第 5 章　计算机视觉基础实践

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。

本章中将通过实践的方式介绍图像数据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。


图像数据预处理实践

5.1　实践一: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 255,0 是黑色,255 是白色,中间值是一些不同等级的灰色。图 5-1-1 展示了灰度图像素值与颜色的变化。

(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。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)3 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有颜色,是目前运用最广的颜色系统之一。图 5-1-2 展示了三通道图的可视化效果。

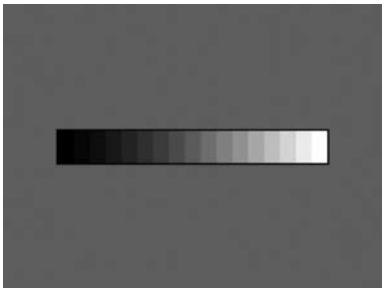
图 5-1-1　灰度图像素值与颜色变化


图 5-1-2　三通道图(RGB)

(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：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。

cv2：一个计算机视觉库，实现了图像处理和计算机视觉方面的很多通用算法。

下面介绍不同通道图像的读取。

(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。

```
# 引入依赖包
import numpy as np
from PIL import Image
# 读取单通道图像
img = Image.open('work/lena_gray.png')
display(img)
print(img)
print(img.size)
```

输出结果如图 5-1-3 所示。

图 5-1-3　图像读取

然后，使用 np.arrary()将图像转化为像素矩阵，既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。

```
# 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)
```

输出结果如图 5-1-4 所示。

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 5-1-5 所示。

```
图像尺寸:  (512, 512)
图像矩阵:
 [[162 162 162 ... 170 155 128]
 [162 162 162 ... 170 155 128]
 [162 162 162 ... 170 155 128]
 ...
 [ 43  43  50 ... 104 100  98]
 [ 44  44  55 ... 104 105 108]
 [ 44  44  55 ... 104 105 108]]
```

图 5-1-4　图像大小与像素值输出

```
162 162 162 161 162 157 163 161 166 162 162
162 162 162 161 162 157 163 161 166 162 162
162 162 162 161 162 157 163 161 166 162 162
162 162 162 161 162 157 163 161 166 162 162
162 162 162 161 162 157 163 161 166 162 162
164 164 158 155 161 159 159 160 161 160 155
160 160 163 158 160 162 159 156 159 163 158
159 159 155 157 158 159 156 157 159 162 162
155 155 158 158 159 160 157 157 163 157 159
```

图 5-1-5　图像数字格式

(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。

```
# 读取彩色图像
img = Image.open('work/lena.png')
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)
```

输出结果如图 5-1-6 所示。

```
<PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=532x528 at 0x7F1128D18A90>
图像尺寸:  (528, 532, 3)
图像矩阵:
 [[[236 242 255]
  [236 242 255]
  [236 242 254]
  ...
  [238 242 251]
  [238 242 251]
  [238 242 251]]
```

图 5-1-6　三通道图像读取

通过运行结果中的输出，可以看出单通道图像与三通道图像的不同，单通道图像的 mode 为“L”，三通道图像 mode 为“RGB”，三通道图像相比于单通道图像增加了一个维度。

步骤 1：彩色图像通道分离

上面已经介绍了彩色三通道图像的读取。彩色图的 RBG 3 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。这里既可以使用 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 3 个颜色通道的数据。

```
# 读取彩色图像
img = Image.open('work/lena.png')
# 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 3 个通道的图像如图 5-1-7 所示。

图 5-1-7　3 个通道图像

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('work/lena.png')

# 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 5-1-8 所示。

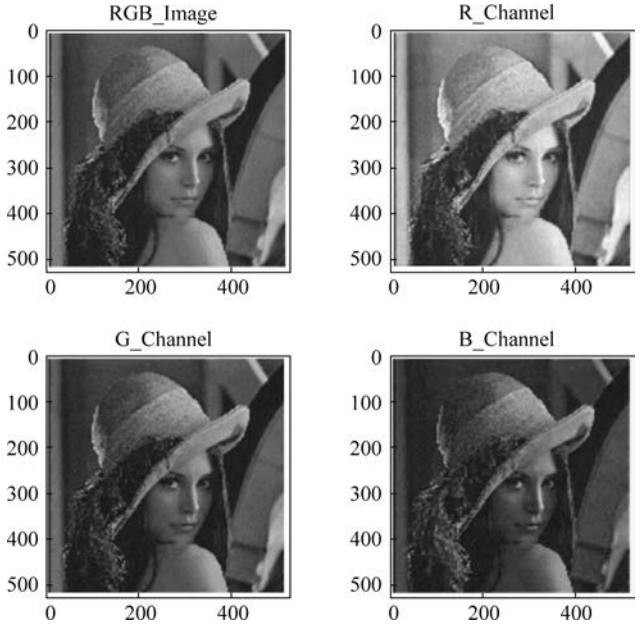

图 5-1-8　各通道图像及合并后的图像

步骤 2：图像的通道转换

上一小节中提前使用了 OpenCV 读取图片并进行通道分离，本节将会对 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化为灰度图像。

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）以及灰度到 RGB 或 BGR（COLOR_GRAY2RGB，COLOR_GRAY2BGR）。

（1）BGR 图像转灰度图像。

```
import numpy as np
import cv2
img = cv2.imread('work/lena.png')                    # 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(528,532,3)。

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()将图像保存。结果如图 5-1-9 所示。

图 5-1-9　灰度图保存结果

```
# 保存灰度图
cv2.imwrite('img_gray.jpg', img_gray)
```

（2）GBR 图像转为 RGB 图像。

将 GBR 图像转为 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('work/lena.png', 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像拼接在一起成为一张图像。本节实践先将一张图像从中间切割成两张图像，然后再进行拼接。

首先，我们用 cv2.imread()读取图 5-1-10，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像。

```
img = cv2.imread('work/test.png')
sum_rows = img.shape[0]
print(img.shape)
sum_cols = img.shape[1]
```

图 5-1-10　原始实践图

```
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))

# 显示分割后图像
plt.subplot(121)
plt.imshow(part1)
plt.title('Image1')
plt.subplot(122)
plt.imshow(part2)
plt.title('Image2')
```

分割后的图像如图 5-1-11 所示。

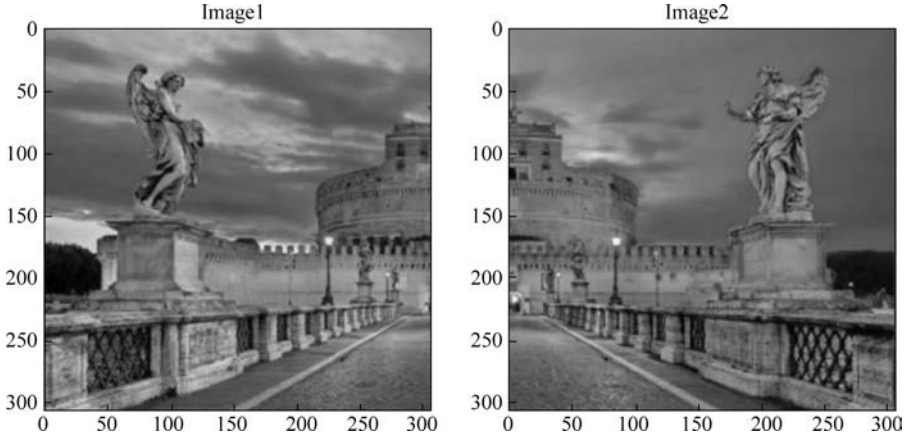

图 5-1-11　分割后的图像

然后，尝试将两张图像进行拼接。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），再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。

```
final_matrix = np.zeros((308, 614, 3), np.uint8)

final_matrix[0:308, 0:307] = part1
final_matrix[0:308, 307:614] = part2
plt.subplot(111)
```

```
plt.imshow(final_matrix)
plt.savefig('final_img.png')
plt.title('final_img')
```

拼接后的图像如图 5-1-12 所示。

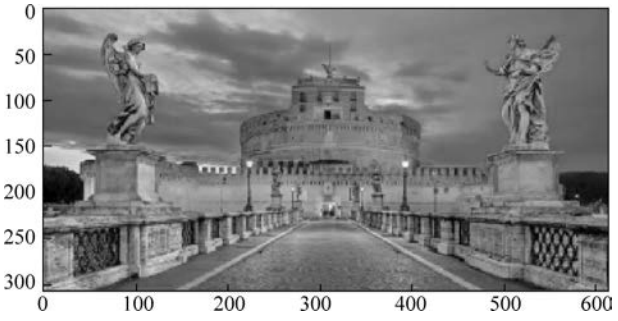

图 5-1-12　拼接后的图像

缩放图像就是调整图像的大小。在本节实践中我们使用 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 轴的缩放系数,interpolation 为缩放插入方法。

cv2.resize()提供了多种图像缩放插入方法,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,读取一张图像并将其转化为 RGB 格式;然后使用 cv2.resize()将图像进行缩放,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。通过输出图像可以看到,将一张尺寸为 308×614 的图像缩放成了 224×224。

```
img = cv2.imread('work/test.png')
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
print(img1.shape)
# 按照指定的宽度、高度缩放图像
img2 = cv2.resize(img1, (224, 224))
plt.figure(figsize=(12,12))
# 显示缩放前后的图像
plt.subplot(121)
plt.imshow(img1)
plt.title('Image1')
plt.subplot(122)
plt.imshow(img2)
plt.title('Image2')
```

缩放前后的图像如图 5-1-13 所示。

下面尝试通过设置 x、y 轴缩放系数来对图像进行缩放。我们尝试将 x 轴放大 5 倍,y 轴放大 2 倍。

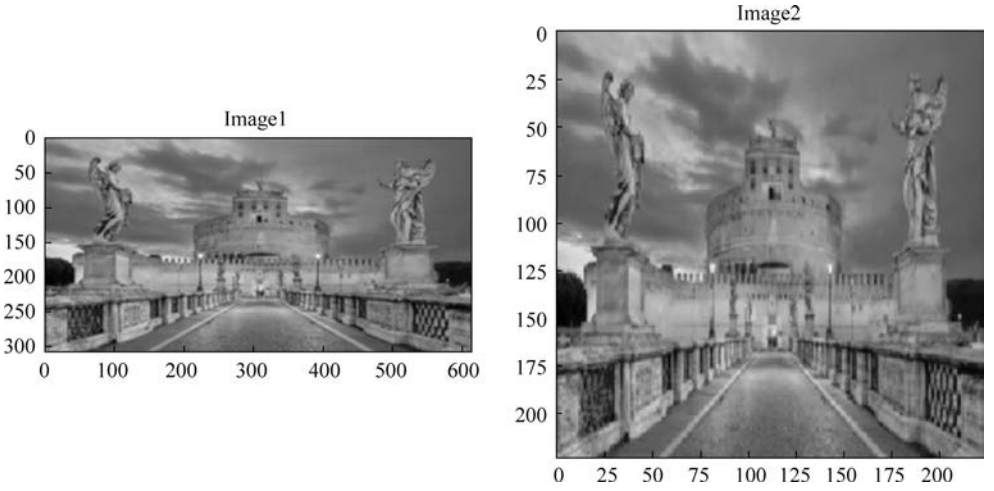

图 5-1-13　缩放前后的图像

```
# 按照比例缩放，如 x,y 轴均放大一倍
img3 = cv2.resize(img, None, fx = 5, fy = 2, interpolation = cv2.INTER_LINEAR)
plt.imshow(img3)
plt.savefig('img3.png')
```

