东软教育科技集团

项目实训总结报告

项 目 名 称: “识食物者”食物识别系统

学 校 名 称: 内蒙古大学

项 目 讲 师: 王志勇

姓 名: 李纪元

学 号: 0191121383

实 训 日 期: 2022年7月14日-8月6日

**1 引言** （一级标题，小三黑体，居中）

1.1项目概述（二级标题，四号黑体，左对齐）

说明项目目的、背景、意义及参考资料（包括企业提供的素材、DEMO）等；

本项目为食品识别程序，意图通过神经网络技术，实现对于牛排、汉堡、冰激凌、甜甜圈、蛋挞、披萨六种视频的自动识别与分类。

该项目可以快速有效地令学生建立起对于网页爬取技术、神经网络技术、网页开发前后端的认识，并通过实践掌握相关技术的基本开发方法，为以后的工作奠定基础。

本项目取用《Python深度学习》、《Flask Web开发：基于Python的Web应用开发实战》等书目作为参考资料。

1.2 需求分析 *(课程目标支撑4 ; 毕业要求支撑10.2)*

为什么做这个题目，有什么实际价值

对于事物的识别与分类是机器学习与深度学习中一个庞大且重要的课题，如何更高效、更准确地识别、分类事物一直是相关研究者所关注的重心。其中，对于图片的分类一直是一个难点，这是因为，图片所包含的信息比文字丰富很多，并且，现实世界的图片在规格、质量方面往往差距较大，种种这些原因，令算法很难捕捉到图片中的有效信息。

我们的课题，意图通过对食物数据集进行分类，来观察并总结卷积神经网络在图片分类方面的表现。比如，当事物的颜色相近时（汉堡、蛋挞、披萨），CNN的分类效果如何？当事物的形状相近时（汉堡、蛋挞、披萨、甜甜圈），CNN又能否正确分类？ 我们不仅关注CNN面对明显不同的事物时的表现，也关注其在相近相似事物上是如何提取特征并正确分类的。

从实用角度来说，该模型提供了一个能够对于六种食物进行识别且正确率相对不低的识别模型，该模型可迁移到任何需要机器自动进行食物识别的场景来节省相应的人力资源，比如说幼教领域（由食物的自动识别程序辅助幼儿独立学习）、信息提取（为用户浏览网页时的图片添加标签，优化浏览体验）、大量数据的自动分类与标记等等。

从其他角度来说，该模型的训练成果十分易于扩展，我们可以很方便地在该模型的基础上扩充食物种类、修改模型结构等等，甚至未来可以尝试让其不止可以识别食物，转而实现对更多类食物的分类功能。这对于未来的学习、工作、研究、开发，都有着十分重要的正面作用。

那么，让我们从功能需求与非功能需求，明确地分析总结我们的模型所要实现的功能：

功能需求：

1. 允许用户通过Keras、Tensorflow、Python环境加载已经训练好的模型，并输入图像，然后返回预测结果；
2. 实现对于汉堡、冰激凌、蛋挞、甜甜圈、披萨、牛排六种食物的正确识别；
3. 允许用户继续使用新数据对模型进行训练，并更新模型。

非功能需求：

1. 图片识别的准确率不能过低，不能低于90%；
2. 图片识别的耗时不能过长，不应长于一分半；
3. 模型的加载与使用不能过于复杂，用户理应可以使用一个良好定义的接口直接使用模型。

1.3 运行环境

软、硬件环境

（正文，小四号宋体，1.25倍行距）

软件环境：

|  |  |
| --- | --- |
| 软件/包名 | 版本 |
| Python | 3.8 |
| Keras | 2.9.0 |
| Tensorflow-gpu | 2.9.1 |
| Jupyter Notebook | 6.4.12 |
| Matplotlib | 3.5.2 |
| Numpy | 1.23.1 |
| Opencv-python | 4.6.0.66 |
| Pandas | 1.4.3 |
| Pip | 22.2 |
| Requests | 2.28.1 |
| BeautifulSoup4 | 4.11.1 |
| Sklearn | 0.0 |

硬件环境：

Window11家庭版，Nvidia RTX3060.

**2 项目设计**（小三黑体，居中）*(课程目标支撑3 ; 毕业要求支撑9.2)*

（以个人主要负责内容为主）

2.1 设计思路

模型的构建与训练分为这样几块内容：

首先是数据的读入与处理。我们的数据将由负责爬取数据的同学进行清洗与切分，不过，在实际训练模型时，我们或许仍然希望对数据进行一些灵活的处理。比如说，我想要在训练程序内部自由地修改图片的大小，并且希望能够轻松对图片进行标准化与归一化，那么，我将会将图片的处理封装为一个函数，并且保证在训练模型前能够容易地调用这些函数来处理数据。我们可以使用Opencv或者Python的Pillow库来处理图片数据，考虑到Opencv更为强大，并且Python本身也有封装好的Opencv包，因此，我选择使用Opencv作为图片处理的工具。

此外，我还希望对于图片进行更丰富的变换，比如翻转、位移等等，我们一般称之为数据增强。Opencv也可以对图片进行这样的变换，但是在训练时去生成这样的数据稍显麻烦，因此，我们可以使用数据生成器，去自动生成一块块随机变换的图片集供模型使用。

我们的数据并不是作为一个整体全部用于训练，而是分为训练用的训练集、训练时评估正确率的验证集、以及训练完成后最终评估准确率的测试集。这些数据的切分工作已经由负责爬取数据的同学完成，我们只需要从不同的文件夹读取并处理即可。

