

# 画出训练过程中的损失值

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.title('Training & Validation loss on X-ray images')

plt.xlabel("Epoch #")

plt.ylabel("Loss")

plt.legend()

plt.savefig(output\_plot\_loss\_path)

plt.close()

执行代码块。

生成的曲线如图10所示。

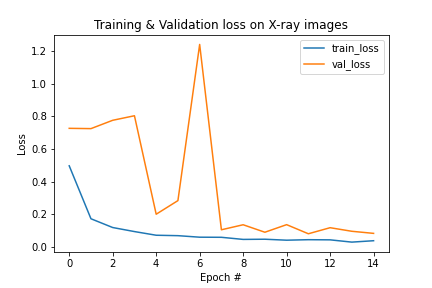


图11 loss曲线图

由图11可以看到，当随着epoch的递增，train\_loss和val\_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。

7.4评估模型分类性能

向test\_model.py文件中添加代码，该部分的目的是在测试集上评估模型分类性能，代码如下：

7.4.1 导入库

把所有用到的库都导入。

【代码7-4-1】test\_model.py

#导入库

import numpy as np

from pathlib import Path

from keras.utils import to\_categorical

from sklearn.metrics import classification\_report

from build\_model import CustomNN

import cv2

执行代码块。

7.4.2 设置全局变量

这些变量用于指定数据源的路径、模型存放等的路径。设置为全局变量比较方便使用。

【代码7-4-2】test\_model.py

#设置全局变量

data\_dir = Path('dataset/chest-xray-pneumonia')#数据集路径

test\_normal\_cases\_dir = data\_dir / 'test/NORMAL'#测试集中NORMAL类样本的子目录

test\_pneumonia\_cases\_dir = data\_dir / 'test/PNEUMONIA'#测试集中PNEUMONIA类样本的子目录

model\_path = 'output/model/best\_model.h5'#最佳模型路径

执行代码块。

7.4.3 图像预处理

该模块的目的是对数据进行预处理，包含读取图片、更改图像尺寸、转换图像通道、图像颜色空间转换、图像标准化等操作。

【代码7-4-3】test\_model.py

# 图像预处理

def image\_preprocess(img\_path):

# 1）读取图片

img = cv2.imread(img\_path)

# 2）将图像尺寸resize到224x224

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

# 3）如果图像通道数为1，将其转换为3个通道的图像

if img.shape[2] ==1:

img = np.dstack([img, img, img])

# 4）图像颜色空间转换

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

# 5）图像标准化

img = img.astype(np.float32)/255.

return img

执行代码块。

7.4.4 处理测试集数据

该模块的作用是对测试集中的图像进行预处理。处理过程和7.3.5节中对训练集数据处理的过程相同。

在代码执行过程中请注意查看x\_test和y\_test变量的变化，这里不再一一展示。

首先，生成x\_test和y\_test两个空列表，以存储测试集的图像和标签信息。

【代码7-4-4-1】test\_model.py

print('[INFO] 处理测试集数据...')

# 生成x\_test和y\_test

x\_test = []

y\_test = []

执行代码块。

其次，获得正常和有肺炎病例样本的所有图像，其中NORMAL类（无肺炎）图像，234张和PNEUMONIA类（有肺炎）图像390张，共624张。

【代码7-4-4-2】test\_model.py

# 获得NORMAL类（无肺炎）和有肺炎病例样本的列表

test\_normal\_cases = test\_normal\_cases\_dir.glob('\*.jpeg')

test\_pneumonia\_cases = test\_pneumonia\_cases\_dir.glob('\*.jpeg')

执行代码块。

然后，调用7.4.3中定义的图像预处理函数对NORMAL类（无肺炎）图像进行预处理，并加载到x\_test中。使用to\_categorical方法将因变量处理为哑变量，并将结果加载到y\_test中。

【代码7-4-4-3】test\_model.py

# 把NORMAL类（无肺炎）图片加载到x\_test和y\_test中

for img in test\_normal\_cases:

x\_test.append(image\_preprocess(str(img)))

y\_test.append(to\_categorical(0, num\_classes=2))

