

垃圾分类识别系统

基于卷积神经网络的图像识别

项目论文

周铠然 · 仅供求职展示

周铠然

目 录

1 项目设计目的 2

2 项目设计题目及要求 2

 2.1 题目：设计一个基于深度学习的图像分类模型 2

 2.2 设计目标 2

 2.3 设计要求 2

3 项目设计背景分析 3

 3.1 项目背景 3

 3.2 技术背景 4

 3.3 模型背景分析 4

4 数据与准备工作 7

 4.1 数据集准备 7

 4.2 介绍数据集 7

 4.3 图像数据处理 8

 4.4 数据集处理的缺点 8

 4.5 数据集处理的改进 8

 4.6 数据准备和预处理 9

5 模型方法 9

 5.1 ResNet50 9

 5.2 MobileNetV2 10

6 评估与测试 11

 6.1 模型评估与测试 11

 6.1.1 ResNet50 11

 6.1.2 MobileNetV2 12

 6.2 交互窗口 14

7 项目设计总结 17

8 参考文献 18

1 项目设计目的

理论知识与实践能力相结合,通过本**项目设计**,我们能够将理论知识应用到实际项目中,特别是迁移学习和深度学习在图像分类中的应用,在解决实际问题方面,垃圾分类是一个现实生活中的重要问题,我们了解如何利用现代技术解决实际问题。

垃圾分类的目的是提高垃圾的资源价值和经济价值,减少垃圾处理量和处理设备的使用,降低处理成本,减少土地资源的消耗,具有社会、经济、生态等几方面的效益。

本**项目设计**的主要目的是通过迁移学习方法,研究并实现一个高效的垃圾分类系统。具体包括以下几个方面:

- (1) 数据集准备方面。收集并处理垃圾分类数据集,包括数据增强和数据集划分。
- (2) 模型训练 选择合适的预训练模型(如 ResNet50),通过迁移学习进行训练和优化。
- (3) 模型评估 对训练好的模型进行评估,包括准确率和损失函数的分析。
- (4) 系统实现 通过简单的交互操作和预加载模型文件实现垃圾识别系统。

2 项目设计题目及要求

2.1 题目:设计一个基于深度学习的图像分类模型

2.2 设计目标

- 高效的垃圾图像分类模型,设计训练一深度学习模型,能够从多类别垃圾图像中学习到不同类别的特征,并实现准确分类。
- 识别未见过的测试图像,确保模型能够识别那些在训练过程中未曾见过的测试图像,准确预测它们所属的类别。
- 迁移学习的应用,采用迁移学习方法,利用预训练模型(如 ResNet50、MobileNetV2)来加速模型训练,提高分类性能,避免从零开始训练。
- 模型评估与优化,通过实验对模型进行评估,使用准确率、混淆矩阵等指标进行性能评估,并对模型进行优化,提升分类精度和泛化能力。
- 实现垃圾分类系统,开发一个能进行交互的自动识别和分类垃圾的窗口,确保其在实际应用价值。

2.3 设计要求

(1) 收集垃圾分类数据集,包括多种类型的垃圾图片,如可回收物、厨余垃圾、有害垃圾和其他垃圾。对数据进行清洗和标注,确保每张图片有正确的标签。使用旋转、缩放、裁剪等方法扩展数据集。

(2) 选择预训练的卷积神经网络(如 ResNet50、MobileNetV2)作为基础模型。冻结基础模型的前几层,只训练后面的全连接层或者添加新的分类层。调整模型的超参数,如学习率、批次大小等,以获得最佳性能。

(3) 对模型训练与评估,将数据集分为训练集、验证集和测试集,进行模型训练。使用交叉熵损失函数和 Adam 优化器进行优化。评估模型在验证集和测试集上的表现,使用准确率、混淆矩阵等指标进行评估。

(4) 模型优化部分,通过调整超参数、增加数据增强策略等方法优化模型。考虑使用迁移学习的其他方法,如微调预训练模型的某些层,或者使用不同的预训练模型进行对比实验。

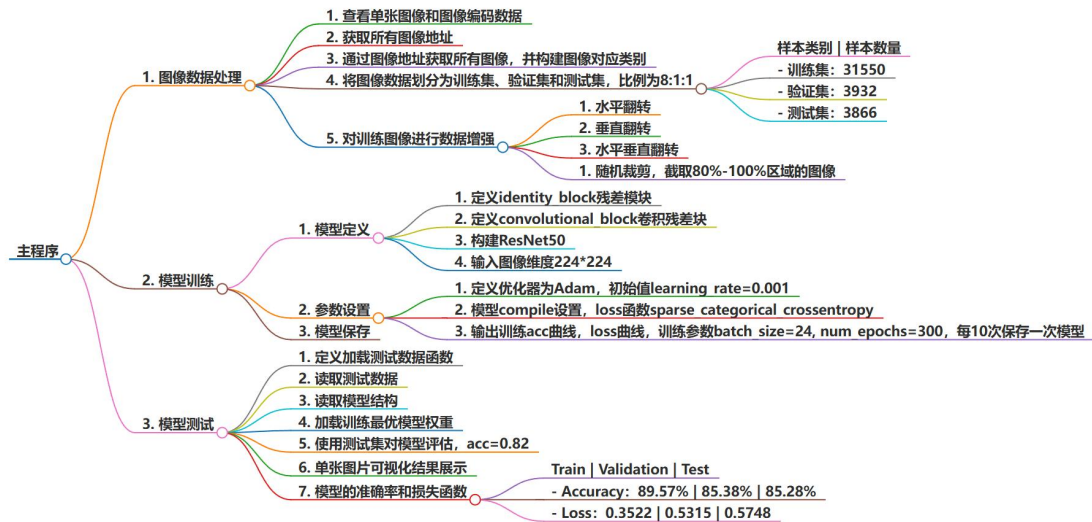


图 1 项目架构

3 项目设计背景分析

3.1 项目背景

随着城市化进程的加快和人口的增加，垃圾处理成为了一个重要的环境问题。有效的垃圾分类不仅可以减少环境污染，还能提高资源的回收利用率。然而，传统的垃圾分类方法主要依赖人工，存在效率低、成本高、易出错等问题。因此，利用人工智能技术进行自动化垃圾分类成为了一种趋势^[1]。通过深度学习和迁移学习技术，可以显著提高垃圾分类的准确率和效率。

随着经济的快速发展，人们生活水平不断提高，在消费与生产的大循环里，垃圾数量随之激增。《国际联合早报》研究报告显示，到 2050 年，全球垃圾量将增加 70%，将从 2016 年的 20.1 亿 t 增至 2050 年的 34 亿 t，垃圾分类任务十分艰巨，研究一种有效的垃圾分类方法已经十分紧迫。国内外学者针对垃圾分类作了大量分析，但提出的方案多为末端回收方法的创新。收集依赖于人们的自觉性，垃圾分类效率很低，研究一个有效的收集方法对于垃圾分类现状具有重要意义^[2]。

本项目实验基于 CNN 的轻量级神经网络，设计垃圾分类系统，该系统可以满足厨余垃圾、有害垃圾、其他垃圾和可回收物垃圾分类的要求；利用迁移学习对模型进行重新设计，利用预训练模型在本文的垃圾数据集上重新训练得到垃圾识别模型，把 MobileNetV2 和 ResNet 网络模型进行对比，验证模型的准确性，通过训练完成的模型文件实现垃圾分类系统。



图 2 垃圾分类

3.2 技术背景

深度学习是当前人工智能领域最为活跃的研究方向之一，已经在图像分类、语音识别、自然语言处理等多个领域取得了显著成果，如康庄等人提出利用迁移学习对 Inception v3 模型进行重新设计，利用预训练模型在本文的垃圾数据集上重新训练得到垃圾识别模型^[2]。迁移学习是一种重要的深度学习方法，通过将预训练模型应用到新的任务上，可以在数据量有限的情况下显著提高模型的性能。利用迁移学习技术进行垃圾分类，可以大大减少训练时间，提高分类精度。

具体背景分析有以下几点：

- 垃圾分类是减少环境污染、提高资源回收效率的重要措施。自动化垃圾分类可以显著提高处理效率和精度。
- 随着深度学习技术的发展，其在图像识别和分类方面的应用越来越广泛。项目通过实践这些技术，准备学生解决类似的实际问题。
- 通过实际操作深度学习模型，学生可以更好地理解理论知识，并提高解决问题的能力。
- 开发有效的自动垃圾分类系统，可以应用于多个领域，如城市管理和环保，具有广泛的社会影响和商业潜力。

3.3 模型背景分析

在进行垃圾分类系统的研究和设计时，选择适当的模型至关重要。本文将总结 CNN、ResNet 和 MobileNet 三个模型，并分析后两个模型是如何基于 CNN 改进的，最后对比它们的优缺点。

(1) 卷积神经网络（CNN）

卷积神经网络（Convolutional Neural Network, CNN）是一种专门用于图像处理和计算机视觉任务的深度学习模型。CNN 通过卷积层、池化层和全连接层提取图像的特征。

模型可进行局部连接和共享权重，其利用局部连接和共享权重的方式减少了参数数量，提高了计算效率。自动特征提取，无需手动设计特征提取器，CNN 可以通过训练自动学习图像的特征。但在数据量不足时，CNN 容易出现过拟合问题。对计算资源要求较高，训练时间较长。

(2) 残差网络（ResNet）

残差网络（Residual Network, ResNet）是由何凯明等人提出的一种改进于卷积神经网络^[3]。ResNet 通过引入“残差块”解决了深层网络训练中的梯度消失问题。

与 CNN 相比改进之处在于通过加入恒等映射，使得网络层可以直接学习残差（即输入和输出的差值），从而避免梯度消失问题。ResNet 可以构建更深的网络，如 ResNet-50、ResNet-101 等，从而在大规模图像分类任务中取得更好的效果。

解决了梯度消失问题，使得网络可以更深，从而提高模型的表达能力。性能优越，在 ImageNet 等多个数据集上取得了优异的性能。但该模型所需计算资源要求高，尽管残差块可缓解梯度消失问题，但网络深度增加也带来更高的计算资源需求。

(3) 移动网络（MobileNet）

移动网络（MobileNet）是由 Google 提出的一种轻量级卷积神经网络，专为移动设备和嵌入式设备设计^[4]。MobileNet 通过深度可分离卷积（Depthwise Separable Convolution）显著减少了参数和计算量。

与以前提出的模型相比其深度可分离卷积，将标准卷积分解为深度卷积和逐点卷积，大幅减少参数数量和计算量。

模型架构上采用轻量级设计，计算量和参数量大幅减少，适合在资源受限的设备上运行。可以通过调整网络宽度和输入图像分辨率来平衡性能和计算量。但在某些复杂任务中，性能可能不如 ResNet 等更复杂的模型。

