第 3 章

图像分类（智能垃圾分拣器）

3.1 任务概述

3.1.1 任务背景

2020 年 11 月，住房和城乡建设部等 12 个部门联合发文，明确了到 2025 年 46 个重点城市要基本建立配套完善的生活垃圾分类法律法规制度体系，地级及以上城市建立生活垃圾分类投放、分类收集、分类运输、分类处理系统，这体现了国家对生活垃圾分类工作的高度重视。

面对每天庞大的生活垃圾，传统的人工分类方法耗时耗力、效率较低。为了解决这个问题，垃圾自动分类处理系统应运而生。通常垃圾分类处理系统由垃圾自动分拣机、可燃垃圾低温磁化炉、可腐烂有机垃圾资源化处理设备三部分组成，可以将城市小区、乡镇社区以及农村的生活垃圾进行分类，并在源头就地把分类后的垃圾资源化处理，起到节能减排、绿色环保的作用。如何设计出实用高效的智能垃圾自动分拣算法显得尤为重要。

按照垃圾分类方法，生活垃圾可以分为厨余垃圾、有害垃圾、可回收物和其他垃圾四大类，如图 3.1 所示。因四类垃圾在外观形态上有明显的不同，又考虑到摄像头成本低、无接触等优势，一般会采用基于图像的垃圾分类方法来构建垃圾分类系统，即通过摄像头捕获垃圾图像并进行自动分类。

基于图像的垃圾分类方法属于典型的图像分类应用，已有的方法主要分为两大类：基于传统特征描述的方法和基于深度学习的方法。基于传统特征描述的方法计算成本小，对硬件要求不高，在早期硬件资源受限条件下易于部署和应用。随着技术的发展以及海量数据的挖掘和利用，基于深度学习的垃圾分类方法逐渐成为主流。在图像样本充足的情况下，基于深度学习的方法识别率高、鲁棒性强，而伴随着各种轻量级深度学习模型的提出，在算法速度、硬件依赖、运算精度上深度学习分类方法均取得了显著进步，使得运算资源不再制约其应用。

本章内容将以垃圾分类任务为主线，使用图像分类套件 PaddleClas 实现高精度垃圾

图 3.1　生活垃圾分类

分类，并最终脱离训练环境，通过 FastDeploy 部署工具，在 Jetson Nano 智能边缘设备上实现算法集成。

3.1.2　安装 PaddleClas 套件

PaddleClas 是飞桨为工业界和学术界所研发的一个图像分类算法库，助力开发者能够快速训练出视觉分类模型并完成部署应用。PaddleClas 功能结构如图 3.2 所示，其中功能图底部是一系列前沿的图像分类算法，每种算法在模型大小、推理速度上均有不同的性能表现，一般根据业务场景选择相应的算法模型。在算法层之上，面向工业界还提供了一些产业级特色方案，这些方案所采用的模型或算法精度高、稳定性好、易部署。针对算法的训练和推理部署，PaddleClas 提供了完备、统一的实现接口，参照官网教程可以快速完成整个算法研发任务。在最顶端的应用场景方面，PaddleClas 提供了诸多真实有效的场景应用案例，如商品识别、车辆属性分析、电动车进电梯识别等，感兴趣的读者可以针对性地借鉴和学习这些案例，并应用到实际的项目中。

下面开始介绍如何安装 PaddleClas 套件。在安装 PaddleClas 套件前，请先确保已正确安装 PaddlePaddle，详细安装方法请参考 2.2 节的内容。

首先下载 PaddleClas 套件，可以使用 GitHub 源进行下载，如因网络原因无法下载，也可以尝试从国内 Gitee 源进行下载，下载网址详见前言二维码。

下载完成后通过 cd 命令切换到 PaddleClas 根目录，然后安装相关的依赖库：

```
pip install -r requirements.txt -i https://mirror.baidu.com/pypi/simple
```

图 3.2　图像分类套件 PaddleClas 功能结构图

最后安装 paddleclas：

```
python setup.py install
```

这样就完成了 PaddleClas 的安装。

3.2　算法原理

图像分类是计算机视觉的重要领域，它的目标是将图像分类到预定义的标签。近些年研究者提出很多不同种类的神经网络模型，极大地提升了图像分类算法的性能。下面着重介绍三个重要的图像分类网络模型及其基本原理。

3.2.1　VGG 算法

Alex 的成功指明了深度卷积神经网络可以取得出色的识别结果，但并没有提供相应的方案来指导后续的研究者如何设计新的网络来进一步提升性能。VGG 算法的提出证明了通过相同卷积模块的堆叠可以有效提升分类性能，这一理念也被延续至今。

VGG 是牛津大学的视觉几何小组（Visual Geometry Group）在 2014 年提出的一种神经网络模型，该模型证明了使用重复的卷积模块并且适当增加模型深度能够有效提高图像分类性能。VGG 有多种不同变体，如 VGG11、VGG13、VGG16 和 VGG19，这些变体本质上没有太大的区别，只是所使用的神经网络层数量不一样。

图 3.3 展示了 VGG16 对应的模型结构，该模型输入是 224×224×3 的 RGB 图像，然后经过一系列的卷积层 convolution、非线性激活层 ReLU 和最大池化层 max pooling

来提取高层语义特征，接下来对提取到的卷积特征使用 3 个全连接层 fully connected 和非线性激活层 ReLU 进行降维处理，映射到分类类别所对应的数量，最后使用 softmax 分类器完成分类。图 3.3 所示的下方列出了每层卷积的核参数和通道参数。

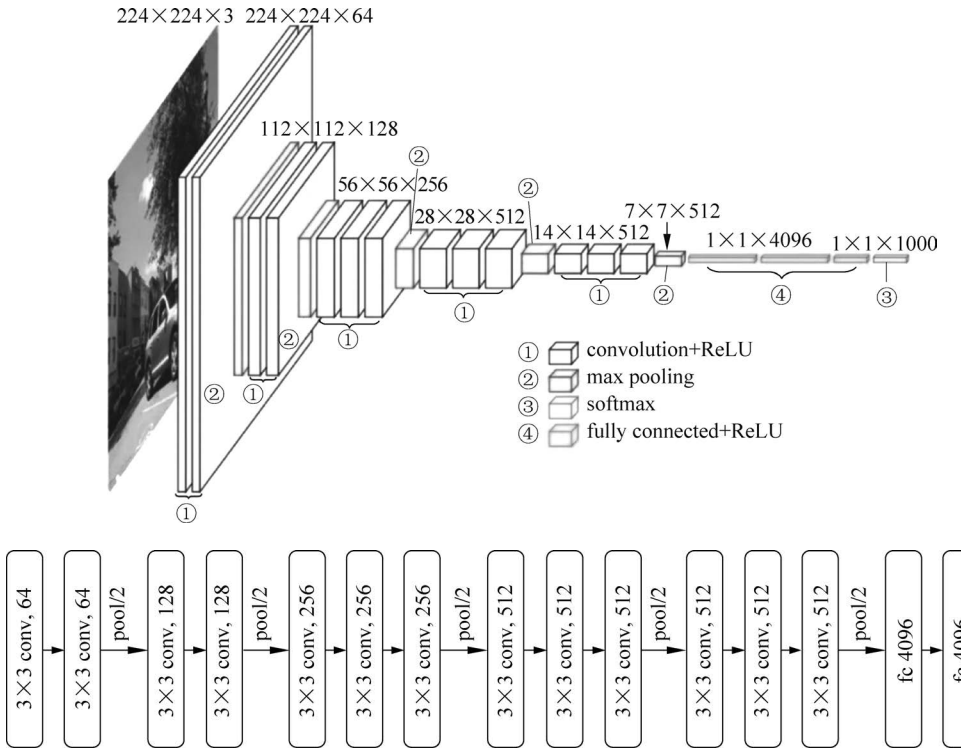


图 3.3　VGG 模型结构图

这里需要说明的是，VGG 所有的卷积层都采用 3×3 的卷积核（kernel_size），步长为 1（stride）、边缘填充为 1（padding），因此，按照 2.5.1 节中介绍的卷积计算公式，使用该类型卷积层不会改变输入特征尺寸。每个卷积模块后面的最大池化层采用的核大小为 2、步长为 2、边缘填充为 0，因此，使用该最大池化层计算后对应的特征图尺寸会变为原来的一半。

VGG 的完整代码请参考 PaddleClas/ppcls/arch/backbone/legendary_models/vgg.py，感兴趣的读者可以深入剖析该源码，提升对 VGG 模型的理解。

3.2.2　ResNet 算法

前面介绍的 VGG 模型证明了增加卷积层数可以提升模型性能。这里自然引申出来一个问题：神经网络堆叠得越深，学习的效果就一定会越好吗？答案无疑是否定的。研究学者发现当模型层数增加到某种程度，模型的效果将会不升反降。为此，研究学者给出了两种可能的解释：过拟合或梯度消失。但是进一步的研究表明，无论是过拟合还是梯度消失，都不能完美地解释这个现象。

目前，对于上述问题的一种直观解释就是神经网络的非线性表达能力让神经网络模

型太发散，一旦走得太远太深，它就“忘记”了初心，使得特征随着层层前向传播越来越发散。这个解释有一定的哲学意味，应该说深度学习目前还存在很多理论上的不足，有些情形只能通过实验和直觉分析来定位，而在理论分析方面深度学习还有很多需要探索的地方。那么到底怎么解决深度学习模型随着网络层数加深而产生的性能下降问题呢？解决方法就是残差学习，这也是 ResNet 算法最核心的思想。

在 ILSVRC2015 图像分类任务竞赛中，由何恺明等提出的深度残差模型 ResNet 首次超越人类水平，斩获竞赛第一名，同时基于 ResNet 所发表的论文也获得 2016 年 CVPR 最佳论文奖。这篇论文提出了一种大道至简的残差模块，成功解决了前面所述的深度学习模型退化问题。

残差模块结构如图 3.4 所示。

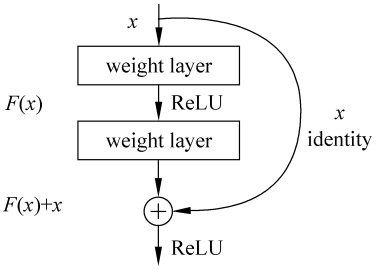


图 3.4　残差模块结构图

从图 3.4 看到，残差网络让输入 x 经过两个神经网络层产生输出 $F(x)$，然后从输入直接引入一个短连接线到输出上。这样一种网络结构可以用下面的公式表示：

$$y = F(x) + x$$

前面提到过，深度学习模型的非线性特性使得神经网络如果堆叠得太深会容易“忘记”初始的输入特征，因此残差模型在输出的地方额外地加入输入特征，构建这样一个恒等映射(identity)，从而让模型在探索求解空间时产生一个强约束，这个约束保证模型参数不会朝着一个过分发散的方向前进。这样的构建方式完全是一种直觉上的改进，为了验证它的有效性，何恺明等通过大量实验进行了证明。