缩放后的效果如图 5-1-14 所示。

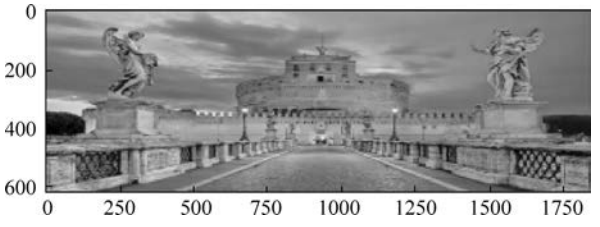

图 5-1-14　x、y 轴缩放后的图像

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 到 255，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的变成一类值，小于阈值的变成另一类值。

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，第一个是原图像，第二个是进行分类的阈值，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。

尝试对一张图像进行二值化。我们设置像素点的灰度值超过 127 则将该像素点的灰度值重新赋值为 255；灰度值小于 127 则将该像素点的灰度值重新赋值为 0。

```
# 原图
img = cv2.imread('work/lena.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 灰度图读入
img_gray = cv2.imread('work/lena.png', 0)
img_ = cv2.cvtColor(img_gray, cv2.COLOR_BGR2RGB)
ret, th = cv2.threshold(img_, 127, 255, cv2.THRESH_BINARY)

plt.figure(figsize=(12,12))
plt.subplot(121)
plt.imshow(img)
plt.title('Image1')
plt.subplot(122)
plt.imshow(th)
plt.title('Image2')
```

二值化前后的图像如图 5-1-15 所示。

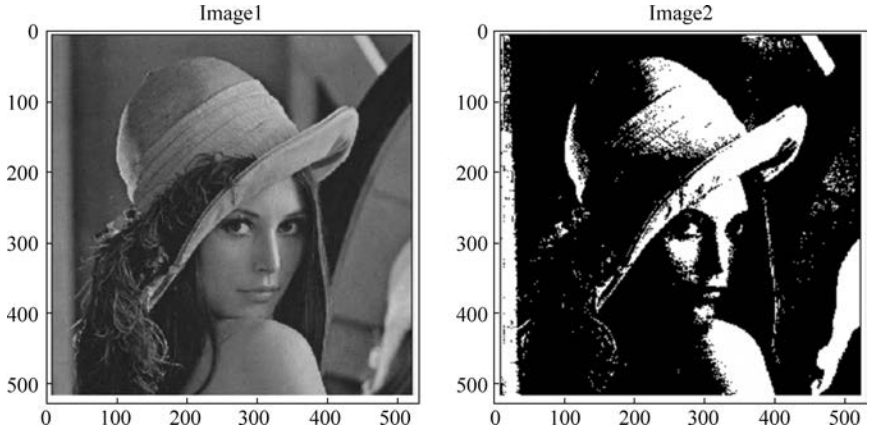

图 5-1-15　二值化前后的图像

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的 paddle.vision.transforms.normalize()方法可以方便地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。

下面尝试对一幅三通道图像进行归一化处理。我们设置图像三个通道的归一化后的均值为 127.5，图像三个通道的标准差为 127.5，图像的格式为“HWC”。

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

img = np.asarray(Image.open('work/lena.png'))

mean = [127.5, 127.5, 127.5]
std = [127.5, 127.5, 127.5]

normalized_img = F.normalize(img, mean, std, data_format='HWC')
print(normalized_img)
```

输出结果如图 5-1-16 所示，可以看到，图像经过归一化后的像素值在 0 至 1。

```
[[[0.8509804  0.8980392  1.       ]
  [0.8509804  0.8980392  1.       ]
  [0.8509804  0.8980392  0.99215686]
  ...
  [0.8666667  0.8980392  0.96862745]
  [0.8666667  0.8980392  0.96862745]
  [0.8666667  0.8980392  0.96862745]]
```

图 5-1-16　归一化前后的图像

步骤 6：图像增强

Gamma 变换采用了非线性函数（指数函数）对图像的灰度值进行幂次方变换，其作用是提升图像暗部细节，可以将漂白（相机曝光）或过暗（曝光不足）的图像，进行矫正。数学公式如下：

$$\boldsymbol{V}_{\text{out}} = A\boldsymbol{V}_{\text{in}}^{\gamma}$$

其中，$\boldsymbol{V}_{\text{in}}$ 是归一化后的图像矩阵，因此像素点取值范围为 0～1，$\boldsymbol{V}_{\text{out}}$ 是经过 Gamma 变换后的像素点矩阵，A 为一个常数，γ 指数为 Gamma。当 Gamma＞1 时，会减小灰度级较高的地方，增大灰度级较低的地方；当 Gamma＜1 时，会增大灰度级较高的地方，减小灰度级较低的地方。

定义函数 gamma_transfer()函数实现对图像的增强，首先用 PIL.Image 读取图像，之后使用 numpy.power()方法对归一化的图像数据进行幂次变换，并将变换前后的图像进行展示。

```
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# 图像增强，Gamma 变换采用了非线性函数（指数函数）对图像的灰度值进行变换
# 当 Gamma＞1 时，会减小灰度级较高的地方，增大灰度级较低的地方；当 Gamma＜1 时，会增大灰度级
  较高的地方，减小灰度级较低的地方
# Gamma 变换对像素值做的是幂次方变换，主要是图像的灰度级发生改变
def gama_transfer(path, power1=1):
    img = np.array(Image.open(img_path))

    img_g = 255*np.power(img/255, power1)
    img_g = np.around(img_g)
    img_g[img_g>255] = 255
    out = img_g.astype(np.uint8)
```

```
    plt.figure(figsize=(12,12))
    plt.subplot(121)
    plt.imshow(img)
    plt.title('Image1')
    plt.subplot(122)
    plt.imshow(out)
    plt.title('Image2')

img_path = 'work/lena.png'
gama_transfer(img_path, 2)
```

图像增强前后的效果如图 5-1-17 所示。

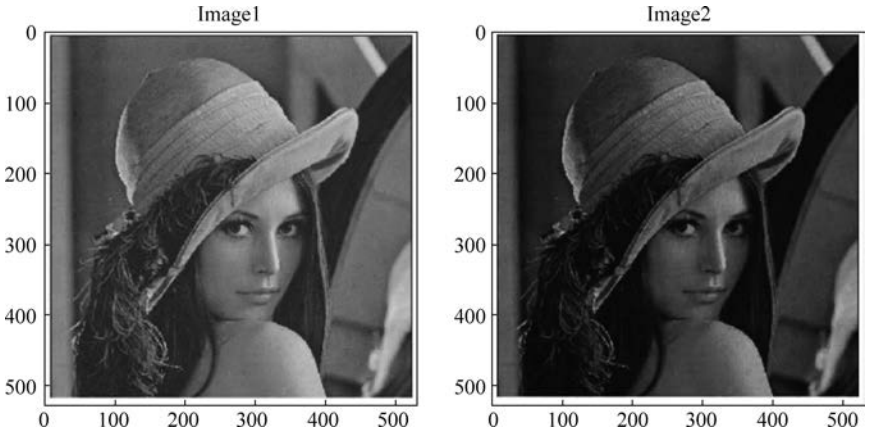

图 5-1-17　图像增强前后的效果

5.2　实践二：基于卷积神经网络实现宝石分类

基于卷积神经网络实现宝石分类

本节实践中，我们使用卷积神经网络(CNN)解决宝石图像的分类问题。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。

本实践代码运行的环境为 Python 3.7、Paddle 2.0，实践的平台为 AI Studio。

步骤 1：数据加载及预处理

宝石分类是一个图像多分类任务，旨在针对所给宝石图像，判断其所属的标签类型。

开源宝石数据集中包含 25 种宝石类别，每个类别的图像被单独保存在一个文件夹下，文件夹命名为其类型名。所有的类型如图 5-2-1 所示。

```
Alexandrite   Carnelian   Emerald     Iolite       Malachite     Rhodochrosite  Zircon
Almandine     Cats Eye    Fluorite    Jade         Onyx Black    Sapphire Blue
Benitoite     Danburite   Garnet Red  Kunzite      Pearl         Tanzanite
Beryl Golden  Diamond     Hessonite   Labradorite  Quartz Beer   Variscite
```

图 5-2-1　宝石分类数据集类型

部分宝石图像如图 5-2-2 所示。

图 5-2-2　部分宝石分类数据集

本实践使用的数据集包含 800 余幅格式为 jpg 的三通道彩色图像。对于本实践中的数据包，具体处理及加载方式与车辆图像分类实践基本相同（代码可参考本书第 4.2 节内容），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 5-2-3 所示。

然后，定义 get_data_list()函数遍历文件夹和图像，按照一定的比例将数据划分为训练集和验证集，并生成 train.txt 以及 eval.txt，文件内的格式为“图像路径 标签类别”，部分内容如图 5-2-4 所示。

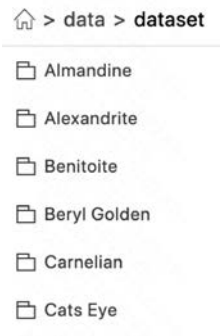

图 5-2-3　解压后的数据集文件夹结构

```
/home/aistudio/data/dataset/Hessonite/hessonite_16.jpg  2
/home/aistudio/data/dataset/Benitoite/benitoite_15.jpg  6
/home/aistudio/data/dataset/Tanzanite/tanzanite_11.jpg  1
/home/aistudio/data/dataset/Danburite/danburite_16.jpg  18
/home/aistudio/data/dataset/Emerald/emerald_35.jpg  16
/home/aistudio/data/dataset/Sapphire Blue/sapphire blue_21.jpg  3
/home/aistudio/data/dataset/Almandine/almandine_17.jpg  23
/home/aistudio/data/dataset/Jade/jade_1.jpg 13
```

图 5-2-4　train.txt 以及 eval.txt 中的部分内容

接下来，定义一个数据加载器 GemReader，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。具体代码包括：

__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 GemReader(Dataset):
    def __init__(self, data_path, mode = 'train'):
```

```
        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):
        # 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((224, 224), 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)
```

之后，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 = GemReader('/home/aistudio/',mode='train')
train_loader = paddle.io.DataLoader(train_dataset, batch_size=16, shuffle=True)
# 测试数据加载
eval_dataset = GemReader('/home/aistudio/',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模型

本实践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 224×224，经过三次“卷积—池化”操作，最后连接一个全连接层作为预测层。具体模型结构如图 5-2-5 所示。

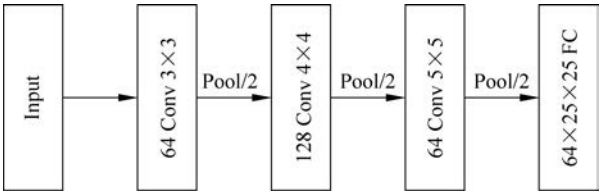

图 5-2-5　自定义卷积神经网络的结构

在了解了本节实践的网络结构后，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。本节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：

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

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

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。本节实践采用最大池化 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，则它必须包含两个整数值 (pool_size_Height，pool_size_Width)；如果它是一个整数，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，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，则可以是“VALID”或者“SAME”，表示填充算法；如果它是一个元组或列表，则可以有 3 种格式。①包含 2 个整数值，即[pad_height，pad_width]；②包含 4 个整数值，即[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)指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,默认值为 NCHW。