(4) 对比 MobileNet V1 与 MobileNetV2^[5] 的微结构

MobileNet V1 的微结构包含两步，Depthwise (DW) 卷积和 Pointwise (PW) 卷积，两者都使用激活函数 ReLU6。Depthwise Convolution (DW)，其中 X 为输入特征图，形状为 $[H \times W \times C_{in}]$ ， W_{DW} 为 Depthwise 卷积核，形状为 $[K_h \times K_w \times C_{in}]$ ：

$$Y_{DW} = ReLU6(X * W_{DW}) \quad (1)$$

DW 卷积保持输入和输出的通道数一致（不改变通道数），只是对每个通道进行卷积操作。

Pointwise Convolution (PW)将特征图从 C_{in} 转换到 C_{out} 实现通道融合，其中 W_{PW} 为 Pointwise 卷积核，形状为 $[1 \times 1 \times C_{in} \times C_{out}]$ ：

$$Y_{PW} = ReLU6(Y_{DW} * W_{PW}) \quad (2)$$

总结构上形如下式：

$$Y = Y_{PW} \quad (3)$$

MobileNet V2 引入 Linear Bottleneck 和 Inverted Residual Block。Pointwise Convolution (PW1) (升维)，其中 W_{PW1} 为 Pointwise 卷积核，形状为 $[1 \times 1 \times C_{in} \times tC_{in}]$ ， t 为扩展因子，通常为 6（升维）：

$$X_1 = ReLU6(X * W_{PW1}) \quad (4)$$

Depthwise Convolution (DW)， W_{DW} 为 Depthwise 卷积核，形状为 $[K_h \times K_w \times tC_{in}]$ ：

$$X_2 = ReLU6(X_1 * W_{DW}) \quad (5)$$

Pointwise Convolution (PW2) (降维)， W_{PW2} ：Pointwise 卷积核，形状为 $[1 \times 1 \times t \cdot C_{in} \times C_{out}]$ ，此处没有激活函数，形成 Linear Bottleneck：

$$X_3 = X_2 * W_{PW2} \quad (6)$$

Residual Connection (Shortcut)，即如果输入输出的形状一致，则进行残差连接：

$$Y = X + X_3 \quad (7)$$

总结构上形如下式：

$$Y = \begin{cases} X + X_3 & \text{如果 stride} = 1 \text{ 且 } C_{in} = C_{out} \\ X_3 & \text{否则} \end{cases} \quad (8)$$

相同点上，都采用 Depth-wise (DW) 卷积搭配 Point-wise (PW) 卷积的方式来提特征。这两个操作合起来也被称为 Depth-wise Separable Convolution，之前在 Xception 中被广泛使用。这么做的好处是理论上可以成倍的减少卷积层的时间复杂度和空间复杂度。由下式可知，因为卷积核的尺寸 K 通常远小于输出通道数 C_{out} ，因此标准卷积的计算复杂度近似为 DW + PW 组合卷积的 K^2 倍。

$$Complexity \frac{Depth-wise Separable CONV}{Standard CONV} = \frac{1}{K^2} + \frac{1}{C_{out}} \sim \frac{1}{K^2} \quad (9)$$

不同点上, Linear Bottleneck。V2 在 DW 卷积之前新加了一个 PW 卷积。这么做的原因, 是因为 DW 卷积由于本身的计算特性决定它自己没有改变通道数的能力, 上一层给它多少通道, 它就只能输出多少通道。所以如果上一层给的通道数本身很少的话, DW 也只能很委屈的在低维空间提特征, 因此效果不够好。现在 V2 为了改善这个问题, 给每个 DW 之前都配备了一个 PW, 专门用来升维, 定义升维系数 $t=6$, 这样不管输入通道数 C_{in} 是多是少, 经过第一个 PW 升维之后, DW 都是在相对的更高维 ($t \cdot C_{in}$) 进行着辛勤工作的。

V2 去掉第二个 PW 的激活函数。论文作者称其为 Linear Bottleneck。这么做的原因, 是因为作者认为激活函数在高维空间能够有效的增加非线性, 而在低维空间时则会破坏特征, 不如线性的效果好。由于第二个 PW 的主要功能就是降维, 因此按照上面的理论, 降维之后就不宜再使用 ReLU6 了。

(5) 对比 ResNet 与 MobileNet V2 的微结构

ResNet 的微结构是标准的 Residual Block, 分为两个 3×3 标准卷积层和残差连接。第一层卷积 (Conv1), 其中 W_1 为标准卷积核, 形状为 $[K_h \times K_w \times C_{in} \times C_{out}]$;

$$X_1 = ReLU(X * W_1) \quad (10)$$

第二层卷积 (Conv2), W_2 为标准卷积核, 形状为 $[K_h \times K_w \times C_{out} \times C_{out}]$:

$$X_2 = X_1 * W_2 \quad (11)$$

残差连接 (Shortcut):

$$Y = X + X_2 \quad (12)$$

若输入和输出通道数不一致, 则通过 1×1 卷积调整输入维度:

$$Y = W_s * X + X_2 \quad (13)$$

相同点上, MobileNet V2 借鉴 ResNet, 都采用了 $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$ 的模式。MobileNet V2 借鉴 ResNet, 同样使用 Shortcut 将输出与输入相加 (未在上式画出)

不同点上, Inverted Residual Block。ResNet 使用**标准卷积**提特征, MobileNet 始终使用**DW 卷积**提特征。ResNet 先**降维** (0.25 倍)、卷积、再升维, 而 MobileNet V2 则是先升维 (6 倍)、卷积、再降维。直观的形象上来看, ResNet 的微结构是**沙漏形**, 而 MobileNet V2 则是**纺锤形**, 刚好相反。因此论文作者将 MobileNet V2 的结构称为 Inverted Residual Block。这么做也是因为使用 DW 卷积而作的适配, 希望特征提取能够在高维进行。

(6) 对比与选择 (ResNet vs. MobileNet)

a. 计算资源上, ResNet 在性能上优于 MobileNet, 但计算资源需求较高; MobileNet 则在保持较低计算量的同时, 仍能提供较好的性能。

b. 适用场景上, ResNet 适用于高性能计算环境, 如服务器和高端 GPU; MobileNet 适用于资源受限的设备, 如移动设备和嵌入式系统。

c. 在垃圾分类系统中, 由于需要在资源有限的硬件上运行, 理论上看 MobileNet 更为合适。MobileNet 在保持高效计算的同时, 能提供足够的分类准确率, 满足项目需求。

通过对比分析, 可以看出 MobileNet 的轻量级设计和高效性使其成为本项目的最佳选择。虽然 ResNet 在性能上略胜一筹, 但在实际应用中, 计算资源的限制使得 MobileNet 更加适用。

4 数据与准备工作

4.1 数据集准备

首先代码导入一些必要的模块和库，requests（用于发送 HTTP 请求），json（用于处理 JSON 格式数据），urllib.parse（用于 URL 解析和编码），os（用于文件操作），以及 time（用于时间操作）。

我们通过百度的搜索引擎，结合人工排查，对数据图片进行爬取。定义存储图片的基础目录、请求的基础 URL 和 HTTP 请求头部信息，上述设置在初始化时就完成，为后续操作提供了必要的信息。

发送 HTTP 请求获取百度图片搜索结果，并解析返回的 JSON 数据，提取其中的图片链接。每次请求返回一组（30 张）图片的链接，通过遍历解析得到所有图片的 URL。

根据搜索关键词创建一个目录，用于存储下载的图片。如果目录不存在，则新建一个。然后开始下载和保存图片，根据提取到的图片链接，逐个发送请求下载图片，并将图片以二进制方式写入到本地文件中。下载过程中对每张图片设置了短暂的休眠时间，以防止 IP 被封。

最后从文件中读取需要搜索的关键词，逐个进行爬取操作。对于每个关键词，创建一个爬虫实例并运行下载操作，最终将所有图片保存到相应的目录中。

4.2 介绍数据集

通过上述爬虫的方法，得到垃圾分类图片数据集（120 类）。根据国内现有垃圾分类标准使用网络爬虫的方式进行数据采集，采集的目标 URL 为百度图片。数据集包含可回收物、厨余垃圾、其他垃圾和有害垃圾四个大类，其他垃圾_PE 塑料袋、其他垃圾_一次性棉签、其他垃圾_唱片、其他垃圾_打火机、其他垃圾_滚筒纸等 120 个小类共计 39348 张，可用于不同的深度学习模型训练。

数据集按比例划分为训练集、验证集和测试集，比例为 8:1:1。训练集包含 31550 张图片，验证集包含 3932 张图片，测试集包含 3866 张图片。

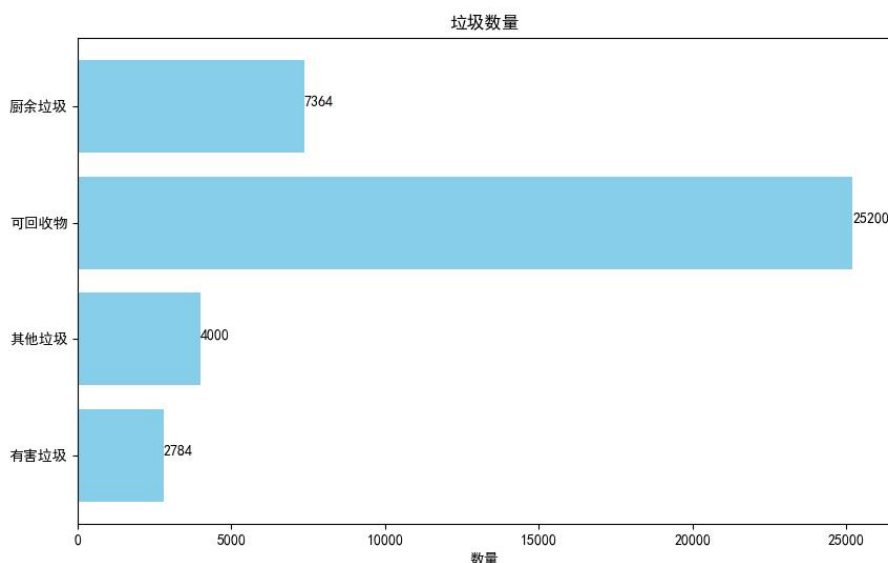


图 3 图片数量预览

在 Garbage_images_spyder 目录下包含着不同大类的垃圾图片爬取代码，并且不同的图片名称通过图片类型制作目录下的代码自动创建。

表 1 数据基及其划分

类别	数量	样本类别	样本数量
有害垃圾	2784	训练集	31550
其他垃圾	4000	验证集	3932
可回收物	25200	测试集	3866
厨余垃圾	7364		

4.3 图像数据处理

爬取完数据集之后需要对图像将进行图像数据处理，处理思路过程分为以下几点：

- (1) 查看单张图像和图像编码数据。
- (2) 获取所有图像地址，通过图像地址获取所有图像，并构建图像对应类别。
- (3) 对训练图像进行数据增强，包括水平翻转、垂直翻转、水平方向垂直翻转、随机裁剪等。

4.4 数据集处理的缺点

没有进行标签处理，图像分类模型需要通过图像和标签的对应关系来学习不同类别的特征。如果没有标签，模型无法识别图像所属的类别，无法进行有监督的学习。

在没有标签的情况下，训练数据是不完整的，模型只能看到图像数据，但无法得知每张图像对应的类别信息。这将导致模型无法计算损失函数，也无法进行梯度下降来更新模型参数。

在实际应用中，垃圾分类系统需要根据模型的预测结果对垃圾进行分类。如果没有标签作为训练依据，模型将无法正确地对新图像进行分类，整个系统将失去作用。

4.5 数据集处理的改进

(1) 添加标签处理步骤

确保在数据加载时正确读取图像和对应的标签，将标签进行编码以适应模型的训练和评估。以下是改进后的代码，包括标签处理的详细步骤。

```

1. # 数据增强
2. datagen = ImageDataGenerator(
3.     horizontal_flip=True,
4.     vertical_flip=True,
5.     rotation_range=20,
6.     width_shift_range=0.2,
7.     height_shift_range=0.2,
8.     zoom_range=0.2
9. )
10. datagen.fit(images)
11.
12. # 划分数据集
13. train_images, test_images, train_labels, test_labels = train_test_split(
14.     images, labels, test_size=0.2, random_state=42)
15. val_images, test_images, val_labels, test_labels = train_test_split(
16.     test_images, test_labels, test_size=0.5, random_state=42)

```

```

16. # 将标签进行独热编码
17. train_labels = to_categorical(train_labels, num_classes=len(os.listdir(data_dir)))
18. val_labels = to_categorical(val_labels, num_classes=len(os.listdir(data_dir)))
19. test_labels = to_categorical(test_labels, num_classes=len(os.listdir(data_dir)))

```

针对以上缺点进行标签编码，使用字典 `class_to_idx` 将类别名称映射为整数标签。加载图像时，根据图像所在文件夹的名称为每张图像分配标签。使用 `ImageDataGenerator` 进行数据增强，增加数据的多样性，提高模型的泛化能力。