有了上述残差模块，就可以进行堆叠，从而得到不同深度的深度残差模型 ResNet。常见的 ResNet 模型有 ResNet18、ResNet34、ResNet50、ResNet101、ResNet152 和 ResNet200 等。大量的实验证明，残差模块堆叠得越深性能表现越强。目前，ResNet 已作为最常见的深度学习模型被广泛应用。图 3.5 展示了 ResNet18 的模型结构，其中 k 表示卷积核大小，s 表示步长，p 表示边界扩充大小。

ResNet 的完整实现代码请参考 PaddleClas/ppcls/arch/backbone/legendary_models/resnet.py。感兴趣的读者可以深入剖析该源码，提升对 ResNet 模型的理解。

3.2.3　MobileNet 算法

目前的深度学习算法已经可以在 GPU 服务器上实时运行，但是如果将训练好的模型直接移植到手机或嵌入式终端，模型的推理速度和内存消耗就是非常致命的问题。因此，只有对深度学习模型进行优化才有可能在这类资源有限的设备中使用。

模型优化主要有三种方法：设计轻量级的网络、压缩剪枝和量化加速。本章重点关注如何设计轻量级神经网络模型。

在轻量级网络设计中，MobileNet 系列算法最为经典。从 MobileNet 的名字也可以看出，该系列算法旨在为移动设备进行 AI 赋能，其系列模型包括 MobileNetV1、MobileNetV2 和 MobileNetV3。通过使用 MobileNet 模型，可以大幅减少计算参数，降低模型推理时

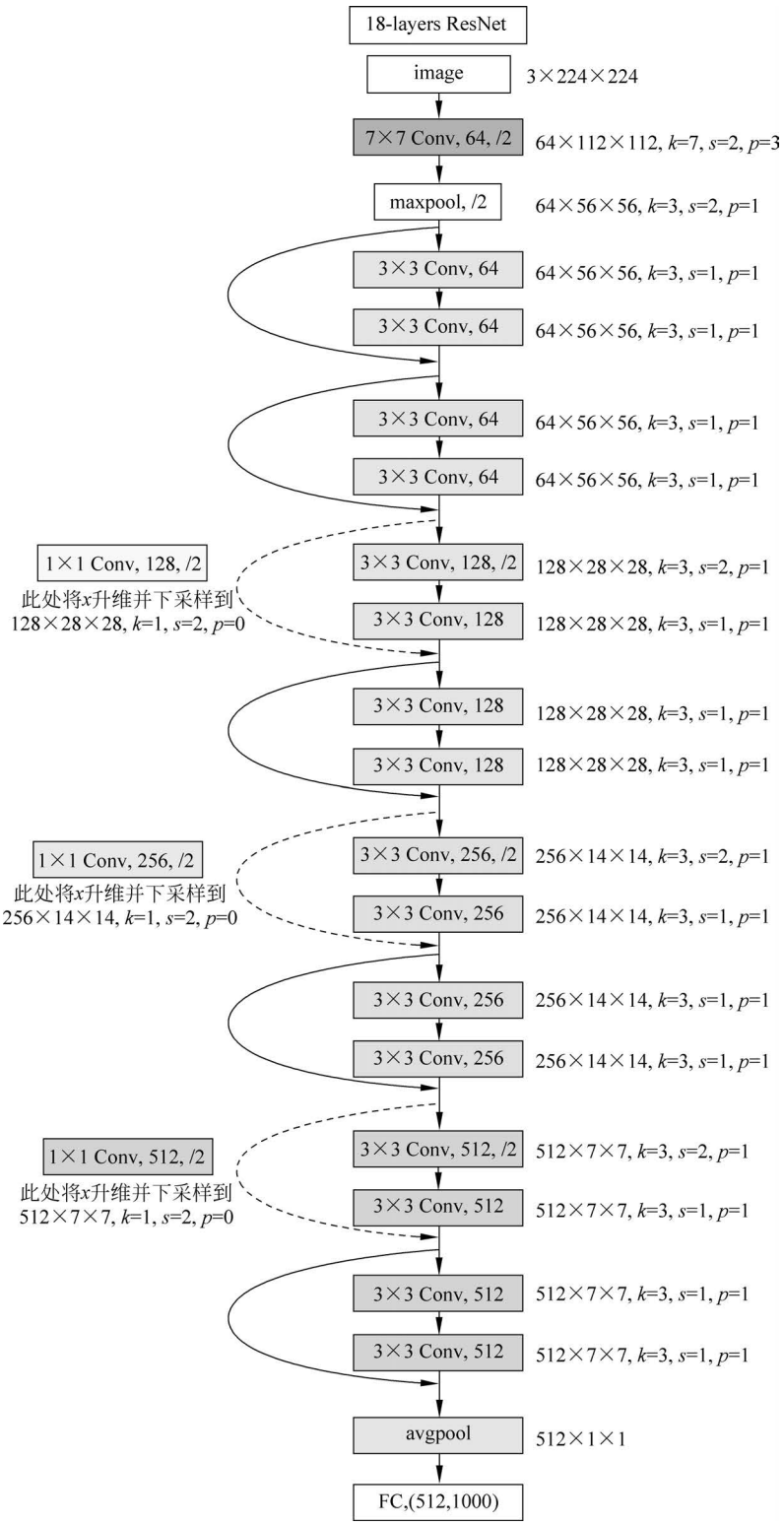


图 3.5　ResNet18 模型结构图

的内存消耗，这在实际工业场景研发过程中尤为重要。如果从产品部署角度考虑，目前深度学习的热潮已经逐步从服务器端转向小型终端，即所谓的边缘计算设备。众多企业纷纷在此发力，力求能够推出带 AI 功能的终端硬件产品，实现离线运行，保护客户的数据安全，其中以英伟达推出的 Jetson 系列开发板最为成功。Jetson 系列开发板不仅体积小巧，而且自带 GPU，因此一经推出便受到广泛关注。尽管有 GPU 加持，但是在这种资源有限的开发板上运行重量级的深度学习模型依然是一个难题，在速度上面也依然难以满足实时性要求。因此，对原模型进行优化使得能够利用低廉的终端设备实现 AI 应用已经成为 AI 工程师必须掌握的技能。本章工程实践部分将采用 MobileNet 算法在 Jetson 开发板上进行高性能深度学习推理。

下面首先讲解 MobileNetV1 和 MobileNetV2 的模型结构，了解 MobileNet 系列模型到底“轻”在何处。

1. MobileNetV1

传统的卷积神经网络在移动设备上运行速度慢且会消耗大量运算资源。因此，MobileNetV1 最大的贡献就是改进传统卷积神经网络的结构，降低卷积操作的计算量。

具体的，MobileNetV1 提出了深度可分离卷积（Depthwise Separable Convolution，DSC），以此来代替传统的二维卷积。假设输入图像是 3 通道图像，长、宽均为 256 像素，采用 5×5 卷积核进行卷积操作，输出通道数为 16（即 16 组卷积），步长为 1，边缘填充为 0，那么进行二维卷积后输出特征形状为 252×252×16，如图 3.6 所示。

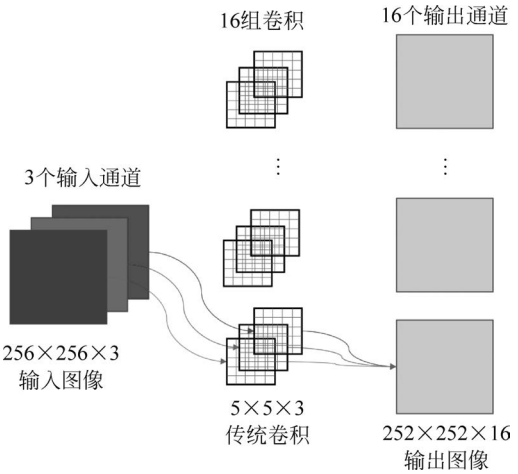


图 3.6　传统 CNN 卷积操作

图 3.6 中的 CNN 卷积使用 16 个不同的 5×5×3 的卷积核以滑窗的形式遍历输入图像，因此需要学习的参数个数为 5×5×3×16，实际对应的计算量为 5×5×3×16×256×256。可以看到，传统卷积计算量还是非常大的。MobileNetV1 的提出就是为了解决这个问题。

在 MobileNetV1 中，采用深度可分离卷积来代替传统的 CNN。深度可分离卷积可以分为两部分：深度卷积（Depthwise Convolution，DW）和逐点卷积（Pointwise Convolution，PW），如图 3.7 所示。

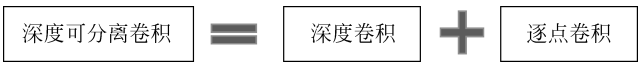


图 3.7　深度可分离卷积

深度卷积在处理特征图时对于特征图的每个通道有一个独立的卷积核，并且这个卷积核仅作用在这个通道上。例如，对于图 3.6 所示的 3 通道输入图像，如果采用深度卷积进行操作，那么就变成了使用 3 个卷积滤波器，每个卷积滤波器单独地作用于图像的某个通道上，每个卷积滤波器得到一个通道特征，最后合并产生 3 通道输出特征。从这个过程可以看出，使用深度卷积不会改变原始特征的通道数。图 3.8 展示了对于 3 通道图像的深度卷积操作步骤。

从计算量上来分析，因为只采用了 3 个 5×5 大小的卷积滤波器，因此对应的参数量为 5×5×3，计算量为 5×5×3×256×256。很明显，计算量降低了很多。那么这种深度卷积如何使用 PaddlePaddle 实现呢？

答案是非常简单的，只需要修改 paddle.nn.Conv2D()函数中的 groups 参数即可（paddle.nn.Conv2D 的详细定义请参考 2.5.1 节），这里 groups 的意思就是将输入通道数分成多少组进行卷积运算。当 groups 等于 1 时（默认值），等价于传统 CNN；当 groups 等于输入通道数时，此时就对应深度卷积计算。因此，如果要将传统 CNN 改为深度卷积，只需要设置这个 groups 参数为输入通道数即可。