详细介绍了各个卷积类与池化类之后,就可以实现分类算法,如下所示。

```
# 定义卷积神经网络实现宝石识别
class MyCNN(nn.Layer):
    def __init__(self):
        super(MyCNN,self).__init__()
        self.conv0 = nn.Conv2D(in_channels = 3,out_channels = 64, kernel_size = 3, stride = 1)
        self.pool0 = nn.MaxPool2D(kernel_size = 2,stride = 2)

        self.conv1 = nn.Conv2D(in_channels = 64, out_channels = 128, kernel_size = 4, stride = 1)
        self.pool1 = nn.MaxPool2D(kernel_size = 2,stride = 2)

        self.conv2 = nn.Conv2D(in_channels = 128,out_channels = 64,kernel_size = 5)
        self.pool2 = nn.MaxPool2D(kernel_size = 2,stride = 2)

        self.fc1 = nn.Linear(in_features = 64 * 25 * 25,out_features = 25)

    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 = paddle.reshape(x,shape = [ - 1,64 * 25 * 25])
        y = self.fc1(x)

        return y
```

步骤 3:模型训练与评估

上一小节中我们已经定义好 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.Adagrad 优化器进行参数梯度的计算。

```
model = MyCNN()                                                    # 模型实例化
model.train()                                                      # 训练模式
cross_entropy = paddle.nn.CrossEntropyLoss()
opt = paddle.optimizer.Adagrad(learning_rate = train_parameters['learning_strategy']['lr'],
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)             # 计算精度
        if batch_id!= 0 and batch_id % 5 == 0:
            Batch = Batch + 5
            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(),acc.numpy()))
        loss.backward()
        opt.step()
        opt.clear_grad()                                        # 重置梯度
paddle.save(model.state_dict(),'MyCNN')                         # 保存模型
```

训练过程中的部分输出如图 5-2-6 所示。

```
epoch:19,step:5,train_loss:[0.7246032],train_acc:[0.8125]
epoch:19,step:10,train_loss:[0.43328825],train_acc:[0.9375]
epoch:19,step:15,train_loss:[0.49731687],train_acc:[0.9375]
epoch:19,step:20,train_loss:[0.40940693],train_acc:[0.9375]
epoch:19,step:25,train_loss:[0.71382046],train_acc:[0.75]
epoch:19,step:30,train_loss:[0.5805962],train_acc:[0.875]
epoch:19,step:35,train_loss:[0.6296073],train_acc:[0.875]
epoch:19,step:40,train_loss:[0.45943573],train_acc:[0.875]
epoch:19,step:45,train_loss:[0.38437143],train_acc:[1.]
```

图 5-2-6　宝石分类模型训练过程中的部分输出

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。

```
# 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)
```

输出结果如图 5-2-7 所示。

```
当前模型在验证集上的准确率为: 0.75
```

图 5-2-7　宝石分类模型在验证集上的准确率

步骤 4：模型预测

在此步骤中，将训练好的宝石图像分类模型应用于验证集。首先将验证集解压缩，之后定义基本的图像处理函数，对输入图像进行预处理，最后加载训练好的模型，在验证模式下进行预测。

```
def unzip_infer_data(src_path,target_path):
    if(not os.path.isdir(target_path)):
        z = zipfile.ZipFile(src_path, 'r')
        z.extractall(path = target_path)
        z.close()

def load_image(img_path):
    img = Image.open(img_path)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    img = img.resize((224, 224), Image.BILINEAR)
    img = np.array(img).astype('float32')
    img = img.transpose((2, 0, 1))                          # HWC to CHW
    img = img/255                                           # 像素值归一化
    return img

infer_src_path = '/home/aistudio/data/data55032/archive_test.zip'
infer_dst_path = '/home/aistudio/data/archive_test'
unzip_infer_data(infer_src_path,infer_dst_path)
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

# 展示预测图片
infer_path = 'data/archive_test/alexandrite_3.jpg'
img = Image.open(infer_path)
plt.imshow(img)                                             # 根据数组绘制图像
plt.show()                                                  # 显示图像
infer_imgs = []                                             # 对预测图片进行预处理
infer_imgs.append(load_image(infer_path))
infer_imgs = np.array(infer_imgs)
label_dic = train_parameters['label_dict']
for i in range(len(infer_imgs)):
    data = infer_imgs[i]
    dy_x_data = np.array(data).astype('float32')
    dy_x_data = dy_x_data[np.newaxis,:, : ,:]
    img = paddle.to_tensor (dy_x_data)
    out = model(img)
    lab = np.argmax(out.numpy())                            # argmax():返回最大数的索引
    print("第{}个样本,被预测为:{},真实标签为:{}".format(i + 1,label_dic[str(lab)],infer_
path.split('/')[ - 1].split("_")[0]))
print("预测结束")
```

输出结果如图 5-2-8 所示。

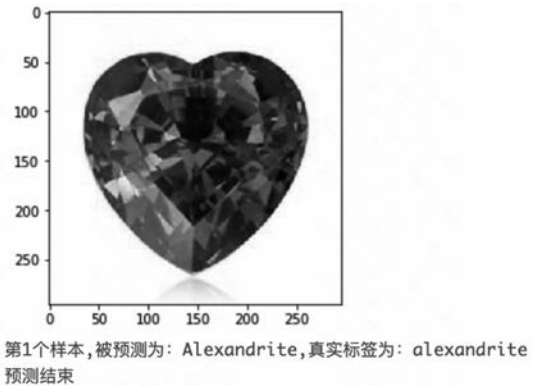

图 5-2-8　宝石分类模型的预测可视化结果

基于 VGGNet 网络模型实现美食分类

5.3　实践三：基于 VGGNet 网络模型实现美食分类

本次实践我们使用 VGG 网络模型解决美食分类问题。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 成功地搭建了 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。

本实践代码运行的环境为 Python 3.7、Python 2.0，实践的平台为 AI Studio。

步骤 1：美食分类数据集准备

本实践使用的数据集包含 5000 余幅格式为 jpg 的三通道彩色图像，共 5 种食物类别。对于本实践中的数据包，处理及加载的主要步骤如下。

首先，我们定义 unzip_data()对数据集的压缩包进行解压。

```
def unzip_data(src_path,target_path):
    '''
    解压原始数据集,将 src_path 路径下的 zip 包解压至 target_path 目录下
    '''
    if(not os.path.isdir(target_path + "foods")):
        z = zipfile.ZipFile(src_path, 'r')
        z.extractall(path = target_path)
        z.close()
```

解压后可以观察到数据集文件夹结构如图 5-3-1 所示。

然后，定义 get_data_list()遍历文件夹和数据集图像，按照一定比例将数据划分为训练集和验证集，并生成相应的 train.txt 以及 eval.txt，同时创建数据集的说明文件。

```
def get_data_list(target_path,train_list_path,eval_list_path):
    # 存放所有类别的信息
    class_detail = []
```

图 5-3-1　美食分类数据集的结构

```
    # 获取所有类别保存的文件夹名称
    data_list_path = target_path + "foods/"
    class_dirs = os.listdir(data_list_path)
    # 总的图像数量
    all_class_images = 0
    # 存放类别标签
    class_label = 0
    # 存放类别数目
    class_dim = 0
    # 存储要写进 eval.txt 和 train.txt 中的内容
    trainer_list = []
    eval_list = []
    # 读取每个类别
    for class_dir in class_dirs:
        if class_dir != ".DS_Store":
            class_dim += 1
            # 每个类别的信息
            class_detail_list = {}
            eval_sum = 0
            trainer_sum = 0
            # 统计每个类别有多少幅图像
            class_sum = 0
            # 获取类别路径
            path = data_list_path  + class_dir
            # 获取所有图像
            img_paths = os.listdir(path)
            for img_path in img_paths:                          # 遍历文件夹下的每幅图像
                name_path = path + '/' + img_path               # 每幅图像的路径
                if class_sum % 8 == 0:                          # 每 8 幅图像取一个做验证数据
                    eval_sum += 1                               # test_sum 为测试数据的数目
                    eval_list.append(name_path + "\t%d" % class_label + "\n")
                else:
                    trainer_sum += 1
                    trainer_list.append(name_path + "\t%d" % class_label + "\n")
                                                                # trainer_sum:测试数据的数目
                class_sum += 1                                  # 每类图像的数目
                all_class_images += 1                           # 所有类图像的数目

            # 说明的 json 文件的 class_detail 数据
            class_detail_list['class_name'] = class_dir      # 类别名称
            class_detail_list['class_label'] = class_label   # 类别标签
            class_detail_list['class_eval_images'] = eval_sum
class_detail_list['class_trainer_images'] = trainer_sum      # 该类数据的训练集数目
            class_detail.append(class_detail_list)
            # 初始化标签列表
```

```
            train_parameters['label_dict'][str(class_label)] = class_dir
            class_label += 1
    # 初始化分类数
    train_parameters['class_dim'] = class_dim
    # 乱序
    random.shuffle(eval_list)
    with open(eval_list_path, 'a') as f:
        for eval_image in eval_list:
            f.write(eval_image)
    random.shuffle(trainer_list)
    with open(train_list_path, 'a') as f2:
        for train_image in trainer_list:
            f2.write(train_image)

    # 说明的 JSON 文件信息
    readjson = {}
    readjson['all_class_name'] = data_list_path                    # 文件父目录
    readjson['all_class_images'] = all_class_images
    readjson['class_detail'] = class_detail
    jsons = json.dumps(readjson, sort_keys = True, indent = 4, separators = (',', ': '))
    with open(train_parameters['readme_path'],'w') as f:
        f.write(jsons)
    print ('生成数据列表完成!')
```

train.txt 中训练样本的格式如图 5-3-2 所示。

```
/home/aistudio/data/foods/beef_carpaccio/560891.jpg 2
/home/aistudio/data/foods/baby_back_ribs/1807563.jpg 4
/home/aistudio/data/foods/beef_carpaccio/3769247.jpg 2
/home/aistudio/data/foods/apple_pie/2598068.jpg 1
/home/aistudio/data/foods/apple_pie/3555118.jpg 1
/home/aistudio/data/foods/baklava/1133981.jpg 0
/home/aistudio/data/foods/apple_pie/2600379.jpg 1
```

图 5-3-2　美食分类数据集 train.txt 中训练样本的格式

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。

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

class FoodDataset(paddle.io.Dataset):
    """
    5 类 food 数据集类的定义
    """
    def __init__(self, mode = 'train'):
        """
        初始化函数
        """
        self.data = []
        with open('data/{}.txt'.format(mode)) as f:
            for line in f.readlines():
                info = line.strip().split('\t')
```

```
                if len(info) > 0:
                    self.data.append([info[0].strip(), info[1].strip()])

    def __getitem__(self, index):
        """
        根据索引获取单个样本
        """
        image_file, label = self.data[index]
        image = Image.open(image_file)
        if image.mode != 'RGB':
            image = image.convert('RGB')
        image = image.resize((224, 224), Image.BILINEAR)
        image = np.array(image).astype('float32')
        image = image.transpose((2, 0, 1)) / 255
        return image, np.array(label, dtype = 'int64')

    def __len__(self):
        """
        获取样本总数
        """
        return len(self.data)
```

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，同时打印训练集和验证集的样本数量。