4.6 数据准备和预处理

主要有数据集训练、验证、测试划分三个部分：

a. 设置划分比例。

定义训练集、验证集和测试集的划分比例。在目标路径下创建 `train`、`val` 和 `test` 文件夹，并在每个文件夹下创建类别文件夹。

`split_rate` 定义了数据集的划分比例，分别为训练集（80%）、验证集（10%）和测试集（10%）。

b. 按比例划分数据并复制文件

遍历每个类别文件夹，获取所有图片文件名，随机打乱后按比例复制到相应的训练集、验证集和测试集文件夹中。

```

1. for class_name in class_names:
2.     # 当前类别数据路径
3.     current_class_data_path = os.path.join(file_path, class_name)
4.     current_all_data = os.listdir(current_class_data_path)
5.     current_data_length = len(current_all_data)
6.
7.     # 创建一个随机索引列表
8.     current_data_index_list = list(range(current_data_length))
9.     random.shuffle(current_data_index_list)
10.    # 目标路径
11.    train_path = os.path.join(new_file_path, 'train', class_name)
12.    val_path = os.path.join(new_file_path, 'val', class_name)
13.    test_path = os.path.join(new_file_path, 'test', class_name)

```

- 遍历每个类别文件夹，获取所有图片文件名并随机打乱。
- 根据设定的比例将图片复制到相应的训练集、验证集和测试集文件夹中。
- 打印每个类别的图片数量及其在不同数据集分布情况。

通过上述步骤，数据集被有效地划分为训练集、验证集和测试集，为后续的模型训练和评估提供了良好的基础。

5 模型方法

5.1 ResNet50

Jie, P 等人提出的 ResNet50 是一种广泛使用的卷积神经网络模型^[7]，已在大规模

数据集（ImageNet）上进行预训练。预训练模型已经学习到一些基础的特征，如边缘、纹理等，上述特征可以在许多不同的任务中复用。

ResNet50 常见的输入形状是 (224, 224, 3)，表示宽度和高度为 224 像素，有 3 个颜色通道（RGB）。模型构建时不包括原模型的顶层（即最后的全连接分类层），由于我们根据新的任务添加自己的分类层，使用在 ImageNet 数据集上预训练的权重，使模型已经具有了良好的初始特征提取能力。

对于 ResNet50 模型参数进行冻结，防止预训练模型的参数在训练过程中被更新，即只使用预训练模型提取特征而不更新其权重，实现减少训练时间，并防止在数据集上过拟合。大体而言，我们在实验时把 Resnet50 视为特征提取器，把归一化层、全局平均池化层和全连接层上述新加入模块视为分类器。

归一化层将输入图像的像素值归一化至[1,1]区间，使得模型的收敛速度提高，优化训练效果。

$$x' = \frac{x}{127.5} - 1 \quad (1)$$

全局平均池化层用于将特征图的空间维度降至 1，输出每个特征图的平均值，有效地减少参数数量且降低过拟合风险，同时还保留全局特征信息。式(2)中 y_k 是第 k 个通道的输出， x_{ijk} 是第 k 个通道在位置 (i,j) 的输入：

$$y_k = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W x_{ijk} \quad (2)$$

全连接层用于分类。该层的输出维度为 class_num 即 120，对应分类任务的类别数量。激活函数 softmax 将输出转换为概率分布，每个值表示对应类别的概率。

在优化器的选择上，选择 Kingma P D 等人提出的 Adam 优化器^[8]。Adam 是基于二阶和二阶矩估计的自适应学习率优化器，结合 AdaGrad 和 RMSProp 的优点，能够处理稀疏梯度和非平稳目标。其主要优点是适应不同特征的学习速率，提高训练效率。式(3)中 $\hat{m}_t$ 和 $\hat{v}_t$ 为二阶动量的偏差校正估计：

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (3)$$

对于交叉熵损失函数，其常用于多类别分类任务，衡量真实标签与预测概率之间的差异。其值越小，说明预测结果与真实标签越接近。式(4)中 N 为类别数， y_i 为真实标签（one-hot 编码）， $\hat{y}_i$ 是预测概率：

$$L = - \sum_{i=1}^N y_i \log(\hat{y}_i) \quad (4)$$

5.2 MobileNetV2

SANDLER, M.等人提出的 MobileNetV2 模型为轻量级深度神经网络，在基于该模型构建时同 5.1 中思路，先加载预训练的模型对其参数进行冻结。模型结构上把 MobileNetV2 模型作为特征提取器加入，而后依次为归一化层、全局平均池化层、Dropout 层和全连接层，Dropout 层在训练过程中以 50% 的概率随机丢弃一半的神经元防止过拟合，通过引入随机性 Dropout 层可以使模型更具鲁棒性。整体上看，基于 MobileNetV2 的优化模型结构及其输出如表 2：

表 2 MobileNetV2 优化模型具体输出解析

	Layer type	Output Shape	Filter Size	Stride
Initial Convolutional Layer	Input	(224,224,3)	—	—
	Conv2D	(112,112,32)	3*3	2
	BatchNormalization & ReLU6			
	Depthwise Conv2D	(112,112,32)	3*3	1
	BatchNormalization & ReLU6			
Inverted Residual Block	Conv2D	(112,112,64)	1*1	1
	Depthwise Conv2D	(56,56,64)	3*3	2
	BatchNormalization & ReLU6			
	Conv2D	(56,56,128)	1*1	1
	BatchNormalization & ReLU6			
	Depthwise Conv2D	(56,56,128)	3*3	1
	BatchNormalization & ReLU6			
	Conv2D	(56,56,128)	1*1	1
	BatchNormalization & ReLU6			
	Depthwise Conv2D	(28,28,128)	3*3	1
	BatchNormalization & ReLU6			
	Conv2D	(28,28,256)	1*1	1
	BatchNormalization & ReLU6			
	Depthwise Conv2D	(28,28,256)	3*3	1
	BatchNormalization & ReLU6			
	Conv2D	(28,28,256)	1*1	1
	BatchNormalization & ReLU6			
	Depthwise Conv2D	(14,14,256)	3*3	2
.....				
Top Convolutional Layer	Conv2D	(7, 7,1024)	1*1	1
Added Classifiers	GlobalAveragePooling2D	(1280)	—	—
	Dropout	(1280)	—	—
	Dens	(1280)	—	—

6 评估与测试

6.1 模型评估与测试

6.1.1 ResNet50

针对 ResNet50 模型其初始参数如表 3:

表 3 ResNet 模型初始参数表

参数名	参数值
image_size	224*224*3
batch_size	16
optimizers	Adam
learning_rate	0.0001
loss	categorical_crossentropy
epochs	30

其训练结果如图 4(左), 经过 30 个 epoch 后, 模型在验证集上的 loss 值为 3.0561, accuracy 为 0.2969, 能够明显的看出, 模型收敛的速度极其缓慢, 为了参考模型的具体

的准确率，重新设定 epoch=200，实验结果如图 4(右)：

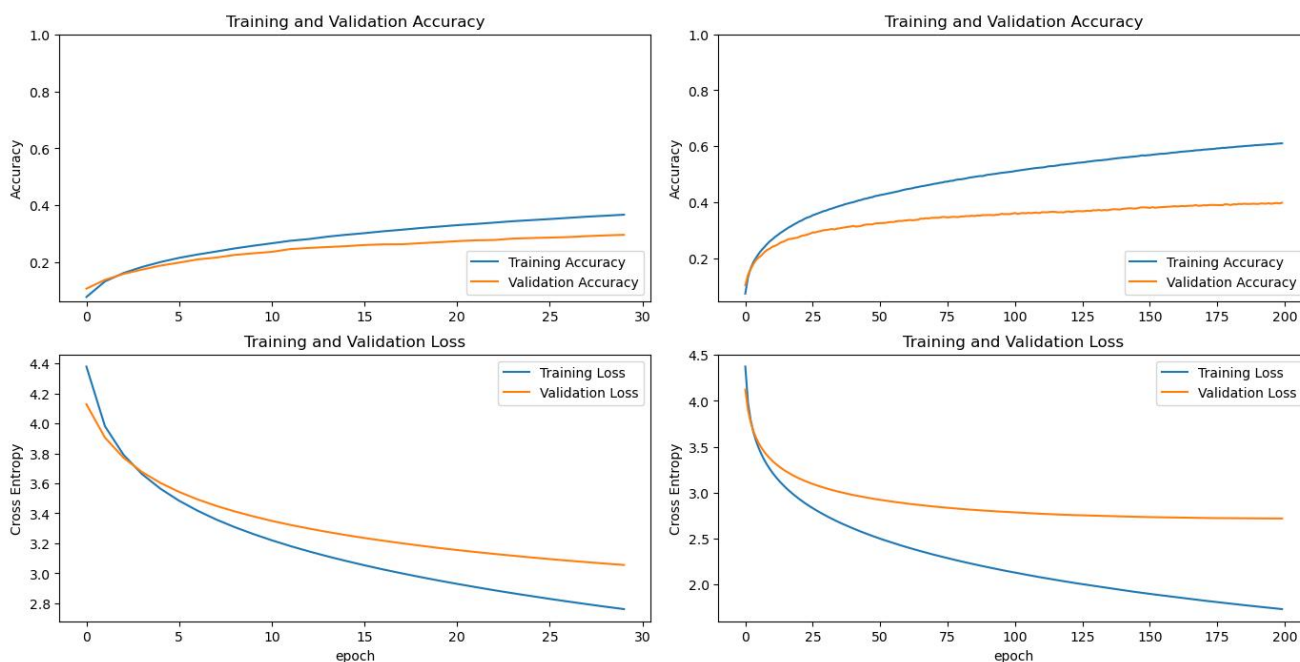


图 4 ResNet 模型训练可视化

看出模型并没有实质性的改变，因此还需要进一步优化模型，考虑 ResNet50 模型的迁移策略问题，在上述实验中，我们冻结了 ResNet50 模型的全部卷积层，只训练针对垃圾分类任务的全连接层，但是由于 ResNet50 模型的参数量较多，模型的层级较深，只训练全连接层并不能很好的获取垃圾分类任务的特征。故对迁移学习策略进行了实验，全部层级都设定为可训练，当然这里的 epoch 依旧设置为 30 次，得图 5：

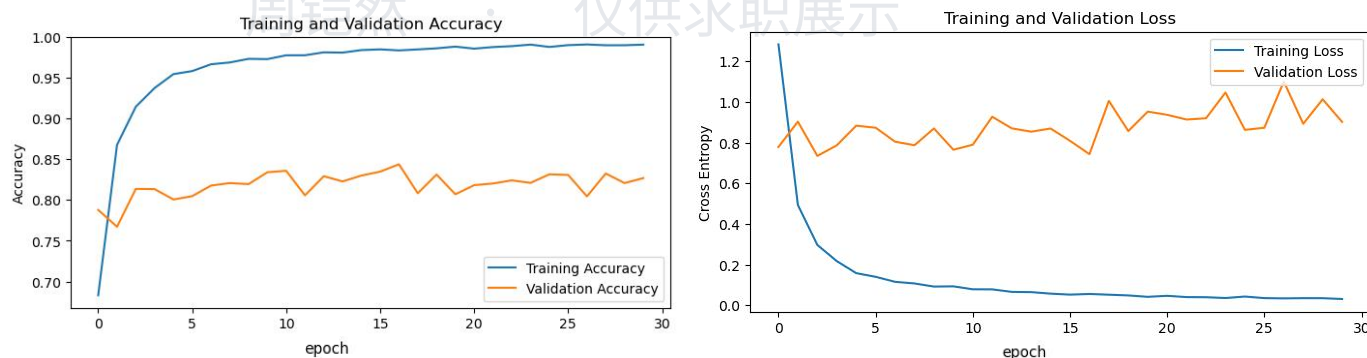


图 5 ResNet50_all_layers_epoch30

可看出模型的收敛速度变快，在训练集上达到了超过 0.95 的准确率，而在验证集准确率达到大约 0.82。训练集上损失在下降，而验证集上损失在波动，表明模型在训练集上学习良好，但可能过度拟合了训练集，根据分析故进入 MobileNetV2 模型实验。