采用深度卷积实现了每个通道的特征计算，但是这些计算是在单一的特征通道上完成的，此时各个通道之间的信息是独立的，那么如何对各通道特征进行融合并改变最终的输出通道数呢？这里就需要通过逐点卷积来完成。逐点卷积使用卷积核为 1×1 的常规 CNN 来实现。使用逐点卷积并不会改变输入特征长宽尺寸，仅改变输入特征通道数。从本质上来说，逐点卷积的作用就是对特征通道进行升维和降维，如图 3.9 所示。

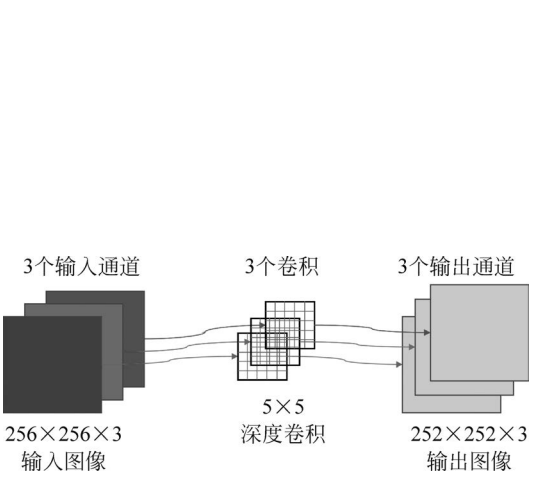


图 3.8　深度卷积操作示意图

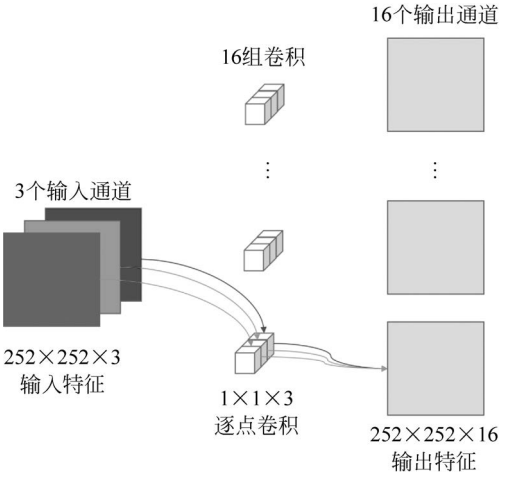


图 3.9　逐点卷积操作示意图

对应图 3.9，逐点卷积的参数量为 1×1×3×16，计算量为 1×1×3×252×252×16。

综合对比下，对于采用深度可分离卷积总的参数量为 5×5×3＋1×1×3×16，相比于普通卷积的 5×5×3×16，占(1/16＋1/25)。从计算量上来看，采用深度可分离卷积总

的计算量为 5×5×3×256×256＋1×1×3×252×252×16，相比于普通卷积的 5×5×3×16×256×256，同样占(1/16＋1/25)左右。如果采用的不是 5×5 卷积，而是常用的 3×3 卷积，那么使用深度可分离卷积只需要普通卷积 1/9 左右的计算量。

MobileNetV1 正是基于深度可分离卷积，实现了模型参数和计算量的减少。值得称赞的是，尽管参数量显著降低了，但是通过大量实验证明，MobileNetV1 算法在精度上并不会下降很多，这也是为何 MobileNetV1 获得研究学者如此青睐的原因。

完整的 MobileNetV1 结构如图 3.10 所示。

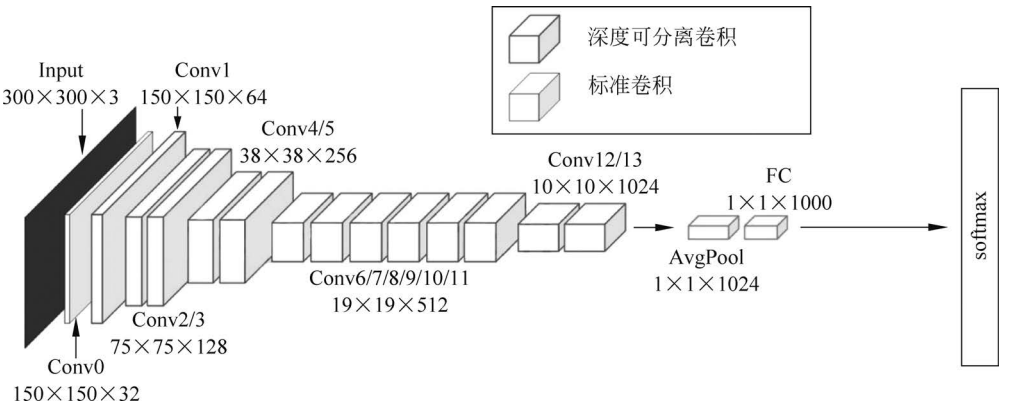


图 3.10　MobileNetV1 模型结构图

MobileNetV1 算法通过堆叠深度可分离卷积完成构建，完整代码请参考 PaddleClas 套件中的实现方案：PaddleClas/ppcls/arch/backbone/legendary_models/mobilenet_v1.py。

2. MobileNetV2

MobileNetV2 是由 Google 团队在 2018 年提出的，相比于 MobileNetV1 而言准确率更高，模型更小。

MobileNetV2 主要创新点就是在 MobileNetV1 中加入了残差模块，同时提出了一种新的激活函数 ReLU6。在 MobileNetV2 论文中指出，当输出特征通道数较少的时候，使用 ReLU 对其进行操作会导致信息严重损耗。为此，MobileNetV2 提出了 ReLU6 激活函数，其数学表达形式如下：

$$\mathrm{ReLU6}=\min(\max(0,x),6)$$

从上述公式可以看到，输入值 x 如果为 0～6，那么输出值不变，还是 x；当 x 超过 6 时输出值将被截断，恒等于 6；如果 x 小于 0，则输出恒等于 0。

ReLU6 对应的函数曲线如图 3.11 所示。

MobileNetV2 通过实验验证了 ReLU6 激活函数的有效性。实际在使用 PaddlePaddle 编程时，可以直接使用已有的接口实现，如下所示：

```
import paddle
import paddle.nn.functional as F
import numpy as np
x = paddle.to_tensor(np.array([-1, 0.3, 6.5]))
```

```
y = F.relu6(x)
print(y.numpy())
```

输出结果如下：

```
[0, 0.3, 6]
```

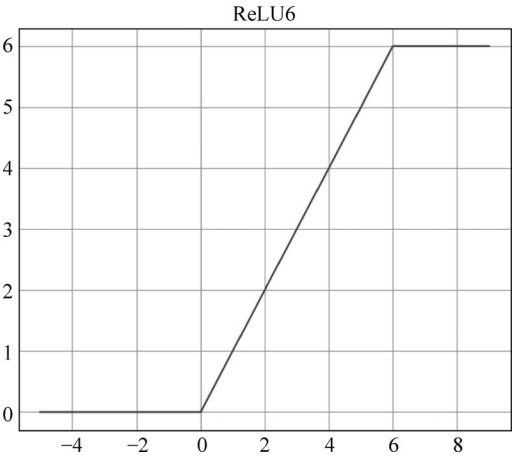


图 3.11　ReLU6 函数曲线

为了进一步提高分类性能，MobileNetV2 使用了前面介绍的残差模块，但是在设计整个模型结构时与 ResNet 算法有明显不同，MobileNetV2 设计了一种反向残差结构模型（Inverted Residuals）。ResNet 中的残差模块和 MobileNetV2 中的反向残差模型如图 3.12 所示。

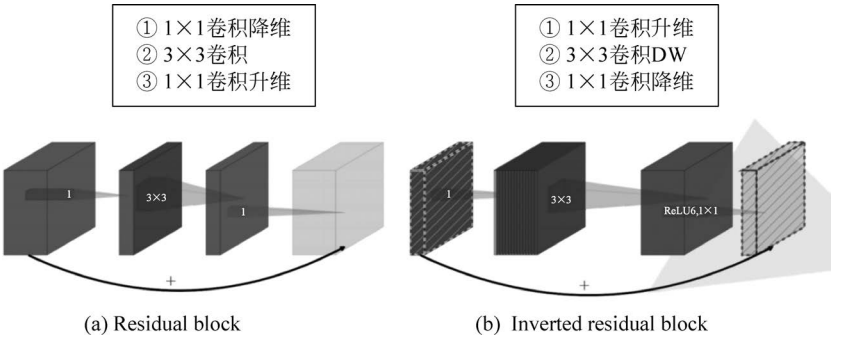


图 3.12　ResNet 中的残差模块和 MobileNetV2 中的反向残差模型

在 ResNet 提出的残差模块结构中，先使用 1×1 卷积实现降维，然后通过 3×3 卷积实现特征融合，最后通过 1×1 卷积实现升维，即两头大中间小。而 MobileNetV2 提出的反向残差模块，将降维和升维的顺序进行了调换，并且将 3×3 卷积替换成了 3×3 深度可分离卷积，即两头小中间大。这样的修改主要是考虑到深度卷积（DW）如果在低维度上工作，特征融合效果不会很好，所以 MobileNetV2 首先会扩张通道。通过前面可以知道逐点卷积（PW）所使用的 1×1 卷积可以用来升维和降维，那就可以在深度卷积（DW）

之前先使用逐点卷积(PW)进行升维(升维倍数为 t,论文中 $t=6$),然后再在一个更高维的空间中进行深度卷积(DW)操作来融合特征,最后再采用逐点卷积(PW)将通道数下降并还原回来。

MobileNetV2 中设计了两种反向模块,如图 3.13 所示。

图 3.13 中只有当步长 Stride=1 且输入特征矩阵与输出特征矩阵形状相同的时候才有残差连接,此时就对应反向残差模块。

MobileNetV2 完整结构如图 3.14 所示。

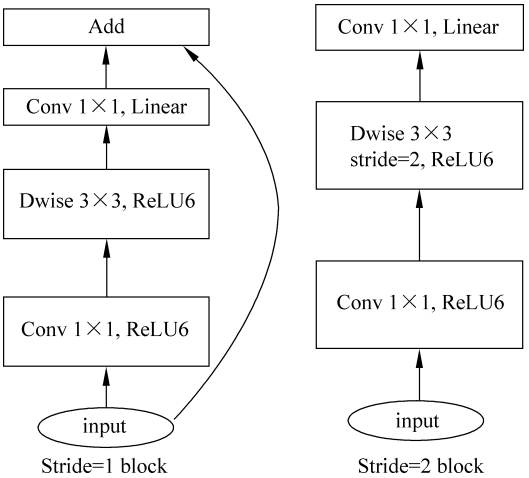


图 3.13　MobileNetV2 反向模块(bottleneck)结构图

Input	Operator	t	c	n	s
$224^2\times3$	conv2d	—	32	1	2
$112^2\times32$	bottleneck	1	16	1	1
$112^2\times16$	bottleneck	6	24	2	2
$56^2\times24$	bottleneck	6	32	3	2
$28^2\times32$	bottleneck	6	64	4	2
$14^2\times64$	bottleneck	6	96	3	1
$14^2\times96$	bottleneck	6	160	3	2
$7^2\times160$	bottleneck	6	320	1	1
$7^2\times320$	conv2d 1×1	—	1280	1	1
$7^2\times1280$	avgpool 7×7	—	—	1	—
1×1×1280	conv2d 1×1	—	k	—	

