第 5 章　计算机视觉基础实验

计算机视觉(Computer Vision)又称为机器视觉(Machine Vision),顾名思义就是要让计算机能够去“看”人类眼中的世界并进行理解和描述。

图像分类是计算机视觉中的一个重要的领域,其核心是向计算机输入一张图像,计算机能够从给定的分类集合中为图像分配一个标签。这里的标签来自预定义的可能类别集。例如,我们预定义类别集合 categories={'猫','狗','其他'},然后我们输入一张图片,计算机给出这幅图片的类别标签‘猫’,或者给出这幅图片属于每个类别标签的概率{'猫': 0.9,'狗': 0.04,'其他': 0.06},这样就完成了一个图像分类任务。

本章通过实验的方式向大家介绍对于图像数据的处理方法以及利用深度学习实现计算机视觉中的图像分类任务。

实践十六：图像数据预处理实践

步骤 1：单通道、多通道图像读取

(1) 单通道图,俗称灰度图,每个像素点只能有一个值表示颜色,它的像素值在 0 到 255 之间,0 是黑色,255 是白色,中间值是一些不同等级的灰色,如图 5-1 所示。

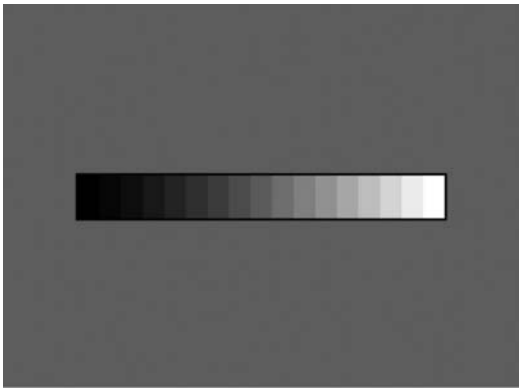

图 5-1　灰度图像素值与颜色变化

(2) 三通道图,每个像素点都有 3 个值表示 ,所以就是 3 通道,也有 4 通道的图,如图 5-2 所示。例如 RGB 图片即为三通道图片,RGB 色彩模式是工业界的一种颜色标准,是通过对红(R)、绿(G)、蓝(B)三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色的,RGB 即是代表红、绿、蓝三个通道的颜色,这个标准几乎包括了人类视力所能感知的所

图 5-2　三通道图(RGB)

有颜色,是目前运用最广的颜色系统之一。

(3) 四通道图像,采用的颜色依然是红(R)、绿(G)、蓝(B),只是多出一个 alpha 通道。alpha 通道一般用作不透明度参数,例如,一个像素的 alpha 通道数值为 0%,那它就是完全透明的(也就是看不见的),而数值为 100%则意味着一个完全不透明的像素(传统的数字图像)。

Python 处理数据图像通常需要使用到以下 3 个库:

Numpy:是 Python 科学计算库的基础。包含了强大的 N 维数组对象和向量运算。

PIL:Python Image Library,是 Python 的第三方图像处理库,提供了丰富的图像处理函数。

cv2:是一个计算机视觉库,实现了图像处理和计算机视觉方面的很多通用算法。

下面介绍多通道图的读取:

(1) 单通道图像。

首先,使用 PIL 的 Image 模块读取图片,获得一个 Image 类实例 img,在 jupyter 中,可使用 display(img)展示图片,也可使用 img.size 查看图片尺寸。具体代码如下:

```
# 引入依赖包
import numpy as np
from PIL import Image
# 读取单通道图像
img = Image.open('lena - gray.jpg')

display(img)
print(type(img))
print(img.size)
```

运行结果如图 5-3 所示。

接下来,使用 np.arrary()将图像转化为像素矩阵,可以将像素矩阵打印查看,也可以通过 shape 属性查看矩阵维度。具体代码和结果如下所示。

```
<PIL.JpegImagePlugin.JpegImageFile image mode=L size=350x350 at 0x7F8FCA198F90>
(350, 350)
```

图 5-3　图像读取

```
# 将图片转为矩阵表示
print("图像尺寸: ", img_np.shape)
img_np = np.array(img)
# 图像尺寸和单通道图像矩阵输出结果如图 5-4 所示
```

```
图像尺寸:  (350, 350)
图像矩阵:
 [[161 162 162 ... 173 168 134]
 [161 162 161 ... 169 167 132]
 [162 163 159 ... 176 171 136]
 ...
 [ 48  51  49 ...  97  97  94]
 [ 41  49  50 ... 101 104 101]
 [ 39  49  53 ... 103 107 106]]
```

图 5-4　图像大小与像素值输出

可以利用 np.savetxt(fname,X,fmt)将矩阵保存为文本,其中,fname 为文件名,X 为要保存到文本中的数据(像素矩阵),fmt 为数据的格式。

```
# 将矩阵保存成文本,数字格式为整数
np.savetxt('lena_gray.txt', img, fmt = '%4d')
# 文本预览如图 5-5 所示
```

(2) 三通道图像。

多通道读取方式与单通道一样,直接用 Image.open()打开即可。

图 5-5　图像数字格式

```
# 读取彩色图像
img = Image.open('lena.jpg')
print(img)
# 将图片转为矩阵表示
img_np = np.array(img)
print("图像尺寸：", img_np.shape)
print("图像矩阵：\n", img_np)
# 运行结果如图 5-6 所示
```

```
<PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=350x350 at 0x7F8FC309E210>
图像尺寸：  (350, 350, 3)
图像矩阵：
 [[[224 136 126]
  [225 137 127]
  [225 137 127]
  ...
  [236 148 136]
  [232 142 131]
  [198 105  97]]
```

图 5-6　三通道图像读取

通过运行结果可以看出单通道图像与三通道图像的不同，单通道图的 mode＝L，三通道图像 mode＝RGB，三通道图像的像素矩阵维度为(350，350，3)。

上面已经介绍了彩色三通道图像的读取。彩色图的 RBG 三个颜色通道是可以分开单独访问的。

第一种方法：使用 PIL 对颜色通道进行分离。这里可以使用 Image 类的 split 方法进行颜色通道分离，也可以使用 Image 类的 getchannel 方法分别获取三个颜色通道的数据。

```
# 读取彩色图像
img = Image.open('lena.jpg')
# 使用 split 分离颜色通道
r,g,b = img.split()
# 使用 getchannel 分离颜色通道
r = img.getchannel(0)
g = img.getchannel(1)
b = img.getchannel(2)
# 展示各通道图像
```

```
display(img.getchannel(0))
display(img.getchannel(1))
display(img.getchannel(2))
# 将矩阵保存成文本,数字格式为整数
np.savetxt('lena-r.txt', r, fmt='%4d')
np.savetxt('lena-g.txt', g, fmt='%4d')
np.savetxt('lena-b.txt', b, fmt='%4d')
# 获取到的R、G、B三个通道的图像展示如图5-7所示
```