```
train_dataset = FoodDataset(mode = 'train')
train_loader = paddle.io.DataLoader(train_dataset, batch_size = 16, shuffle = True)
eval_dataset = FoodDataset(mode = 'eval')
eval_loader = paddle.io.DataLoader(eval_dataset, batch_size = 8, shuffle = False)

print("训练集样本数量为:", train_dataset.__len__())
print("验证集样本数量为:", eval_dataset.__len__())
```

输出结果如图 5-3-3 所示，可以看出，训练集总共包含 4375 个样本，验证集总共包含 625 个样本。

```
训练集样本数量为:  4375
验证集样本数量为:  625
```

图 5-3-3　美食分类数据集的划分结果

步骤 2：VGG 网络模型搭建

VGGNet 引入“模块化”的设计思想，将不同的层进行简单的组合构成网络模块，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。VGGNet 中的经典模型包含 VGG-16 和 VGG-19。以 VGG-16 为例，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，其通道数增加一倍。最后通过全连接与 Softmax 层输出结果。

（1）VGG-16 网络模型。

VGG-16 整体包含 16 层，其网络结构如图 5-3-4 所示。

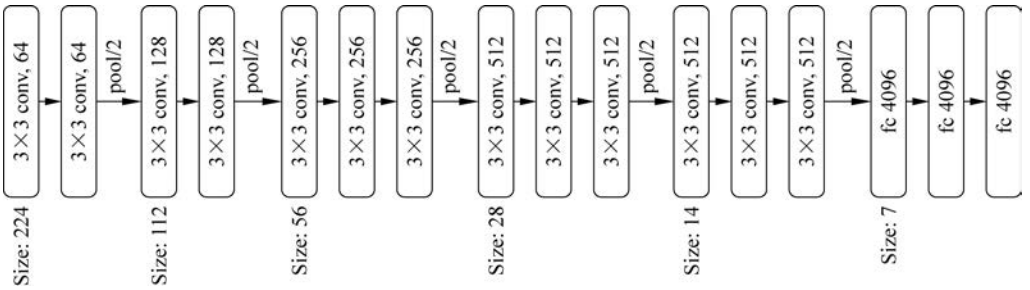

图 5-3-4　VGG-16 模型结构

在具体实现过程中，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，使用了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 类的实现如下所示。

```
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。

```
class VGG16(paddle.nn.Layer):
    def __init__(self):
        super(VGG16, self).__init__()
        # 3 为通道数,64 为卷积核个数,3 为卷积核大小,2 为池化核大小,2 为池化步长,2 为连续卷积个数
        self.convpool01 = ConvPool(3, 64, 3, 2, 2, 2)
        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):
        """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:
            label = paddle.unsqueeze(label, axis = - 1)
            acc = paddle.metric.accuracy(input = out, label = label)
            return out, acc
        else:
            return out
```

(2) VGG-19 网络模型。

VGG-19 相比于 VGG-16 增加了三层卷积层，同样使用 Convpool 模块定义 VGG-19 网

络模型。

```
class VGG19(paddle.nn.Layer):
    def __init__(self):
        super(VGG19, self).__init__()
        self.convpool01 = ConvPool(3, 64, 3, 2, 2, 2)
        self.convpool02 = ConvPool(64, 128, 3, 2, 2, 2)
        self.convpool03 = ConvPool(128, 256, 3, 2, 2, 4)
        self.convpool04 = ConvPool(256, 512, 3, 2, 2, 4)
        self.convpool05 = ConvPool(512, 512, 3, 2, 2, 4)
        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):
        """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:
            label = paddle.unsqueeze(label, axis = - 1)
            acc = paddle.metric.accuracy(input = out, label = label)
            return out, acc
        else:
            return out
```

步骤 3：模型训练与评估

第 5.2 节中我们已经定义好 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, beta1, beta2, epsilon, parameters, weight_decay, grad_clip, name, lazy_mode)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 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。以 VGG-16 为例的训练代码如下所示。

```
# VGG-16 模型训练
model1 = VGG16()
model1.train()
cross_entropy = paddle.nn.CrossEntropyLoss()
optimizer = paddle.optimizer.Adam(learning_rate = train_parameters['learning_strategy']['lr'],
parameters = model1.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 = model1(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(model1.state_dict(),save_path)

paddle.save(model1.state_dict(),train_parameters["checkpoints"] + "/" + "save_dir_final.
pdparams")
```

训练过程的部分输出如图 5-3-5 所示。

```
epo: 0, step: 200, loss is: [1.444011], acc is: [0.3125]
save model to: /home/aistudio/work/checkpoints/save_dir_200.pdparams
epo: 1, step: 400, loss is: [1.2611442], acc is: [0.5625]
save model to: /home/aistudio/work/checkpoints/save_dir_400.pdparams
epo: 2, step: 600, loss is: [0.81553876], acc is: [0.6875]
save model to: /home/aistudio/work/checkpoints/save_dir_600.pdparams
epo: 2, step: 800, loss is: [0.71179104], acc is: [0.625]
save model to: /home/aistudio/work/checkpoints/save_dir_800.pdparams
```

图 5-3-5　美食分类模型训练过程中的部分输出

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，以 VGG-16 为例的评估代码如下所示。

```
model1__state_dict = paddle.load('work/checkpoints/save_dir_final.pdparams')
model1_eval = VGG16()
model1_eval.set_state_dict(model1__state_dict)
model1_eval.eval()
accs1 = []

for _, data in enumerate(eval_loader()):
    x_data = data[0]
    y_data = data[1]
    predicts1 = model1_eval(x_data)
    y_data = paddle.unsqueeze(y_data, axis = -1)
    acc1 = paddle.metric.accuracy(predicts1, y_data)
    accs1.append(acc1.numpy()[0])
print('模型在验证集上的准确率为:',np.mean(accs1))
```

VGG-16 与 VGG-19 的评估结果如图 5-3-6 和图 5-3-7 所示。可以看到，VGG-19 由于具有更深的网络层数，因此在验证集上的准确率更高。

```
模型在验证集上的准确率为: 0.5949367
```

图 5-3-6　VGG-16 模型在验证集上的准确率

```
模型在验证集上的准确率为: 0.6075949
```

图 5-3-7　VGG-19 模型在验证集上的准确率

基于 ResNet 网络模型实现中草药分类

5.4　实践四：基于 ResNet 网络模型实现中草药分类

本实践使用 ResNet 网络模型实现中草药分类。ResNet 网络模型是于 2015 年由微软实践室中的何凯明等研究员提出，其致力于解决由于深度卷积神经网络层数加深带来的梯度消失问题。

本实践代码运行的环境为 Python 3.7、Paddle 2.0，实践的平台为 AI Studio。

步骤 1：中草药分类数据集准备

本实践使用的数据集包含 900 余幅格式为 jpg 的三通道彩色图像，共 5 种中草药类别。我们在 AI Studio 上提供了本实践的数据集压缩包 Chinese Medicine.zip。对于本实践中的数据包，具体处理与加载方式与美食分类实践类似（代码可参考第 5.3 节内容），主要步骤如下：

首先，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 5-4-1 所示。

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

图 5-4-1　中草药分类数据集的结构

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 train.txt 以及 eval.txt。train.txt 中训练样本的格式如图 5-4-2 所示。

```
/home/aistudio/data/Chinese Medicine/baihe/b (34).jpg   0
/home/aistudio/data/Chinese Medicine/jinyinhua/jyh_69.jpg   3
/home/aistudio/data/Chinese Medicine/gouqi/cgcjyfyj (84).jpg    2
/home/aistudio/data/Chinese Medicine/jinyinhua/jyh_52.jpg   3
/home/aistudio/data/Chinese Medicine/jinyinhua/jyh_96.jpg   3
```

图 5-4-2　中草药分类数据集 train.txt 中训练样本的格式

最后，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，并利用 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：ResNet 网络模型搭建

ResNet 全名 Residual Network，意为残差网络。经典的 ResNet 结构有 ResNet18、ResNet34、ResNet50 等，其结构如图 5-4-3 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 5-4-3　ResNet 网络结构

以 ResNet50 网络模型为例。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，具体参数如上表中所示；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型，可以通过如下代码进行查看。

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

输出结果如图 5-4-4 所示。

```
飞桨内置网络： ['ResNet', 'resnet18', 'resnet34', 'resnet50', 'resnet101', 'resnet152', 'VGG', 'vgg11', 'vgg13', 'vgg16', 'vgg19', 'MobileNet
V1', 'mobilenet_v1', 'MobileNetV2', 'mobilenet_v2', 'LeNet']
```

图 5-4-4　飞桨深度学习平台内置的经典模型

由此可以看出，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。

```
model1 = paddle.vision.models.resnet50()                    # 获取模型实例
paddle.summary(model1, (1, 3, 224, 224))                    # 打印模型参数结构

model2 = paddle.vision.models.resnet101()                   # 获取模型实例
paddle.summary(model2, (1, 3, 224, 224))                    # 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，本实践采用飞桨深度学习平台提供的便捷的高层 API 来实现。以 ResNet50 为例，首先，需要用 paddle.Model()方法封装实例化的模型。

```
model1 = paddle.Model(model1)
```

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。

```
model1.prepare(optimizer = paddle.optimizer.Adam(parameters = model1.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。

```
model1.fit(train_dataset,                                   # 训练数据集
    eval_dataset,                                           # 评估数据集
    epochs = 0,                                             # 总的训练轮次
    batch_size = 16,                                        # 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_points1/',                            # 分阶段的训练模型存储路径
    )
```

训练过程的部分输出如图 5-4-5 所示。

```
Epoch 9/10
step 50/50 [==============================] - loss: 0.6258 - acc: 0.9327 - 135ms/step
save checkpoint at /home/aistudio/chk_points1/8
Eval begin...
step 8/8 [==============================] - loss: 0.0217 - acc: 0.8261 - 98ms/step
Eval samples: 115
Epoch 10/10
step 50/50 [==============================] - loss: 0.2113 - acc: 0.9301 - 134ms/step
save checkpoint at /home/aistudio/chk_points1/9
Eval begin...
step 8/8 [==============================] - loss: 0.0418 - acc: 0.9217 - 100ms/step
Eval samples: 115
save checkpoint at /home/aistudio/chk_points1/final
```

图 5-4-5　中草药分类模型训练过程的部分输出

也可以单独调用 Model 对象的 evaluate 方法对模型进行评估。

```
model1.evaluate(eval_dataset, batch_size = 16, verbose = 1)
```

ResNet50 以及 ResNet101 的评估结果如图 5-4-6 与图 5-4-7 所示，可以看出，尽管 ResNet101 具有更深的层数，但是 ResNet50 取得了更准确的结果。

```
Eval begin...
step 8/8 [==============================] - loss: 0.0418 - acc: 0.9217 - 98ms/step
Eval samples: 115
{'loss': [0.041824978], 'acc': 0.9217391304347826}
```

图 5-4-6　ResNet50 模型的评估结果

```
Eval begin...
step 8/8 [==============================] - loss: 0.9725 - acc: 0.6261 - 101ms/step
Eval samples: 115
{'loss': [0.972527], 'acc': 0.6260869565217392}
```