然后是模型的定义与构建。首先是模型的结构；CNN模型一般包括卷积基与分类器两部分，卷积基由若干卷积层与池化层交叠而成，而分类器由若干密集连接层组成。我已经尝试过朴素的CNN网络在数据集上的效果，并不能使人满意，因此，在正式的程序中，我将对于模型的结构做出一些改变，比如，我会使用预训练模型来替换卷积基，并且向分类器中加入更多的神经元与正则化层次，来提高正确率、降低过拟合。神经网络的结构等一般称之为超参数，相应的，我们还需要调整网络的参数，比如优化器、损失函数、评估函数、训练周期、训练步长等。许多参数是需要我们在实际的训练中才能根据实际表现调整的，但另一些参数目前也可以确定，或者至少有大致的范围。比如，对于损失函数，我们的项目是多分类问题，所以我们应当选择多元交叉熵；对于优化器，我们可以选择RMS、Adam、SGD等等；对于学习率与训练轮次等参数，人工的调整不仅复杂，且训练过程不可干预，我们可以通过设置Callback list，让模型自动地调整这些参数。我们将会一一测试不同参数的效果，选择其中最好的作为结果。

最后，我们将会在测试集上测试模型的正确率，如果正确率可以接受，我们将会将模型保存为h5文件。用户可以通过搭建好的Keras环境加载模型并使用。

2.2 模块功能介绍

我们将训练程序分为这样几个模块：

**数据读入模块**：负责读入数据集，在数据集通过文件夹正确分类的情况下，该模块可以为不同文件夹顺序编号，并为文件夹内的图片打上标记。

**数据处理模块**：负责调整图片的大小等属性，方便模型在不同大小的数据上测试效果；此外，为了模型能够快速收敛，该模块还需要负责对数据进行标准化与归一化。

**数据生成器定义模块**：该模块会在需要数据增强的情景下调用。该模块能够定义数据生成器并自定义增强参数，并且能够从指定的目录下读取图片进行处理。

**模型定义模块**：该模块负责定义模型的结构，比如卷积基结构、是否需要预训练模型、选择哪个预训练模型、如何定义分类器、各层的神经元大小及激活函数、正则化选项等等。

**模型编译模块**：负责定义模型的优化器、损失函数、评估函数，并编译模型。

**模型展示模块**：负责展示模型的整体结构。

**模型训练模块**：负责指定模型训练的周期、步长、Callback List，传入训练集与验证集，并进行训练，保存训练过程的数据。

**训练图像绘制模块**：负责读取训练过程记录，绘制训练集与验证集的正确率与损失图像。

**模型测试模块**：负责使用测试集对模型的正确率进行测试，测试集应保证是全新的数据，没有直接或间接地与模型接触过。

**模型更新模块**：负责对模型进行保存到指定目录，方便日后读取使用。

2.3 模块结构图

以上各模块之间的关系如下图所示。

模型的训练过程可以视作两个阶段，一个是准备阶段，一个是训练与测试阶段。第一个阶段所做的工作是数据准备、模型准备，第二个阶段所做的工作是模型的训练与测试。图中的数据读入模块、数据处理模块、数据生成器定义模块、模型定义模块、模型编译模块、模型展示模块都属于这个阶段，实际上，它们承担了大部分的工作；而图中的模型训练模块、训练图像绘制模块、模型测试模块、模型更新模块则属于第二个阶段，第一阶段的模块大部分传入第二阶段作为参数使用。

图示, 示意图

描述已自动生成

图2.3 模块结构图

2.4 程序流程图

模型训练程序的流程如下图所示。

我们首先读入数据并进行处理，然后定义训练集、验证集、测试集的数据生成器并将数据传入其中；其中，对于训练集，我们应当定义其数据增强的参数，而对于验证集与测试集，则不应该进行数据增强。

接着，我们定义模型结构，这一步我们导入预训练模型并将其作为模型的卷积基加入，然后在其头部加入分类器；这一步我们还可以对模型进行一些微调，比如解冻部分卷积基。模型结构定义好后，我们可以调用展示模块对模型结构进行展示，以此检查模型定义是否正确。