图 5-7　三个通道图像展示

第二种方法：使用 cv2.split()分离颜色通道。首先，使用 cv2.imread()读取图片信息，获取图片的像素矩阵；然后，使用 cv2.split()对图像的像素矩阵进行分离；最后，使用 matplotlib.pyplot 将分离和合并结果展示出来。

```
# 引入依赖包
%matplotlib inline
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('lena.jpg')

# 通道分割
b, g, r = cv2.split(img)

# 通道合并
RGB_Image = cv2.merge([b,g,r])
RGB_Image = cv2.cvtColor(RGB_Image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(12,12))

# 绘图展示各通道图像及合并后的图像
plt.subplot(141)
plt.imshow(RGB_Image,'gray')
plt.title('RGB_Image')
plt.subplot(142)
plt.imshow(r,'gray')
plt.title('R_Channel')
plt.subplot(143)
```

```
plt.imshow(g,'gray')
plt.title('G_Channel')
plt.subplot(144)
plt.imshow(b,'gray')
plt.title('B_Channel')
# 输出结果如图 5-8 所示
```

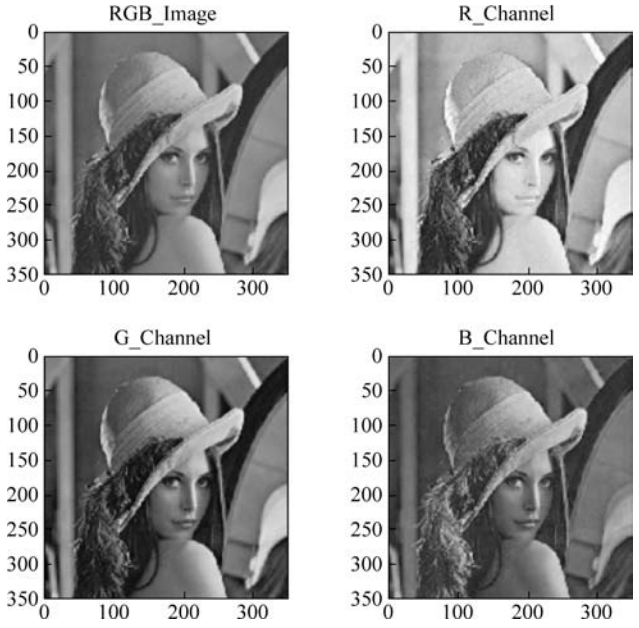

图 5-8　各通道图像及合并后的图像

步骤 2：图像的通道转换

上一步骤我们提前使用了 OpenCV 读取图片并进行通道分离，本步骤将会对 OpenCV 库的使用做更详细的介绍。

- cv2.imread()：用来读取图片，第一个参数是图片路径，第二个参数是一个标识，用来指定图像的读取方式。
- cv2.imshow()：用来显示图像，第一个参数是窗口的名字，第二个参数是图像数据。
- cv2.imwrite()：用来保存图像，第一个参数是要保存的文件名，第二个参数是要保存的图像。可选的第三个参数，它针对特定的格式：对于 JPEG，其表示的是图像的质量，用 0～100 的整数表示，默认为 95；对于 png，第三个参数表示的是压缩级别，默认为 3。

我们在使用 cv2.imread()读取图像时，cv2 会默认将三通道彩色图像转化为 GBR 格式，因此经常需要将其转化为 RGB 格式。本节的内容主要介绍如何使用 cv2.cvtColor()对图像通道进行转化以及如何将彩色图像转换为灰度图像。

在图像处理中最常用的颜色空间转换如下：

- RGB 或 BGR 到灰度(COLOR_RGB2GRAY，COLOR_BGR2GRAY)。

- RGB 或 BGR 到 YcrCb(或 YCC)(COLOR_RGB2YCrCb,COLOR_BGR2YCrCb)。
- RGB 或 BGR 到 HSV(COLOR_RGB2HSV,COLOR_BGR2HSV)。
- RGB 或 BGR 到 Luv(COLOR_RGB2Luv,COLOR_BGR2Luv)。
- 灰度到 RGB 或 BGR(COLOR_GRAY2RGB,COLOR_GRAY2BGR)。

(1) BGR 图像转换为灰度图像。

```
import numpy as np
import cv2
img = cv2.imread('lena.jpg') #默认为彩色图像
# 打印图片的形状
print(img.shape)
# 形状中包括行数、列数和通道数
height, width, channels = img.shape
print('图片高度:{},宽度:{},通道数:{}'.format(height,width,channels))
# 转换为灰度图
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print(img_gray.shape)
```

读取的三通道彩色图像图片维度为(350,350,3)。

通过 cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)获取到灰度图像的维度为(350,350),可以使用 cv2.imwrite()将图像进行保存。

```
# 保存灰度图,如图 5-9 所示
cv2.imwrite('img_gray.jpg', img_gray)
```

图 5-9 写灰度图

(2) GBR 图像转换为 RGB 图像。

将 GBR 图像转换为 RGB 图像的方式十分简单,只需要将 cv2.cvtColor()中第二个参数设置为 cv2.COLOR_BGR2RGB 即可。

```
import cv2
# 加载彩色图
img = cv2.imread('lena.jpg', 1)
# 将彩色图的 BGR 通道顺序转成 RGB
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

步骤 3:图像拼接与缩放

图像拼接,顾名思义就是将两张图片拼接在一起成为一张图像。本实验先将一张图片从中间切割成两张图片,然后再进行拼接。

首先,我们用 cv2.imread()读取图 5-10,即获得图像的像素矩阵,然后,通过 numpy array 的 shape 方法获取矩阵的尺寸,根据尺寸将图像分割成两张图片。

```
img = cv2.imread('test.jpg')
# the image height
sum_rows = img.shape[0]
print(img.shape)
```

图 5-10　原始实验图