图 5-4-7　ResNet101 模型的评估结果

5.5　实践五：基于 Faster RCNN 模型实现目标检测

基于 Faster RCNN 模型实现目标检测

在人工智能领域众多的热点任务之中，目标检测和图像分割算法发展迅速，一直占据着重要的地位，同时也是最具有挑战性的两大任务。区别于图像分类任务仅仅判断出目标实例的类别，目标检测任务不仅需要判断出目标实例的类别，同时也要给出目标实例的具体位置坐标。随着近年来深度学习技术的不断发展，早期的手工特征的方法被深度特征所取代，涌现出了大量的新的方法，在目标检测领域取得了显著性的突破。

当前的基于深度学习的检测方法根据检测的原理可以分为一阶段和两阶段的目标检测方法。一阶段的目标检测通常的作法是在特征图上根据设定不同的大小和长宽比首先预定义出一些候选边界框，之后对于两个子任务来说，定位任务是指对于候选边界框的中心位置以及长宽作回归矫正。两阶段的目标检测方法一般包含两个网络，一个是候选区域生成网络，另一个是目标检测以及识别网络。候选区域生成网络用于在特征图上自动生成候选边界框，之后定位和分类两个子任务就由检测以及识别网络完成。接下来具体介绍两种目标检测方法。

(1) 一阶段目标检测方法。

典型的一阶段的目标检测方法包括 YOLO、SSD(Single Shot Detector)以及 RetinaNet 模型等。一阶段的方法首先会在特征图的每个位置上根据不同的大小和长宽比预定义固定数量的候选边界框，之后再对候选边界框的中心位置和长宽进行回归矫正，并对每个候选边界框中包含的目标对象进行分类。一阶段目标检测方法的主要优点是其具有很高的计算效率，缺点是其检测的精度通常低于二阶段的检测方法，其中主要原因之一是预定义的候选边界框中可能大部分都包含着背景，只有一小部分包含前景区域，也就是说可能会产生类别不平衡的问题。

以 SSD 模型为例，其提出的出发点是希望能够在不影响太多检测精度的前提下实现实时的检测速度。SSD 设置了多种宽高比的锚点框以及数据增强策略，它有效地结合了 Faster R-CNN、YOLO 和多尺度卷积特性中的思想，能够在达到与当时最先进的两阶段的检测方法相当的检测精度的同时，达到实时检测的要求。SSD 从 YOLO 中继承了将检测转化为回归的思路，一次性地端到端完成目标的定位和分类任务。与 YOLO 一样，SSD

对预先设定好的固定数量的候选边界框进行类别预测和边界框的回归，并采用非极大值抑制的处理方法生成最终的目标检测结果。SSD 其前面几层的设计是基于标准网络结构，在去掉网络的分类层之后，将几个尺度逐渐减小的卷积层添加到标准网络的末端构成 SSD 的主干网络。对于不同大小的目标对象，SSD 能够在多个不同尺度的特征图上进行预测。

本节实践以两阶段的目标检测网络 Faster R-CNN 模型为例。通过调用 PaddleDetection 来完成目标检测任务。本节实践的平台为 AI Studio，实践环境为 Python 3.7。

（2）两阶段目标检测方法。

典型的两阶段的目标检测方法有 R-CNN、Fast R-CNN、Faster R-CNN、R-FCN 及 Cascade R-CNN 等。这类方法通常包含两个阶段，第一阶段使用一个候选区域生成网络，第二阶段使用一个检测和识别网络。第一阶段首先通过采用区域提议的方法来生成候选边界框（与类别无关），然后在第二阶段根据得到的候选边界框从特征图中得到相应的特征，之后使用检测和识别网络来进行回归矫正，包括中心位置和长宽的回归，以进一步修正得到的区域的位置，并完成分类任务，确定目标对象所属的类别标签。

以 Faster R-CNN 模型为例，其是 Fast R-CNN 的进阶版本，虽然 Fast R-CNN 在检测速度方面显著提升，同时具有较高的检测精度、端到端的训练过程以及不需要存储特征到磁盘的优点，但是它的候选边界框提取的过程仍然是分离的，仍然依赖独立于网络之外的选择性搜索的区域提议方法。区域提议成为 Fast R-CNN 的瓶颈。因此，一个高效准确的区域提议网络（Region Proposal Network，RPN）被提出，其通过全卷积神经网络来生成区域提议，替代之前的基于选择性搜索的生成候选框的方法。Faster R-CNN 框架图在最后一个卷积层后面插入一个 RPN，直接产生区域提议，不需要额外算法产生区域提议，在 RPN 之后，像 Fast R-CNN 中一样，使用 ROI 池化和后续的分类器以及回归器。其中，RPN 和 Fast R-CNN 共享大量的卷积层。RPN 首先在特征图的每个位置根据不同大小和宽高比初始化 k 个 $n \times n$ 的锚点框（也就是所谓的 anchor boxes，k 代表特征图中每个像素点提取的框的个数，n 为特征图的尺寸），这些锚点框是平移不变的，在每个位置处都相同，每个锚点框会被映射到一个低维向量，同时该低维向量被并行地送入两个全连接层，回归器为每个锚点框计算偏移，分类器评测每个锚点框是物体的可能性。Faster R-CNN 通过采用 RPN 来生成区域提议，同时与检测网络共享卷积部分，使得它在检测效果和检测速度上都能得以提升。

步骤 1：认识 PASCAL VOC 数据集

在 Faster RCNN 实践中，我们将使用在目标检测领域中十分著名和经典的数据集 PASCAL VOC 目标检测数据集。PASCAL VOC 目标检测数据集包含 20 个类别，被看成目标检测问题的一个基准数据集。

本实践主要研究 PASCAL VOC 2007 和 PASCAL VOC 2012 两部分。其中，VOC 2007 包含 9963 张图片，共 24640 个物体；VOC 2012 包含 11540 张图片，共 27450 个物体。数据集共有 20 个类 person，bird，cat，cow，dog，horse，sheep，aeroplane，bicycle，boat，bus，car，motorbike，train，bottle，chair，dining table，potted plant，sofa，tv/monitor。

以 2007 数据集为例，数据集格式如图 5-5-1 所示，JPEGImages 目录下存放的是所有的图片，Annotations 文件下存储的是与图片对应的 xml 标注文件，ImageSets 下含 3 个子文件夹 Layout、Main、Segmentation，其中，Main 存放的是数据集划分的文件，分别对应训练、验证和测试集。

```
├─ Annotations 进行 detection 任务时的标签文件，xml 形式，文件名与图片名一一对应
├─ ImageSets 包含三个子文件夹 Layout、Main、Segmentation，其中 Main 存放的是分类和检测的数据集分割文件
├─ JPEGImages 存放 .jpg 格式的图片文件
├─ SegmentationClass 存放按照 class 分割的图片
└─ SegmentationObject 存放按照 object 分割的图片

├─ Main
│   ├─ train.txt 写着用于训练的图片名称，  共 2501 个
│   ├─ val.txt 写着用于验证的图片名称，共 2510 个
│   ├─ trainval.txt train与val的合集，共 5011 个
│   ├─ test.txt 写着用于测试的图片名称，共 4952 个
```

图 5-5-1　PASCAL VOC 2007 数据集结构

PASCAL VOC 创建了一个经典的目标检测标注格式，如图 5-5-2 所示：每个 object 代表图像中的一个目标实例，name 中是目标的类别，bndox 中存储着目标的位置（左上角，右下角）。

PASCAL VOC 数据集可在其官网下载：

http：//host. robots. ox. ac. uk/pascal/VOC/

步骤 2：PaddleDetection

PaddleDetection 飞桨目标检测开发套件，旨在帮助开发者更快更好地完成检测模型的组建、训练、优化及部署等全开发流程。

PaddleDetection 模块化地实现了多种主流目标检测算法，提供了丰富的数据增强策略、网络模块组件（如骨干网络）、损失函数等，并集成了模型压缩和跨平台高性能部署能力。

经过长时间产业实践打磨，PaddleDetection 已拥有顺畅、卓越的使用体验，被工业质检、遥感图像检测、无人巡检、新零售、互联网、科研等十多个行业广泛使用。图 5-5-3 展示了 PaddleDetection 的应用。

PaddleDetection 具有以下特点：

模型丰富：包含目标检测、实例分割、人脸检测等 100＋个预训练模型，涵盖多种全球竞赛冠军方案。

使用简洁：模块化设计，解耦各个网络组件，可以使开发者轻松搭建、试用各种检测模型及优化策略，快速得到高性能、定制化的算法。

端到端打通：从数据增强、组网、训练、压缩、部署端到端打通，并完备支持云端/边缘端多架构、多设备部署。

高性能：基于飞桨的高性能内核，模型训练速度及显存占用优势明显。支持 FP16 训练，支持多机训练。

PaddleDetection 如图 5-5-4 所示，可以实现一阶段目标检测方法：SSD、YOLOv3 和 PP-YOLO 等；两阶段目标检测方法：Faster RCNN、FPN 和 Cascade RCNN 等；实例分割模型 Mask RCNN、SOLOv2，以及人脸检测模型 FaceBoxes 等。

```
<annotation>
    <folder>VOC2007</folder>
    <filename>000001.jpg</filename>  # 文件名
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
        <flickrid>341012865</flickrid>
    </source>
    <owner>
        <flickrid>Fried Camels</flickrid>
        <name>Jinky the Fruit Bat</name>
    </owner>
    <size>  # 图像尺寸，用于对 bbox 左上和右下坐标点做归一化操作
        <width>353</width>
        <height>500</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>  # 是否用于分割
    <object>
        <name>dog</name>  # 物体类别
        <pose>Left</pose>  # 拍摄角度：front, rear, left, right, unspecified
        <truncated>1</truncated>  # 目标是否被截断（比如在图片之外），或者被遮挡（超过15%）
        <difficult>0</difficult>  # 检测难易程度，这个主要是根据目标的大小，光照变化，图片质量来判断
        <bndbox>
            <xmin>48</xmin>
            <ymin>240</ymin>
            <xmax>195</xmax>
            <ymax>371</ymax>
        </bndbox>
    </object>
    <object>
        <name>person</name>
        <pose>Left</pose>
        <truncated>1</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>8</xmin>
            <ymin>12</ymin>
            <xmax>352</xmax>
            <ymax>498</ymax>
        </bndbox>
    </object>
</annotation>
```

图 5-5-2　数据标注实例

图 5-5-3　PaddleDetection 应用示例

Architectures	Backbones	Components	Data Augmentation

• Two-Stage Detection
  ○ Faster RCNN
  ○ FPN
  ○ Cascade-RCNN
  ○ Libra RCNN
  ○ Hybrid Task RCNN
  ○ PSS-Det
• One-Stage Detection
  ○ RetinaNet
  ○ YOLOv3
  ○ YOLOv4
  ○ PP-YOLO
  ○ SSD
• Anchor Free
  ○ CornerNet-Squeeze
  ○ FCOS
  ○ TTFNet
• Instance Segmentation
  ○ Mask RCNN
  ○ SOLOv2
• Face-Detction
  ○ FaceBoxes
  ○ BlazeFace
  ○ BlazeFace-NAS
	• ResNet(&vd)
	• ResNeXt(&vd)
	• SENet
	• Res2Net
	• HRNet
	• Hourglass
	• CBNet
	• GCNet
	• DarkNet
	• CSPDarkNet
	• VGG
	• MobileNetv1/v3
	• GhostNet
	• Efficientnet
		• Common
		  ○ Sync-BN
		  ○ Group Norm
		  ○ DCNv2
		  ○ Non-local
		• FPN
		  ○ BiFPN
		  ○ BFP
		  ○ HRFPN
		  ○ ACFPN
		• Loss
		  ○ Smooth-L1
		  ○ GIoU/DIoU/CIoU
		  ○ IoUAware
		• Post-processing
		  ○ SoftNMS
		  ○ MatrixNMS
		• Speed
		  ○ FP16 training
		  ○ Multi-machine training
			• Resize
			• Flipping
			• Expand
			• Crop
			• Color Distort
			• Random Erasing
			• Mixup
			• Cutmix
			• Grid Mask
			• Auto Augment