图 3.14　MobileNetV2 完整结构图

图 3.14 中,t 表示扩展因子(维度扩增倍数);c 表示输出特征通道数;n 表示 bottleneck 的模块重复次数;s 表示步长,bottleneck 表示反向模块。

完整的 MobileNetV2 代码请参考 PaddleClas 套件中的实现方案,代码位于 PaddleClas/ppcls/arch/backbone/model_zoo/mobilenet_v2.py。感兴趣的读者可以深入剖析该源码,提升对 MobileNetV2 模型的理解。

至此,已介绍完 VGG、ResNet、MobileNet 等常见的深度学习图像分类算法。VGG 通过适当堆叠卷积模块来提高分类精度;ResNet 通过恒等映射连接来解决网络层数过度加深后出现的性能退化问题;MobileNet 则在传统 CNN 卷积结构基础上提出了深度可分离卷积来减少计算量。深度学习分类算法近些年一直在快速更新中,读者只有掌握这些经典的图像分类算法原理,了解其背后所面临的核心问题,才能洞察本质,提高实践能力。

下面将以图像分类算法套件 PaddleClas 为主,研发一款智能垃圾分类器应用,同时结合英伟达推出的 Jetson Nano 智能开发板,打造一款实战级的图像分类产品。为了能够在性能有限的 Jetson Nano 开发板上运行深度学习算法,将使用轻量级 MobileNetV2 算法来完成这个任务。

3.3　算法研发

本节开始将采用前面安装好的 PaddleClas 图像分类套件进行项目研发。

3.3.1 数据集准备

经过近20年的发展，垃圾分类方法已经作了数次调整，目前一种典型的方法就是把垃圾分为四个类别：有害垃圾、厨余垃圾、其他垃圾和可回收物。有害垃圾典型的有废旧干电池、蓄电池等；厨余垃圾典型的有剩菜、果皮等；其他垃圾典型的有厕所废纸、香烟头等；可回收物典型的有易拉罐、牛奶盒等。本项目依据这个分类准则进行模型研发。

首先从本章配套资源中获取数据集garbage265.zip，下载网址详见前言二维码。下载后解压并放置在PaddleClas/dataset目录下。

该数据集包含近15万张常见的生活垃圾图像，覆盖食品、厨房用品、塑料制品等265个垃圾小类，其中训练集文件夹garbage265/train中包含132674张图像，验证集文件夹garbage265/val中包含14612张图像，数据集总大小接近13GB空间。

为了能够方便地使用PaddleClas套件进行算法研发，可以使用TXT格式文件来指定训练集和验证集的图片索引，在文件中每一行列出图片路径及其所属类别，如下所示：

```
val/262/4190939a8d26_520.jpg 3
val/57/fb4cfa85cf84_484.jpg 1
val/46/17ca888157ac_225.jpg 0
...
```

每一行图片路径后面的数字表示图片的类别序号。每一行采用空格来分隔图片路径与类别序号。一般使用train.txt表示训练集，val.txt表示验证集。在算法训练时，只需要设置好这两个文件，PaddleClas套件就会调用固定的接口去逐行读取这些数据。在2.5.3节中，也采用了类似的方法实现自动驾驶小车的数据采集和读取，只不过在第2章中每张图片对应的是一个转向角回归值，而本章对应的是所属垃圾类别序号。两个任务在本质上异曲同工。

本章将垃圾分类具体拆分为四个类别：厨余垃圾、可回收物、其他垃圾、有害垃圾。对应的，令这四个类别的类别索引分别为0、1、2、3。下面写一个数据处理脚本，根据训练集图片和验证集图片生成对应的train.txt和val.txt文件。

代码如下（PaddleClas/gen_list.py）：

```
import os
import random
import cv2

# 全局参数设置
dataset_folder = "./dataset/garbage265"          # 数据集路径

def writeLst(lstpath, namelst):
    """保存文件列表"""
    print("正在写入 " + lstpath)
    random.shuffle(namelst)                      # 打乱顺序写入
    f = open(lstpath, "a", encoding="utf-8")
    for i in range(len(namelst)):
```

```
            text = namelst[i] + "\n"
            f.write(text)
        f.close()
        print(lstpath + "已完成写入")

def get_label(foldername):
    """类标签映射"""
    label = int(foldername)
    if 0 <= label <= 51:                    # 厨余垃圾
        return "0"
    if 52 <= label <= 200:                  # 可回收物
        return "1"
    if 201 <= label <= 250:                 # 其他垃圾
        return "2"
    if 251 <= label <= 264:                 # 有害垃圾
        return "3"
    return "0"

def gen_list(mode = "train"):
    '''生成文件列表'''
    filelst = list()
    # 查找文件夹
    mode_folder = os.path.join(dataset_folder, mode)
    subfolderlst = os.listdir(mode_folder)
    for foldername in subfolderlst:
        subfolderpath = os.path.join(mode_folder, foldername)
        if os.path.isfile(subfolderpath):
            continue
        label = get_label(foldername)
        # 查找图像文件
        imglst = os.listdir(subfolderpath)
        for imgname in imglst:
            imgpath = os.path.join(subfolderpath, imgname)
            print("正在检查图像  " + imgpath)
            img = cv2.imread(imgpath, cv2.IMREAD_COLOR)
            if img is None:
                continue
            text = os.path.join(mode, foldername, imgname) + " " + label
            filelst.append(text)
    # 生成并写入文件
    filelst_path = os.path.join(dataset_folder, mode + ".txt")
    writeLst(filelst_path, filelst)

# 生成训练集和验证集列表
gen_list("train")
gen_list("val")
```

通过 cd 命令进入 PaddleClas 目录内，然后运行上述脚本：

```
python gen_list.py
```

运行结束后在 dataset/garbage265 目录下会生成项目所需的 train.txt 和 val.txt 文件。

到这里就完成了整个数据集的准备工作，训练集图片总数为 132674，验证集图片总数为 14612。

3.3.2 算法训练

1. 准备配置文件

使用 PaddleClas 算法套件可以让开发者使用统一的 Python 接口快速实现各种模型的训练和推理。PaddleClas 采用结构化的方式将分散的脚本代码抽象成固定的模块，各个模块之间的协调由后缀为 yaml 的配置文件衔接。因此，如果要切换模型或者调整参数，只需要修改配置文件即可。

PaddleClas 套件的相关配置文件存放在 PaddleClas/ppcls/configs 文件夹中，该文件夹里面区分了不同任务对应的配置文件，如用于轻量级物体识别任务的 PULC、用于车辆属性分析任务的 Vehicle、用于传统图像分类任务的 ImageNet、用于行人重识别任务的 reid 等，如图 3.15 所示。

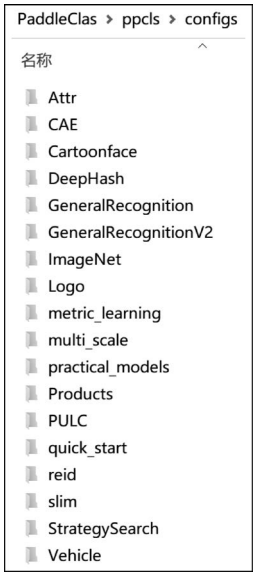


图 3.15 PaddleClas 相关任务配置文件夹

随着 PaddleClas 的不断迭代更新，上述文件夹也在不断扩充。由于 ImageNet 是比较权威的图像分类数据集，基本上大部分分类模型都会在这个数据集上进行验证和测试，因此，对于本章任务，读者可以参考 ImageNet 文件夹下对应的配置文件。