```
# print(sum_rows)
# the image length
sum_cols = img.shape[1]
# print(sum_cols)
part1 = img[0:sum_rows, 0:int(sum_cols/2)]
print(part1.shape)
part2 = img[0:sum_rows, int(sum_cols/2):sum_cols]
print(part2.shape)
plt.figure(figsize=(12,12))
# 显示分割后图片
plt.subplot(121)
plt.imshow(part1)
plt.title('Image1')
plt.subplot(122)
plt.imshow(part2)
plt.title('Image2')
# 分割后形成的图片如图 5-11 所示
```

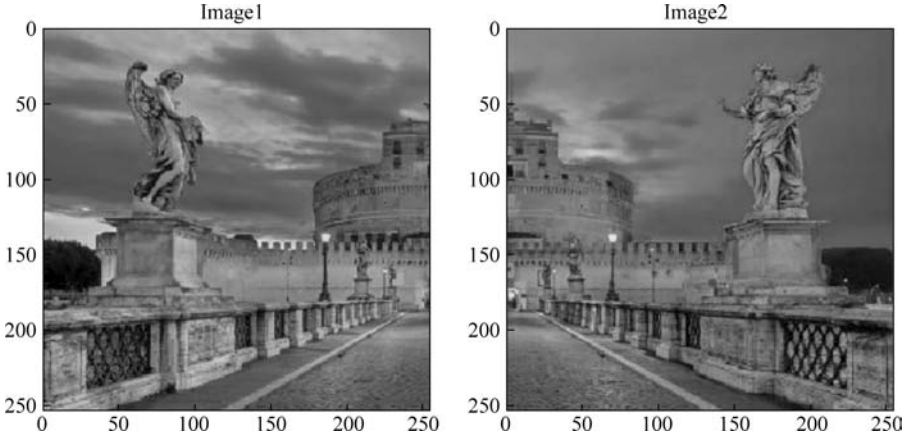

图 5-11　分割后的图片

接下来，尝试将两张图片进行拼接。我们根据原始图像的大小用 np.zeros()初始化一个全为 0 的矩阵 final_matrix，其尺寸大小为 254×510×3(与原始图像大小相同)；然后，将两张图像的像素矩阵赋值到 final_matrix 的响应位置，形成一个完整的像素矩阵，也就完成了图像的拼接。

```
# new image
final_matrix = np.zeros((254, 510, 3), np.uint8)
# change
final_matrix[0:254, 0:255] = part2
final_matrix[0:254, 255:510] = part2
plt.subplot(111)
plt.imshow(final_matrix)
plt.title('final_img')
# 拼接后的图像展示如图 5-12 所示
```

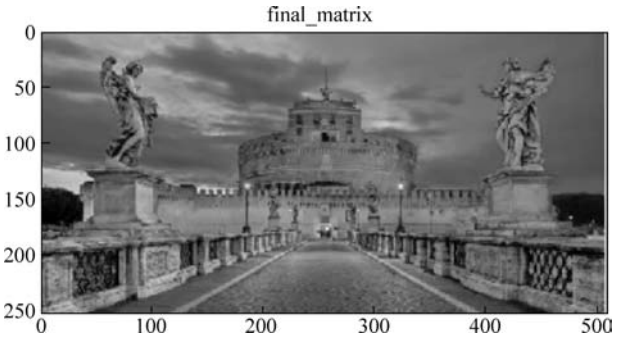

图 5-12　拼接后的图片

缩放图片就是调整图片的大小。在实验中我们使用 cv2.resize(input,output,size,fx,fy,interpolation)函数实现缩放。其中，input 为输入图片，output 为输出图片，size 为输出图片尺寸，fx 和 fy 为沿 x 轴和 y 轴缩放系数，interpolation 为缩放插入方法。

cv2.resize()提供了多种图片缩放插入方法，如下所示：

- cv2.INTER_NEAREST：最近邻插值。
- cv2.INTER_LINEAR：线性插值(默认缩放方式)。
- cv2.INTER_AREA：基于局部像素的重采样，区域插值。
- cv2.INTER_CUBIC：基于邻域 4×4 像素的三次插值。
- cv2.INTER_LANCZOS4：基于 8×8 像素邻域的 Lanczos 插值。

首先，读取一张图片并将其转换为 RGB 格式；然后使用 cv2.resize()将图片进行缩放，若缩放方法没有设置，则默认为线性插值方法，如下面代码所示。通过输出图片我们可以看到，我们将一张尺寸为 2180×1911 像素的图像缩放成了 400×500 像素，如图 5-13 所示。

```
img = cv2.imread('cat.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
print(img.shape)
# 按照指定的宽度、高度缩放图片
res = cv2.resize(img, (400, 500))
plt.imshow(res)
```

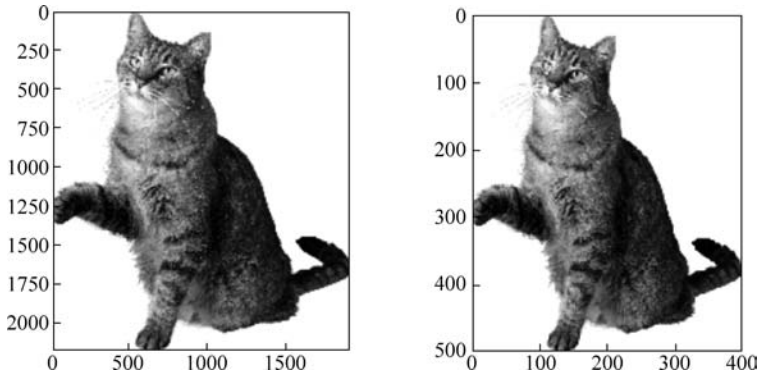

图 5-13　缩放前后的图片

尝试通过设置 x、y 轴缩放系数来对图像进行缩放，例如将 x，y 轴的取值分别放大 5 倍与 2 倍。

```
# 按照比例缩放，如 x，y 轴均放大 1 倍
res2 = cv2.resize(img, None, fx = 5, fy = 2, interpolation = cv2.INTER_LINEAR)
plt.imshow(res)
# 缩放后效果如图 5-14 所示
```

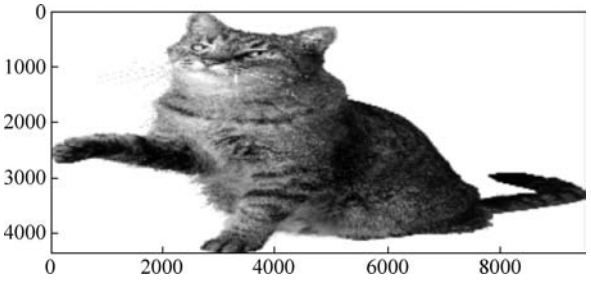

图 5-14　缩放后的图片

步骤 4：图像二值化处理

图像二值化是为了方便提取图像中的信息，二值图像在进行计算机识别时可以增加识别效率。我们已经知道图像像素点的灰度值在 0～255 之间，图像的二值化简单来说就是通过设定一个阈值，将像素点灰度值大于阈值变成一类值，小于阈值变成另一类值。

Opencv 提供的 cv2.threshold()可以用来方便地实现图像的二值化。该函数有四个参数，第一个原图像，第二个进行分类的阈值，第三个是高于(低于)阈值时赋予的新值，第四个是一个方法选择参数，常用的有：

- 0：cv2.THRESH_BINARY，当前点值大于阈值时，取 Maxval，也就是第四个参数，否则设置为 0。
- 1：cv2.THRESH_BINARY_INV，当前点值大于阈值时，设置为 0，否则设置为 Maxval。
- 2：cv2.THRESH_TRUNC，当前点值大于阈值时，设置为阈值，否则不改变。

- 3：cv2.THRESH_TOZERO，当前点值大于阈值时，不改变，否则设置为 0。
- 4：cv2.THRESH_TOZERO_INV，当前点值大于阈值时，设置为 0，否则不改变。

尝试对一张图片进行二值化，我们设置像素点的灰度值超过 127 则将该像素点灰度值重新赋值为 255，灰度值小于 127 的，我们将该像素点的灰度值重新赋值为 0。