图 5-5-4　PaddleDetection 模型库

同时，PaddleDetection 集成了 VGG、ResNet、MobileNet、Efficientnet 等一系列经典和前沿的骨干网络以及一些网络中所需要的各种组件。除此之外，还集成了视觉任务的多种数据增强方式。

步骤 3：使用 PaddleDetection 实现目标检测

（1）准备环境。

首先需要通过 git 和 pip 命令下载并安装 PaddleDetection。

#PaddleDetection 的代码库下载，同时支持 github 源和 gitee 源，为了在国内网络环境更快下载，此处使用 gitee 源。

```
# ! git clone https://github.com/PaddlePaddle/PaddleDetection.git
! git clone https://gitee.com/paddlepaddle/PaddleDetection.git
%cd PaddleDetection
# 安装其他依赖
! pip install paddledet==2.0.1 -i https://mirror.baidu.com/pypi/simple
```

（2）数据下载。

通过执行 download_voc.py 下载数据。

```
! python PaddleDetection/dataset/voc/download_voc.py
```

（3）执行训练。

通过 train.py 可以直接开始网络的训练，其中涉及的参数如图 5-5-5 所示，需要给定配置文件的路径：配置文件中存储着网络训练过程涉及的一些超参数。除此之外，还可以通过给定 trian 文件传入参数设置是否验证、是否加载预训练模型、是否进行模型压缩等。

FLAG	支持脚本	用途	默认值	备注
-c	ALL	指定配置文件	None	必选，例如-c configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.yml
-o	ALL	设置或更改配置文件里的参数内容	None	相较于 -c 设置的配置文件有更高优先级，例如：-o use_gpu=False
--eval	train	是否边训练边测试	False	如需指定，直接 --eval 即可
-r/--resume_checkpoint	train	恢复训练加载的权重路径	None	例如：-r output/faster_rcnn_r50_1x_coco/10000
--slim_config	ALL	模型压缩策略配置文件	None	例如 --slim_config configs/slim/prune/yolov3_prune_l1_norm.yml
--use_vdl	train/infer	是否使用VisualDL记录数据，进而在VisualDL面板中显示	False	VisualDL需Python>=3.5
--vdl_log_dir	train/infer	指定 VisualDL 记录数据的存储路径	train: vdl_log_dir/scalar infer: vdl_log_dir/image	VisualDL需Python>=3.5
--output_eval	eval	评估阶段保存json路径	None	例如 --output_eval=eval_output，默认为当前路径
--json_eval	eval	是否通过已存在的bbox.json或者mask.json进行评估	False	如需指定，直接 --json_eval 即可，json文件路径在 --output_eval 中设置
--classwise	eval	是否评估单类AP和绘制单类PR曲线	False	如需指定，直接 --classwise 即可
--output_dir	infer/export_model	预测后结果或导出模型保存路径	./output	例如 --output_dir=output
--draw_threshold	infer	可视化时分数阈值	0.5	例如 --draw_threshold=0.7
--infer_dir	infer	用于预测的图片文件夹路径	None	--infer_img 和 --infer_dir 必须至少设置一个
--infer_img	infer	用于预测的图片路径	None	--infer_img 和 --infer_dir 必须至少设置一个，--infer_img 具有更高优先级

图 5-5-5　训练参数列表

```
! python tools/train.py -c \
./configs/faster_rcnn/faster_rcnn_r50_1x_coco.yml \
-- eval -- use_vdl = True -- vdl_log_dir = "./output"
```

（4）模型评估与预测。

通过执行 eval.py 开始验证模型，需要给定模型的配置文件和训练好的权重。

```
!python -u tools/eval.py \
-c ./configs/faster_rcnn/faster_rcnn_r50_1x_coco.yml  \
-o weights = output/faster_rcnn_r50_1x_coco/best_model.pdparams
```

通过执行 infer.py 开始用模型进行预测，需要给定模型的配置文件、训练好的权重和用于预测的图像。

```
!python tools/infer.py -c ./configs/faster_rcnn/faster_rcnn_r50_1x_coco.yml -o\
weights = output/faster_rcnn_r50_1x_coco/model_final.pdparams \
-- infer_img = dataset/roadsign_voc/images/road114.png
```

基于 U-Net 模型实现宠物图像分割

5.6　实践六：基于 U-Net 模型实现宠物图像分割

图像分割作为一项热点的研究任务，也是最具有挑战的任务之一，它可以构成很多其他更复杂任务的基础或核心任务。例如，对于场景理解任务来说，分割的精度直接决定着诸如

自动驾驶、三维重建、无人机控制与目标识别等应用的质量。图像分割包含了计算机视觉中的一类精细相关的问题，其中最经典的版本是语义分割。在语义分割中，每个像素被分类为一组预定的类别中的一个，以使得属于同一类别的像素作为图像中的唯一语义实体。近年来，基于深度学习的方法在计算机视觉、模式识别等领域取得了令人鼓舞的成就，其强大的特征表示学习能力使得其在图像分类、目标检测等方面的精度已经超过人类手工操作，越来越多的方法致力于提高图像分割领域的性能，推动了图像分割技术的发展。

近年来，鉴于深度学习技术的成熟并广泛应用，多种基于深度学习的图像语义分割方法被相继设计出来。与深度卷积神经网络相结合的图像语义分割，通常采用卷积神经网络的形式将图像进行像素级的分类并分割为表示不同语义类别的区域。

本实践选择了一个在医学图像分割领域最为熟知的 U-Net 网络结构，以其 U 形结构命名，如图 5-6-1 所示。网络结构主要分为三部分：编码器部分、解码器部分以及跳跃连接。编码器部分主要通过卷积和下采样来提取特征；解码器部分通过卷积和上采样来恢复图像的分辨率；中间通过跳跃连接的方式融合编码器和解码器的特征，并在最后的特征图上进行预测。

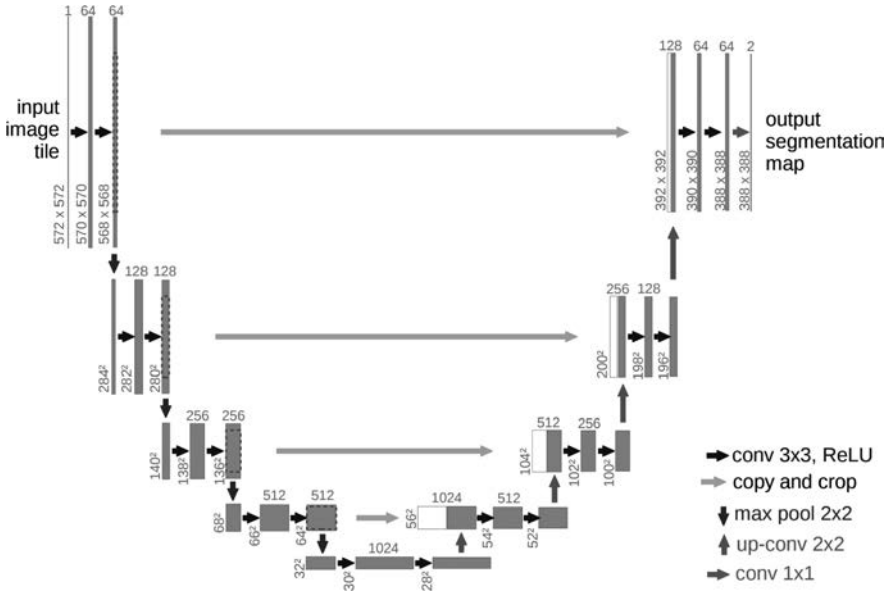

图 5-6-1　U-Net 网络结构

步骤 1：Oxford-IIIT Pet 数据集

（1）数据集概述。

本次实践中使用 Oxford-IIIT Pet 数据集，其包含 37 类宠物，每个类别大约有 200 张图像。数据集统计的分布如图 5-6-2 所示。

数据分为原始图像和标签两个部分，每张图像对应着一个 mask 图像，图像中用不同的像素值代表着不同的类别和背景。数据集标注的可视化效果如图 5-6-3 所示。原始图像和标签分别存储在 image 和 annotation 目录下，image 目录下存储着所有的图像，annotation

下分别存储着 list. txt（存储着所有的样本列表）、trainval. txt（存储着用于训练的样本列表）、test. txt（存储着需要预测的图像名单），trimaps 目录下存储的则是与训练图像命名相同的标注图像，如图 5-6-4 所示。

Breed	Count
American Bulldog	200
American Pit Bull Terrier	200
Basset Hound	200
Beagle	200
Boxer	199
Chihuahua	200
English Cocker Spaniel	196
English Setter	200
German Shorthaired	200
Great Pyrenees	200
Havanese	200
Japanese Chin	200
Keeshond	199
Leonberger	200
Miniature Pinscher	200
Newfoundland	196
Pomeranian	200
Pug	200
Saint Bernard	200
Samyoed	200
Scottish Terrier	199
Shiba Inu	200
Staffordshire Bull Terrier	189
Wheaten Terrier	200
Yorkshire Terrier	200
Total	4978

1.Dog Breeds

Breed	Count
Abyssinian	198
Bengal	200
Birman	200
Bombay	200
British Shorthair	184
Egyptian Mau	200
Main Coon	190
Persian	200
Ragdoll	200
Russian Blue	200
Siamese	199
Sphynx	200
Total	2371

2.Cat Breeds

Family	Count
Cat	2371
Dog	4978
Total	7349

3.Total Pets

图 5-6-2　Oxford-IIIT Pet 数据集统计分布

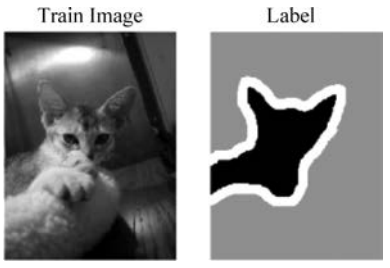

图 5-6-3　数据集标注可视化

```
.
├── README
├── list.txt
├── test.txt
├── trainval.txt
├── trimaps
│   ├── Abyssinian_1.png
│   ├── Abyssinian_10.png
│   ├── ......
│   └── yorkshire_terrier_99.png
└── xmls
    ├── Abyssinian_1.xml
    ├── Abyssinian_10.xml
    ├── ......
    └── yorkshire_terrier_190.xml
```

```
.
├── samoyed_7.jpg
├── ......
└── samoyed_81.jpg
```

图 5-6-4　数据文件结构展示

（2）数据集下载。

数据可从其官网：https://www.robots.ox.ac.uk/~vgg/data/pets 下载。

（3）数据集类定义。

接下来，通过继承 paddle.io.Dataset 类来定义数据集类 PetDataset，通过继承父类 paddle.