具体的，仿照 PaddleClas/ppcls/configs/ImageNet/MobileNetV2 文件夹中的 MobileNetV2.yaml 文件，对应地在 PaddleClas 目录下创建一个 config.yaml 文件，其完整内容如下（PaddleClas/config.yaml）：

```
################ 全局配置 ################
Global:
  checkpoints: null                                  # 断点模型路径
  pretrained_model: null                             # 预训练模型路径
  output_dir: ./output/                              # 训练结果保存目录
  device: gpu                                        # 采用什么设备训练模型：cpu 或 gpu
  save_interval: 1                                   # 模型保存间隔
  eval_during_train: True                            # 是否边训练边评估
  eval_interval: 1                                   # 评估间隔
  epochs: 800                                        # 训练总轮数
  print_batch_step: 10                               # 打印间隔
  use_visualdl: True                                 # 是否开启可视化工具 visualdl
  image_shape: [3, 224, 224]                         # 静态图模型导出尺寸
  save_inference_dir: ./output/inference             # 静态图模型保存路径
  to_static: False                                   # 是否采用静态图模式训练模型
################ 模型结构配置 ################
Arch:
  name: MobileNetV2                                  # 模型名称
```

```
  class_num: 4                                    # 类别数
################### 损失函数配置 ################
Loss:
  Train:
    - CELoss:                                     # 交叉熵损失
        weight: 1.0
  Eval:
    - CELoss:
        weight: 1.0
################### 优化器设置 ################
Optimizer:
  name: Momentum                                  # 动量优化算法(属于梯度下降)
  momentum: 0.9
  lr:                                             # 学习率设置
    name: Cosine
    learning_rate: 0.045                          # 学习率大小,调整时与 GPU 卡数成正比
  regularizer:                                    # 正则化(防止过拟合)
    name: 'L2'
    coeff: 0.00004
################### 数据读取管道 ################
DataLoader:
  # 训练
  Train:
    dataset:
      name: ImageNetDataset                             # 数据集类型
      image_root: ./dataset/garbage265/                 # 数据集根路径
      cls_label_path: ./dataset/garbage265/train.txt    # 训练集列表文件路径
      # 数据预处理
      transform_ops:
        - DecodeImage:
            to_rgb: True                              # 转换到 RGB 空间
            channel_first: False
        - RandCropImage:                              # 随机中心化裁剪
            size: 224
        - RandFlipImage:                              # 随机水平翻转
            flip_code: 1
        - NormalizeImage:                             # 归一化: (x * scale - mean)/std
            scale: 1.0/255.0
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''
    # 数据读取
    sampler:
      name: DistributedBatchSampler                   # 分布式读取
      batch_size: 32                                  # 单次读取数量
      drop_last: False        # 图片总数不能被 batch_size 整除时是否丢弃最后剩余的样本
      shuffle: True                                   # 采集图片时是否将数据打乱顺序
    # 数据加载
    loader:
      num_workers: 2                                  # 是否开启多个进程来加载
```

```
    use_shared_memory: True                          # 是否共享内存
  # 评估
  Eval:
    dataset:
      name: ImageNetDataset                          # 数据集类型
      image_root: ./dataset/garbage265/              # 数据集根路径
      cls_label_path: ./dataset/garbage265/val.txt   # 验证集列表文件路径
      # 图像预处理(需要与图像训练时设置的参数基本保持一致)
      transform_ops:
        - DecodeImage:
            to_rgb: True                             # 是否转换为 RGB 空间
            channel_first: False                     # 是否调整通道
        - ResizeImage:                               # 调整图像大小
            resize_short: 256                        # 短边对齐到 256,长边等比缩放
        - CropImage:                                 # 中心化裁剪
            size: 224
        - NormalizeImage:                            # 归一化: (x * scale - mean)/std
            scale: 1.0/255.0
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''
    # 数据读取
    sampler:
      name: DistributedBatchSampler                  # 分布式读取
      batch_size: 32                                 # 单次读取图像数量
      drop_last: False        # 图片总数不能被 batch_size 整除时是否丢弃最后剩余的样本
      shuffle: False                                 # 采集图片时是否将数据打乱顺序
    loader:
      num_workers: 2                                 # 是否开启多个进程来加载
      use_shared_memory: True                        # 是否共享内存
##################### 推理测试 ###############
Infer:
  infer_imgs: dataset/garbage265/val/0/1be1245fa11c_1019.jpg   # 测试图片路径
  batch_size: 1                                      # 每批次读取的图片数量
  # 图像预处理(需要与图像训练时设置的参数基本保持一致)
  transforms:
    - DecodeImage:
        to_rgb: True                                 # 转换到 RGB 空间
        channel_first: False                         # 是否调整通道
    - ResizeImage:                                   # 调整图像大小
        resize_short: 256                            # 短边对齐到 256,长边等比缩放
    - CropImage:                                     # 中心化裁剪
        size: 224
    - NormalizeImage:                                # 归一化: (x * scale - mean)/std
        scale: 1.0/255.0
        mean: [0.485, 0.456, 0.406]
        std: [0.229, 0.224, 0.225]
        order: ''
    - ToCHWImage:
  PostProcess:                                       # 后处理
    name: Topk                                       # 采用 TopK 准确率来评测结果
```

```
    topk: 1
################## 评估指标 ###############
Metric:
  Train:
    - TopkAcc:        # TopK 准确率：预测结果中概率最大的前 K 个结果包含正确标签的占比
        topk: [1, ]
  Eval:
    - TopkAcc:
        topk: [1, ]
```

本书针对上述配置给出了相关参数注释。对于一般的图像分类任务，大部分参数都可以沿用官方套件给出的配置文件进行修改。

PaddleClas 通过读取配置文件中的参数来衔接各个模块。首先通过 cd 命令切换到 PaddleClas 文件夹下面，然后按照前面的方式创建好配置文件，下面就可以使用 PaddleClas 的统一接口实现模型训练、验证和推理。

2. 训练

如果是单 GPU 的服务器，那么可以使用下面的命令来启动训练：

```
python tools/train.py -c config.yaml
```

如果使用多 GPU 服务器，为了最大化利用多卡训练优势、加快训练速度，可以使用分布式方式来训练。假设服务器有 2 个 GPU，可以使用下面的命令实现分布式训练：

```
export CUDA_VISIBLE_DEVICES=0,1
python -m paddle.distributed.launch tools/train.py -c config.yaml
```

最终的模型输出结果会保存到 output/MobileNetV2 文件夹下面。训练结束后会自动保存所有训练过程中性能最佳的模型文件 best_model（包含 3 个文件），如图 3.16 所示。

best_model.pdopt	2023/12/10 21:30	PDOPT 文件	8,730 KB
best_model.pdparams	2023/12/10 21:30	PDPARAMS 文件	8,875 KB
best_model.pdstates	2023/12/10 21:30	PDSTATES 文件	1 KB

图 3.16　训练后的动态图模型文件

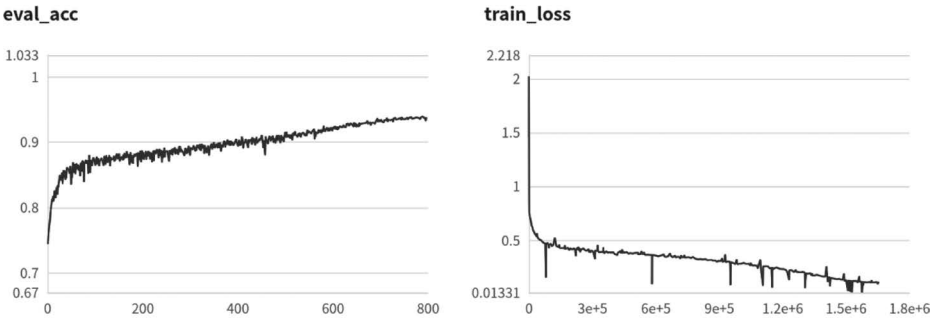
另外，在 YAML 配置文件中设置了 use_visualdl：True，即在训练过程中使用 visualdl 可视化工具保存训练结果。visualdl 是 PaddlePaddle 提供的非常强大的深度学习可视化工具，方便在训练时记录中间结果，并且可以以图形化方式展示。训练结束后在 output/vdl 目录下会自动保存对应的可视化训练结果。

可以使用下面的命令来启动 visualdl 查看训练过程（也可以在训练过程中开启 visualdl 实时查看）：

```
visualdl --logdir output/vdl
```

正常开启 visualdl 后可以使用浏览器输入网址 http://127.0.0.1:8040/进行查看，

效果如图 3.17 所示。



(a) 验证集准确率　　(b) 训练集损失

图 3.17　使用 visualdl 查看训练结果

可以看到，随着不断迭代训练，在验证集上的准确率越来越高，最佳 Top1 准确率达到 0.93909，训练集损失函数也越来越小。

从精度上可以看出，本章所选择的 MobileNetV2 模型基本能够满足任务需求。读者如果想进一步提高精度，可以尝试延长训练的迭代次数或者更改学习率等参数。整个训练过程大概需要 2 天时间，读者可以自行训练也可以直接使用本书配套资源中训练好的模型进行接下来的学习，下载网址详见前言二维码。

3. 推理测试

下面针对实际的任意单张图片进行测试，测试图片如图 3.18 所示。config.yaml 文件的 infer_imgs 参数设置了对应的测试图片路径，可以通过下面的命令使用训练好的动态图模型对测试图片进行推理：

```
python tools/infer.py -c config.yaml \
    -o Global.pretrained_model=output/MobileNetV2/best_model
```

图 3.18　测试图片

输出结果如下：

```
[{'class_ids': [0], 'scores': [0.99946], 'file_name': 'dataset/garbage265/val/0/
1be1245fa11c_1019.jpg', 'label_names': []}]
```

其中，class_ids 对应类别序号；scores 对应置信度；file_name 对应图片路径。这里的类

别序号输出为0，参考3.3.1节数据集准备部分，0代表的是厨余垃圾。

训练好的AI模型最后输出的并不仅仅是某个类别对应的类别序号，也会包含所属类别的概率。例如上述输出scores为0.99946，说明该模型以99.946%的概率确定这张图片类别序号为0，对应厨余垃圾。在实际项目中，往往需要根据经验过滤掉scores过低的预测结果。读者可以按照上述代码自行修改路径预测其他图片。

到这里就完成了算法的研发工作。可以看到，通过PaddleClas套件的使用，整个研发过程是非常简单便捷的。本章项目使用的分类模型为MobileNetV2，如果想要尝试其他模型，只需要修改对应的YAML配置文件参数即可。

读者如果感兴趣也可以采用debug单步调试的方法，跟踪相应的Python脚本，逐行分析每个模块的运行机制，进一步学习PaddleClas套件的工程化实现方法。

4. 动态图转静态图

深度学习框架分为静态图框架和动态图框架，早期的TensorFlow、Caffe和PaddlePaddle都是静态图框架，而PyTorch则是典型动态图框架。随着PyTorch动态图框架在深度学习领域逐渐流行起来，其他框架也相继进行了大的版本改进，纷纷转变为动态图框架，其中就包括PaddlePaddle。从2020年开始，PaddlePaddle从版本1系列过渡到2系列，正式全面迈入动态图框架行列。

静态图和动态图最大的区别就是他们采用不同的计算图表现形式。静态图框架需要先定义计算图，然后再调用它，而动态图则会根据当前输入情况动态生成计算图。

一般来说，在模型开发阶段，推荐采用动态图编程，这样可获得更好的编程体验、更易用的接口、更友好的调试交互机制。而在模型部署阶段，可将动态图模型转换为静态图模型，并在底层使用静态图执行器运行，这样可获得更好的模型运行性能。前面使用了PaddlePaddle的动态图框架完成了模型的训练，接下来，为了能够在Jetson Nano开发板上实现高性能推理，需要先将该模型转换为静态图模型。

PaddlePaddle的各个算法套件都提供动态图转静态图的高级命令接口，可以使用下面的命令将前面训练好的动态图模型转换为静态图模型。

```
python tools/export_model.py \
    -c config.yaml \
    -o Global.pretrained_model=output/MobileNetV2/best_model \
    -o Global.save_inference_dir=output/inference
```

运行完上述命令后，在output/inference文件夹下会生成转换完的静态图模型，具体包含3个文件。

- inference.pdiparams：参数文件。
- inference.pdiparams.info：参数信息文件。
- inference.pdmodel：模型结构文件。

这里注意，对于图像分类任务来说，为了后续能够使用FastDeploy工具部署PaddleClas套件训练出来的模型，还需要额外提供一个名为inference_cls.yaml的配置文件，用于提供预处理的相关信息。在旧版本的PaddleClas套件中，使用静态图导出命令后会自动生成用于部署的inference_cls.yaml文件，但是最新版本的PaddleClas并没有提供这个功

能。为了方便后面部署，需要手动生成这个文件。

具体的，在 PaddleClas/deploy/configs 文件夹中提供了一个 inference_cls.yaml 模板，只需要根据前面配置的 config.yaml 文件内容，针对性地修改这个 inference_cls.yaml 文件即可。