6.1.2 MobileNetV2

预训练模型的复用中，我们删除原始的分类器，加入针对于垃圾分类任务的分类器。并根据常用的三种迁移策略，对模型进行微调，其参数如表 4。

策略 1：冻结全部卷积层，只训练针对垃圾分类任务的全连接层

策略 2：冻结前 80 层网络，训练剩余卷积层以及全连接层

策略 3：训练模型的全部卷积层以及全连接层

表 4 MobileNetV2 模型微调参数表

模型	训练参数	冻结参数
策略 1	153720	2257984
策略 2	2192504	219200
策略 3	2377592	34112

训练结果如图 6:



图 6 MobileNetV2 三策略模型训练可视化

各策略下模型准确率如表 5:

表 5 MobileNetV2 三策略下模型准确率

模型	训练集	验证集	测试集
策略 1	0.9954	0.8643	0.8609
策略 2	0.9909	0.8457	0.8483
策略 3	0.9874	0.8319	0.8253

经过对比实验,分析实验结果,在训练的参数方面,策略 1 仅有策略 3 的 6.5%、策略 2 的 7.01%。预训练模型的迁移过程中,并不是训练的参数越多越好,由于预训练模型的部分参数固定,因此实验的准确率更多看垃圾分类数据集与参数的拟合情况。

根据图 6 的实验结果来看,策略 1 下模型表现更加稳定,在 30 轮的训练中,模型并未出现抖动情况,不管是准确率还是损失函数,稳定性都为三者中最优。策略 2 中,虽然准确率的表现整体上与策略 1 相差较小,但损失函数不稳定出现抖动。

垃圾分类的实际应用中,模型的稳定性也是一个重要因素,模型的不稳定容易导致垃圾分类错误,投放到错误的垃圾类别中,造成资源的浪费等。对于策略 3 下的模型来说,虽训练参数最多,但整体的准确率不如其他两个模型,因为训练的参数较多数据集的图片数量并不能有效提供给模型进行学习,且数据分类较多,难以达到很高的识别准确率,然而扩充数据集是一项复杂的工程,且在实际的分类任务中,数据集的大小很难达到深度学习模型的要求,并不适合应用与此次垃圾分类任务中。

通过上述实验,最终确定策略 1 进行后续研究。策略 1 下模型在训练集和验证集上出现明显差异,训练集的准确率为 99.53%,而验证集仅有 85.38%,说明模型存在过拟合问题。由于在训练集上的拟合度过高,导致模型应用中若使用与训练集相仿的图片时识别准确率很高,但将同一垃圾类型图片风格改变会导致模型的准确率降低,因此,需要对模型进行一定的优化。防止过拟合的主要机制有数据增强、正则化、随机权重初始化、Dropout 等。在策略 1 下的模型中,由于只是训练全连接层,垃圾分类任务下输入全连接层的所有神经元均被拟合进模型中,模型在训练集上才会出现过拟合情况,故在

冻结全部卷积层的实验策略里首要考虑丢弃一部分全连接层前的神经元，即在全连接层前加入 Dropout 机制，以 0.5 的概率丢弃一部分神经元，以达到防止模型过拟合的效果，加入 Dropout 机制后训练结果如图 7：

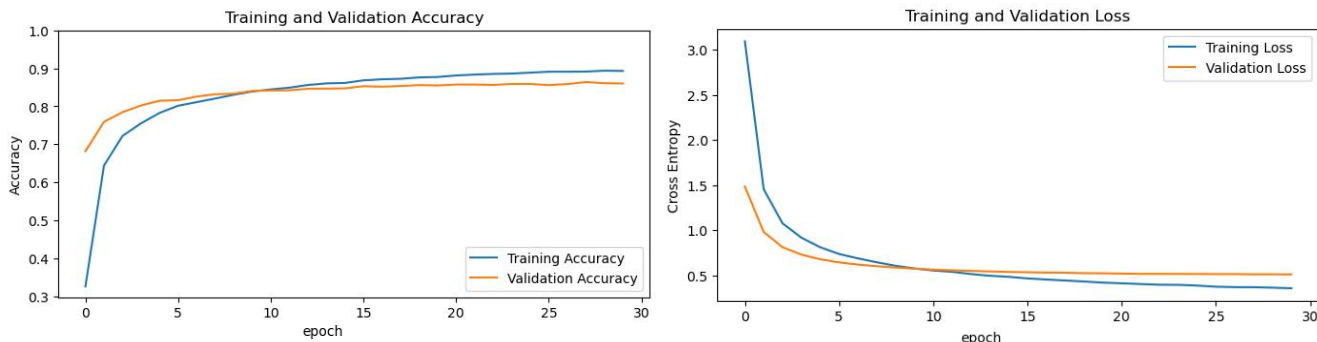


图 7 策略 1 下加入 Dropout 层模型训练可视化

模型的准确率、loss 函数的大小如表 6，策略 1 下如表 7：

表 6 策略 1 下加入 Dropout 层模型评估指标

	训练集	验证集	测试集
准确率	0.8961	0.8587	0.8552
损失值	0.3934	0.5144	0.5271

表 7 策略 1 下未加入 Dropout 层模型评估指标

	训练集	验证集	测试集
准确率	0.9954	0.8643	0.8609
损失值	0.0597	0.5438	0.5697

在加入 Dropout 机制后，模型表现稳定且准确率在验证集和测试集上优于原来进一个百分点，选定该模型确定为实际垃圾分类任务中的模型。

6.2 交互窗口

基于 PyQt5 和上述实验已得到模型的图形用户界面（GUI）。主窗口继承自 QTabWidget 类，包含多个标签页，每个标签页都有不同的功能和布局。

(1) 界面设计

主窗口标题为“智能垃圾分类系统”，设置了窗口图标。包含一个水平布局的主窗口，分为左右两个部分：左侧部分展示一个用于显示待识别图片的标签，右侧部分包含了多个按钮和用于显示识别结果的标签和文本框。实现如图 8。

(2) 功能构成

- **上传图片按钮**：允许用户从文件系统中选择要识别的图片，并显示在左侧的图片标签中。
- **开始识别按钮**：对当前显示的图片进行垃圾分类识别，通过预加载的 MobileNetV2_Droup 模型进行推断，将识别结果显示在右侧的结果标签中，并显示相关的垃圾信息。
- **打开摄像头按钮**：允许用户从计算机的摄像头获取实时视频流，视频流显示在左侧的图片标签中。用户可以通过这个按钮打开或关闭摄像头。
- **截屏按钮**：在打开摄像头后，用户可以通过此按钮捕获当前摄像头画面的静态图像。



图 8 窗口主界面

(3) 识别和信息显示

使用 MobileNetV2_Droup 模型文件，对图片进行预处理后进行分类推断。根据模型输出的分类结果，更新界面上的垃圾种类、物品名称和概率大小信息。根据不同类别的垃圾，显示相应的垃圾信息，例如垃圾的处理方式和分类方法。

(4) 实时视频流处理

通过 OpenCV 库捕获摄像头的视频流，将每一帧图像转换为适合在 PyQt 界面中显示的格式。在每次新的视频帧到来时，同时进行垃圾分类的推断，实时更新界面上的识别结果和垃圾信息。

关于摄像头使用触发频率为 300 毫秒的定时器，以达在摄像头进行动态识别其所获取的图像进行识别时较为稳定，减小光线或人体动作对物体识别的影响。具体代码如下：

```
1. def toggle_camera(self):
2.     if self.cap is None:
3.         self.cap = cv2.VideoCapture(0)
4.         self.timer.start(30)
5.     else:
6.         self.timer.stop()
7.         self.cap.release()
8.         self.cap = None
9.         self.img_label.clear()
```

(5) 事件处理

- a. 上传图片：点击“上传图片”按钮时，程序调用文件对话框让用户选择一个图

片文件。若用户选择了文件，程序将该文件复制到指定的临时目录，并更新待识别图片的路径，左侧显示的图片标签会根据新的图片路径加载和显示图片。

b. 开始识别：点击“开始识别”按钮后，程序会使用预加载模型文件对当前显示的图片进行垃圾分类识别。结果会显示在右侧标签中，包括垃圾种类、物品名称和识别概率。根据识别结果的垃圾种类，相应的垃圾信息会显示在文本框中，帮助用户了解该类垃圾的处理方法和环保意识。

c. 打开摄像头：点击“打开摄像头”按钮后，程序会尝试打开计算机上的摄像头，并实时捕获视频流。捕获到的视频流将显示在左侧的图片标签中，并使用定时器定期更新显示，形成实时视频效果，再次点击该按钮会关闭摄像头，并停止视频流的显示。

d. 截屏：当摄像头处于打开状态时，点击“截屏”按钮，将当前摄像头捕获的静态图像保存到临时目录中。图片会被显示在左侧的图片标签中，代替实时视频流的显示，同时定时器会暂停更新，保持静态图像的显示。

e. 键盘事件的处理：按下键盘上的 `Escape` 键时，程序会检查当前是否正在显示摄像头捕获的静态图像。如果是，程序会恢复定时器的更新，重新开始实时视频流的显示，实现切换实时和静态图像的显示状态，提升用户体验。

f. 窗口关闭：当尝试关闭窗口时，程序会弹出对话框，询问是否确认退出程序。如果用户确认退出，程序会释放摄像头资源（若打开），然后关闭应用程序。如果用户取消退出，程序会忽略关闭事件，继续运行应用程序，保持当前状态。

具体实现示例如图 9-10：



图 9 传图片识别示例

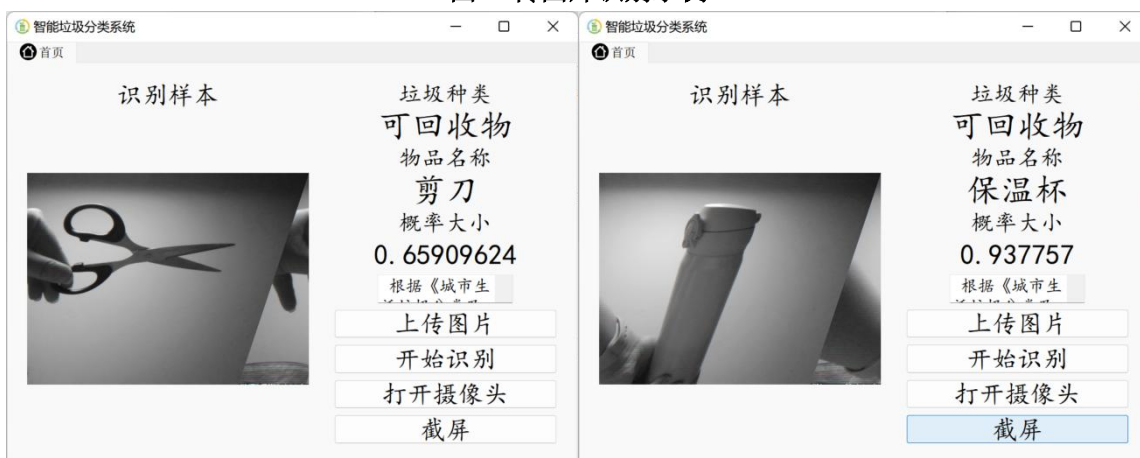


图 10 摄像头捕获识别示例

7 项目设计总结

在本项目中，设计了一个智能垃圾分类系统，旨在实现高准确率的垃圾分类识别并提供用户友好的交互体验

为了构建高准确率的垃圾分类模型，我们选择迁移学习的方法，使用预训练的卷积神经网络 ResNet50 和 MobileNetV2 作为基础模型。通过冻结基础模型的部分层，并调整超参数进行训练和优化，对比各模型在训练集、验证集与测试集上的准确率和损失值，最后选择了表现效果较稳定且准确率表现较好的模型。数据集以 8:1:1 比例被分为训练集、验证集和测试集，并采用交叉熵损失函数和 Adam 优化器进行模型优化。通过数据增强策略（如旋转、缩放、裁剪）扩展数据集，确保模型具有良好的泛化能力。**该选定模型**基于 MobileNetV2，将其视为特征提取器添加新的层级作为分类器，并以冻结全部卷积层只训练针对垃圾分类任务的全连接层作为策略加入 Dropout 层。