```
import cv2

# 灰度图读入
img = cv2.imread('lena.jpg', 0)
# 颜色通道转换
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
ret, th = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
# 阈值分割

plt.imshow(th)
# 二值化后的图像效果如图 5-15 所示
```

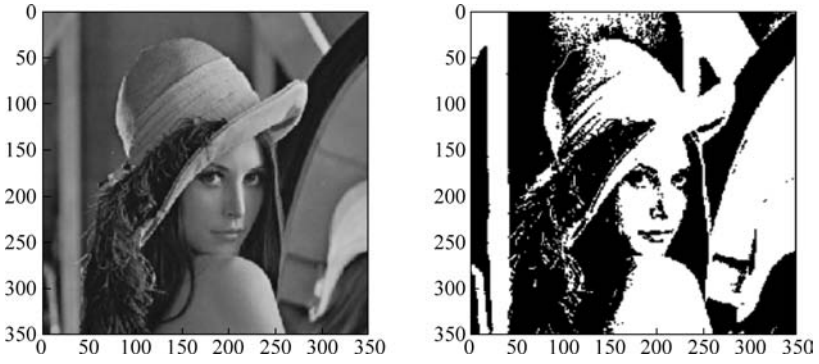

图 5-15　二值化前后的图片

步骤 5：图像归一化处理

图像的归一化是对图像的像素矩阵进行一系列的变化，使像素的灰度值都落入一个特定的区间。在机器学习中，对数据进行归一化可以加快训练网络的收敛性。

飞桨深度学习平台提供了 paddle.vision.transforms.normalize()方法可以方便的实现对图像数据的归一化。该方法包含四个参数，第一个参数为图像的 np.array 格式数据，第二个参数为用于每个通道归一化的均值，第三个参数为用于每个通道归一化的标准差值，第三个参数为数据的格式，第四个参数为是否转换为 rgb 的格式，默认为 False。

下面尝试对一张三通道图片进行归一化处理。我们设置图像三个通道的归一化后的均值分别为 0.31169346、0.25506335、0.12432463，图像三个通道的标准差分别为 0.34042713、0.29819837、0.1375536，图像的格式为‘HWC’，具体代码及归一化效果如下所示：

```
import numpy as np
from PIL import Image
from paddle.vision.transforms import functional as F
```

```
img = np.asarray(Image.open('lena.jpg'))

mean = [0.31169346, 0.25506335, 0.12432463]
std = [0.34042713, 0.29819837, 0.1375536]
normalized_img = F.normalize(img, mean, std, data_format='HWC')

normalized_img = Image.fromarray(np.uint8(normalized_img))
normalized_img.save('normalized_img.jpg')

# 归一化后效果对比如图 5-16 所示
```

图 5-16　归一化前后的图片

实践十七：基于卷积神经网络实现美食分类

本次实验中，我们使用卷积神经网络(CNN)解决美食图片的分类问题。CNN 由纽约大学的 Yann LeCun 于 1998 年提出。CNN 本质上是一个多层感知机，其成功的原因关键在于它所采用的局部连接和共享权值的方式，一方面减少了权值的数量使得网络易于优化，另一方面降低了过拟合的风险。

本实验代码运行的环境配置如下：Python 版本为 3.7，飞桨版本为 2.0，操作平台为 AI Studio。

步骤 1：美食识别数据集加载

本实验使用的数据集包含 5000 余张格式为 jpg 的三通道彩色图像，共 5 种食物类别。对于本实验中的数据包，具体处理及加载方式与宝石分类实验基本相同，主要步骤如下：

首先，我们定义 unzip_data()对数据集的压缩包进行解压，解压后可以观察到数据集文件夹结构如图 5-17 所示。

```
aistudio@jupyter-44484-2011726:~/data/foods$ tree -L 1
.
├── apple_pie
├── baby_back_ribs
├── baklava
├── beef_carpaccio
└── beef_tartare
```

图 5-17　数据集文件夹结构

然后，定义 get_data_list()遍历文件夹和图片，按照一定比例将数据划分为训练集和验证集，并生成图片 label、train.txt、eval.txt；最终生成的训练样本格式如图 5-18 所示，每条训练样本都由该图片的存储位置和对应的标签组成。

```
/home/aistudio/data/foods/apple_pie/2328227.jpg 1
/home/aistudio/data/foods/beef_carpaccio/1932385.jpg    0
/home/aistudio/data/foods/baby_back_ribs/2275499.jpg    3
/home/aistudio/data/foods/apple_pie/2602468.jpg 1
/home/aistudio/data/foods/baby_back_ribs/2878757.jpg    3
/home/aistudio/data/foods/apple_pie/1097378.jpg 1
/home/aistudio/data/foods/beef_tartare/1577426.jpg  2
```

图 5-18　训练样本形式

接下来，定义一个数据加载器 FoodDataset，用于加载训练和评估时要使用的数据。这里需要继承基类 Dataset。具体代码如下：

__init__：构造函数，实现数据的读取逻辑。

__getitem__：实现对数据的处理操作，返回图像的像素矩阵和标签值。

__len__：返回数据集样本个数。

```
Class FoodDataset(paddle.io.Dataset):
  def __init__(self, data_path, mode = 'train'):
    """
    数据读取器
    :param data_path: 数据集所在路径
    :param mode: train or eval
    """
    super().__init__()
    self.data_path = data_path
    self.img_paths = []
    self.labels = []

    if mode == 'train':
      with open(os.path.join(self.data_path, "train.txt"), "r", encoding = "utf - 8") as f:
        self.info = f.readlines()
      for img_info in self.info:
        img_path, label = img_info.strip().split('\t')
        self.img_paths.append(img_path)
        self.labels.append(int(label))

    else:
      with open(os.path.join(self.data_path, "eval.txt"), "r", encoding = "utf - 8") as f:
        self.info = f.readlines()
      for img_info in self.info:
        img_path, label = img_info.strip().split('\t')
        self.img_paths.append(img_path)
        self.labels.append(int(label))

def __getitem__(self, index):
  """
```

```
    获取一组数据
    :param index: 文件索引号
    :return:
    """
    # 第一步打开图像文件并获取 label 值
    img_path = self.img_paths[index]
    img = Image.open(img_path)
    if img.mode != 'RGB':
      img = img.convert('RGB')
    img = img.resize((64, 64), Image.BILINEAR)
    img = np.array(img).astype('float32')
    img = img.transpose((2, 0, 1)) / 255
    label = self.labels[index]
    label = np.array([label], dtype="int64")
    return img, label

  def print_sample(self, index: int = 0):
    print("文件名", self.img_paths[index], "\t 标签值", self.labels[index])

  def __len__(self):
    return len(self.img_paths)
```

最后，利用 paddle.io.DataLoader()方法定义训练数据加载器 train_loader 和验证数据加载器 eval_loader，并设置 batch_size 大小。

