第3章

卷积神经网络

计算机视觉是深度学习技术应用和发展的重要领域，而卷积神经网络(Convolutional Neural Network，CNN)作为典型的深度神经网络在图像和视频处理、自然语言处理等领域发挥着重要的作用。本章将介绍卷积神经网络的基本概念、组成及经典的卷积神经网络架构。此外，本章还将 VGG 神经网络应用到任务——中草药识别，结合真实案例情景和代码，剖析如何使用 PaddlePaddle 搭建卷积神经网络。学习本章，希望读者能够：

- 掌握卷积神经网络的基本组成和相关概念；
- 了解经典的卷积神经网络架构；
- 使用 PaddlePaddle 搭建卷积神经网络。

3.1 图像分类问题描述

图像分类是计算机视觉基本任务之一。顾名思义，图像分类即给定一幅图像，计算机利用算法找出其所属的类别标签。相较于目标检测、实例分割、行为识别、轨迹跟踪等难度较大的计算机视觉任务，图像分类只需要让计算机看出图片里的物体类别，更为基础但极为重要。图像分类在许多领域都有着广泛的应用，例如，安防领域的智能视频分析和人脸识别、医学领域的中草药识别、互联网领域基于内容的图像检索和相册自动归类、农业领域的害虫识别等。

图像分类的过程主要包括图像的预处理、图像的特征提取以及使用分类器对图像进行分类，其中图像的特征提取是至关重要的一步。传统的图像分类算法提取图像的色彩、纹理和空间等特征，其在简单的图像分类任务中表现较好，但在复杂图像分类任务中表现不尽人意。在 CNN 提出之前，人们通过人工设计的图像描述符对图像特征进行提取，

效果卓有成效，例如，尺度不变特征变换（Scale-Invariant Feature Transform，SIFT）、方向梯度直方图（Histogram of Oriented Gradient，HOG）和词袋模型（Bag-of-Words，BoW）等，但是人工设计特征通常需要花费很大精力，并且不具有普适性。

随着智能信息时代的来临，深度学习应运而生。深度学习作为机器学习的一个分支，旨在模拟人类的神经网络系统构建深度人工神经网络，对输入的数据进行分析和解释，将数据的底层特征组合成抽象的高层特征，深度学习在计算机视觉、自然语言处理等人工智能领域发挥了不可替代的作用。作为深度学习的典型代表，卷积神经网络在计算机视觉任务中大放异彩，与人工提取特征的传统图像分类算法相比，卷积神经网络使用卷积操作对输入图像进行特征提取，有效地从大量样本中学习特征表达，模型泛化能力更强。这种基于“输入-输出”的端到端的学习方法通常可以获得非常理想的效果，受到了学术界和工业界的广泛关注。本章将对卷积神经网及其应用加以详细论述。

3.2 卷积神经网络

在第 2 章已经介绍，深度学习的模型框架包括三个部分：建立模型、损失函数和参数学习。在此，本章损失函数和参数学习与第 2 章类似，不再另设章节特意说明，在本章建立的模型即是卷积神经网络。本节从卷积神经网络结构上的三大特点出发，详细介绍卷积神经网络的概念和结构。