形成实现用户友好的垃圾分类系统方面，设计了一个交互式界面。主窗口标题为“智能垃圾分类系统”，包含了上传图片、开始识别、打开摄像头和截屏等功能按钮。界面通过实时更新的识别结果和垃圾信息，帮助用户理解和分类各种类型的垃圾，提升环保意识。在界面设计方面，我们使用了 PyQt 库，通过事件处理实现了上传图片、开始识别、实时视频流处理、截屏等功能。用户可以通过界面交互，方便快捷地进行垃圾分类识别，并获取相关的环保信息和建议。

由于时间等问题，本项目仍有完善空间。数据集部分，查找收集整理好数据集时发现大部分经过验证的良好垃圾图片数据集垃圾类别基本小于 20 类，故决定进行爬虫获取数据。通过爬虫在百度图片上收集数据，数据中会出现目录关键词相关性小影响模型训练的数据，虽我们有进行手动删除部分明显与目标不符的图片，但仍有可能存在影响模型训练的数据。

总体而言，本项目不仅成功构建了一个高准确率的垃圾分类模型，还实现了一个功能完善、用户友好的垃圾分类系统，为环保工作提供了实用的技术支持和推广应用。

8 参考文献

- [1] 邓银莹,徐勇.基于 Tensorflow 的垃圾分类回收系统设计[J].电脑知识与技术, 2021, 017(023):50-52.
- [2] 康庄, 杨杰, 郭濠奇. 基于机器视觉的垃圾自动分类系统设计[J]. 浙江大学学报(工学版), 2020, 54(7): 1272-1280. doi:10.3785/j.issn.1008-973X.2020.07.004.
- [3] He K ,Zhang X ,Ren S , et al.Deep Residual Learning for Image Recognition.[J].CoRR,2015,abs/1512.03385
- [4] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications[EB/OL]. 2017.
- [5] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks[C]//2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Salt Lake City, UT, USA: IEEE, 2018: 4510-4520. doi: 10.1109/CVPR.2018.00474.
- [6] Michael Yuan. MobileNet V2 论文初读[EB/OL]. (2018)[2018-04-14]. 知乎, <https://zhuanlan.zhihu.com/p/33075914>.
- [7] Jie P ,Shuai K ,Zhengyuan N , et al.Residual convolutional neural network for predicting response of transarterial chemoembolization in hepatocellular carcinoma from CT imaging.[J].European radiology,2020,30(1):413-424.
- [8] Kingma P D ,Ba J .Adam: A Method for Stochastic Optimization.[J].CoRR,2014, abs/1412.6980

周铠然 · 仅供求职展示

附录

附录-目录

介绍:

附件 1: 源码

附件 2: 爬虫关键词目录

附录 1

介绍: 附件 1 源码进行部分代码删减的代码示例

```

1.      #数据爬虫
2.      import requests  # 模拟请求
3.      import json  # 轻量级的数据交换格式，易于人阅读和编写
4.      from urllib import parse  # 用于 url 的解析，合并，编码，解码
5.      import os  # 用于对文件进行操作了模块
6.      import time  # 时间模块
7.
8.
9.      class BaiduImageSpider(object):  # 创建一个类
10.         def __init__(self):
11.             self.directory_base = r"E:\test\garbage2\garbage_images\{" # 存储目录 这里
需要修改为希望保存的目录  }不要丢
12.             self.json_count = 0  # 请求到的 json 文件数量（一个 json 文件包含 30 个图像文件）
13.             self.url = 'https://image.google.com/search/acjson?tn=resultjson_com&logid=5
179920884740494226&ipn=rj&ct' \
14.                 '=201326592&is=&fp=result&queryWord={' \
15.                 '}&cl=2&lm=-1&ie=utf-8&oe=utf-8&adpicid=&st=-1&z=&ic=0&hd=&latest
=@right=&word={' \
16.                 '}&s=&se=&tab=&width=&height=&face=0&istype=2&qc=&nc=1&fr=&experim
ode=&nojc=&pn={' \
17.                 '}&rn=30&gsm=1e&1635054081427= '
18.             self.header = {
19.                 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537
.36 (KHTML, like Gecko) '
20.                 'Chrome/95.0.4638.54 Safari/537.36 Edg/95.0.1020.30 '
21.             }
22.
23.         # 获取图像链接
24.         def get_image_link(self, url):
25.             list_image_link = []  # 建立一个图像列表
26.             strhtml = requests.get(url, headers=self.header)  # Get 方式获取网页数据
27.
28.             try:
29.                 jsonInfo = json.loads(strhtml.text)  # json.loads 将已编码的 JSON 字符串解
码为 Python 对象

```

```

30.         except json.JSONDecodeError as e:
31.             print(f"JSONDecodeError: {e}")
32.             print(f"Response text: {strhtml.text}")
33.             return list_image_link
34.
35.         # 获取图像的链接存到了列表中
36.         for index in range(30):
37.             try:
38.                 list_image_link.append(jsonInfo['data'][index]['thumbURL']) # 在列表末尾添加新的对象，将图像的链接放在列表中
39.             except (KeyError, IndexError):
40.                 continue
41.         return list_image_link
42.
43.     # 创建存储文件夹
44.     def create_directory(self, name):
45.         directory = self.directory_base.format(name.strip())
46.         if not os.path.exists(directory):
47.             os.makedirs(directory)
48.         return directory
49.
50.     # 下载图片
51.     def save_image(self, img_link, filename):
52.         res = requests.get(img_link, headers=self.header) # 模拟 get 请求 返回信息
53.         if res.status_code == 404:
54.             print(f"图片{img_link}下载出错----->")
55.             return
56.         with open(filename, "wb") as f: # 使用二进制形式 覆盖写文件
57.             f.write(res.content) # requests 模块中的 content 返回的是二进制的的数据
58.             print("存储路径: " + filename) # 打印存储路径
59.
60.     # 入口函数
61.     def run(self, name):
62.         searchName_parse = parse.quote(name) # 编码 将中文转换为 url 编码格式
63.         directory = self.create_directory(name) # 调用创建文件夹的函数，根据查询内容创建文件
64.
65.         pic_number = 0 # 图像数量
66.         for index in range(self.json_count):
67.             pn = (index + 1) * 30 # pn 表示一组文件，一组包含 30 个图像内容
68.             request_url = self.url.format(searchName_parse, searchName_parse, str(pn))
69.

```

```

70.         list_image_link = self.get_image_link(request_url) # 通过新的 url 调用图
像链接函数，获取新的图像链接
71.         for link in list_image_link:
72.             pic_number += 1
73.             self.save_image(link, os.path.join(directory, "img_" + name + "_" +
str(pic_number) + '.jpg'))
74.             time.sleep(0.2) # 休眠 0.2 秒，防止封 ip
75.             print(name + "图像下载成功")
76.             print("图片存入{}".format(directory))
77.
78.
79.     if __name__ == '__main__':
80.         filename = open("./source_refuse_name.txt", encoding="utf-8")
81.         str_all = filename.readlines()
82.         for i in str_all:
83.             line = i.strip("\n")
84.             print("正在爬取" + line + "类型的图片")
85.             spider = BaiduImageSpider()
86.             spider.json_count = 8 # 默认下载一组图片（30 张）
87.             spider.run(line)
88.         filename.close()
89.
90.     #数据集训练、验证、测试划分
91.     import os
92.     import random
93.     from shutil import copy2
94.
95.     # 源文件路径
96.     file_path = r"E:/test/garbage2/garbage_images"
97.     # 新文件路径
98.     new_file_path = r"E:/test/garbage2/garbage_image"
99.
100.    # 比例
101.    split_rate = [0.8, 0.1, 0.1]
102.    print("Starting...")
103.    print("Ratio= {}:{}:{}".format(int(split_rate[0] * 10), int(split_rate[1] * 10), int
(split_rate[2] * 10)))
104.    class_names = os.listdir(file_path)
105.    # 在目标目录下创建文件夹
106.    split_names = ['train', 'val', 'test']
107.    # 判断是否存在本文件夹
108.    if os.path.isdir(new_file_path):
109.        pass
110.    else:

```

```

111.     os.mkdir(new_file_path)
112.     for split_name in split_names:
113.         # split_path = os.path.join(new_file_path, split_name)
114.         split_path = new_file_path + "/" + split_name
115.         if os.path.isdir(split_path):
116.             pass
117.         else:
118.             os.mkdir(split_path)
119.         # 然后在 split_path 的目录下创建类别文件夹
120.         for class_name in class_names:
121.             class_split_path = os.path.join(split_path, class_name)
122.             if os.path.isdir(class_split_path):
123.                 pass
124.             else:
125.                 os.mkdir(class_split_path)
126.
127.         # 按照比例划分数据集，并进行数据图片的复制
128.         # 首先进行分类遍历
129.         for class_name in class_names:
130.             current_class_data_path = os.path.join(file_path, class_name)
131.             current_all_data = os.listdir(current_class_data_path)
132.             current_data_length = len(current_all_data)
133.             current_data_index_list = list(range(current_data_length))
134.             random.shuffle(current_data_index_list)
135.             # 周铠然 · 仅供求职展示
136.             train_path = os.path.join(os.path.join(new_file_path, 'train'), class_name)
137.             val_path = os.path.join(os.path.join(new_file_path, 'val'), class_name)
138.             test_path = os.path.join(os.path.join(new_file_path, 'test'), class_name)
139.             train_stop_flag = current_data_length * split_rate[0]
140.             val_stop_flag = current_data_length * (split_rate[0] + split_rate[1])
141.             current_idx = 0
142.             train_num = 0
143.             val_num = 0
144.             test_num = 0
145.             for i in current_data_index_list:
146.                 src_img_path = os.path.join(current_class_data_path, current_all_data[i])
147.                 if current_idx <= train_stop_flag:
148.                     copy2(src_img_path, train_path)
149.                     train_num = train_num + 1
150.                 elif (current_idx > train_stop_flag) and (current_idx <= val_stop_flag):
151.                     copy2(src_img_path, val_path)
152.                     val_num = val_num + 1
153.                 else:
154.                     copy2(src_img_path, test_path)

```

```

155.         test_num = test_num + 1
156.
157.         current_idx = current_idx + 1
158.
159.         print("<{}> has {} pictures,train:val:test={}:{}:{}".format(class_name, current_
data_length, train_num, val_num,
160.                                     test_num))
161.     print("Done")
162.
163.     #ResNet50_train 后续其他优化模型代码仅放涉及参数部分，其他部分同 ResNet50_train
164.     import tensorflow as tf
165.     import matplotlib.pyplot as plt
166.     from time import *
167.     import os
168.     from pathlib import Path
169.
170.     os.environ["CUDA_VISIBLE_DEVICES"] = "0"
171.
172.     image_path = "E:/test/garbage2/garbage_image" # 代码所用图集的文件夹
173.     image_path = Path(image_path)
174.     train_dir = image_path / "train"
175.     validation_dir = image_path / "val"
176.
177.     img_height = 224
178.     img_width = 224
179.     batch_size = 16
180.
181.     # 加载数据集，按照 8 比 2 的比例划分数据集，其中 8 份作为训练集，2 份作为验证集
182.     def data_load(train_dir, validation_dir,img_height, img_width, batch_size):
183.         train_ds = tf.keras.preprocessing.image_dataset_from_directory(
184.             train_dir,
185.             label_mode='categorical',
186.             seed=123,
187.             color_mode="rgb",
188.             image_size=(img_height, img_width),
189.             batch_size=batch_size)
190.
191.         val_ds = tf.keras.preprocessing.image_dataset_from_directory(
192.             validation_dir,
193.             label_mode='categorical',
194.             seed=123,
195.             color_mode="rgb",
196.             image_size=(img_height, img_width),
197.             batch_size=batch_size)