为了使模型可以自动调整学习率与训练轮数，我们在这里设置一下Callback List。

在编译时，我们需要指定优化器、损失函数、评估函数。

接下来，我们传入数据生成器、Callback list，指定模型训练的周期上限、数据块大小与步长，然后开始训练。

训练结束后，我们可以借助保存好的历史记录绘制出训练集与测试集的准确率与损失图像，帮助我们直观地看到二者的变化。

最后，我们在测试集上评估一下模型，并对模型进行保存。

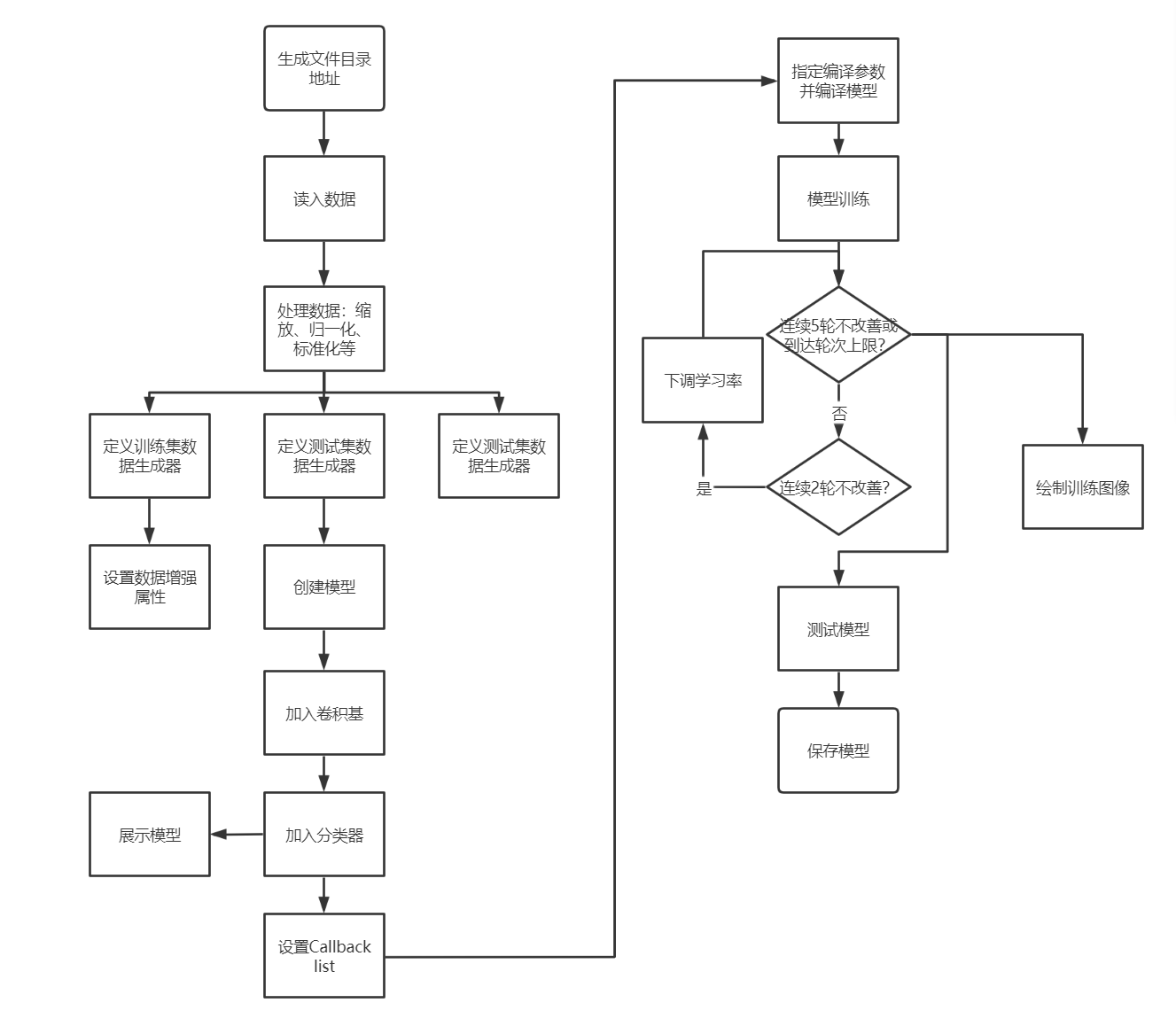


图2.4模型训练流程图

2.5 功能设计分工

需详细标注每部分的内容，具体任务及任务安排等。

|  |  |  |
| --- | --- | --- |
| **序号** | **任务名称** | **工时** |
| 1 | 设计实现数据读入模块与数据处理模块 | 0.5天 |
| 2 | 设计实现数据生成器定义模块 | 0.5天 |
| 3 | 设计实现模型定义模块、模型编译模块、模型展示模块 | 0.5天 |
| 4 | 设计实现模型训练模块、训练图像测试模块、模型更新模块 | 0.5天 |
| 5 | 调整模型参数提高准确率 | 0.5天 |
| 6 | 部分前端业务逻辑函数 | 0.5天 |

**3 详细设计**（小三黑体，居中）*(课程目标支撑3 ; 毕业要求支撑9.2)*

详细写明项目的设计（系统设计、数据库设计、页面等）等，并写明对所涉技术、核心功能等描述；

在我所负责的模型训练模块，我使用Opencv进行数据处理，并通过Keras框架搭建CNN卷积神经网络，使用Matplotlib.pyplot库绘制训练图像，接下来，我将对该模块的详细设计、相关技术、核心功能进行详细的描述。

首先，下图展示了该模块的详细架构：

图示

描述已自动生成

图3.1 模块详细架构图

对于数据集的读入，我们可以直接使用ImageDataGenerator进行读取，但是如果我们要对数据进行标准化的话，那么预先将数据集读入并转化为Numpy数组也必不可少。借助Opencv或者PIL，我们可以写一个简单的脚本来做到这一点。

Opencv允许直接在读入时指定图片的大小，我们也可以利用其所提供的接口在读入后再缩放图片。

最重要的是对数据进行归一化：

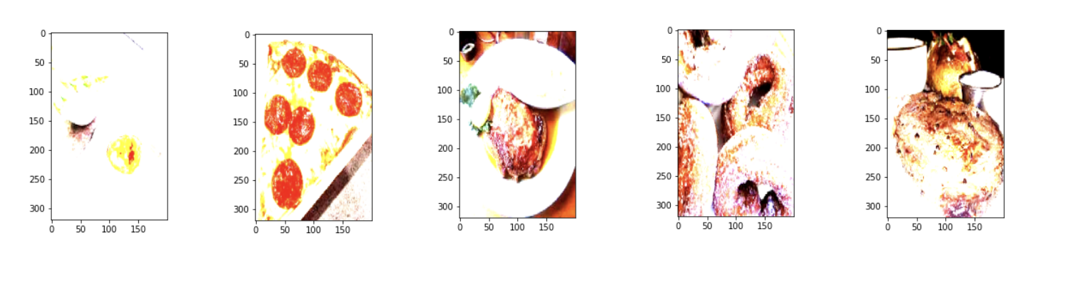
图表

描述已自动生成

图3.2 数据归一化

这里，我们使用的是最大最小归一化，由于最小值等于0，所以等价于令图像的所有元素除以255。我们可以在将图像转化为Numpy数组后手动进行，也可以直接在ImageDataGenerator中完成。