```
# 训练数据加载
train_dataset = FoodDataset(data_path='data/',mode='train')
train_loader = paddle.io.DataLoader(train_dataset, batch_size=train_parameters['train_
batch_size'], shuffle=True)
# 测试数据加载
eval_dataset = FoodDataset(data_path='data/',mode='eval')
eval_loader = paddle.io.DataLoader(eval_dataset, batch_size = 8, shuffle=False)
```

步骤 2：自定义卷积神经网络

本任务使用的卷积网络结构(CNN)，输入的是归一化后的 RGB 图像样本，每张图像的尺寸被裁切到了 64×64，经过三次“卷积-池化”操作，最后连接一个输出层，具体模型结构如图 5-19 所示。

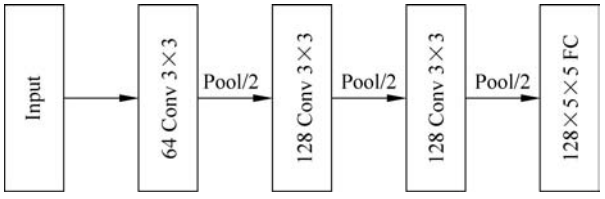

图 5-19　卷积神经网络实现美食分类结构

在了解了本实践的网络结构后，接下来就可以使用飞桨深度学习框架搭建该网络来解决美食识别的问题。本实践主要使用卷积神经网络进行图像的分类，自定义模型类

MyCNN，该类继承 nn.Layer 抽象类，实现模型训练、验证模式的切换等功能。在飞桨中，paddle.nn.Conv2D(in_channels，out_channels，kernel_size，stride＝1，padding＝0，dilation＝1，groups＝1，padding_mode＝'zeros'，weight_attr＝None，bias_attr＝None，data_format＝'NCHW')可实现二维卷积，根据输入通道数(in_channels)、输出通道数(out_channels)、卷积核大小(kernel_size)、步长(stride)、填充(padding)、空洞大小(dilations)等参数计算输出特征层大小。输入和输出是 NCHW 或 NHWC 格式，其中 N 是批大小，C 是通道数，H 是特征高度，W 是特征宽度，卷积核是 MCHW 格式，M 是输出图像通道数，C 是输入图像通道数，H 是卷积核高度，W 是卷积核宽度，如果组数(groups)大于 1，C 等于输入图像通道数除以组数的结果，其中，输入的单个通道图像维度与输出的单个通道图像维度的对应计算关系如下：

$$H_{out}=\frac{(H_{in}+2*paddings[0]-(dilations[0]*(kernel\_size[0]-1)+1))}{strides[0]}+1$$

$$W_{out}=\frac{(W_{in}+2*paddings[1]-(dilations[1]*(kernel\_size[1]-1)+1))}{strides[1]}+1$$

在应用卷积操作之后，可以对卷积后的特征映射进行下采样，以达到降维的效果，本实践采用最大池化 paddle.nn.MaxPool2D(kernel_size，stride＝None，padding＝0，ceil_mode＝False，return_mask＝False，data_format＝'NCHW'，name＝None)类实现特征的下采样，其中，kernel_size 为池化核大小。如果它是一个元组或列表，它必须包含两个整数值 (pool_size_Height，pool_size_Width)。如果为一个整数，则它的平方值将作为池化核大小，比如若 pool_size＝2，则池化核大小为 2×2；stride(可选)为池化层的步长，使用规则同 pool_size 相同，默认值为 None，这时会使用 kernel_size 作为 stride；padding (可选) 为池化填充，如果它是一个字符串，可以是"VALID"或者"SAME"，表示填充算法，如果它是一个元组或列表，它可以有 3 种格式：

(1) 包含 2 个整数值：[pad_height，pad_width]；

(2) 包含 4 个整数值：[pad_height_top，pad_height_bottom，pad_width_left，pad_width_right]；

(3) 包含 4 个二元组：当 data_format 为"NCHW"时为 [[0,0]，[0,0]，[pad_height_top，pad_height_bottom]，[pad_width_left，pad_width_right]]，当 data_format 为"NHWC"时为[[0,0]，[pad_height_top，pad_height_bottom]，[pad_width_left，pad_width_right]，[0,0]]，若为一个整数，则表示 H 和 W 维度上均为该值。

ceil_mode (可选)表示是否用 ceil 函数计算输出高度和宽度。

如果是 True，则使用 ceil 计算输出形状的大小；return_mask (可选)指示是否返回最大索引和输出，默认为 False；data_format (可选)：输入和输出的数据格式，可以是"NCHW"和"NHWC"，N 是批尺寸，C 是通道数，H 是特征高度，W 是特征宽度，默认值："NCHW"。

详细介绍了各个卷积类与池化类之后，我们可以实现分类算法如下：

```
import paddle.nn as nn
# 定义卷积网络
```

```
class MyCNN(nn.Layer):
  def __init__(self):
    super(MyCNN, self).__init__()
    self.conv0 = nn.Conv2D(in_channels = 3,
                           out_channels = 64,
                           kernel_size = 3,
                           padding = 0,
                           stride = 1)
    self.pool0 = nn.MaxPool2D(kernel_size = 2, stride = 2)
    self.conv1 = nn.Conv2D(in_channels = 64,
                           out_channels = 128,
                           kernel_size = 4,
                           padding = 0,
                           stride = 1)
    self.pool1 = nn.MaxPool2D(kernel_size = 2, stride = 2)
    self.conv2 = nn.Conv2D(in_channels = 128,
                           out_channels = 128,
                           kernel_size = 5,
                           padding = 0)
    self.pool2 = nn.MaxPool2D(kernel_size = 2, stride = 2)
    self.fc1 = nn.Linear(in_features = 128 * 5 * 5, out_features = 5)

  def forward(self,input):
    x = self.conv0(input)
    x = self.pool0(x)
    x = self.conv1(x)
    x = self.pool1(x)
    x = self.conv2(x)
    x = self.pool2(x)
    x = fluid.layers.reshape(x,shape = [ - 1,50 * 5 * 5])
    y = self.fc1(x)
    return y
```

步骤 3：模型训练与评估

以上我们已经定义好 MyCNN 模型结构，接下来实例化一个模型并进行迭代训练，对于分类问题，依旧使用交叉熵损失函数，使用 paddle.optimizer.SGD 优化器进行参数梯度的计算，具体代码如下：