```

```

198.
199.     class_names = train_ds.class_names
200.     return train_ds, val_ds, class_names
201.
202.
203.
204.     # 模型加载, 指定图片处理的大小和是否进行迁移学习
205.     def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
206.         # 微调的过程中不需要进行归一化的处理
207.         base_model = tf.keras.applications.ResNet50(input_shape=IMG_SHAPE,
208.                                                       include_top=False,
209.                                                       weights='imagenet') # 加载
mobilenetv2 模型
210.         base_model.trainable = False # 将主干的特征提取网络参数进行冻结
211.         model = tf.keras.models.Sequential([
212.             tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,
input_shape=IMG_SHAPE),
213.             # 归一化处理, 将像素值处理为-1 到 1 之间
214.             base_model,
215.             tf.keras.layers.GlobalAveragePooling2D(), # 全局平均池化
216.             tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层,
用于分类
217.         ])
218.         model.summary() # 输出模型信息
219.         Adam = tf.keras.optimizers.Adam(learning_rate=0.0001)
220.         model.compile(optimizer=Adam, loss='categorical_crossentropy',
221.                       metrics=['accuracy']) # 用于编译模型, 指定模型的优化器是 adam 优化器,
模型的损失函数是交叉熵损失函数
222.         return model # 返回模型
223.
224.
225.     # 展示训练过程的曲线
226.     def show_loss_and_acc(history):
227.         # 从 history 参数中提取准确率和误差信息
228.         acc = history.history['accuracy']
229.         val_acc = history.history['val_accuracy']
230.         loss = history.history['loss']
231.         val_loss = history.history['val_loss']
232.         # 绘制准确率曲线图
233.         plt.figure(figsize=(8, 8))
234.         plt.subplot(2, 1, 1)
235.         plt.plot(acc, label='Training Accuracy')
236.         plt.plot(val_acc, label='Validation Accuracy')
237.         plt.legend(loc='lower right')

```

```

238.     plt.ylabel('Accuracy')
239.     plt.ylim([min(plt.ylim()), 1])
240.     plt.title('Training and Validation Accuracy')
241.     # 绘制模型误差曲线图
242.     plt.subplot(2, 1, 2)
243.     plt.plot(loss, label='Training Loss')
244.     plt.plot(val_loss, label='Validation Loss')
245.     plt.legend(loc='upper right')
246.     plt.ylabel('Cross Entropy')
247.     plt.title('Training and Validation Loss')
248.     plt.xlabel('epoch')
249.     plt.savefig("E:/test/garbage2/result_images/ResNet50_epoch30.png", dpi=100)
250.
251.
252.     # 训练模型主流程
253.     def train(epochs):
254.         begin_time = time() # 记录开始时间
255.         train_ds, val_ds, class_names = data_load(train_dir, validation_dir, img_height,
            img_width, batch_size)
256.         print(class_names) # 输出类名
257.         model = model_load(class_num=len(class_names)) # 加载模型
258.         history = model.fit(train_ds, validation_data=val_ds, epochs=epochs) # 开始训练
259.         model.save("E:/test/garbage2/models/ResNet50_epoch30.h5")
260.         end_time = time() # 记录结束时间
261.         run_time = end_time - begin_time # 记录时间花费
262.         print('该循环程序运行时间: ', run_time, "s") # 输出程序运行时间
263.         show_loss_and_acc(history) # 展示曲线图
264.
265.
266.     if __name__ == '__main__':
267.         train(epochs=30)
268.
269.     #ResNet_all_layers train
270.     # 模型加载, 指定图片处理的大小和是否进行迁移学习
271.     def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
272.         # 微调的过程中不需要进行归一化的处理
273.         base_model = tf.keras.applications.ResNet50(input_shape=IMG_SHAPE,
274.             include_top=False,
275.             weights='imagenet') # 加载
mobilenetv2 模型
276.         base_model.trainable = True # 将主干的特征提取网络参数进行冻结
277.         model = tf.keras.models.Sequential([
278.             tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,

```

```

input_shape=IMG_SHAPE),
279.         # 归一化处理, 将像素值处理为-1 到 1 之间
280.         base_model,
281.         tf.keras.layers.GlobalAveragePooling2D(), # 全局平均池化
282.         tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层,
用于分类
283.     ])
284.     model.summary() # 输出模型信息
285.     Adam = tf.keras.optimizers.Adam(learning_rate=0.0001)
286.     model.compile(optimizer=Adam, loss='categorical_crossentropy',
287.                   metrics=['accuracy']) # 用于编译模型, 指定模型的优化器是 adam 优化器,
模型的损失函数是交叉熵损失函数
288.     return model # 返回模型
289.
290. #ResNet50_epoch_200 train
291. # 模型加载, 指定图片处理的大小和是否进行迁移学习
292. def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
293.     # 微调的过程中不需要进行归一化的处理
294.     base_model = tf.keras.applications.ResNet50(input_shape=IMG_SHAPE,
295.                                                  include_top=False,
296.                                                  weights='imagenet') # 加载
mobilenetv2 模型
297.     base_model.trainable = False # 将主干的特征提取网络参数进行冻结
298.     model = tf.keras.models.Sequential([
299.         tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,
input_shape=IMG_SHAPE),
300.         # 归一化处理, 将像素值处理为-1 到 1 之间
301.         base_model,
302.         tf.keras.layers.GlobalAveragePooling2D(), # 全局平均池化
303.         tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层,
用于分类
304.     ])
305.     model.summary() # 输出模型信息
306.     Adam = tf.keras.optimizers.Adam(learning_rate=0.0001)
307.     model.compile(optimizer=Adam, loss='categorical_crossentropy',
308.                   metrics=['accuracy']) # 用于编译模型, 指定模型的优化器是 adam 优化器,
模型的损失函数是交叉熵损失函数
309.     return model # 返回模型
310.
311. #MobileNetV2 train
312. # 模型加载, 指定图片处理的大小和是否进行迁移学习
313. def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
314.     # 微调的过程中不需要进行归一化的处理
315.     base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,

```

```

316.                                     include_top=False,
317.                                     weights='imagenet') # 加载
mobilenetv2 模型
318.     base_model.trainable = False # 将主干的特征提取网络参数进行冻结
319.     model = tf.keras.models.Sequential([
320.         tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,
input_shape=IMG_SHAPE),
321.         # 归一化处理, 将像素值处理为-1 到 1 之间
322.         base_model,
323.         tf.keras.layers.GlobalAveragePooling2D(), # 全局平均池化
324.         tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层,
用于分类
325.     ])
326.     model.summary() # 输出模型信息
327.     Adam=tf.keras.optimizers.Adam(learning_rate=0.0001)
328.     model.compile(optimizer=Adam, loss='categorical_crossentropy',
329.                   metrics=['accuracy']) # 用于编译模型, 指定模型的优化器是 adam 优化器,
模型的损失函数是交叉熵损失函数
330.     return model # 返回模型
331.
332. #MobileNetV2 test 后续其他模型测试代码不进行示例, 同本部分代码
333. import tensorflow as tf
334. import matplotlib as mpl
335.
336. # 可以显示中文
337. mpl.rcParams['font.sans-serif'] = ['KaiTi']
338. mpl.rcParams['font.serif'] = ['KaiTi']
339.
340. # 垃圾类名
341. class_names=['其他垃圾_PE 塑料袋', '其他垃圾_一次性杯子', '其他垃圾_一次性棉签', '其他垃圾
_口罩', '其他垃圾_唱片', '其他垃圾_干燥剂', '其他垃圾_打火机', '其他垃圾_搓澡巾', '其他垃圾_滚筒纸
', '其他垃圾_点燃的香烟', '其他垃圾_眼镜', '其他垃圾_笔', '其他垃圾_胶带', '其他垃圾_验孕棒', '其
他垃圾_鸡毛掸', '厨余垃圾_圣女果', '厨余垃圾_地瓜', '厨余垃圾_大蒜', '厨余垃圾_梨', '厨余垃圾_橙子
', '厨余垃圾_火龙果', '厨余垃圾_瓜子壳', '厨余垃圾_番茄', '厨余垃圾_白菜叶', '厨余垃圾_苹果', '厨
余垃圾_草莓', '厨余垃圾_菠萝', '厨余垃圾_菠萝蜜', '厨余垃圾_蘑菇', '厨余垃圾_蛋', '厨余垃圾_蛋挞
', '厨余垃圾_蛋糕', '厨余垃圾_西瓜皮', '厨余垃圾_豆', '厨余垃圾_豆腐', '厨余垃圾_辣椒', '厨余垃圾
_面包', '厨余垃圾_饼干', '厨余垃圾_香蕉皮', '可回收物_A4 纸', '可回收物_一次性筷子', '可回收物_不
锈钢管', '可回收物_乒乓球拍', '可回收物_书', '可回收物_保温杯', '可回收物_信封', '可回收物_充电头
', '可回收物_充电宝', '可回收物_充电线', '可回收物_凳子', '可回收物_刀', '可回收物_剪刀', '可回收
物_勺子叉子', '可回收物_包', '可回收物_卡', '可回收物_台灯', '可回收物_吹风机', '可回收物_呼啦圈
', '可回收物_塑料瓶', '可回收物_塑料盆', '可回收物_头戴式耳机', '可回收物_尺子', '可回收物_布制品
', '可回收物_帽子', '可回收物_手机', '可回收物_手电筒', '可回收物_手表', '可回收物_打气筒', '可回
收物_拉杆箱', '可回收物_插线板', '可回收物_收音机', '可回收物_放大镜', '可回收物_易拉罐', '可回收
物_望远镜', '可回收物_木制切菜板', '可回收物_木质梳子', '可回收物_木质锅铲', '可回收物_枕头', '可