在数据增强模块中，我们需要借助ImageDataGenerator完成两项工作：数据标准化与数据增强。



图形用户界面, 图表

中度可信度描述已自动生成

图形用户界面

描述已自动生成

图3.3 数据标准化

上面三张图展示了不同的数据标准化方式所导致的不同效果。数据标准化可以令图像的特征更加鲜明，提高模型的泛化能力。

由于我们的数据集并不算大，所以我们也需要使用数据增强技术，对于图像进行平移、旋转、错切等变换，变相地扩充数据集的大小，从而让模型更不容易过拟合。

比如，下面几张图就是数据增强后的效果：

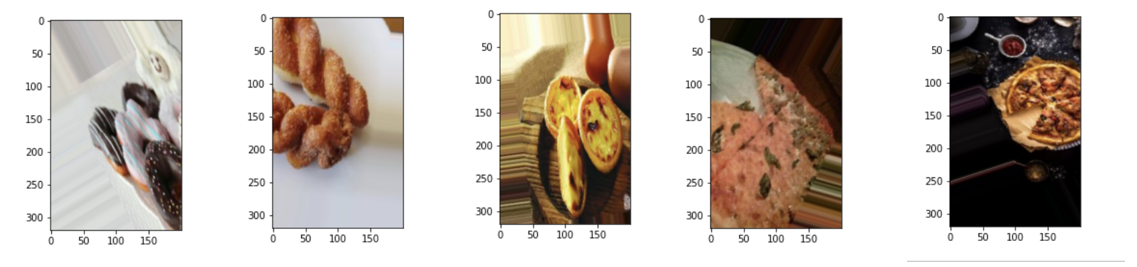


图3.4数据增强

数据增强同样可以使用ImageDataGenerator完成，我们只需要指定相关参数即可。

模型搭建部分我们分为三步进行。首先是搭建预训练模型。Keras框架内置了许多预训练模型，比如VGG16、VGG19、ResNet等，我们可以直接导入，只需要指定是否带分类器、输入大小等参数即可。

作为例子，下图展示了VGG16模型的架构：

表格

描述已自动生成

图3.5 VGG16结构图

然后是分类器，这部分我们可以根据需要自行发挥，我们既可以加入若干密集连接层，也可以用GlobalAveragePooling2D来代替，根据神经元、层数、激活函数的不同，可以得到不同的训练结果。

下图展示了两种典型的分类器结构：

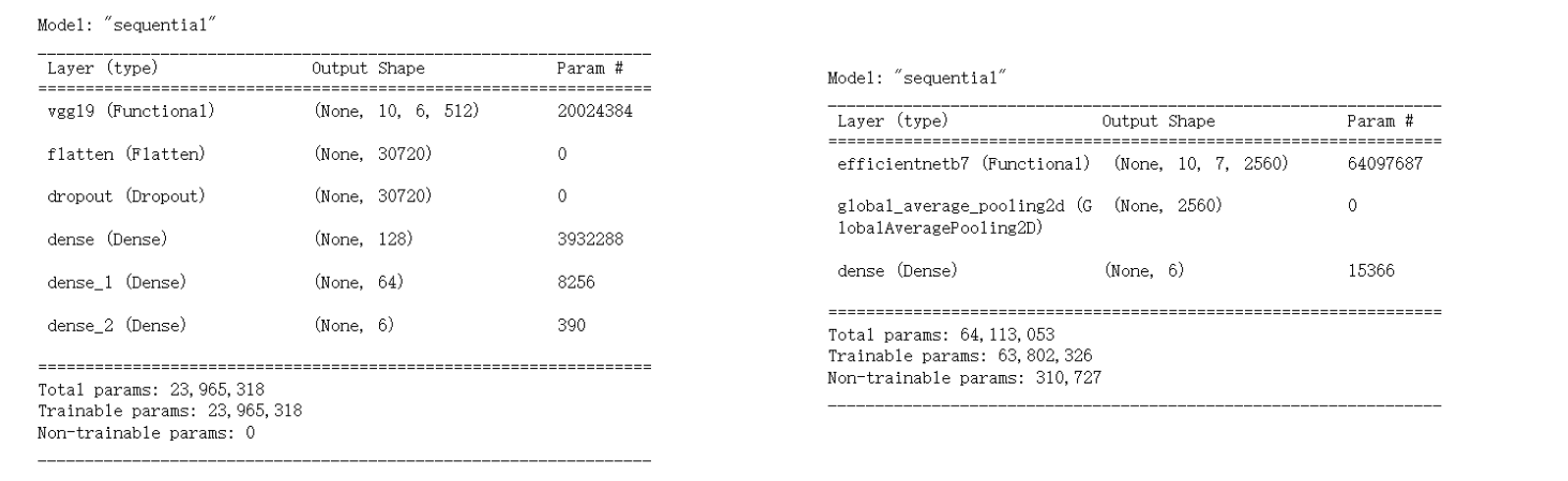


图3.6 分类器结构

最后，为了防止模型过拟合，我们还可以视情况加入一些Dropout层次或者对某些层加入L1与L2正则化。

模型训练模块同样有三部分主要工作。

首先是模型编译，在这里我们需要选择模型的优化器、损失函数等参数。由于我们的模型是多分类问题，所以损失函数我们选取多元交叉熵，而对于优化器，我们可以选择SGD、Adam、RMSprop等等，不同的优化器也会对模型的训练效果有所影响。