```
model = MyCNN()                                                    # 模型实例化
model.train()                                                      # 训练模式
cross_entropy = paddle.nn.CrossEntropyLoss()
opt = paddle.optimizer.SGD(learning_rate = 0.001, parameters = model.parameters())

epochs_num = train_parameters['num_epochs']                        # 迭代次数
for pass_num in range(train_parameters['num_epochs']):
  for batch_id,data in enumerate(train_loader()):
    image = data[0]
    label = data[1]
```

```
        predict = model(image)                                          # 数据传入 model
        loss = cross_entropy(predict,label)
        acc = paddle.metric.accuracy(predict,label.reshape([-1,1]))     # 计算精度
        # acc = np.mean(label == np.argmax(predict,axis = 1))

        if batch_id!= 0 and batch_id % 10 == 0:
          Batch = Batch + 10
          Batchs.append(Batch)
          all_train_loss.append(loss.numpy()[0])
          all_train_accs.append(acc.numpy()[0])
          print("epoch:{},step:{},train_loss:{},train_acc:{}".format(pass_num,batch_id,loss.
    numpy()[0],acc.numpy()[0]))
        loss.backward()
        opt.step()
        opt.clear_grad()                                    # opt.clear_grad()来重置梯度
    paddle.save(model.state_dict(),'MyCNN')                 # 保存模型
```

保存模型之后，接下来我们对模型进行评估。模型评估就是在验证数据集上计算模型输出结果的准确率。与训练部分代码不同，评估模型时不需要进行参数优化，因此，需要使用验证模式，具体代码如下：

```
# 模型评估
para_state_dict = paddle.load("MyCNN")
model = MyCNN()
model.set_state_dict(para_state_dict)               # 加载模型参数
model.eval()                                        # 验证模式

accs = []

for batch_id,data in enumerate(eval_loader()):      # 测试集
  image = data[0]
  label = data[1]
  predict = model(image)
  acc = paddle.metric.accuracy(predict,label)
  accs.append(acc.numpy()[0])
  avg_acc = np.mean(accs)
print("当前模型在验证集上的准确率为:",avg_acc)
```

实践十八：基于 VGG-16 实现中草药分类

本实验我们使用 VGG 网络模型解决中草药的分类问题。VGGNet 是牛津大学计算机视觉组和 Google DeepMind 公司的研究员一起研发的深度卷积神经网络。VGG 主要探究了卷积神经网络的深度和其性能之间的关系，通过反复堆叠 3×3 的小卷积核和 2×2 的最大池化层，VGGNet 成功的搭建了 16～19 层的深度卷积神经网络，通过不断加深网络来提升性能。

本实验代码运行的环境配置如下：Python 版本为 3.7，飞桨版本为 2.0，操作平台为 AI Studio。

步骤 1：中草药分类数据集准备

本实验使用的数据集包含 900 余张格式为 jpg 的三通道彩色图像，共 5 种中草药类别。我们在 AIstudio 上提供了本实验的数据集压缩包 Chinese Medicine.zip。对于本实验中的数据包，具体处理与加载方式与美食分类实验基本相同，主要步骤如下：

首先，我们定义 unzip_data()对数据集的压缩包进行解压，解压后可以观察到数据集文件夹结构如图 5-20 所示。

```
aistudio@jupyter-44484-2011752:~/data/Chinese Medicine$ tree -L 1
.
├── baihe
├── dangshen
├── gouqi
├── huaihua
└── jinyinhua
```

图 5-20　数据集文件夹结构

然后，定义 get_data_list()遍历文件夹和图片，按照一定比例将数据划分为训练集和验证集，并生成图片 label、train.txt、eval.txt；训练数据的格式如图 5-21 所示，其与实践十七中的样本格式一致。

```
/home/aistudio/data/Chinese Medicine/jinyinhua/jyh_129.jpg  0
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_32.jpg   1
/home/aistudio/data/Chinese Medicine/gouqi/cgcjyfyj (2).jpg 2
/home/aistudio/data/Chinese Medicine/gouqi/u=1010307075,2293841367&fm=26&gp=0.jpg   2
/home/aistudio/data/Chinese Medicine/huaihua/huaihua_61.jpg 3
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_157.jpg  1
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_122.jpg  1
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_144.jpg  1
/home/aistudio/data/Chinese Medicine/gouqi/cgcjyfyj (34).jpg    2
```

图 5-21　训练集样本格式

接下来，定义一个数据加载器 dataset，用于加载训练和评估时要使用的数据。数据加载器定义方式与 Reader 定义方式相同；

最后，利用 paddle.io.DataLoader()方法定义训练数据加载器 train_loader 和验证数据加载器 eval_loader，并设置 batch_size 大小。

```
# 训练数据加载
train_dataset = dataset('/home/aistudio/data',mode = 'train')
train_loader = paddle.io.DataLoader(train_dataset, batch_size = 16, shuffle = True)
# 测试数据加载
eval_dataset = dataset('/home/aistudio/data',mode = 'eval')
eval_loader = paddle.io.DataLoader(eval_dataset, batch_size = 8, shuffle = False)
```

步骤 2：VGG-16 网络搭建

VGGNet 引入“模块化”的设计思想，将不同的层进行简单组合构成网络模块，再用模块来组装成完整网络，而不再是以“层”为单元组装网络。本实验使用的是 VGG-16 网络模型，

输入是归一化后的 RGB 图像样本，每张图像的尺寸被裁切到了 224×224，使用 ReLU 作为激活函数，在全连接层使用 Dropout 防止过拟合。VGGNet 中所有的 3×3 卷积(conv3)都是等长卷积(步长 1，填充 1)，因此特征图的尺寸在模块内是不变的。特征图每经过一次池化，其高度和宽度减少一半，作为弥补，其通道数增加 1 倍，最后通过全连接与 Softmax 层输出结果。VGG-16 结构如图 5-22 所示。

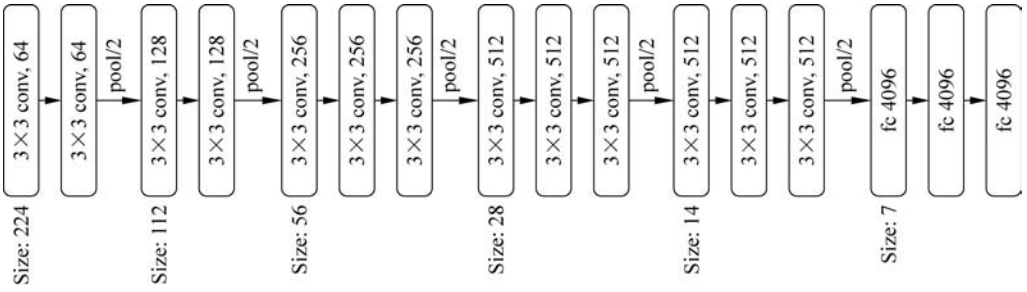

图 5-22　VGG-16 网络结构

在了解了 VGG-16 的网络结构后，接下来就可以使用飞桨深度学习框架来搭建一个 VGG-16 网络来解决中草药识别问题。

首先，根据“模块化”的思想，我们定义 VGG-16 要使用的“卷积池化”模块 ConvPool，在该模块中，使用一种新的定义可训练层的方法，即 paddle.nn.Layer.add_sublayer(name, sublayer)，该方法为封装在 Layer 类中的函数，实现子层实例的添加，需要传递两个参数：子层名 name(str)与 Layer 实例 sublayer(Layer)，可以通过 self.name 访问该 sublayer，ConvPool 类实现如下：