执行代码块。

再调用7.4.3中定义的图像预处理函数对PNEUMONIA类（有肺炎）的图像进行预处理，并加载到x\_test中。使用to\_categorical方法将因变量处理为哑变量，将结果加载到y\_test中。

【代码7-4-4-4】test\_model.py

# 把肺炎图片加载到x\_test和y\_test中

for img in test\_pneumonia\_cases:

x\_test.append(image\_preprocess(str(img)))

y\_test.append(to\_categorical(1, num\_classes=2))

执行代码块。

最后，将list转换为Numpy类型。

【代码7-4-4-5】test\_model.py

# list转换为Numpy类型

x\_test = np.array(x\_test)

y\_test = np.array(y\_test)

执行代码块。

7.4.5 加载模型

该部分的作用是加载训练好的模型参数。

【代码7-4-5】test\_model.py

#加载模型

model = CustomNN.build\_model()

model.load\_weights(model\_path)

执行代码块。

7.4.6 评估模型分类性能

当数据样本不平衡时，accuracy不是最佳的评价指标。比如，当数据集中包含95个负样本，5个正样本时，95%的准确度根本没有意义。分类器可能会将所有的样本都标记为负样本，但仍然可以达到95%的准确度。因此，需要使用更好的评价指标，precision、recall、F1-score能够很好的解决此类问题，并计算对应的macro avg（宏平均）和weighted avg(加权平均)。

评估模型分类性能。

【代码7-4-6】test\_model.py

#评估模型分类性能

print("[INFO] 评估模型分类性能...")

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

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

predictions.argmax(axis=1)))

执行代码块。

执行完毕后，得到的结果如图12所示。

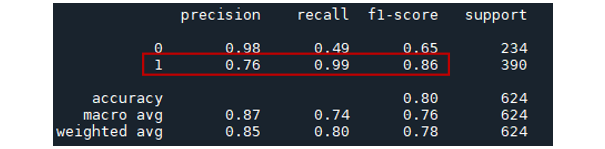


图12 precision和recall结果

图12 中，0表示NORMAL类（无肺炎），1表示PNEUMONIA类（有肺炎），可以看到PNEUMONIA类（有肺炎）的precision、recall、F1-score的值分别达到了0.76、0.99、0.86，在整个测试集上，macro avg（宏平均）和weighted avg(加权平均)都在0.74以上，证明模型能够比较有效的对肺炎和非肺炎样本进行分类，但还有进一步提升的空间。

结论：在本案例中，首先，定义了自己的网络结构，其次，在训练模型时，巧妙的使用数据增强技术平衡了训练数据，并利用了迁移学习技术，使用在Imagenet数据集上预训练的模型的前几层初始化了网络的几层。最后，在测试集上对模型的分类性能进行评价，证明了模型对胸透X片分类的有效性。同学们如果有时间可以将本案例用到的技术在别的分类场景中进行实验。

8.案例代码

【案例代码】build\_model.py

#导入库

from keras.models import Model

from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input, Flatten, SeparableConv2D

from keras.layers.normalization import BatchNormalization

class CustomNN:

#建立模型

def build\_model():

# 输入层，输入图像大小为224\*224

input\_img = Input(shape=(224,224,3), name='ImageInput')

# 卷积层 => 卷积层 => 池化层

x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1\_1')(input\_img)

x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1\_2')(x)

x = MaxPooling2D((2,2), name='pool1')(x)

# 深度可分离卷积层 => 深度可分离卷积层 => 池化层

x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2\_1')(x)

x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2\_2')(x)

x = MaxPooling2D((2,2), name='pool2')(x)

# 深度可分离卷积层 => BN层（归一化）=> 深度可分离卷积层 => BN层 => 深度可分离卷积层 => 池化层

x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3\_1')(x)

x = BatchNormalization(name='bn1')(x)

x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3\_2')(x)

x = BatchNormalization(name='bn2')(x)

x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3\_3')(x)

x = MaxPooling2D((2,2), name='pool3')(x)