其次是模型训练，这里我们需要传入训练集与测试集的数据生成器，指定训练步长与训练周期，另外，我们也可以加入回调函数来自动控制学习率与训练周期。

最后是模型测试，我们同样为测试集也制作一个数据生成器，然后简单地调用相关接口即可显示测试效果。

在模型训练和测试完成后，我们可以使用matplotlib.pyplot来绘制训练过程中训练集、验证集的准确率和损失图像。这是我们观察并改善模型的一个依据。

在保存模型后，我们可以通过Opencv来载入单张图片，转化为Numpy数组后调用模型自身的接口返回预测结果。

此外，这里也展示了在实际训练时，模块运行的详细时序图：

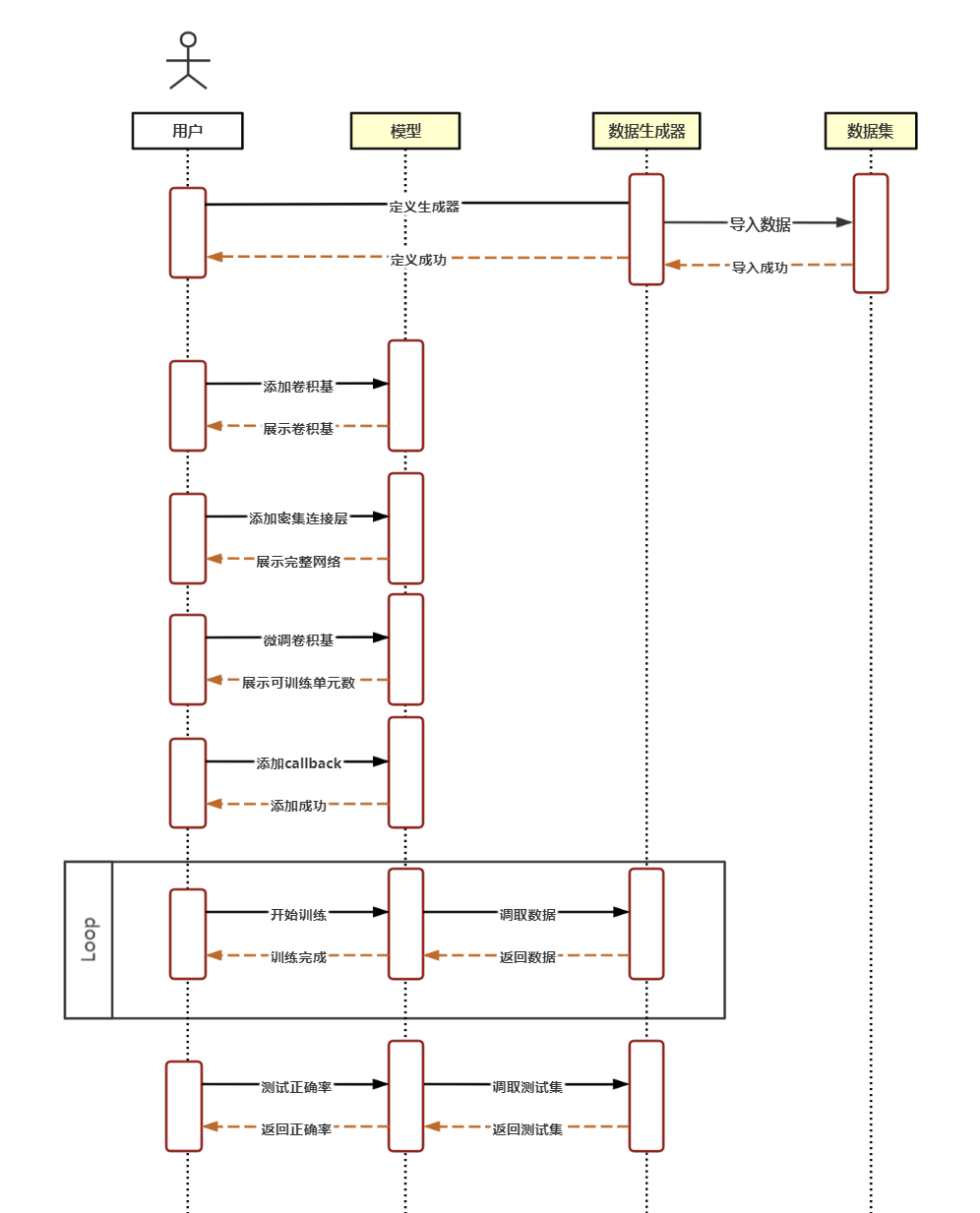


图3.7 模型训练时序图

可以看到该模块是一个几乎线性的过程。我们首先导入数据，进行数据处理；然后定义数据生成器，进行数据标准化与数据增强；接着，我们建立模型，导入卷积基和密集连接层，然后微调卷积基之后，开始进行模型编译与训练；训练完成后，我们对模型进行测试，并绘制训练图像，保存模型。

**4 实现方案**（小三黑体，居中）*(课程目标支撑3 ; 毕业要求支撑9.2)*

详细写明项目功能实现描述、核心代码及实现结果等；

并对调试好的程序，从执行程序弹出界面开始，每一步操作截一个图，并附加说明（图片格式为JPG格式）。

接下来，我将对详细设计中所涉及的功能，在代码层面上进行描述，并展示核心代码的运行结果。

首先是数据集的读取。在这里，我们用PIL库来进行图片数据的读取，并转换为Numpy数组。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图4.1 数据集读取代码

在getDataset中，由于训练集、验证集、测试集的数据均有六类且放置于不同的文件夹中，所以我们使用循环分别读取六类数据的子文件夹；getPic的功能是单独读取其中一个子文件夹的内容：遍历文件夹内所有图片，放到列表中，最后将列表转换为Numpy数组后返回。