io.Dataset，实现父类中的两个抽象方法：__getitem__和__len__。通过__getitem__在每次迭代的过程中返回数据和其对应的分割标签，并通过__len__返回数据集的数量。

在 init 函数中，我们通过输入的 mode 参数决定输入生成的数据是用于训练、验证还是测试，并将对应的图像和标注图像地址添加到列表中。

```
class PetDataset(Dataset):
    """
    数据集定义
    """
    def __init__(self, mode = 'train'):
        """
        构造函数
        """
        self.image_size = IMAGE_SIZE
        self.mode = mode.lower()

        assert self.mode in ['train', 'test', 'predict'], \
            "mode should be 'train' or 'test' or 'predict', but got {}".format(self.mode)

        self.train_images = []
        self.label_images = []

        with open('./{}.txt'.format(self.mode), 'r') as f:
            for line in f.readlines():
                image, label = line.strip().split('\t')
                self.train_images.append(image)
                self.label_images.append(label)
```

load_image 函数共有 3 个输入参数，分别是 path（需要加载图像或标注的路径）、color_mode='rgb'（加载图像或标注的方式）和 transforms=[]（图像增强的方式）。在 load_image 函数中，首先通过 PilImage 来加载图像，对于加载格式不符合要求的图像进行格式上的转换，然后通过 paddle.vision.transforms 对读入的图像进行各种转换。

```
    def _load_img(self, path, color_mode = 'rgb', transforms = []):
        """
        统一的图像处理接口封装，用于规整图像大小和通道
        """
        with open(path, 'rb') as f:
            img = PilImage.open(io.BytesIO(f.read()))
            if color_mode == 'grayscale':
                # if image is not already an 8 - bit, 16 - bit or 32 - bit grayscale image
                # convert it to an 8 - bit grayscale image.
                if img.mode not in ('L', 'I;16', 'I'):
                    img = img.convert('L')
            elif color_mode == 'rgba':
                if img.mode != 'RGBA':
                    img = img.convert('RGBA')
            elif color_mode == 'rgb':
                if img.mode != 'RGB':
                    img = img.convert('RGB')
            else:
                raise ValueError('color_mode must be "grayscale", "rgb", or "rgba"')
```

```
            return T.Compose([
                T.Resize(self.image_size)
            ] + transforms)(img)
```

getitem 通过调用 load_image 函数，在每次迭代的时候返回图像和标注图像。由于加载进来的图像不一定都符合自己的需求（可能会是 RGBA 格式的图像，不符合 3 通道的需求），因此需要进行图像的格式转换。除此之外，卷积神经网络的输入维度一般默认为 CHW（通道数、长和宽），而图像一般加载出来的默认的维度是 HWC，这个时候对加载的图像的维度进行调整，从 HWC 转换成了 CHW。

因此在调用 load_image 加载时，依次输入 paddle.vision.transforms.Transpose()和 paddle.vision.transforms.Normalize()对图像的进行维度上的转换和数值上的归一化。

```
    def __getitem__(self, idx):
        """
        返回 image, label
        """
        train_image = self._load_img(self.train_images[idx], transforms=[ T.Transpose(),
T.Normalize(mean=127.5, std=127.5)])                    # 加载原始图像
        label_image = self._load_img(self.label_images[idx],
color_mode='grayscale', transforms=[T.Grayscale()])     # 加载 Label 图像
        # 返回 image, label
        train_image = np.array(train_image, dtype='float32')
        label_image = np.array(label_image, dtype='int64')
        return train_image, label_image

    def __len__(self):
        """
        返回数据集总数
        """
        return len(self.train_images)
```

步骤 2：U-Net 模型搭建

本次实践中，需要用到的 paddle 接口包括：

（1）paddle.nn.Upsample(size=None, scale_factor=None, mode='nearest', align_corners=False, align_mode=0, data_format='NCHW', name=None)。

用于调整一个 batch 中图片的大小，可以选择最近邻插值、线性插值、双线性插值、三线性插值、双三次线性插值等方法。

size(list|tuple|Variable|None)：输出 Tensor。输入为 3D 张量时，形状为(out_w)的 1D Tensor。输入为 4D 张量时，形状为(out_h, out_w)的 2D Tensor。输入为 5D Tensor 时，形状为(out_d, out_h, out_w)的 3D Tensor。如果 out_shape 是列表，则每个元素可以是整数或者形状为 1 的变量。如果 out_shape 是变量，则其维度大小为 1。默认值为 None。

scale_factor(float|Tensor|list|tuple|None)：输入的高度或宽度的乘数因子。out_shape 和 scale 至少要设置一个。out_shape 的优先级高于 scale。默认值为 None。

mode(str，可选)：插值方法。支持"bilinear"、"trilinear"、"nearest"、"bicubic"、"linear"

或"area"。默认值为"nearest"。

align_corners(bool,可选)：一个可选的 bool 型参数，如果为 True，则将输入和输出张量的 4 个角落像素的中心对齐，并保留角点像素的值。默认值为 True。

align_mode(int,可选)：双线性插值的可选项。它可以是'0'，代表 src_idx = scale *(dst_index + 0.5)−0.5；如果为'1'，则代表 src_idx = scale * dst_index。

data_format(str,可选)：指定输入的数据格式，输出的数据格式将与输入保持一致。对于 3D Tensor，支持 NCHW(num_batches,channels,width)；对于 4D Tensor，支持 NCHW(num_batches,channels,height,width)或者 NHWC(num_batches,height,width,channels)；对于 5D Tensor，支持 NCDHW(num_batches,channels,depth,height,width)或者 NDHWC(num_batches,depth,height,width,channels)，默认值为'NCHW'。

(2) paddle.nn.Conv2DTranspose(in_channels,out_channels,kernel_size,stride=1,padding=0,output_padding=0,groups=1,dilation=1,weight_attr=None, bias_attr=None, data_format='NCHW')。

二维转置卷积层，该层根据输入(input)、卷积核(kernel)和空洞大小(dilations)、步长(stride)、填充(padding)来计算输出特征层大小或者通过 output_size 指定输出特征层大小。

in_channels(int)：输入图像的通道数。

out_channels(int)：卷积核的个数，和输出特征图通道数相同。

kernel_size(int|list|tuple)：卷积核大小。它可以为单个整数或包含两个整数的元组或列表，分别表示卷积核的高和宽。如果为单个整数，则表示卷积核的高和宽都等于该整数。

stride(int|tuple,可选)：步长大小。如果 stride 为元组或列表，则必须包含两个整型数，分别表示垂直和水平滑动步长；否则，表示垂直和水平滑动步长均为 stride。默认值为 1。

padding(int|tuple,可选)：填充大小。如果 padding 为元组或列表，则必须包含两个整型数，分别表示竖直和水平边界填充大小；否则，表示竖直和水平边界填充大小均为 padding。如果它是一个字符串，则可以是"VALID"或者"SAME"，表示填充算法，计算细节可参考下方形状 padding="SAME"或 padding="VALID"时的计算公式。默认值为 0。

output_padding(int|list|tuple,optional)：输出形状上一侧额外添加的大小。默认值为 0。

groups(int,可选)：二维卷积层的组数。根据 Alex Krizhevsky 的深度卷积神经网络(CNN)论文中的分组卷积：当 group=2 时，卷积核的前一半仅和输入特征图的前一半连接，卷积核的后一半仅和输入特征图的后一半连接。默认值为 1。

dilation(int|tuple,可选)：空洞大小。它可以是单个整数或包含两个整数的元组或列表，分别表示卷积核中的元素沿着高和宽的空洞。如果为单个整数，则表示高和宽的空洞都等于该整数。默认值为 1。

weight_attr(ParamAttr,可选)：指定权重参数属性的对象。默认值为 None，表示使用默认的权重参数属性。

bias_attr(ParamAttr|bool,可选)：指定偏置参数属性的对象。默认值为 None,表示使用默认的偏置参数属性。

data_format(str,可选)：指定输入的数据格式,输出的数据格式将与输入保持一致,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。默认值为"NCHW"。

(3) paddle.nn.functional.pad(x,pad,mode='constant',value=0.0,data_format='NCHW',name=None)。

pad 函数依据 pad 和 mode 属性对 x 进行 pad。如果 mode 为 constant,并且 pad 的长度为 x 维度的 2 倍时,则会根据 pad 和 value 对 x 从前面的维度向后依次补齐;否则只会对 x 在除 batchsize 和 channel 之外的所有维度进行补齐。如果 mode 为 reflect,则 x 对应维度上的长度必须大于对应的 pad 值。

x(Tensor)：Tensor,format 可以为 NCL,NLC,NCHW,NHWC,NCDHW 或 NDHWC,默认值为 NCHW,数据类型支持 float16,float32,float64,int32,int64。

pad(Tensor|List[int])：填充大小。当输入维度为 3 时,pad 的格式为[pad_left,pad_right];当输入维度为 4 时,pad 的格式为[pad_left,pad_right,pad_top,pad_bottom];当输入维度为 5 时,pad 的格式为[pad_left,pad_right,pad_top,pad_bottom,pad_front,pad_back]。

mode(str)：padding 的 4 种模式,分别为 constant、reflect、replicate 和 circular。constant 表示填充常数 value;reflect 表示填充以 x 边界值为轴的映射;replicate 表示填充 x 边界值;circular 为循环填充 x。默认值为 constant。

value(float32)：以 constant 模式填充区域时填充的值。默认值为 0.0。

data_format(str)：指定 x 的 format,可为 NCL,NLC,NCHW,NHWC,NCDHW 或 NDHWC,默认值为 NCHW。

paddle.nn.Sequential(*layers)：是一个顺序容器。子 Layer 将按构造函数参数的顺序添加到此容器中。传递给构造函数的参数可以是 Layers 或可迭代的 name Layer 元组。在 DoubleConv 中将需要搭建的网络结构按顺序作为 paddle.nn.Sequential 的输入。

layers(tuple)：Layers 或可迭代的 name Layer 元组。

paddle.concat(x,axis=0,name=None)：该 OP 对输入沿 axis 轴进行联结,返回一个新的 Tensor。

x(list|tuple)：待联结的 Tensor list 或者 Tensor tuple,支持的数据类型为：bool、float16、float32、float64、int32、int64、uint8,x 中所有 Tensor 的数据类型应该一致。

axis(int|Tensor,可选)：指定对输入 x 进行运算的轴,可以是整数或者形状为 1 的 Tensor,数据类型为 int32 或者 int64。axis 的有效范围是$[-R,R)$,R 是输入 x 中 Tensor 的维度,axis 为负值时与 axis+Raxis+R 等价。默认值为 0。

(4) paddle.optimizer.RMSProp(learning_rate,rho=0.95,epsilon=1e-06,momentum=0.0,centered=False,parameters=None,weight_decay=None,grad_clip=None,name=None)。