```
class ConvPool(paddle.nn.Layer):
  '''卷积+池化'''
  def __init__(self,
           num_channels,
           num_filters,
           filter_size,
           pool_size,
           pool_stride,
           groups,
           conv_stride=1,
           conv_padding=1,
           ):
    super(ConvPool, self).__init__()

    for i in range(groups):
      self.add_sublayer(                          # 添加子层实例
        'bb_%d' % i,
        paddle.nn.Conv2D(                         # layer
        in_channels=num_channels,                 # 通道数
        out_channels=num_filters,                 # 卷积核个数
        kernel_size=filter_size,                  # 卷积核大小
        stride=conv_stride,                       # 步长
```

```
            padding = conv_padding,                    # padding
            )
        )
        self.add_sublayer(
            'relu%d' % i,
            paddle.nn.ReLU()
        )
        num_channels = num_filters


    self.add_sublayer(
        'Maxpool',
        paddle.nn.MaxPool2D(
        kernel_size = pool_size,                       # 池化核大小
        stride = pool_stride                           # 池化步长
        )
    )

  def forward(self, inputs):
    x = inputs
    for prefix, sub_layer in self.named_children():
      x = sub_layer(x)
    return x
```

接下来，我们利用 Convpool 模块定义 VGG-16 网络模型，具体代码如下：

```
class VGGNet(paddle.nn.Layer):
  def __init__(self):
    super(VGGNet, self).__init__()
    self.convpool01 = ConvPool(
      3, 64, 3, 2, 2, 2) #3:通道数,64: 卷积核个数,3:卷积核大小,2:池化核大小,2:池化步长,
2:连续卷积个数
    self.convpool02 = ConvPool(
      64, 128, 3, 2, 2, 2)
    self.convpool03 = ConvPool(
      128, 256, 3, 2, 2, 3)
    self.convpool04 = ConvPool(
      256, 512, 3, 2, 2, 3)
    self.convpool05 = ConvPool(
      512, 512, 3, 2, 2, 3)
    self.pool_5_shape = 512 * 7 * 7
    self.fc01 = paddle.nn.Linear(self.pool_5_shape, 4096)
    self.fc02 = paddle.nn.Linear(4096, 4096)
    self.fc03 = paddle.nn.Linear(4096, train_parameters['class_dim'])

  def forward(self, inputs, label = None):
    # print('input_shape:', inputs.shape) #[8, 3, 224, 224]
    """前向计算"""
    out = self.convpool01(inputs)
    out = self.convpool02(out)
```

```
        out = self.convpool03(out)
        out = self.convpool04(out)
        out = self.convpool05(out)

        out = paddle.reshape(out, shape=[-1, 512*7*7])
        out = self.fc01(out)
        out = self.fc02(out)
        out = self.fc03(out)

        if label is not None:
            acc = paddle.metric.accuracy(input=out, label=label)
            return out, acc
        else:
            return out
```

步骤 3：模型训练与评估

前面我们已经定义好 VGGNet 模型结构，接下来实例化一个模型并进行迭代训练，本实践使用交叉熵损失函数，使用 paddle.optimizer.Adam(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08, parameters=None, weight_decay=None, grad_clip=None, name=None, lazy_mode=False)优化器，该优化器能够利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率，其中，learning_rate 为学习率，用于参数更新的计算，可以是一个浮点型值或者一个_LRScheduler 类，默认值为 0.001；beta1 为一阶矩估计的指数衰减率，是一个 float 类型或者一个 shape 为[1]，默认值为 0.9；beta2 为二阶矩估计的指数衰减率，默认值为 0.999；epsilon 为保持数值稳定性的短浮点类型值，默认值为 1e-08；parameters 指定优化器需要优化的参数，在动态图模式下必须提供该参数，在静态图模式下默认值为 None，这时所有的参数都将被优化；weight_decay 为正则化方法，可以是 L2 正则化系数或者正则化策略；grad_clip 为梯度裁剪的策略，支持三种裁剪策略：paddle.nn.ClipGradByGlobalNorm、paddle.nn.ClipGradByNorm、paddle.nn.ClipGradByValue，默认值为 None，此时将不进行梯度裁剪；lazy_mode 设为 True 时，仅更新当前具有梯度的元素。VGGNet 具体代码如下：

```
model = VGGNet()
model.train()
cross_entropy = paddle.nn.CrossEntropyLoss()
optimizer = paddle.optimizer.Adam(learning_rate=train_parameters['learning_strategy']
['lr'],parameters=model.parameters())

steps = 0
Iters, total_loss, total_acc = [], [], []

for epo in range(train_parameters['num_epochs']):
    for _, data in enumerate(train_loader()):
        steps += 1
        x_data = data[0]
        y_data = data[1]
```

```
        predicts, acc = model(x_data, y_data)
        loss = cross_entropy(predicts, y_data)
        loss.backward()
        optimizer.step()
        optimizer.clear_grad()
        if steps % train_parameters["skip_steps"] == 0:
          Iters.append(steps)
          total_loss.append(loss.numpy()[0])
          total_acc.append(acc.numpy()[0])
          # 打印中间过程
          print('epo: {}, step: {}, loss is: {}, acc is: {}'.format(epo, steps, loss.numpy(), acc.
numpy()))
        # 保存模型参数
        if steps % train_parameters["save_steps"] == 0:
          save_path = train_parameters["checkpoints"] + "/" + "save_dir_" + str(steps) + '.pdparams'
          print('save model to: ' + save_path)
          paddle.save(model.state_dict(),save_path)
paddle.save(model.state_dict(),train_parameters["checkpoints"] + "/" + "save_dir_final.
pdparams")
```

保存模型之后，接下来我们对模型进行评估。模型评估就是在验证数据集上计算模型输出结果的准确率。与训练部分代码不同，评估模型时不需要进行参数优化，因此，需要使用验证模式，具体代码如下：