为了演示这部分代码，我们首先生成一下数据集文件的路径：



图4.2 数据集文件路径生成代码及结果

我们调用上面的函数，可以看到从训练集的路径中读出了六类图片：

图片包含 图形用户界面

描述已自动生成

图4.3 数据读取代码及结果

对于验证集与测试集，我们也调用该接口读取即可。

接下来是数据生成器的定义：

图形用户界面, 文本, 应用程序

描述已自动生成

图4.4 数据生成器定义代码

这里我们展示了训练集数据生成器的定义。我们利用rescale参数进行了数据归一化，然后进行了一系列的数据增强，并且进行了数据标准化。

由于数据标准化需要用到数据的均值和标准差，所以我们需要调用fit接口计算这些数值，这也是我们需要事先读入数据的原因，如果不进行数据标准化，则不需要这一步。

然后，我们指定batchsize，并从文件夹将图片导入到数据生成器中：

图形用户界面, 文本, 应用程序

描述已自动生成

图4.5 导入数据到生成器代码及结果

在这里，我们选取生成器中第一张图片进行展示：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图4.6 图片展示代码及结果

然后是模型搭建部分。我们首先导入预训练模型，这里以VGG16作为例子。

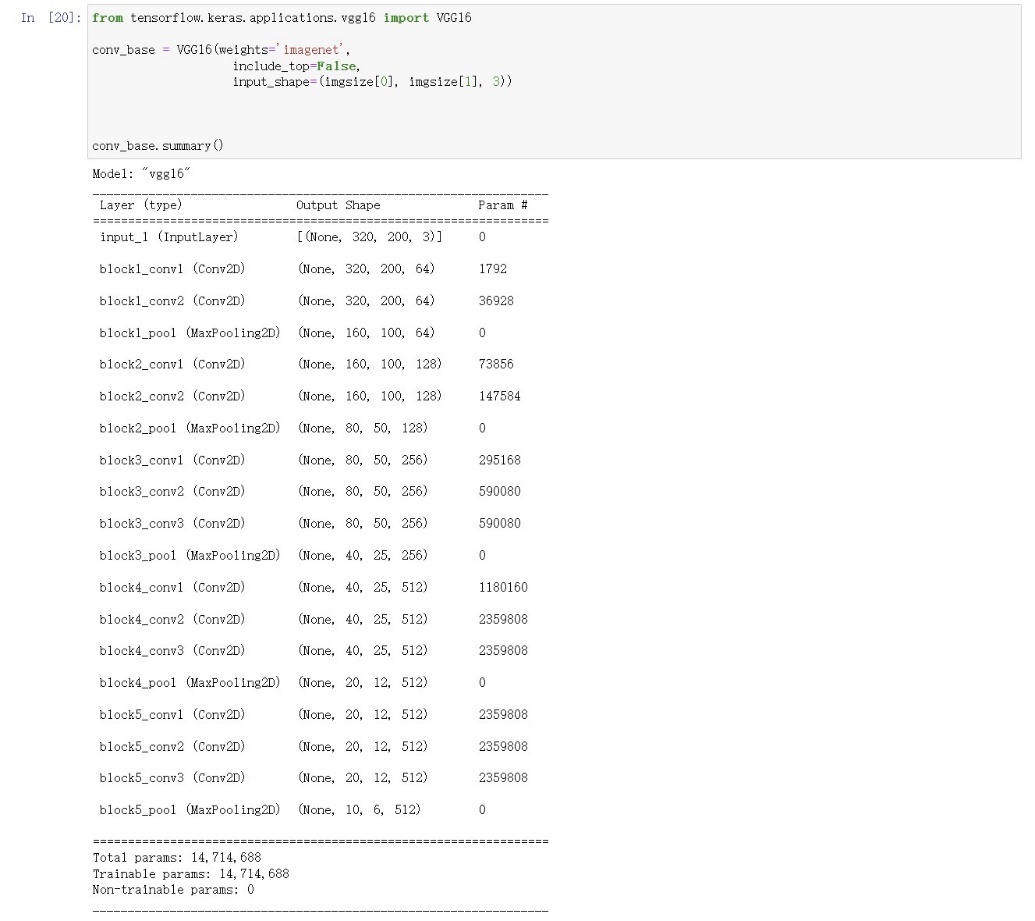


图4.7 预训练模型导入代码及结果

我们指定了预训练模型初始化的权重检查点，指定不需要VGG16自己的分类器，指定了输入图片的大小，并输出了预训练模型的结构。

然后，我们将其作为卷积基加入模型，并且在模型顶部加入分类器。下图中，我们以加入若干密集连接层、使用Dropout正则化为例，展示了卷积基、分类器的整体结构：

表格

描述已自动生成

图4.8 模型搭建代码及结果

然后，为了让卷积基更好地提取图像特征，我们需要对部分卷积基进行冻结，而对另一部分进行解冻，如下图所示：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图4.9 模型微调代码及结果

为了更轻松地控制训练过程，我们可以事先设置一下回调函数：



图4.10 回调函数设置代码

我们设置了两个回调函数，RefuceLROnPlateau的意思是，当val\_loss连续1轮不改善时，将其调整为原来的0.2倍；EarlyStopping的意思是，当val\_loss连续两轮不改善时，中断训练。

在模型编译部分，我们需要指定模型的优化器、初始的学习率、损失函数和评估函数；而在模型训练部分，我们需要传入训练集和验证集的数据生成器，指定步长和训练周期，如果有回调函数的话，也要传入其中。下图展示了模型编译及训练部分的代码和训练过程：