```

```
回收物_桌子', '可回收物_水壶', '可回收物_水杯', '可回收物_泡沫板', '可回收物_灭火器', '可回收物_热水瓶', '可回收物_电动剃须刀', '可回收物_电子秤', '可回收物_电熨斗', '可回收物_电磁炉', '可回收物_电路板', '可回收物_电风扇', '可回收物_电饭煲', '可回收物_盘子', '可回收物_碗', '可回收物_箱子', '可回收物_纸板', '可回收物_衣架', '可回收物_袜子', '可回收物_裙子', '可回收物_裤子', '可回收物_计算器', '可回收物_订书机', '可回收物_话筒', '可回收物_路由器', '可回收物_轮胎', '可回收物_遥控器', '可回收物_钥匙', '可回收物_铁丝球', '可回收物_键盘', '可回收物_镊子', '可回收物_闹钟', '可回收物_雨伞', '可回收物_鞋', '可回收物_鼠标', '有害垃圾_太阳能电池', '有害垃圾_灯', '有害垃圾_电池', '有害垃圾_纽扣电池', '有害垃圾_胶水', '有害垃圾_药品包装', '有害垃圾_蓄电池']
```

```
342.     print(len(class_names))
343.
344.     # 数据加载, 按照 8:2 的比例加载垃圾数据
345.     def data_load(data_dir, img_height, img_width, batch_size):
346.         test_ds = tf.keras.preprocessing.image_dataset_from_directory(
347.             data_dir,
348.             label_mode='categorical',
349.             seed=123,
350.             image_size=(img_height, img_width),
351.             batch_size=batch_size)
352.         class_names = test_ds.class_names
353.         return test_ds, class_names
354.
355.
356.     # 测试 mobilenet 的准确率
357.     def my_MobileNet():
358.         test_ds, class_names = data_load("E:/test/garbage2/garbage_image/test", 224, 224, 8)
359.         print(class_names)
360.         model = tf.keras.models.load_model("E:/test/garbage2/models/MobileNet_epoch30.h5")
361.         model.summary()
362.         loss, accuracy = model.evaluate(test_ds)
363.         print('my_MobileNet test accuracy :', accuracy)
364.
365.
366.     if __name__ == '__main__':
367.         my_MobileNet()
368.
369.     #MobileNetV2_80layers_epoch30 train
370.     # 模型加载, 指定图片处理的大小和是否进行迁移学习
371.     def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
372.         # 微调的过程中不需要进行归一化的处理
373.         base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
374.                                                         include_top=False,
375.                                                         layers=tf.keras.layers,
```

```

376.                                     weights='imagenet') # 加载
mobilenetv2 模型
377.     for layer in base_model.layers[:80]:
378.         layer.trainable=False
379.     for layer in base_model.layers[80:]:
380.         layer.trainable=True
381.     #base_model.trainable = True # 将主干的特征提取网络参数进行冻结
382.     model = tf.keras.models.Sequential([
383.         tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,
input_shape=IMG_SHAPE),
384.         # 归一化处理, 将像素值处理为-1 到 1 之间
385.         base_model,
386.         tf.keras.layers.GlobalAveragePooling2D(), # 全局平均池化
387.         tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层,
用于分类
388.     ])
389.     model.summary() # 输出模型信息
390.     Adam=tf.keras.optimizers.Adam(learning_rate=0.0001)
391.     model.compile(optimizer=Adam, loss='categorical_crossentropy',
392.                   metrics=['accuracy']) # 用于编译模型, 指定模型的优化器是 adam 优化器,
模型的损失函数是交叉熵损失函数
393.     return model # 返回模型
394.
395. #MobileNetV2_all_layers train
396. # 模型加载, 指定图片处理的大小和是否进行迁移学习
397. def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
398.     # 微调的过程中不需要进行归一化的处理
399.     base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
400.                                                     include_top=False,
401.                                                     weights='imagenet') # 加载
mobilenetv2 模型
402.     base_model.trainable = True
403.     model = tf.keras.models.Sequential([
404.         tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,
input_shape=IMG_SHAPE),
405.         # 归一化处理, 将像素值处理为-1 到 1 之间
406.         base_model,
407.         tf.keras.layers.GlobalAveragePooling2D(), # 全局平均池化
408.         tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层,
用于分类
409.     ])
410.     model.summary() # 输出模型信息
411.     Adam=tf.keras.optimizers.Adam(learning_rate=0.0001)
412.     model.compile(optimizer=Adam, loss='categorical_crossentropy',

```

```

413.             metrics=['accuracy']) # 用于编译模型，指定模型的优化器是 adam 优化器，
模型的损失函数是交叉熵损失函数
414.         return model # 返回模型
415.
416.     #MobileNetV2_Droup train
417.     # 模型加载，指定图片处理的大小和是否进行迁移学习
418.     def model_load(IMG_SHAPE=(224, 224, 3), class_num=120):
419.         # 微调的过程中不需要进行归一化的处理
420.         tf.keras.backend.set_learning_phase(0)
421.         base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
422.                                                         include_top=False,
423.                                                         weights='imagenet') # 加载
mobilenetv2 模型
424.         base_model.trainable = False # 将主干的特征提取网络参数进行冻结
425.         tf.keras.backend.set_learning_phase(1)
426.         model = tf.keras.models.Sequential([
427.             tf.keras.layers.experimental.preprocessing.Rescaling(1. / 127.5, offset=-1,
input_shape=IMG_SHAPE),
428.             # 归一化处理，将像素值处理为-1 到 1 之间
429.             base_model,
430.             tf.keras.layers.GlobalAveragePooling2D(),# 全局平均池化
431.             tf.keras.layers.Dropout(rate=0.5),
432.             tf.keras.layers.Dense(class_num, activation='softmax') # 设置最后的全连接层，
用于分类
433.         ])
434.         model.summary() # 输出模型信息
435.         Adam=tf.keras.optimizers.Adam(learning_rate=0.0001)
436.         model.compile(optimizer=Adam, loss='categorical_crossentropy',
437.                       metrics=['accuracy']) # 用于编译模型，指定模型的优化器是 adam 优化器，
模型的损失函数是交叉熵损失函数
438.         return model # 返回模型
439.
440.     #交互窗口
441.     import tensorflow as tf
442.     from PyQt5.QtGui import *
443.     from PyQt5.QtCore import *
444.     from PyQt5.QtWidgets import *
445.     import sys
446.     import cv2
447.     from PIL import Image
448.     import numpy as np
449.     import shutil
450.
451.

```

```

452.     class MainWindow(QTabWidget):
453.         def __init__(self):
454.             super().__init__()
455.             self.setWindowIcon(QIcon('E:/test/garbage2/images/logo.png'))
456.             self.setWindowTitle('智能垃圾分类系统')
457.             self.model = tf.keras.models.load_model("E:/test/garbage2/models/Dropout_MobileNet_epoch30.h5") # 修改为自己的模型路径
458.             self.to_predict_name = "E:/test/garbage2/images/Init.jpeg"
459.             self.class_names = ['其他垃圾_PE 塑料袋', '其他垃圾_一次性杯子', '其他垃圾_一次性棉签', '其他垃圾_口罩', '其他垃圾_唱片', '其他垃圾_干燥剂', '其他垃圾_打火机', '其他垃圾_搓澡巾', '其他垃圾_滚筒纸', '其他垃圾_点燃的香烟', '其他垃圾_眼镜', '其他垃圾_笔', '其他垃圾_胶带', '其他垃圾_验孕棒', '其他垃圾_鸡毛掸', '厨余垃圾_圣女果', '厨余垃圾_地瓜', '厨余垃圾_大蒜', '厨余垃圾_梨', '厨余垃圾_橙子', '厨余垃圾_火龙果', '厨余垃圾_瓜子壳', '厨余垃圾_番茄', '厨余垃圾_白菜叶', '厨余垃圾_苹果', '厨余垃圾_草莓', '厨余垃圾_菠萝', '厨余垃圾_菠萝蜜', '厨余垃圾_蘑菇', '厨余垃圾_蛋', '厨余垃圾_蛋挞', '厨余垃圾_蛋糕', '厨余垃圾_西瓜皮', '厨余垃圾_豆', '厨余垃圾_豆腐', '厨余垃圾_辣椒', '厨余垃圾_面包', '厨余垃圾_饼干', '厨余垃圾_香蕉皮', '可回收物_A4 纸', '可回收物_一次性筷子', '可回收物_不锈钢管', '可回收物_乒乓球拍', '可回收物_书', '可回收物_保温杯', '可回收物_信封', '可回收物_充电头', '可回收物_充电宝', '可回收物_充电线', '可回收物_凳子', '可回收物_刀', '可回收物_剪刀', '可回收物_勺子叉子', '可回收物_包', '可回收物_卡', '可回收物_台灯', '可回收物_吹风机', '可回收物_呼啦圈', '可回收物_塑料瓶', '可回收物_塑料盆', '可回收物_头戴式耳机', '可回收物_尺子', '可回收物_布制品', '可回收物_帽子', '可回收物_手机', '可回收物_手电筒', '可回收物_手表', '可回收物_打气筒', '可回收物_拉杆箱', '可回收物_插线板', '可回收物_收音机', '可回收物_放大镜', '可回收物_易拉罐', '可回收物_望远镜', '可回收物_木制切菜板', '可回收物_木质梳子', '可回收物_木质锅铲', '可回收物_枕头', '可回收物_桌子', '可回收物_水壶', '可回收物_水杯', '可回收物_泡沫板', '可回收物_灭火器', '可回收物_热水瓶', '可回收物_电动剃须刀', '可回收物_电子秤', '可回收物_电熨斗', '可回收物_电磁炉', '可回收物_电路板', '可回收物_电风扇', '可回收物_电饭煲', '可回收物_盘子', '可回收物_碗', '可回收物_箱子', '可回收物_纸板', '可回收物_衣架', '可回收物_袜子', '可回收物_裙子', '可回收物_裤子', '可回收物_计算器', '可回收物_订书机', '可回收物_话筒', '可回收物_路由器', '可回收物_轮胎', '可回收物_遥控器', '可回收物_钥匙', '可回收物_铁丝球', '可回收物_键盘', '可回收物_镊子', '可回收物_闹钟', '可回收物_雨伞', '可回收物_鞋', '可回收物_鼠标', '有害垃圾_太阳能电池', '有害垃圾_灯', '有害垃圾_电池', '有害垃圾_纽扣电池', '有害垃圾_胶水', '有害垃圾_药品包装', '有害垃圾_蓄电池']
460.             self.resize(1080, 800)
461.             self.cap = None
462.             self.timer = QTimer()
463.             self.timer.timeout.connect(self.update_frame)
464.             self.is_static_frame = False
465.             self.initUI()
466.
467.         def initUI(self):
468.             main_widget = QWidget()
469.             main_layout = QHBoxLayout()
470.             font = QFont('楷体', 18)
471.

```

```

472.         left_widget = QWidget()
473.         left_layout = QVBoxLayout()
474.         img_title = QLabel("识别样本")
475.         img_title.setFont(font)
476.         img_title.setAlignment(Qt.AlignCenter)
477.         self.img_label = QLabel()
478.         img_init = cv2.imread(self.to_predict_name)
479.         h, w, c = img_init.shape
480.         scale = 400 / h
481.         img_show = cv2.resize(img_init, (0, 0), fx=scale, fy=scale)
482.         cv2.imwrite("E:/test/garbage2/images_test/show.png", img_show)
483.         img_init = cv2.resize(img_init, (224, 224))
484.         cv2.imwrite('E:/test/garbage2/images_test/target.png', img_init)
485.         self.img_label.setPixmap(QPixmap("E:/test/garbage2/images_test/show.png"))
486.         left_layout.addWidget(img_title)
487.         left_layout.addWidget(self.img_label, 1, Qt.AlignCenter)
488.         left_widget.setLayout(left_layout)
489.
490.         right_widget = QWidget()
491.         right_layout = QVBoxLayout()
492.         btn_change = QPushButton("上传图片")
493.         btn_change.clicked.connect(self.change_img)
494.         btn_change.setFont(font)
495.         btn_predict = QPushButton("开始识别")
496.         btn_predict.setFont(font)
497.         btn_predict.clicked.connect(self.predict_img)
498.         btn_camera = QPushButton("打开摄像头")
499.         btn_camera.setFont(font)
500.         btn_camera.clicked.connect(self.toggle_camera)
501.         btn_screenshot = QPushButton("截屏")
502.         btn_screenshot.setFont(font)
503.         btn_screenshot.clicked.connect(self.capture_image)
504.
505.         label_result = QLabel('垃圾种类')
506.         self.result = QLabel("等待识别")
507.         label_result.setFont(QFont('楷体', 16))
508.         self.result.setFont(QFont('楷体', 24))
509.
510.         label_result_f = QLabel('物品名称')
511.         self.result_f = QLabel("等待识别")
512.         label_result_g = QLabel('概率大小')
513.         self.result_g = QLabel("等待识别")
514.
515.         self.label_info = QTextEdit()