```
model__state_dict = paddle.load('work/checkpoints/save_dir_final.pdparams')
model_eval = VGGNet()
model_eval.set_state_dict(model__state_dict)
model_eval.eval()
accs = []

for _, data in enumerate(eval_loader()):
  x_data = data[0]
  y_data = data[1]
  predicts = model_eval(x_data)
  acc = paddle.metric.accuracy(predicts, y_data)
  accs.append(acc.numpy()[0])
print('模型在验证集上的准确率为：',np.mean(accs))
```

实践十九：基于 ResNet-50 实现 CIFAR10 数据集分类

步骤 1：CIFAR10 数据集介绍与使用

本实验使用的是 CIFAR-10 数据集，该数据集是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适对象的小型数据集。数据集中一共包含 10 个类别的 RGB 彩色图片：飞机（airplane）、汽车（automobile）、鸟类（bird）、猫（cat）、鹿（deer）、狗（dog）、蛙类（frog）、马（horse）、船（ship）和卡车（truck）。每张图片的尺寸为 32×32 ，每个类

别有 6000 个图像，数据集中一共有 50 000 张训练图片和 10 000 张测试图片。CIFAR-10 数据是一个非常经典的数据集，飞桨深度学习平台中对该数据集进行了内置。因此，调用飞桨提供的 paddle.vision.datasets.Cifar10()接口就可以直接使用该数据集。paddle.vision.datasets.Cifar10()接口的参数 mode 用来设定选择加载训练数据或测试数据，参数 transform 用来设定图像的预处理方式。ToTensor()函数可以将 PIL.Image 或 numpy.ndarray 转换成 paddle.Tensor。Cifar10 数据加载的代码如下所示：

```
train_dataset = paddle.vision.datasets.Cifar10(mode = 'train', transform = ToTensor())
eval_dataset = paddle.vision.datasets.Cifar10(mode = 'test',
                transform = ToTensor())
```

步骤 2：ResNet-50 模型

ResNet 全名 Residual Network 残差网络。经典的 ResNet 结构有 ResNet18、ResNet34、ResNet50 等等，其结构如图 5-23 所示。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7,64, stride2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3,64; 3×3,64]×2	[3×3,64; 3×3,64]×3	[1×1,64; 3×3,64; 1×1,256]×3	[1×1,64; 3×3,64; 1×1,256]×3	[1×1,64; 3×3,64; 1×1,256]×3
conv3_x	28×28	[3×3,128; 3×3,128]×2	[3×3,128; 3×3,128]×4	[1×1,128; 3×3,128; 1×1,512]×4	[1×1,128; 3×3,128; 1×1,512]×4	[1×1,128; 3×3,128; 1×1,512]×8
conv4_x	14×14	[3×3,256; 3×3,256]×2	[3×3,256; 3×3,256]×6	[1×1,256; 3×3,256; 1×1,1024]×6	[1×1,256; 3×3,256; 1×1,1024]×23	[1×1,256; 3×3,256; 1×1,1024]×36
conv5_x	7×7	[3×3,512; 3×3,512]×2	[3×3,512; 3×3,512]×3	[1×1,512; 3×3,512; 1×1,2048]×3	[1×1,512; 3×3,512; 1×1,2048]×3	[1×1,512; 3×3,512; 1×1,2048]×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 5-23　ResNet 网络组成

本节实验使用 ResNet50 结构。在 ResNet50 结构中，首先是一个卷积核大小为 7×7 的卷积层；接下来是 4 个 Block 结构，其中每个 block 都包含 3 个卷积层，具体参数如图 5-23 所示；最后是一个用于分类的全连接层。

飞桨深度学习平台对于计算机视觉领域内置集成了很多经典模型，可以通过如下代码进行查看：

```
print('飞桨内置网络:', paddle.vision.models.__all__)
```

通过查看结果，可以看到 ResNet50 已经内置于 paddle.vision 中，通过如下代码可以直接获取模型实例：

```
model = paddle.vision.models.resnet50()                    # 获取模型实例
paddle.summary(model,(1,3,32,32))                          # 打印模型参数结构
```

步骤 3：模型训练与评估

对于模型的训练和评估，除了之前介绍的基础方式外，飞桨深度学习平台提供了便捷的高层 API，本实验的模型训练与评估就使用高层 API 对模型进行训练和评估。

首先，需要用 paddle.Model()方法封装实例化的模型：

```
# 用 Model 封装模型
model = paddle.Model(model)
```

然后，通过 Model 对象的 prepare 方法对优化方法、损失函数、评估方法进行设置：

```
# 定义损失函数
model.prepare(optimizer = paddle.optimizer.Adam(parameters = model.parameters()), loss =
              paddle.nn.CrossEntropyLoss(), metrics = paddle.metric.Accuracy())
```

最后，通过 Model 对象的 fit 方法对训练数据、验证数据、训练轮次、批次大小进行加载、日志打印、模型保存等参数进行设置，并进行模型训练和评估。具体代码如下所示：

```
# 启动模型全流程训练
model.fit(train_dataset,                    # 训练数据集
  eval_dataset,                             # 评估数据集
  epochs = epoch_num,                       # 总的训练轮次
  batch_size = batch_size,                  # 批次计算的样本量大小
  shuffle = True,                           # 是否打乱样本集
  verbose = 1,                              # 日志展示格式
  save_dir = './chk_points/',               # 分阶段的训练模型存储路径
  )
# 运行输出结果示例如图 5-24 所示
```

```
The loss value printed in the log is the current step, and the metric is the average value of previous step.
Epoch 1/1
step 782/782 [==============================] - loss: 1.1956 - acc: 0.5071 - 106ms/step
save checkpoint at /home/aistudio/chk_points/0
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 157/157 [==============================] - loss: 1.7141 - acc: 0.4521 - 90ms/step
Eval samples: 10000
save checkpoint at /home/aistudio/chk_points/final
```

图 5-24　训练过程部分输出

我们也可以单独调用 Model 对象的 evaluate 方法对模型进行评估，代码如下：

```
model.evaluate(eval_dataset, batch_size = batch_size, verbose = 1)
# 运行结果示例如图 5-25 所示
```

```
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step
step 157/157 [==============================] - loss: 1.7141 - acc: 0.4521 - 65ms/step
Eval samples: 10000
{'loss': [1.7140898], 'acc': 0.4521}
```

图 5-25　验证结果