表格

描述已自动生成

图4.11 模型编译、模型训练代码及结果

然后，我们使用测试集的数据生成器对模型的正确率进行测试：

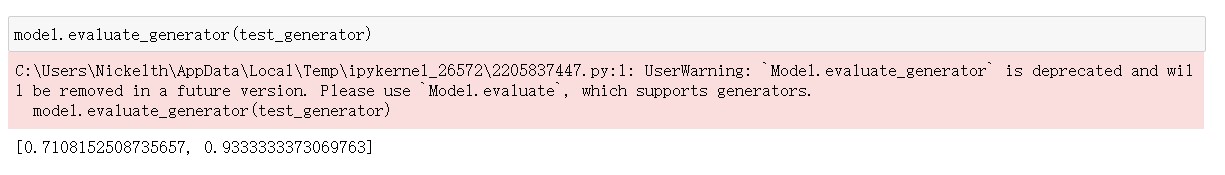


图4.12 模型测试代码及结果

我们将模型训练过程的数据保存到了history中，因此，我们可以从中提取数据，并利用matplotlib.pyplot库绘制训练准确率和损失的图像：

图形用户界面, 应用程序

描述已自动生成

图4.12 图像绘制代码及结果

在这里，我们绘制了训练集准确率（图1圆点）、训练集损失（图2圆点）、验证集准确率（图1折线）、验证集损失（图2折线）。

我们可以对模型进行保存，以便之后加载使用：

背景图案

描述已自动生成

图4.13 模型保存代码

最后，我们使用Opencv加载单张图片，转化为Numpy数组传入模型中进行预测，并返回预测结果：



图4.14 图片预测代码代码

**5 系统测试**（小三黑体，居中）*(课程目标支撑3 ; 毕业要求支撑9.2)*

详细写明测试方法、测试描述（bug与修正方法等）、预期结果等；

在该模块的开发过程中，可能会出现两种问题，一种是程序编写中的bug，另一种是模型训练时准确率的问题。

对于前者，我们一般可以通过对bug描述进行搜索，来找到相应的解决方案，以下是几个例子：

1. ValueError: Error when checking target: expected dense\_2 to have 2 dimensions, but got array with shape (5000, 1, 10)

这是数据集维度上的错误，密集连接层只接受两维的数据，我们可以在卷积基的末尾，加入Flatten层来展平维度。

1. ValueError: shapes (24,1) and (3,) not aligned: 1 (dim 1) != 3 (dim 0)

层次间的输入与输出必须吻合，该bug意味着某两层间的数据维度不相对应，这时我们需要检查数据集的shape，如果不吻合，就调用reshape改一下。

1. ValueError: all the input array dimensions except for the concatenation axis must match exactly

这是在将图像转化为Numpy数组并拼接时可能会产生的bug，在拼接的维度上，两个数组的维度必须对应，我们可以通过检查两个数组的shape来检查维度。

1. PermissionError : [Error 13] Permission denied : 'D:/0.jpg'

这是读取图片时的问题。在Windows中，这大概率不会是权限问题，而是找不到文件所导致的，仔细检查文件路径即可。

1. TensorFlow Error: InternalError: Failed copying input tensor

模型训练时，如果图像较大、数据较多可能会遇到这种问题，需要把batchsize调小。

1. RuntimeError: cuDNN error: CUDNN\_STATUS\_EXECUTION\_FAILED

这意味着cnDNN的版本与其他软件不兼容，在安装时，必须检查Keras,tensorflow,cuda,cudnn四者的软件兼容表。

1. Error：OOM when allocating tensor with shape[......]

训练时内存溢出的问题，这可能是图片过大、数据过大、batchsize过大、解冻了太多预训练模型层次，这些都可能导致占据的内存过大，视情况调小即可。

1. Could not load dynamic library ‘cudart64\_110.dll‘； dlerror: cudart64\_110.dll

同样是cuda和cudnn的版本问题，如果重装以后，确定有该文件且在正确位置，但依然报错，那么重启电脑即可。

还有一些问题，虽然它们不会像bug一样报错，但也令我们十分困扰，以下是几个例子：

1. 过拟合

如下图所示，训练集准确率不断升高，验证集准确率却停滞不前，我们称这种现象为过拟合。

针对过拟合，我们有这样几种解决方法：

* 减小网络容量
* 增大数据集
* 使用数据增强
* 使用正则化
* 减少训练周期

图形用户界面

描述已自动生成

图5.1 过拟合图像

1. 训练准确率不增长

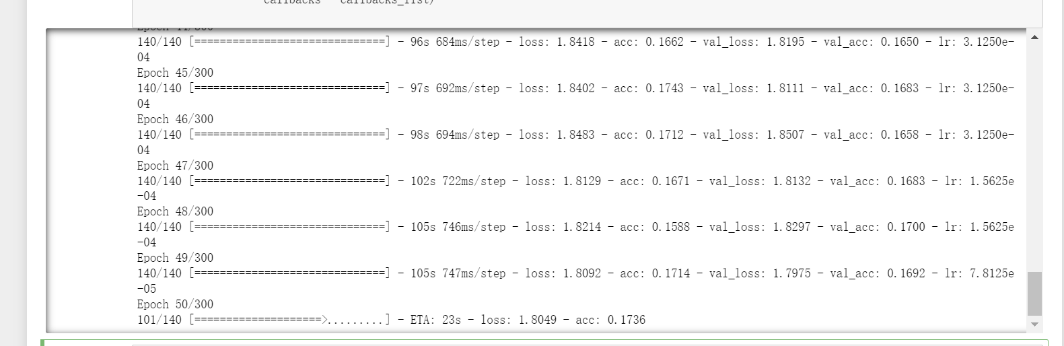


图5.2 准确率异常图像

上图展示了一个相关例子：训练周期已经达到了50轮，准确率仍然只有16%。

导致这个问题的原因可能是多样的，我的解决方案是对图像进行标准化，事实证明，进行标准化以后，大概率就不会出现这种情况了。

1. 准确率抖动

表格

低可信度描述已自动生成

图5.3 准确率抖动图像

上图展示了一个准确率抖动的例子，我们可以看到验证集的准确率在90%上下波动。

这可能是由于过拟合导致的，但也可能是其他原因，比如优化器的选择，再比如过大的学习率。

我的解决方案是，使用SGD优化器（经验上效果最好），使用callback自动调整学习率（经验上检测val\_loss并设置较小的调整系数可以更快地停止训练）。

**6 前沿分析**（小三黑体，居中）*(课程目标支撑6 ; 毕业要求支撑12.2)*

详细写实训过程的心得

本次实训开始前，我对人工智能一无所知，在当时的我看来，让一个程序从0开始自动地学习数据中的特征然后做出预测，这是一件很难想象的事情，也正是这种好奇促使我选择了这门课，我将这当作一个了解人工智能的宝贵机会。