# 深度可分离卷积层 => BN层=> 深度可分离卷积层 => BN层 => 深度可分离卷积层 => 池化层

x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4\_1')(x)

x = BatchNormalization(name='bn3')(x)

x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4\_2')(x)

x = BatchNormalization(name='bn4')(x)

x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4\_3')(x)

x = MaxPooling2D((2,2), name='pool4')(x)

# FC => Dropout => FC => Dropout => FC

x = Flatten(name='flatten')(x)

x = Dense(1024, activation='relu', name='fc1')(x)

x = Dropout(0.7, name='dropout1')(x)

x = Dense(512, activation='relu', name='fc2')(x)

x = Dropout(0.5, name='dropout2')(x)

x = Dense(2, activation='softmax', name='fc3')(x)

#模型输入输出，并返回模型结构

model = Model(inputs=input\_img, outputs=x)

return model

【案例代码】train\_model.py

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

#导入库

import h5py

import numpy as np

import matplotlib.pyplot as plt

import imgaug.augmenters as iaa

from keras.optimizers import Adam

from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.utils import to\_categorical

from build\_model import CustomNN

import cv2

from pathlib import Path

#设置超参数

BATCH\_SIZE = 32

EPOCHS = 15

LR = 1e-4 #学习率

DECAY = 1e-5 #衰减率

ES\_PATIENCE = 5 #在模型训练过程中，如果超过5个epoch，模型性能还没有提升，则终止训练

#设置全局变量

data\_dir = Path('dataset/chest-xray-pneumonia') # 数据集路径

train\_normal\_cases\_dir = data\_dir / 'train/NORMAL' # 训练集中正常样本的子目录

train\_pneumonia\_cases\_dir = data\_dir / 'train/PNEUMONIA' # 训练集中有肺炎病例样本的子目录

val\_normal\_cases\_dir = data\_dir / 'val/NORMAL' # 验证集中正常样本的子目录

val\_pneumonia\_cases\_dir = data\_dir / 'val/PNEUMONIA' # 验证集中正常样本的子目录

vgg16\_path = 'state\_of\_art\_model/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'#vgg16预训练模型路径

saved\_model\_name = 'output/model/best\_model.h5'#保存最佳训练模型路径

output\_plot\_loss\_path = 'output/plots/plot\_loss.png' #保存loss曲线路径

output\_plot\_accuracy\_path = 'output/plots/plot\_accuracy.png' #保存accuracy曲线路径

# 图像预处理

def image\_preprocess(img\_path):

# 1）读取图片

img = cv2.imread(img\_path)

# 2）将图像尺寸resize到224x224

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

# 3）如果图像通道数为1，将其转换为3个通道的图像

if img.shape[2] ==1:

img = np.dstack([img, img, img])

# 4）图像颜色空间转换

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

# 5）图像标准化

img = img.astype(np.float32)/255.

return img

print('[INFO] 处理训练集数据...')

# 生成x\_train和y\_train

x\_train = []

y\_train = []

# 获得无肺炎和有肺炎的所有图像

train\_normal\_cases = train\_normal\_cases\_dir.glob('\*.jpeg')

train\_pneumonia\_cases = train\_pneumonia\_cases\_dir.glob('\*.jpeg')

# 把正常图片加载到x\_train和y\_train中

for img\_path in train\_normal\_cases:

x\_train.append(image\_preprocess(str(img\_path)))

y\_train.append(to\_categorical(0, num\_classes=2))

# 把肺炎图片加载到x\_train和y\_train中

for img\_path in train\_pneumonia\_cases:

x\_train.append(image\_preprocess(str(img\_path)))

y\_train.append(to\_categorical(1, num\_classes=2))

# list转换为Numpy类型

x\_train = np.array(x\_train)

y\_train = np.array(y\_train)

print('[INFO] 处理验证集数据...')

# 生成x\_val和y\_val

x\_val = []

y\_val = []

# 获得无肺炎和有肺炎的所有图像

val\_normal\_cases = val\_normal\_cases\_dir.glob('\*.jpeg')

val\_pneumonia\_cases = val\_pneumonia\_cases\_dir.glob('\*.jpeg')

# 把正常图片加载到x\_val和y\_val中

for img\_path in val\_normal\_cases:

x\_val.append(image\_preprocess(str(img\_path)))

y\_val.append(to\_categorical(0, num\_classes=2))

# 把肺炎图片加载到x\_val和y\_val中

for img\_path in val\_pneumonia\_cases:

x\_val.append(image\_preprocess(str(img\_path)))

y\_val.append(to\_categorical(1, num\_classes=2))

# list转换为Numpy类型

x\_val = np.array(x\_val)

y\_val = np.array(y\_val)

print('[INFO] 图像增强...')