该接口实现均方根传播(RMSProp)法。

learning_rate(float)：全局学习率。

rho(float,可选)：是等式中的 rhorho,默认值为 0.95。

epsilon(float,可选)：等式中的 epsilon 是平滑项,避免被零除,默认值为 1e-6。

momentum(float,可选)：方程中的 β 是动量项,默认值为 0.0。

centered(bool,可选)：如果为 True,则通过梯度的估计方差对梯度进行归一化；如果为 False,则由未 centered 的第二个 moment 归一化。将此设置为 True 有助于模型训练,但会消耗额外计算和内存资源。默认为 False。

parameters(list,可选)：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(float|WeightDecayRegularizer,可选)：正则化方法。它可以是 float 类型的 L2 正则化系数或者正则化策略：cn_api_fluid_regularizer_L1Decay、cn_api_fluid_regularizer_L2Decay。如果一个参数已经在 ParamAttr 中设置了正则化,则这里的正则化设置将被忽略；如果没有在 ParamAttr 中设置正则化,则这里的设置才会生效。默认值为 None,表示没有正则化。

grad_clip(GradientClipBase,可选)：梯度裁剪的策略,支持三种裁剪略：paddle.nn.ClipGradByGlobalNorm、paddle.nn.ClipGradByNorm、paddle.nn.ClipGradByValue。默认值为 None,此时将不进行梯度裁剪。

U-Net 是一个 U 型网络结构,可以看作左右两部分：

左边网络为特征提取网络：采用两个 conv 和一个 pooling 组合的方式,从图像中提取输入图像的特征。

右边网络为特征融合网络：对输入的特征图进行上采样并与左侧特征图进行融合,然后通过两次卷积提取特征。(pooling 层会丢失图像的一些信息和降低图像的分辨率,且是永久性的。上采样可以让包含高级抽象特征的低分辨率图片在保留高级抽象特征的同时变为高分辨率图片,然后再与左边低级表层特征高分辨率特征进行 concatenate 操作,获得高分辨率高语义信息的特征)。再经过两次卷积操作,生成特征图。在最后,通过 n(类别数目)个大小为 1×1 的卷积核生成最后的 n 个通道的特征图。每个特征图代表一种类别,每个类别特征图的每个像素代表着对应图像中该像素位置归属该类的概率。

(1) 网络子模块搭建。

首先是连续两次卷积子模块 DoubleConv。通过 DoubleConv 可以构建一组两层的卷积神经网络,in_channels 表示输入特征图的通道数,out_channels 表示输出特征图的通道数。DoubleConv 函数继承了 paddle.nn.Layer,包含 init 和 forward 两个函数。

init 函数中定义了两层卷积结构：在 DoubleConv 中第一个卷积层输入为 in_channels 的特征图,使用 out_channels 个 kernel_size=3 的卷积核进行卷积,同时使用 padding=1 保持特征图的大小。紧接着对输出的特征图进行 BatchNorm,并经过 ReLU 层激活。第二个卷积层以激活后的特征图为输入,用 out_channels 个 kernel_size=3 的卷积核进行卷积,之

后通过 BatchNorm 和 ReLU 得到最后的特征图。通过 forward()函数实现 DoubleConv 正向传播的过程。

```
class DoubleConv(paddle.nn.Layer):
    """(convolution => [BN] => ReLU) * 2"""
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()

        self.double_conv = paddle.nn.Sequential(
            paddle.nn.Conv2D(in_channels, out_channels, kernel_size=3, padding=1),
            paddle.nn.BatchNorm2D(out_channels),
            paddle.nn.ReLU(),
            paddle.nn.Conv2D(out_channels, out_channels, kernel_size=3, padding=1),
            paddle.nn.BatchNorm2D(out_channels),
            paddle.nn.ReLU()
        )

    def forward(self, x):
        return self.double_conv(x)
```

（2）左侧网络的部分子层结构。

通过 Down 这个类，我们实现右侧部分的子模块。对特征图实现分辨率的两倍下采样，并结合 DoubleConv 提取特征。具体来说，在 Down 类中，输入的特征图，首先会通过最大值池化将分辨率下采样两倍，之后通过 DoubleConv 类对下采样后的特征图进行 conv-> BN-> ReLU-> Conv-> BN-> ReLU 操作，最终得到输出的特征图。

```
class Down(paddle.nn.Layer):
    """Downscaling with maxpool then double conv"""
    def __init__(self, in_channels, out_channels):
        super(Down, self).__init__()
        self.maxpool_conv = paddle.nn.Sequential(
            paddle.nn.MaxPool2D(kernel_size=2, stride=2, padding=0),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)
```

（3）右侧网络的部分子层结构。

我们通过 Up 类实现右侧网络的子模块。Up 类将实现特征图的上采样，并与之前的特征图融合，同理 Up 类也继承 paddle.nn.Layer，并包含 init 和 forward 两部分。

在 init 函数中，在上采样部分，可以提供两种上采样的方式，分别是双线性插值和反卷积的方式将图像的分辨率上采样两倍，并调用了 DoubleConv 实现两次卷积提取特征。

Forward 函数有两个输入参数：x1 和 x2。x1 为需要上采样的特征图，x2 为与右侧网络相对应的左侧网络输出的特征图。对于 x1 首先通过 Init 中定义的 Up 实例进行上采样，之后再通过 paddle.nn.functional.pad 函数对上采样后的 x1 进行 pad 与 x2 特征图大小对齐，再通过 paddle.concat 将 x1 与 x2 特征图在通道上连接起来，最后再通过两次卷积提取特征。

```
class Up(paddle.nn.Layer):
    def __init__(self, in_channels, out_channels, bilinear=True):
        super(Up, self).__init__()

        if bilinear:
            self.up = paddle.nn.Upsample(scale_factor=2, mode='bilinear', align_corners=
True)
        else:
            self.up = paddle.nn.ConvTranspose2d(in_channels // 2, in_channels // 2, kernel_
size=2, stride=2)

        self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        diffY = paddle.to_tensor([x2.shape[2] - x1.shape[2]])
        diffX = paddle.to_tensor([x2.shape[3] - x1.shape[3]])
        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2, diffY // 2, diffY - diffY // 2])
        x = paddle.concat([x2, x1], axis=1)
        return self.conv(x)
```

（4）网络结构搭建。

通过 U-net 类定义 U-net 的整体网络结构。在 init 函数中首先定义网络中需要的每个卷积模组 inc、down1……up4，以及最后用于预测分割的 output_conv，在 forward 中搭建前向传播的过程。

对于输入的图像，首先经过 DoubleConv 得到特征图 x1，再通过 down1……down4 将特征图像下采样 2 倍、4 倍、8 倍、16 倍得到 x2、x3、x4、x5。接下来，通过 up1、up2、up3、up4 实现特征图的上采样 16 倍，同时在每次上采样的过程中分别与 x2、x3、x4、x5 融合，然后通过输出层，将通道数与类别数相对应（每个通道分别对应一个类别），最终通过 softmax 输出最后的结果。

```
class U_Net(paddle.nn.Layer):
    def __init__(self, num_classes, bilinear=True):
        super(U_Net, self).__init__()
        self.num_classes = num_classes
        self.bilinear = bilinear
        self.inc = DoubleConv(3, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        self.down4 = Down(512, 512)
        self.up1 = Up(1024, 256, bilinear)
        self.up2 = Up(512, 128, bilinear)
        self.up3 = Up(256, 64, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.output_conv = paddle.nn.Conv2D(64, num_classes, kernel_size=1)

    def forward(self, inputs):
        x1 = self.inc(inputs)
        x2 = self.down1(x1)
```

```
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        y = self.output_conv(x)
        return y
```

步骤 3：训练 U-Net 网络

在上面的步骤中定义好了数据集、网络模型，接下来就开始模型训练的部分。

首先通过前面定义的 PetDataset 生成训练集和验证集，通过 U-net 类实例化网络模型，并通过 paddle.optimizer.RMSProp[实现均方根传播（RMSProp）法的接口，学习率在该方法中是自适应学习的]来实例的优化器。

然后通过 prepare 方法，给我们的模型绑定优化器和损失函数，损失函数采用的交叉熵损失（paddle.nn.CrossEntropyLoss 用于计算输入 input 和标签 label 间的交叉熵损失，它结合了 LogSoftmax 和 NLLLoss 的 OP 计算，可用于训练一个 n 类分类器）。

最后通过 model.fit 开始训练和验证。其中 epochs=1 表示全部数据训练一次，batch_size=32 表示每个批次训练 32 张图像，verbose 表示的则是保存的日志格式。

```
num_classes = 4
network = U_Net(num_classes)
model = paddle.Model(network)
train_dataset = PetDataset(mode = 'train')                    # 训练数据集
val_dataset = PetDataset(mode = 'test')                       # 验证数据集
optim = paddle.optimizer.RMSProp(learning_rate = 0.001, rho = 0.9, momentum = 0.0, epsilon =
1e-07, centered = False, parameters = model.parameters())
model.prepare(optim, paddle.nn.CrossEntropyLoss(axis = 1))
model.fit(train_dataset, val_dataset, epochs = 1, batch_size = 32, verbose = 2)
```

训练过程的部分输出如图 5-6-5 所示。

```
step 170/197 - loss: 0.4966 - 378ms/step
step 180/197 - loss: 0.5091 - 379ms/step
step 190/197 - loss: 0.4286 - 378ms/step
step 197/197 - loss: 0.4800 - 376ms/step
Eval begin...
step 10/35 - loss: 0.3994 - 273ms/step
step 20/35 - loss: 0.4503 - 270ms/step
step 30/35 - loss: 0.4448 - 264ms/step
step 35/35 - loss: 0.4885 - 260ms/step
Eval samples: 1108
```

图 5-6-5　U-Net 模型训练过程中的部分输出

步骤 4：宠物分割结果预测

通过 PetDataset 设置用于预测的数据集，并通过 model.predict 进行预测，通过可视化可以对比标签和我们预测的结果，输出结果如图 5-6-6 所示。

```
predict_dataset = PetDataset(mode = 'predict')
predict_results = model.predict(predict_dataset)
```

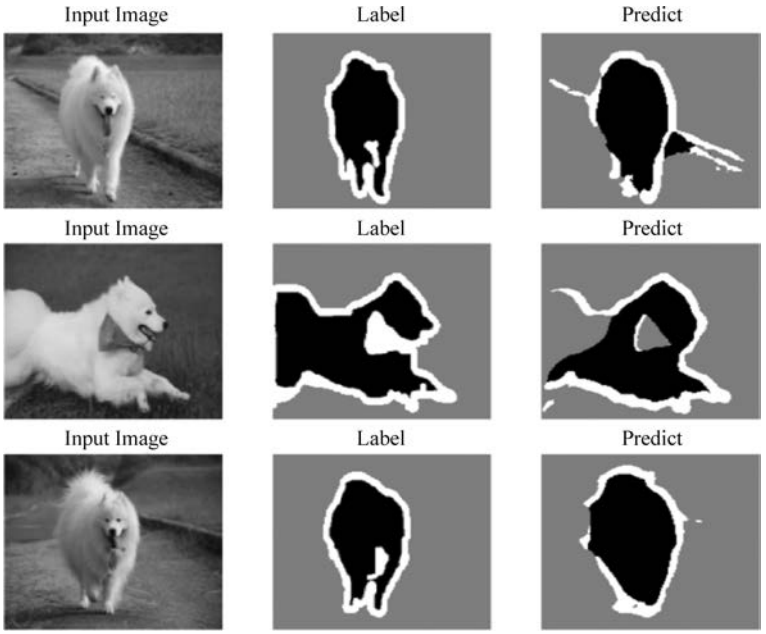

图 5-6-6　U-Net 模型的预测结果可视化