事实证明，这门课没有辜负我的期望，在一个月的实训后，我不仅了解了人工智能的基本原理，甚至能够搭建出属于自己的神经网络，并且，它的预测准确率还不低，这一切都令我十分鼓舞。

我印象最深刻的是，神经网络是如此简洁、又如此复杂的统一体。就拿卷积神经网络来说，对一张图求卷积、池化的过程，看起来简直不能再简单了，但是谁能想到这样简单的一个过程，可以把图像的特征如此高效地提取出来？这种简洁性贯穿神经网络的始终，比如，relu作为一个再简单不过的激活函数，却具有最好的性能，而其他更为复杂的激活函数，却反而略逊一筹——神经网络似乎总是偏爱简单的事物，乍一看，这与我从前所了解的算法、数据结构简直大相径庭，但是仔细想想，这又是奥卡姆剃刀原则的最佳诠释，真是令人惊奇。但这并不代表神经网络完全没有复杂性，即使最简单的VGG16预训练模型，都有着16个层次的堆叠，而其他更为复杂的模型，有几十上百层更是常事，这样庞大的模型结构就是神经网络的复杂所在。但是，这样一个原理简单、架构复杂的网络，又可以被现代良好封装的框架和接口以十分友好的形式展现给用户。即使是不懂神经网络内在原理的开发者，也可以像堆积木一样轻松搭建出一个简单的神经网络，所有的复杂性都对用户隐去了，转移到了框架的内部。我在使用Keras框架时，每每发出感慨：在书本上如此复杂的处理过程，放到程序中居然就是几行代码、甚至一行参数的事！在这种情况下，开发者根本不必关心数据处理、模型层次的内部实现，只需要根据数据与模型表现，优化模型的结构与参数即可，所有不必要的工作都由框架代劳了，这节省了开发者多少宝贵的精力！

但是，这也不意味着模型的训练就是一个“无脑”的简单过程，恰恰相反，在一版又一版的模型训练中，各种各样的问题层出不穷，无论是过拟合、欠拟合、梯度消失、梯度爆炸、准确率抖动等等等等，这些都提醒着开发者，只会盲目堆叠参数是无法训练出好的模型的，开发者必须细心观察数据的特征，耐心地清洗、处理数据，才能一点一点地提高模型的准确率。我想起，在我训练模型举步维艰的时候，不断尝试从各种各样的地方寻找优化点，一度认为手头的模型已经达到的极限、无法再前进一步时，一些看起来不起眼的优化，却总是带来令人惊喜的结果，颇有“柳暗花明又一村”的感觉，令我不禁提振信心，继续寻找优化的突破口。

在这次实训中，我也不是一个人在战斗。我总是会将模型的最新进展分享给队友，模型每前进一个百分点，都令我们倍感开心，他们的陪伴与鼓励，让我拥有了极大的动力。此外，我很久没有过这样良好的组队体验了，在过去的组队作业中，我往往孤立无援，需要一个人完成大部分工作，但是这一次，任务得到了十分平衡的分配，更可贵的是，所有人都十分努力地完成着自己份内的工作，大家一起开八九小时甚至十几小时的组会都是常有的事，如果没有队友的相互扶持，我可能真的不会坚持下去。

这次实训不仅让我了解了人工智能技术，让我收获了良好的组队体验，还让我体会到了企业开发是怎样一个氛围、怎样去寻找、摸索一个未知问题的解决方案。我希望在未来，我也能够成为一个出色的软件工程师，不仅能够对类似神经网络这样原理复杂、架构庞大的事物信手拈来，还能写出像Keras一样将庞大的复杂性隐藏起来的、让用户轻松上手的框架，经过实际的体会，我认为这样的工作富有极大的价值。

一个月还是太过短暂了，实际上，我们也只是触摸到了卷积神经网络的冰山一角。因此，在最后，我想就目前神经网络的前沿进展做一些介绍。目前，业界已经有相当多的新型网络结构，我列举了其中一部分，其中有些我们使用过，有些则因为时间问题没有尝试使用:

* ResNet：残差神经网络，由何恺明提出并实现，解决了模型深度过深时梯度弥散或者梯度爆炸问题，使几百甚至上千层的模型训练成为可能。
* InceptionV1-V4：由Google提出，在池化层后添加多路非线性映射（传统网络只添加单路非线性映射），增加了模型的非线性表达能力。
* DenseNet：同样意图解决梯度弥散与梯度爆炸问题，解决方式是在每一层后单独添加一个shortcut层。
* Squeeze-and-Excitation block：一种可以嵌到其他网络中的子结构，通过网络根据loss学习特征图的权重，令有效的特征图权重大、无效或效果小的特征图权重小。
* MobileNet/ShuffleNet：意在解决模型的效率问题，通过轻量化的网络设计，令网络在不损失精度的同时获得更高的效率。

最后，感谢老师、队友的支持与陪伴！