# 对训练集做数据增强-指定数据增强的3种方法

seq = iaa.OneOf([

iaa.Fliplr(), # 水平翻转

iaa.Affine(rotate=20), # 旋转

iaa.Multiply((1.2, 1.5))]) # 随机亮度

# 对训练集做数据增强-训练数据生成器

# 有肺炎的样本是无肺炎样本数量的3倍，这里增强无肺炎样本

def data\_generator(x\_train, y\_train, batch\_size):

# 获取数据的样本总数

x\_train\_len = len(x\_train)

# 得到总共多少批

steps = x\_train\_len//batch\_size

# 定义两个numpy数组以包含批处理数据和标签，作为最终返回值

batch\_data = np.zeros((batch\_size, 224, 224, 3), dtype=np.float32)

batch\_labels = np.zeros((batch\_size,2), dtype=np.float32)

# 获取输入数据所有索引的numpy数组

indices = np.arange(x\_train\_len)

i =0 # 初始化批的计数器

while True:

# 随机打乱数据顺序

np.random.shuffle(indices)

# 获取一批数据

count = 0

next\_batch = indices[(i\*batch\_size):(i+1)\*batch\_size] # 一批数据的索引

# 遍历一批数据，同时获得索引和值

for j, idx in enumerate(next\_batch):

img = x\_train[idx] # 预处理后的数据

encoded\_label = y\_train[idx] # 独热编码后的标签

# 将数据及标签保存在两个数组中

batch\_data[count] = img

batch\_labels[count] = encoded\_label

#正常样本（无肺炎）量较少，采用数据增强

if encoded\_label[0]==1 and count < batch\_size-2: # encoded\_label[0]==1 相当于label==0

aug\_img1 = seq.augment\_image(img) # 图像增强

aug\_img2 = seq.augment\_image(img)

# 将增强后的样本保存至数组

batch\_data[count+1] = aug\_img1

batch\_labels[count+1] = encoded\_label

batch\_data[count+2] = aug\_img2

batch\_labels[count+2] = encoded\_label

count +=2

# 肺炎样本不需要数据增强

else:

count+=1

#处理完当前批，跳出该循环

if count==batch\_size-1:

break

i+=1

yield batch\_data, batch\_labels

if i>=steps:

i=0

# 加载网络

model = CustomNN.build\_model()

# 使用keras构建深度学习模型，我们会通过model.summary()输出模型各层的参数状况,通过这些参数，可以看到模型各个层的组成（dense表示全连接层）。也能看到数据经过每个层后，输出的数据维度。

# model.summary()

# 模型迁移（参数迁移） 将在Imagement数据集上预训练的模型的前4层卷积层的参数迁移到我们自己网络的对应层，作为初始化权重。

f = h5py.File(vgg16\_path, 'r')

#将VGG16的13层卷积层分为5个block

#block1

#得到预训练模型的第一层的权重和偏置，block1\_conv1

w,b = f['block1\_conv1']['block1\_conv1\_W\_1:0'], f['block1\_conv1']['block1\_conv1\_b\_1:0']

#将权重和偏置分别赋给新网络的第一层，卷积层

model.layers[1].set\_weights = [w,b]

#得到预训练模型的第二层的权重和偏置，block1\_conv2

w,b = f['block1\_conv2']['block1\_conv2\_W\_1:0'], f['block1\_conv2']['block1\_conv2\_b\_1:0']

#将权重和偏置分别赋给新网络的第二层，卷积层

model.layers[2].set\_weights = [w,b]

#block2

#得到预训练模型的第三层的权重和偏置，block2\_conv1

w,b = f['block2\_conv1']['block2\_conv1\_W\_1:0'], f['block2\_conv1']['block2\_conv1\_b\_1:0']