对于本章任务，代码如下（output/inference/inference_cls.yaml）：

```
PreProcess:
  transform_ops:
    - ResizeImage:                          # 短边对齐至 256,长边等比缩放
        resize_short: 256
    - CropImage:                            # 中心化裁剪
        size: 224
    - NormalizeImage:                       # 归一化
        scale: 0.003921
        mean: [0.485, 0.456, 0.406]
        std: [0.229, 0.224, 0.225]
        order: ""
        channel_num: 3
    - ToCHWImage:
```

修改好以后，将上述 inference_cls.yaml 文件放置在 output/inference 文件夹下面，与其他 3 个转换好的静态图文件同目录。

下面就可以使用飞桨的高性能部署工具 FastDeploy 进行推理了。通过 FastDeploy 工具，可以脱离繁重的 PaddlePaddle 环境依赖，甚至可以脱离 Python 语言本身，能够使用 C++等高性能编程语言完成深度学习推理，这对于工业级产品开发来说尤为重要。

下面将分别介绍使用 FastDeploy 在 Jetson Nano 开发板上完成 Python 和 C++推理。


视频讲解

3.4 Jetson Nano 智能终端部署（Linux GPU 推理）

3.3 节内容完成了算法的训练和推理实践，本节开始将详细阐述如何将训练好的模型一步步地部署到真实的智能终端设备上，实现完整的项目闭环，打造出一款实用的智能垃圾分拣器。具体的，本章将使用 Jetson Nano 开发板完成部署。

Jetson Nano 是一款体积小巧、功能强大的 64 位 ARM 开发板，于 2019 年 3 月由英伟达推出，预装 Ubuntu 18.04LTS 系统，搭载英伟达研发的 128 核 Maxwell GPU，可以快速将 AI 技术落地并应用于多种智能化场景。由于 Jetson Nano 售价低、性能强、生态完善，一经推出，便受到了广泛的关注。Jetson Nano 产品外观如图 3.19 所示。

相比其他终端硬件，使用 Jetson Nano 有一个特别的优势，在英伟达 GPU 服务器上所研发的算法模型几乎不用做适配性调整，就可以直接迁移到 Jetson Nano 上。

一般的，会采用带有英伟达显卡的服务器并且使用 CUDA 库来进行算法训练，训练完的模型如果想要迁移到其他智能终端硬件上往往需要做模型的转换，但这些模型转换工作并不是一件容易完成的事。如果使用 Jetson Nano，几乎可以不用考虑因硬件环境所造成的迁移成本。英伟达团队打通了 NVIDIA 显卡和 Jetson 终端产品的依赖库一致

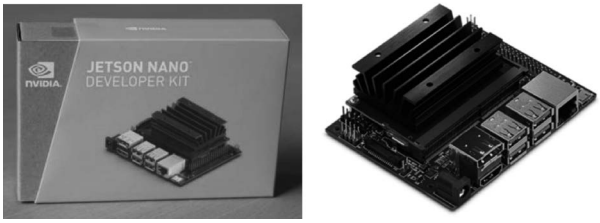


图 3.19　Jetson Nano 产品外观图

性，顶层接口做了统一的封装，在 Jetson Nano 上通过 JetPack 打包好 CUDA、CUDNN、TensorRT 等库环境，对于应用层的用户来说，其使用体验是高度一致的。

由于篇幅原因，本书不再深入阐述 Jetson Nano 这款产品的基本安装和使用方法，读者如果不熟悉 Jetson Nano，可以先参考线上教程进行学习，学习教程网址详见前言二维码。

本项目采用 Jetson Nano 来部署前面训练好的垃圾分类模型，采用 USB 摄像头实时捕获待分类的垃圾图像，然后将图像交给 Jetson Nano 上的 GPU 进行推理，最后将推理结果展示在屏幕上。整个执行流程如图 3.20 所示。

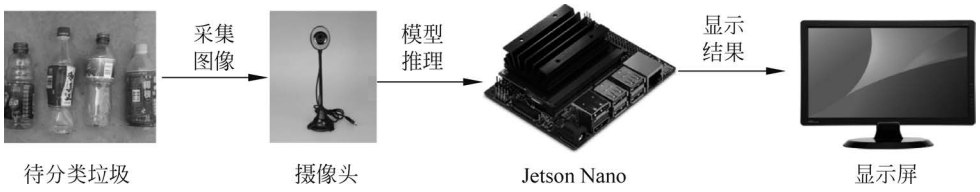


图 3.20　智能垃圾分拣器执行流程图

那么究竟如何将训练好的模型部署到 Jetson Nano 上呢？这里将采用 FastDeploy 工具来实现。

3.4.1　部署工具 FastDeploy 介绍

FastDeploy 是飞桨团队开源的一款全场景、灵活易用的 AI 部署套件，提供了开箱即用的端、边、云部署方案，官网网址详见前言二维码。

FastDeploy 对多个飞桨基础部署工具进行了整合，屏蔽了复杂的前后处理逻辑，针对各种硬件平台封装了统一的调用接口，可以快速完成各个硬件平台上的模型部署任务。目前，FastDeploy 已支持众多文本、视觉、语音和跨模态模型部署任务，典型的有图像分类、物体检测、图像分割、人脸检测、人脸识别、关键点检测、抠图、文字识别、自然语言处理、文图生成等，可以满足多场景、多硬件、多平台的产业部署需求。

FastDeploy 完整功能架构如 3.21 所示。

目前，FastDeploy 部署套件正在高速迭代中，将逐步支持越来越多的产业级模型以及产业级硬件的部署。对于大部分使用飞桨套件训练出来的模型，基本可以无缝、快速地使用 FastDeploy 完成落地部署，这极大地简化了繁杂的部署工作。对于本章任务，通过使用 FastDeploy，只需要几个简单的步骤就可以把训练好的垃圾分类模型部署到智能终端硬件设备 Jetson Nano 中。

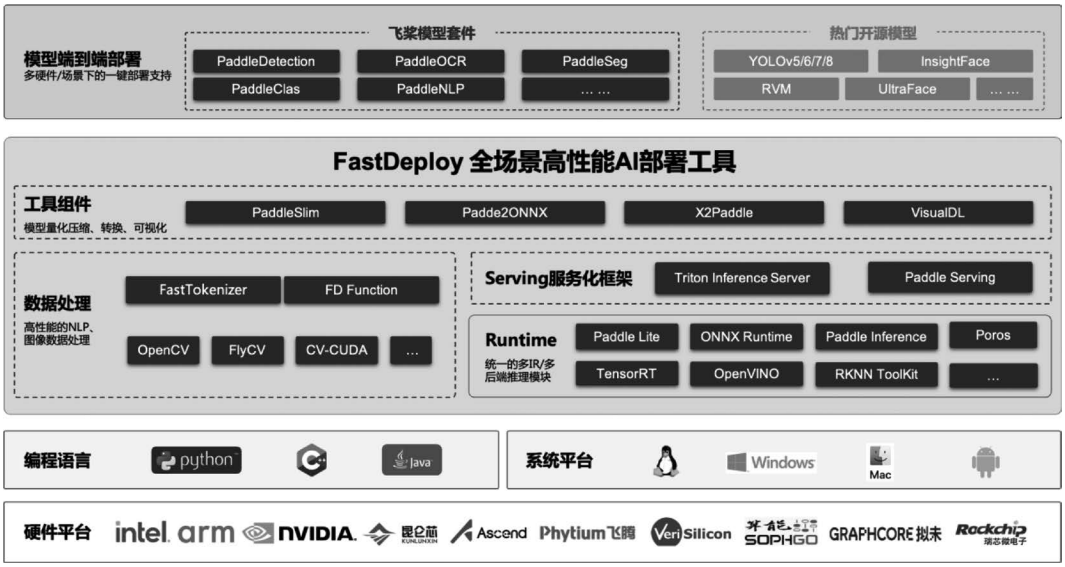


图 3.21　FastDeploy 完整功能架构

3.4.2　Jetson Nano 上 Python 推理

1. 在 Jetson Nano 上编译 FastDeploy 的 Python 预测库

为了能够在 Jetson Nano 上使用 FastDeploy 进行快速部署，需要自行编译 FastDeploy 的 Python 预测库。本书配套资料包中提供了配套的编译好的库（适配 jetpack 4.6.1），但是建议读者还是要掌握该编译方法，这样未来就可以自行编译最新的 FastDeploy 版本。

目前 Jetson Nano 支持 jetpack 4.6.1 及以上版本，因此，如果 Jetson Nano 的 jetpack 版本较低，请从英伟达官网重新下载镜像并安装。

编译前请先在 Jetson Nano 上更新软件列表：

```
sudo apt-get update
sudo apt-get upgrade
```

如果 Python 环境不完整，可以提前安装：

```
sudo apt-get install python-dev python3-dev
pip3 install --upgrade pip
```

从 FastDeploy 的 GitHub 官网拉取完整代码：

```
git clone https://github.com/PaddlePaddle/FastDeploy.git
```

接下来执行下述命令安装两个必要的依赖库：

```
pip3 install wheel tqdm -i https://mirror.baidu.com/pypi/simple
```

安装完以后开始编译，具体编译方法如下：

```
cd FastDeploy/python                                    # 进入目录
export PATH = /usr/local/cuda/bin/: $ PATH              # 设置 CUDA 路径
export BUILD_ON_JETSON = ON                             # 设置编译硬件为 Jetson
export ENABLE_VISION = ON                               # 设置编译视觉模块
export WITH_GPU = ON                                    # 设置支持 GPU 模式
export ENABLE_TRT_BACKEND = ON                          # 设置支持 TensorRT 后端
python3 setup.py build                                  # 开始编译，需要等待较长耗时
python3 setup.py bdist_wheel                            # 生成 whl 文件
```

编译过程中，如若修改编译参数，为避免缓存带来的影响，可删除 FastDeploy/python 目录下的 build 和. setuptools-cmake-build 两个子目录后再重新编译。编译完成后，在 FastDeploy/python/dist 文件夹下面会生成对应的 Python 预测库，如图 3.22 所示。

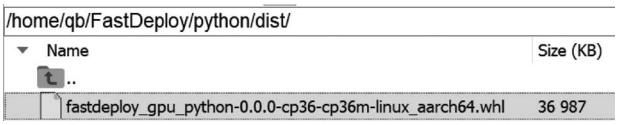


图 3.22　编译好的 Python 版 FastDeploy 预测库

编译好的 Python 版 FastDeploy 预测库本质上就是一个 whl 安装文件，可以使用 pip 工具离线安装。具体的，切换到生成的 dist 目录下，然后使用下面的命令安装：

```
pip3 install cython - i https://mirror.baidu.com/pypi/simple
pip3 install ./fastdeploy_gpu_python - 0.0.0 - cp36 - cp36m - linux_aarch64.whl \
   - i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装完成后就可以在 Jetson Nano 上使用 FastDeploy 了。进入 Python 环境，然后导入该包，如果没有任何输出，则表示安装成功。