```

```

516.         self.label_info.setFont(QFont('楷体', 12))
517.
518.         label_result_f.setFont(QFont('楷体', 16))
519.         self.result_f.setFont(QFont('楷体', 24))
520.
521.         label_result_g.setFont(QFont('楷体', 16))
522.         self.result_g.setFont(QFont('楷体', 20))
523.
524.         right_layout.addStretch()
525.         right_layout.addWidget(label_result, 0, Qt.AlignCenter)
526.         right_layout.addStretch()
527.         right_layout.addWidget(self.result, 0, Qt.AlignCenter)
528.         right_layout.addStretch()
529.         right_layout.addWidget(label_result_f, 0, Qt.AlignCenter)
530.         right_layout.addStretch()
531.         right_layout.addWidget(self.result_f, 0, Qt.AlignCenter)
532.         right_layout.addStretch()
533.         right_layout.addWidget(label_result_g, 0, Qt.AlignCenter)
534.         right_layout.addStretch()
535.         right_layout.addWidget(self.result_g, 0, Qt.AlignCenter)
536.         right_layout.addStretch()
537.         right_layout.addWidget(self.label_info, 1, Qt.AlignCenter)
538.         right_layout.addStretch()
539.         right_layout.addWidget(btn_change)
540.         right_layout.addWidget(btn_predict)
541.         right_layout.addWidget(btn_camera)
542.         right_layout.addWidget(btn_screenshot)
543.         right_layout.addStretch()
544.         right_widget.setLayout(right_layout)
545.
546.         main_layout.addWidget(left_widget)
547.         main_layout.addWidget(right_widget)
548.         main_widget.setLayout(main_layout)
549.         self.addTab(main_widget, '首页')
550.         self.setTabIcon(0, QIcon('E:/test/garbage2/images/首页.jpg'))
551.
552.         def change_img(self):
553.             openfile_name = QFileDialog.getOpenFileName(self, 'chose files', '', 'Image
files(*.jpg *.png *jpeg)')
554.             img_name = openfile_name[0]
555.             if img_name == '':
556.                 pass
557.             else:
558.                 target_image_name = "E:/test/garbage2/images_test/tmp_single" + img_name

```

```

.split(".")[ -1]
559.         shutil.copy(img_name, target_image_name)
560.         self.to_predict_name = target_image_name
561.         img_init = cv2.imread(self.to_predict_name)
562.         h, w, c = img_init.shape
563.         scale = 400 / h
564.         img_show = cv2.resize(img_init, (0, 0), fx=scale, fy=scale)
565.         cv2.imwrite("E:/test/garbage2/images_test/show.png", img_show)
566.         img_init = cv2.resize(img_init, (224, 224))
567.         cv2.imwrite('E:/test/garbage2/images_test/target.png', img_init)
568.         self.img_label.setPixmap(QPixmap("E:/test/garbage2/images_test/show.png"
569. ))
570.     def predict_img(self):
571.         img = Image.open('E:/test/garbage2/images_test/target.png')
572.         img = np.asarray(img)
573.         outputs = self.model.predict(img.reshape(1, 224, 224, 3))
574.         result1 = np.squeeze(outputs)
575.         result_index = int(np.argmax(outputs))
576.         Prob = result1[result_index]
577.         result = self.class_names[result_index]
578.         names = result.split("_")
579.         if names[0] == "厨余垃圾":
580.             self.label_info.setText(
581.                 "厨余垃圾包括剩菜剩饭、骨头、菜根菜叶、果皮等食品类废物。经生物技术就地处理堆肥，每吨可生产 0.6~0.7 吨有机肥料。")
582.         if names[0] == "有害垃圾":
583.             self.label_info.setText(
584.                 "有害垃圾含有对人体健康有害的重金属、有毒的物质或者对环境造成现实危害或者潜在危害的废弃物。包括电池、荧光灯管、灯泡、水银温度计、油漆桶、部分家电、过期药品及其容器、过期化妆品等。这些垃圾一般使用单独回收或填埋处理。")
585.         if names[0] == "可回收物":
586.             self.label_info.setText(
587.                 " 根据《城市生活垃圾分类及其评价标准》行业标准以及参考德国垃圾分类法，可回收物是指适宜回收循环使用和资源利用的废物。主要包括：纸类，塑料，金属，玻璃，织物等。主要的处理方式有：1.垃圾再生法；2.垃圾焚烧法；3.垃圾堆肥法；4.垃圾生物降解法。")
588.         if names[0] == "其他垃圾":
589.             self.label_info.setText(
590.                 "其他垃圾指危害比较小，没有再次利用的价值的垃圾，其他垃圾包括砖瓦陶瓷、渣土、卫生间废纸、瓷器碎片、动物排泄物、一次性用品等难以回收的废弃物。一般都采取填埋、焚烧、卫生分解等方法处理，部分还可以使用生物分解的方法解决")
591.         self.result.setText(names[0])
592.         self.result_f.setText(names[1])
593.         self.result_g.setText(str(Prob))

```

```

594.
595.     def toggle_camera(self):
596.         if self.cap is None:
597.             self.cap = cv2.VideoCapture(0)
598.             self.timer.start(300)
599.         else:
600.             self.timer.stop()
601.             self.cap.release()
602.             self.cap = None
603.             self.img_label.clear()
604.
605.     def update_frame(self):
606.         if not self.is_static_frame:
607.             ret, frame = self.cap.read()
608.             if ret:
609.                 rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
610.                 h, w, c = rgb_image.shape
611.                 scale = 400 / h
612.                 img_show = cv2.resize(rgb_image, (0, 0), fx=scale, fy=scale)
613.                 q_image = QImage(img_show.data, img_show.shape[1], img_show.shape[0]
, QImage.Format_RGB888)
614.                 self.img_label.setPixmap(QPixmap.fromImage(q_image))
615.
616.                 # Predict
617.                 img = cv2.resize(rgb_image, (224, 224))
618.                 img = np.asarray(img)
619.                 outputs = self.model.predict(img.reshape(1, 224, 224, 3))
620.                 result1 = np.squeeze(outputs)
621.                 result_index = int(np.argmax(outputs))
622.                 Prob = result1[result_index]
623.                 result = self.class_names[result_index]
624.                 names = result.split("_")
625.                 if names[0] == "厨余垃圾":
626.                     self.label_info.setText(
627.                         "厨余垃圾包括剩菜剩饭、骨头、菜根菜叶、果皮等食品类废物。经生物技术
就地处理堆肥，每吨可生产 0.6~0.7 吨有机肥料。")
628.                 if names[0] == "有害垃圾":
629.                     self.label_info.setText(
630.                         "有害垃圾含有对人体健康有害的重金属、有毒的物质或者对环境造成现实危
害或者潜在危害的废弃物。包括电池、荧光灯管、灯泡、水银温度计、油漆桶、部分家电、过期药品及其容器、过
期化妆品等。这些垃圾一般使用单独回收或填埋处理。")
631.                 if names[0] == "可回收物":
632.                     self.label_info.setText(
633.                         " 根据《城市生活垃圾分类及其评价标准》行业标准以及参考德国垃圾分类

```

法，可回收物是指适宜回收循环使用和资源利用的废物。主要包括：纸类，塑料，金属，玻璃，织物等。主要的处理方式有：1.垃圾再生法；2.垃圾焚烧法；3.垃圾堆肥法；4.垃圾生物降解法。")

```

634.         if names[0] == "其他垃圾":
635.             self.label_info.setText(
636.                 "其他垃圾指危害比较小，没有再次利用的价值的垃圾，其他垃圾包括砖瓦陶
                瓷、渣土、卫生间废纸、瓷器碎片、动物排泄物、一次性用品等难以回收的废弃物。一般都采取填埋、焚烧、卫生
                分解等方法处理，部分还可以使用生物分解的方法解决")
637.             self.result.setText(names[0])
638.             self.result_f.setText(names[1])
639.             self.result_g.setText(str(Prob))
640.
641.     def capture_image(self):
642.         if self.cap is not None and not self.is_static_frame:
643.             self.timer.stop()
644.             self.is_static_frame = True
645.
646.     def keyPressEvent(self, event):
647.         if event.key() == Qt.Key_Escape and self.is_static_frame:
648.             self.is_static_frame = False
649.             self.timer.start(30)
650.
651.     def closeEvent(self, event):
652.         reply = QMessageBox.question(self,
653.                                     '退出',
654.                                     "是否要退出程序?",
655.                                     QMessageBox.Yes | QMessageBox.No,
656.                                     QMessageBox.No)
657.         if reply == QMessageBox.Yes:
658.             if self.cap is not None:
659.                 self.cap.release()
660.                 self.close()
661.                 event.accept()
662.         else:
663.             event.ignore()
664.
665.
666. if __name__ == "__main__":
667.     app = QApplication(sys.argv)
668.     x = MainWindow()
669.     x.show()
670.     sys.exit(app.exec_())

```

附录 2

介绍：附件 2：爬虫关键词目录即代码中 source_refuse_name.txt 内容

其他垃圾_PE 塑料袋
 其他垃圾_一次性杯子
 其他垃圾_一次性棉签
 其他垃圾_口罩
 其他垃圾_唱片
 其他垃圾_干燥剂
 其他垃圾_打火机
 其他垃圾_搓澡巾
 其他垃圾_滚筒纸
 其他垃圾_点燃的香烟
 其他垃圾_眼镜
 其他垃圾_笔
 其他垃圾_胶带
 其他垃圾_验孕棒
 其他垃圾_鸡毛掸
 厨余垃圾_圣女果
 厨余垃圾_地瓜
 厨余垃圾_大蒜
 厨余垃圾_梨
 厨余垃圾_橙子
 厨余垃圾_火龙果
 厨余垃圾_瓜子壳
 厨余垃圾_番茄
 厨余垃圾_白菜叶
 厨余垃圾_苹果
 厨余垃圾_草莓
 厨余垃圾_菠萝
 厨余垃圾_菠萝蜜
 厨余垃圾_蘑菇
 厨余垃圾_蛋
 厨余垃圾_蛋挞
 厨余垃圾_蛋糕
 厨余垃圾_西瓜皮
 厨余垃圾_豆
 厨余垃圾_豆腐
 厨余垃圾_辣椒
 厨余垃圾_面包
 厨余垃圾_饼干
 厨余垃圾_香蕉皮
 可回收物_A4 纸
 可回收物_一次性筷子
 可回收物_不锈钢管

可回收物_乒乓球拍
 可回收物_书
 可回收物_保温杯
 可回收物_信封
 可回收物_充电头
 可回收物_充电宝
 可回收物_充电线
 可回收物_凳子
 可回收物_刀
 可回收物_剪刀
 可回收物_勺子叉子
 可回收物_包
 可回收物_卡
 可回收物_台灯
 可回收物_吹风机
 可回收物_呼啦圈
 可回收物_塑料瓶
 可回收物_塑料盆
 可回收物_头戴式耳机
 可回收物_尺子
 可回收物_布制品
 可回收物_帽子
 可回收物_手机
 可回收物_手电筒
 可回收物_手表
 可回收物_打气筒
 可回收物_拉杆箱
 可回收物_插线板
 可回收物_收音机
 可回收物_放大镜
 可回收物_易拉罐
 可回收物_望远镜
 可回收物_木制切菜板
 可回收物_木质梳子
 可回收物_木质锅铲
 可回收物_枕头
 可回收物_桌子
 可回收物_水壶
 可回收物_水杯
 可回收物_泡沫板
 可回收物_灭火器
 可回收物_热水瓶
 可回收物_电动剃须刀
 可回收物_电子秤

可回收物_电熨斗
可回收物_电磁炉
可回收物_电路板
可回收物_电风扇
可回收物_电饭煲
可回收物_盘子
可回收物_碗
可回收物_箱子
可回收物_纸板
可回收物_衣架
可回收物_袜子
可回收物_裙子
可回收物_裤子
可回收物_计算器
可回收物_订书机
可回收物_话筒
可回收物_路由器
可回收物_轮胎
可回收物_遥控器
可回收物_钥匙
可回收物_铁丝球
可回收物_键盘
可回收物_镊子
可回收物_闹钟
可回收物_雨伞
可回收物_鞋
可回收物_鼠标
有害垃圾_太阳能电池
有害垃圾_灯
有害垃圾_电池
有害垃圾_纽扣电池
有害垃圾_胶水
有害垃圾_药品包装
有害垃圾_蓄电池