#将权重和偏置分别赋给新网络的第三层，深度可分离卷积

model.layers[4].set\_weights = [w,b]

#得到预训练模型的第四层的权重和偏置，block2\_conv2

w,b = f['block2\_conv2']['block2\_conv2\_W\_1:0'], f['block2\_conv2']['block2\_conv2\_b\_1:0']

#将权重和偏置分别赋给新网络的第四层，深度可分离卷积

model.layers[5].set\_weights = [w,b]

f.close()

# model.summary()

#编译模型

opt = Adam(lr=LR, decay=DECAY)

es = EarlyStopping(monitor='val\_loss', patience = ES\_PATIENCE)

chkpt = ModelCheckpoint(filepath=saved\_model\_name, save\_best\_only=True, save\_weights\_only=True)

model.compile(loss='binary\_crossentropy', metrics=['accuracy'],optimizer=opt)

# 训练模型

print('[INFO] 训练模型...')

H = model.fit(data\_generator(x\_train, y\_train, BATCH\_SIZE),

validation\_data=(x\_val, y\_val),

steps\_per\_epoch=len(x\_train)//BATCH\_SIZE,

epochs=EPOCHS,

callbacks=[es, chkpt],

verbose = 1)

# 画出训练过程中的准确率

N = len(H.history['loss'])

plt.figure()

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

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

plt.title('Training & Validation accuracy on X-ray images')

plt.xlabel("Epoch #")

plt.ylabel("Accuracy")

plt.legend()

plt.savefig(output\_plot\_accuracy\_path)

plt.close()

# 画出训练过程中的损失值

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.title('Training & Validation loss on X-ray images')

plt.xlabel("Epoch #")

plt.ylabel("Loss")

plt.legend()

plt.savefig(output\_plot\_loss\_path)

plt.close()

【案例代码】test\_model.py

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

#在测试集上评估模型性能

# Load the model weights

#导入训练好的模型

#导入库

import numpy as np

from pathlib import Path

from keras.utils import to\_categorical

from sklearn.metrics import classification\_report

from build\_model import CustomNN

import cv2

#from train\_model import image\_preprocess

#设置全局变量

data\_dir = Path('dataset/chest-xray-pneumonia')#数据集路径

test\_normal\_cases\_dir = data\_dir / 'test/NORMAL'#测试集中NORMAL类样本的子目录

test\_pneumonia\_cases\_dir = data\_dir / 'test/PNEUMONIA'#测试集中PNEUMONIA类样本的子目录

model\_path = 'output/model/best\_model.h5'#最佳模型路径

# 图像预处理

def image\_preprocess(img\_path):

# 1）读取图片

img = cv2.imread(img\_path)

# 2）将图像尺寸resize到224x224

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

# 3）如果图像通道数为1，将其转换为3个通道的图像

if img.shape[2] ==1:

img = np.dstack([img, img, img])

# 4）图像颜色空间转换

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

# 5）图像标准化

img = img.astype(np.float32)/255.

return img

print('[INFO] 处理测试集数据...')

# 生成x\_test和y\_test

x\_test = []

y\_test = []

# 获得无肺炎和有肺炎的所有图像

test\_normal\_cases = test\_normal\_cases\_dir.glob('\*.jpeg')

test\_pneumonia\_cases = test\_pneumonia\_cases\_dir.glob('\*.jpeg')

# 把正常图片加载到x\_test和y\_test中

for img in test\_normal\_cases:

x\_test.append(image\_preprocess(str(img)))

y\_test.append(to\_categorical(0, num\_classes=2))

# 把肺炎图片加载到x\_test和y\_test中

for img in test\_pneumonia\_cases:

x\_test.append(image\_preprocess(str(img)))

y\_test.append(to\_categorical(1, num\_classes=2))

# list转换为Numpy类型

x\_test = np.array(x\_test)

y\_test = np.array(y\_test)

#加载模型

model = CustomNN.build\_model()

model.load\_weights(model\_path)

#评估模型分类性能

print("[INFO] 评估模型分类性能...")

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

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

predictions.argmax(axis=1)))