```
python3
import fastdeploy as fd
```

2. 捕捉摄像头图像实时分类

本小节将使用 USB 摄像头实时捕获图像并使用深度学习算法进行垃圾图像分类，最终将分类结果输出展示。编写代码前确保 Jetson Nano 上已经准确连接 USB 摄像头，并且确保摄像头拍摄区域内背景是纯色干净的。

在 Jetson Nano 上新建一个推理文件夹 python，在该文件夹内新建推理脚本 infer. py，然后将前面转换好的静态图模型文件夹 inference 复制到 python 目录下面。

完整代码如下(deploy/python/infer. py)：

```
import cv2
import fastdeploy as fd
# 创建摄像头
cap = cv2.VideoCapture(0)
# 设置采集分辨率
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
```

```
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
# 创建采集视窗
window_handle = cv2.namedWindow("USB Camera", cv2.WINDOW_AUTOSIZE)
# 配置算法模型
option = fd.RuntimeOption()
option.use_gpu()                                          # 使用 GPU 推理
option.use_trt_backend()                                  # 使用 TensorRT 进行加速
option.trt_option.serialize_file = './tensorrt_cache' # 设置 TensorRT 缓存文件路径
# 创建分类器
model_file = "./inference/inference.pdmodel"
params_file = "./inference/inference.pdiparams"
config_file = "./inference/inference_cls.yaml"
model = fd.vision.classification.PaddleClasModel(model_file,
                                                  params_file,
                                                  config_file,
                                                  runtime_option=option)
# 逐帧分析
while cv2.getWindowProperty("USB Camera", 0) >= 0:
    # 捕获图片
    _, img = cap.read()
    # 垃圾分类预测
    result = model.predict(img.copy(), topk=1)
    # 解析分类结果
    score = result.scores[0]
    label = result.label_ids[0]
    text = ''
    if score < 0.6:                                       # 小于阈值认为没有任何物品
        print('请摆放垃圾到分类台上')
    else:
        if label == 0:
            print('厨余垃圾')
        elif label == 1:
            print('可回收物')
        elif label == 2:
            print('其他垃圾')
        elif label == 3:
            print('有害垃圾')
    # 显示图像
    cv2.imshow("USB Camera", img)
    keyCode = cv2.waitKey(30) & 0xFF
    if keyCode == 27:                                     # 按 Esc 键退出
        break
#释放资源
cap.release()
cv2.destroyAllWindows()
```

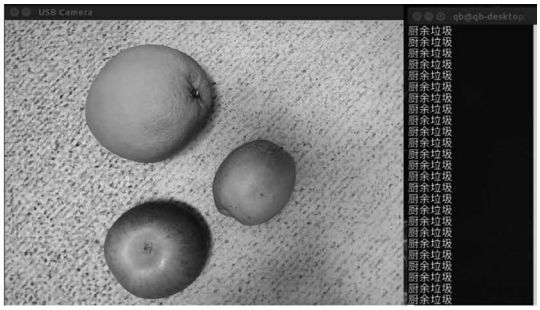
上述代码使用 Jetson Nano 上的 GPU 进行逐帧推理，并且开启了 TensorRT 加速功能。那么 TensorRT 是什么呢？简单来说，TensorRT 是英伟达提供的一套模型推理加速包。如果部署环境是英伟达的 GPU 服务器或者是 Jetson 硬件产品，那么使用 TensorRT 可以将各类深度学习框架训练出来的模型进一步地优化和加速。对于一些常

见的深度学习模型，使用 TensorRT 往往可以提速 3 倍以上。

第一次执行上述代码时，由于 TensorRT 会进行静态图优化操作，所以启动时间会很久。等第二次再启动时速度就会加快了，这是因为在代码中设置了 option. trt_option. serialize_file＝'. /tensorrt_cache'，使用这个设置后 TensorRT 会将模型优化后的结果缓存在当前目录的 tensorrt_cache 文件中，下次再启动时程序会优先启用缓存中的模型文件。需要注意的是，如果静态图模型发生了变化，例如重新训练了模型或者更换了算法，那么在这里部署时需要手动删掉当前目录下的 tensorrt_cache 文件，否则程序还是会自动加载以前的缓存模型。

最终实际测试效果如图 3. 23 所示。

(a) 有害垃圾——废旧电池



(b) 厨余垃圾——果蔬果皮

(c) 可回收物——饮料瓶

(d) 其他垃圾——餐巾纸

图 3. 23　Jetson Nano 开发板上真实场景预测

从整体预测结果来看，预测精度基本满足要求。但是如果更换复杂背景，精度就会有所降低。主要原因还是样本数不够，没有充分挖掘每类样本的固有特征，易受背景干扰。后期通过扩充样本数据，可以有效解决这个问题。

本章使用的是一个轻量级的 MobileNetV2 模型，使用这个模型推理速度快、资源消耗低，非常适合小型终端产品的部署，尤其是结合 Jetson Nano 的 TensorRT 库，基本可以达到实时分类的性能。除了 Jetson Nano 以外，Jetson 系列还有一些性能更强的硬件终端，如 Jetson NX、Jetson AGX Xavier 等，对于这些硬件，可以考虑使用更复杂的模型，如 Resnet50、ResNet101 等，从而获得更高的分类精度。

读者如果对本章任务感兴趣，可以在本章任务基础上继续扩充数据集，采集更多复杂场景下的垃圾图片用于训练，进一步提升真实环境下垃圾分类模型的预测精度和鲁棒性。

3.4.3 Jetson Nano 上 C++推理

前面使用 Python 语言在 Jetson Nano 上完成了垃圾实时分类。对于真实的终端产品来说，使用 Python 其性能会受到一定的影响，并且很难做到完善的版权保护。因此，本小节使用性能更佳的 C++语言来完成同样的任务。

1. 在 Jetson Nano 上编译 FastDeploy 的 C++预测库

为了能够在 Jetson Nano 上使用 FastDeploy 进行快速部署，需要自行编译 FastDeploy 的 C++预测库。本书配套资料包中提供了配套的编译好的库（适配 jetpack 4.6.1）。

目前 Jetson Nano 支持 jetpack 4.6.1 及以上版本，具体编译命令如下：

```
git clone https://github.com/PaddlePaddle/FastDeploy.git
cd FastDeploy
mkdir build && cd build                                          # 进入编译目录
export PATH=/usr/local/cuda/bin/:$PATH                           # 设置 CUDA 路径
cmake .. -DBUILD_ON_JETSON=ON \                                  # 设置编译硬件为 Jetson
         -DENABLE_VISION=ON \                                    # 设置编译视觉模块
     -DWITH_GPU=ON \                                             # 设置支持 GPU 模式
     -DENABLE_TRT_BACKEND=ON \                                   # 设置支持 TensorRT 后端
         -DCMAKE_INSTALL_PREFIX=${PWD}/installed_fastdeploy      # 设置输出目录
make -j8                                                         # 开始编译，需要等待较长耗时
make install                                                     # 安装
```

编译完成后，在 FastDeploy/build/installed_fastdeploy 文件夹下面会生成对应的 C++预测库，包括头文件夹 include、库文件夹 lib、工具文件夹 utils、第三方库文件夹 third_libs 以及一些中间文件，如图 3.24 所示。

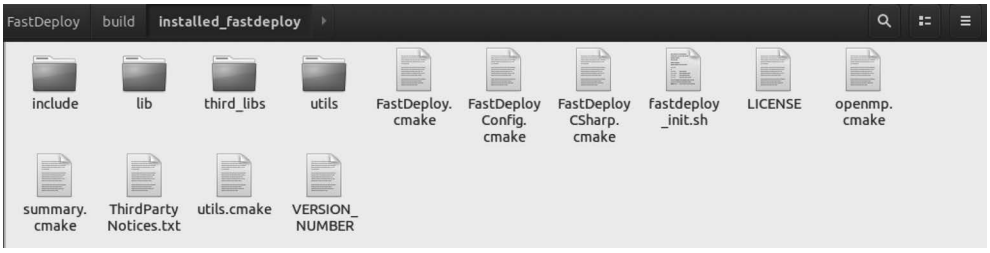


图 3.24 编译好的 FastDeploy C++预测库

接下来就可以在 Jetson Nano 上使用 FastDeploy 的 C++预测库了。

2. 在 Jetson Nano 上安装 Qt

为了能方便地在 Jetson Nano 上编写和编译 C++代码，本书推荐安装 Qt。

Qt 是一个跨平台的 C++开发库，主要用来开发图形用户界面程序，也可以开发不带界面的命令行程序。Qt 支持的操作系统有很多，如通用操作系统 Windows、Linux，智能手机系统 Android、iOS 以及嵌入式系统 QNX、VxWorks 等。当然，Qt 也完全支持 Jetson Nano 的 Ubuntu 环境。

在 Jetson Nano 上安装 Qt 比较简单，只需要输入下述命令即可：

```
sudo apt-get install qt5-default qtcreator -y
```

此时默认安装的是 Qt 5.9.5 版本。安装完成后，在 Jetson Nano 的搜索菜单中搜索 Qt，会出现 Qt Creator，这个即为 Qt 的编程工具，如图 3.25 所示。后续将借助这个编程工具来开发 C++程序。

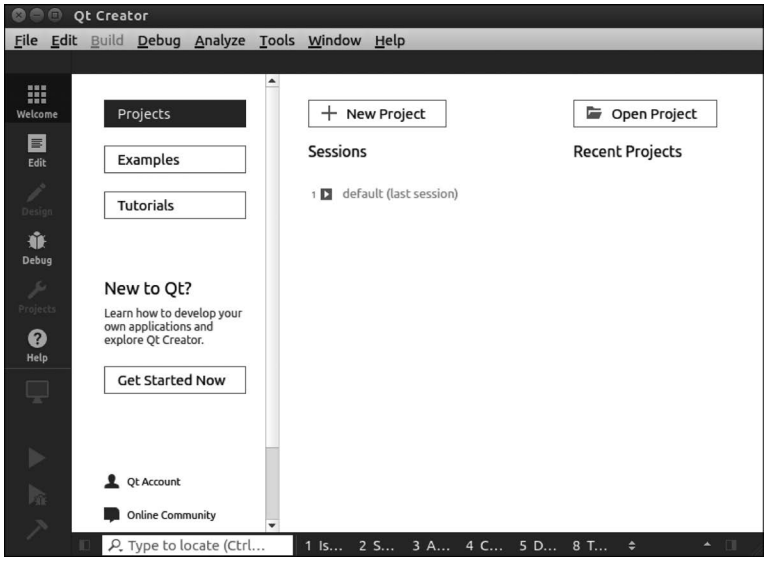


图 3.25　Qt 界面

3. 捕捉摄像头图像实时分类结果

跟前面 Python 版一致，本小节将使用 USB 摄像头实时捕获图像并使用深度学习算法进行垃圾图像分类，最终将分类结果输出展示，整个部署代码使用 C++来编写。

打开 Qt 后，单击 New Project 按钮来创建一个 C++项目。由于本章内容并不需要编写界面程序，因此在项目类型上选择 Non-Qt Project，然后在右侧选择 Plain C++ Application，如图 3.26 所示。

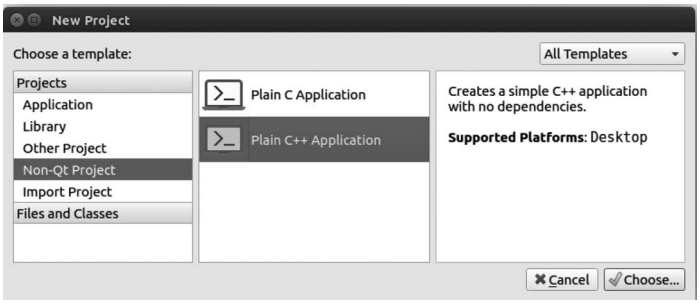


图 3.26　选择项目类型

单击 Choose 按钮，输入本项目名称 InferenceCPlus 并选择项目路径，如图 3.27 所示。

最后选择编译器，为了与 FastDeploy 官方教程一致，这里选择 CMake 编译器，如图 3.28 所示。

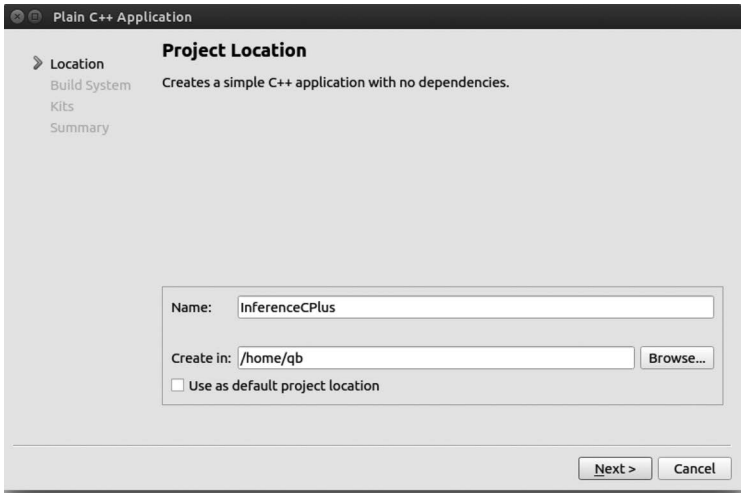


图 3.27　确定项目名称和路径

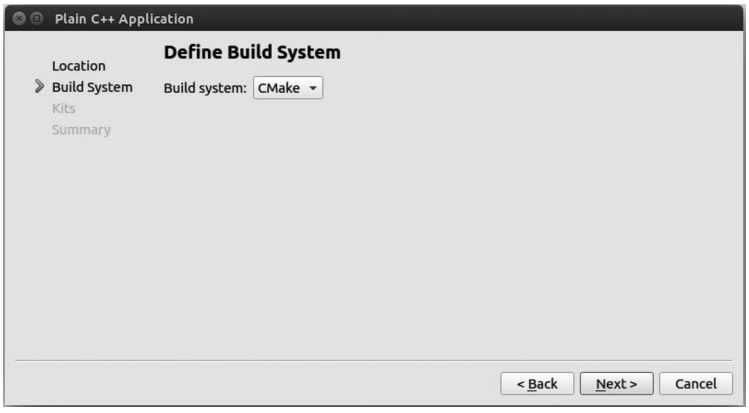


图 3.28　选择编译器

然后默认一直选择 Next 按钮即可成功创建项目。创建成功后，在左侧项目树形列表中可以展开看到创建好的一系列文件，其中 CMakeLists.txt 和 main.cpp 文件就是要编辑的文件，如图 3.29 所示。

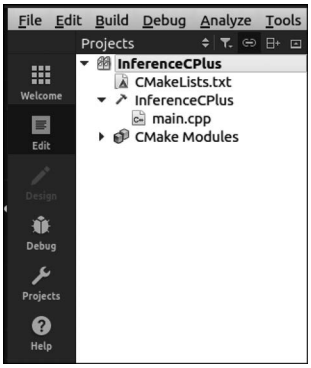


图 3.29　项目列表文件

项目中的 CMakeLists.txt 负责定义项目的基本配置，main.cpp 则负责编写 C++逻辑代码。下面首先修改 CMakeLists.txt 文件(deploy/cplusplus/CMakeLists.txt)：

```
cmake_minimum_required(VERSION 3.10)
project(InferenceCPlus)
# 指定编译后的 FastDeploy 库路径
include(/home/qb/FastDeploy/build/installed_fastdeploy/FastDeploy.cmake)
# 添加 FastDeploy 依赖头文件
include_directories( ${FASTDEPLOY_INCS})
# 绑定 C++代码文件
add_executable( ${PROJECT_NAME} "main.cpp")
# 添加 FastDeploy 库依赖
target_link_libraries(InferenceCPlus ${FASTDEPLOY_LIBS})
```

上述配置添加了 FastDeploy 的 C++库，其中 FastDeploy 库路径需要与前面编译好的 FastDeploy 库目录一致。

接下来修改 main.pp 代码，其主体实现逻辑与前面的 Python 版推理代码一致，完整代码如下(deploy/cplusplus/main.cpp)：

```
#include <iostream>
#include <opencv4/opencv2/opencv.hpp>
#include <opencv4/opencv2/core.hpp>
#include <opencv4/opencv2/highgui.hpp>
#include <opencv4/opencv2/imgproc.hpp>
// 添加 FastDeploy 头文件
#include "fastdeploy/vision.h"
using namespace std;
using namespace cv;
int main(int argc, char **argv)
{
    // 打开摄像头
    VideoCapture cap(0);
    // 设置采集分辨率
    cap.set(cv::CAP_PROP_FRAME_WIDTH, 640);
    cap.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
    // 创建显示窗口
    namedWindow("USB Camera", WINDOW_AUTOSIZE);
    // 配置算法模型的 runtime
    auto option = fastdeploy::RuntimeOption();
    option.UseGpu();
    option.UseTrtBackend();
    // 设置 TensorRT 缓存文件路径
    option.trt_option.serialize_file = './tensorrt_cache';
    // 加载部署模型
    string model_file = "./inference/inference.pdmodel";
    string params_file = "./inference/inference.pdiparams";
    string config_file = "./inference/inference_cls.yaml";
    auto model = fastdeploy::vision::classification::PaddleClasModel(model_file,
                    params_file,config_file, option);
    // 逐帧显示
```

```
    Mat img;
    while (true)
    {
        if (!cap.read(img))
        {
            std::cout << "捕获失败" << std::endl;
            break;
        }
        // 模型预测获取检测结果
        fastdeploy::vision::ClassifyResult result;
        model.Predict(&img, &result, 1);
        // 设置阈值
        int label = result.label_ids[0];
        float score = result.scores[0];
        if (score >= 0.6)
        {
            if (label == 0)
                std::cout << "厨余垃圾" << std::endl;
            else if (label == 1)
                std::cout << "可回收物" << std::endl;
            else if (label == 2)
                std::cout << "其他垃圾" << std::endl;
            else if (label == 3)
                std::cout << "有害垃圾" << std::endl;
        }
        imshow("USB Camera", img);
        int keycode = cv::waitKey(30) & 0xff;              // 按 Esc 键退出
        if (keycode == 27)
            break;
    }
    cap.release();
    destroyAllWindows();
}
```

图 3.30　项目编译

编写完以后按 Ctrl+S 组合键保存所有修改。然后在 Qt 界面的左下角切换项目为 Release 版，并且单击锤子状按钮进行项目编译，如图 3.30 所示。

编译完成后，在项目同目录下会生成编译好的文件夹 build-InferenceCPlus-Desktop-Release，该文件夹下生成的 InferenceCPlus 文件就是最终的可执行程序。

在运行可执行程序前，需要将前面训练好的垃圾分类模型文件夹 inference 放置在 build-InferenceCPlus-Desktop-Release 目录下面，然后在命令窗口中切换到该目录下，最后使用下面的命令运行程序：

```
./InferenceCPlus
```

运行程序时有可能会出现相关.so 文件找不到的情况，如下所示：

```
error while loading shared libraries: libpaddle2onnx.so.1.0.8rc: cannot open shared object
file: No such file or directory
```

这些文件是 FastDeploy 的第三方库依赖文件，需要在 FastDeploy 的 C++库目录下找到对应文件，然后将其复制到/usr/lib/目录下面，这样 Jetson Nano 上的程序才可以加载到这些依赖库文件。

具体的，以 libpaddle2onnx.so.1.0.8rc 文件为例，首先通过 cd 命令切换到 FastDeploy 的 C++库目录下面，然后使用下面的命令进行复制：

```
sudo cp ./third_libs/install/paddle2onnx/lib/libpaddle2onnx.so.1.0.7 /usr/lib/
```

当把所有缺失的.so 文件都复制好以后就可以正常运行了。运行效果和前面 Python 版是一致的。

3.5　小结

本章围绕图像分类，重点介绍了基于深度学习的几种典型分类算法及其实现原理，包括 VGG、ResNet、MobileNetV1 和 MobileNetV2。在算法原理基础上，重点讲解了如何使用 PaddlePaddle 的图像分类套件 PaddleClas 来开发一款智能垃圾分拣器，全流程地实现了数据预处理、训练、验证和推理，并最终将研发的模型通过 FastDeploy 套件在 Jetson Nano 开发板上实现了部署。读者学完本章后，应掌握基本的图像分类算法原理，能够利用 PaddleClas 套件按照本章流程研发图像分类算法。

图像分类作为深度学习图像领域最基础的研究方向，所提出的一些网络模型常作为其他领域的基础模型(Backbone)来使用。例如在图像检测领域，经常会利用 ResNet 模型来提取特征。因此，掌握好本章内容是必要的，也将为后面章节知识打下良好的基础。