卷积神经网络三大特点：

（1）局部连接：相比全连接神经网络，卷积神经网络在进行图像识别的时候，不需要对整个图像进行处理，只需要关注图像中某些特殊的区域，如图 3-1 所示。

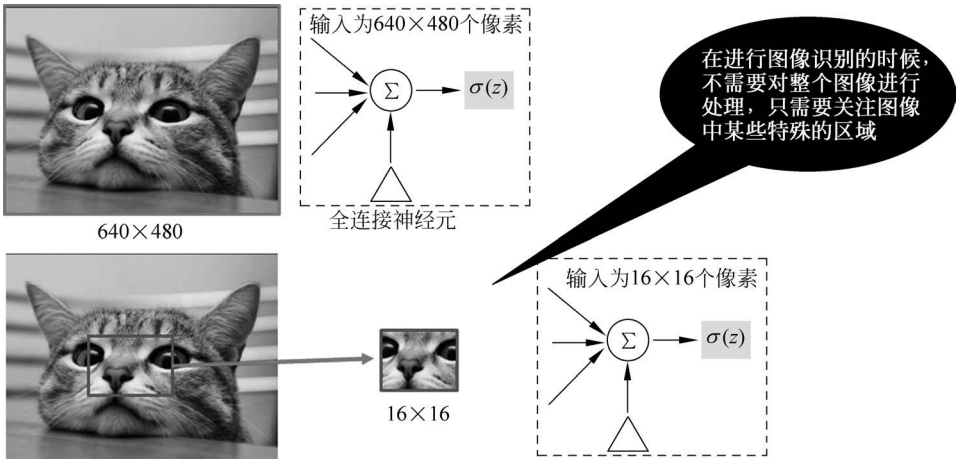


图 3-1 局部连接

（2）权重共享：卷积神经网络的神经元权重相同，如图 3-2 所示。

（3）下采样：对图像像素进行下采样，并不会对物体进行改变。虽然下采样之后的图像尺寸变小了，但是并不影响对图像中物体的识别，如图 3-3 所示。

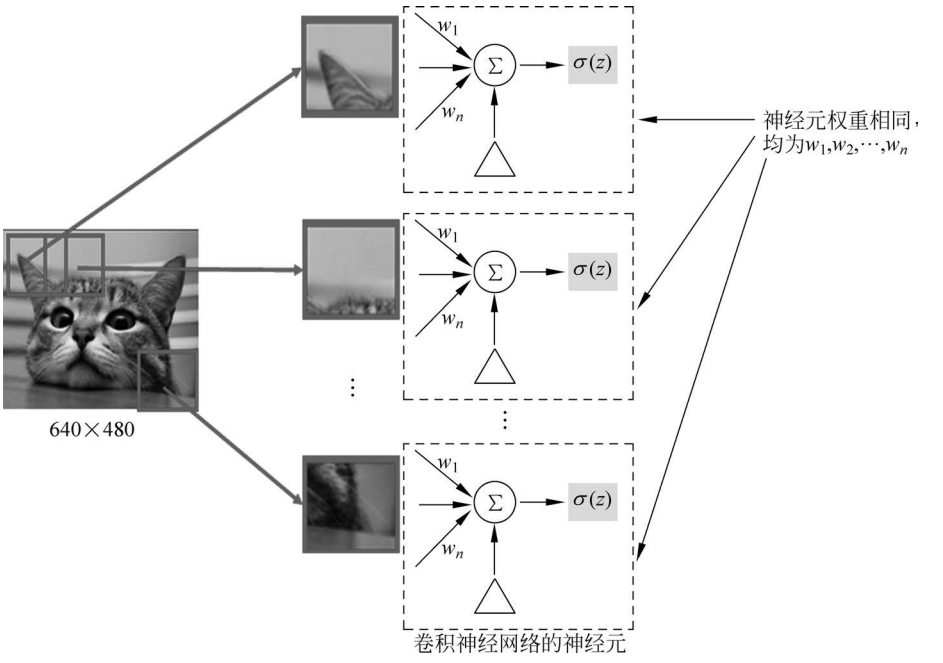


图 3-2　权重共享

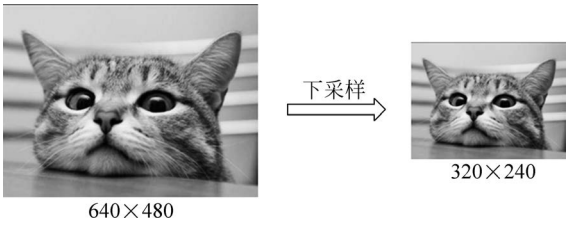


图 3-3　下采样

卷积神经网络利用其三大特点，实现减少网络参数，从而加快训练速度。下面介绍卷积神经网络的模型结构，并说明模型中的各层是如何实现了三大特点的。图 3-4 是一个典型的卷积神经网络结构，多层卷积和池化层组合作用在输入图片上，在网络的最后通常会加入一系列全连接层，ReLU 激活函数一般加在卷积或者全连接层的输出上，网络中通常还会加入 Dropout 来防止过拟合。

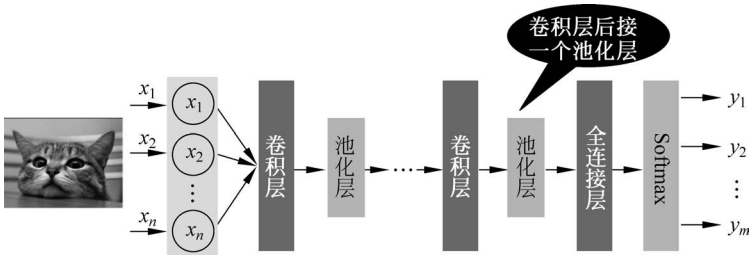


图 3-4　卷积神经网络结构

(1) 卷积层：卷积层用于对输入的图像进行特征提取。卷积的计算范围是在像素点的空间邻域内进行的，因此可以利用输入图像的空间信息。卷积核本身与输入图片大小无关，它代表了对空间邻域内某种特征模式的提取。比如，有些卷积核提取物体边缘特征，有些卷积核提取物体拐角处的特征，图像上不同区域共享同一个卷积核。当输入图片大小不一样时，仍然可以使用同一个卷积核进行操作。

(2) 池化层：池化层通过对卷积层输出的特征图进行约减，实现了下采样。同时对感受域内的特征进行筛选，提取区域内最具代表性的特征，保留特征图中最主要的信息。

(3) 激活函数：激活函数给神经元引入了非线性因素，对输入信息进行非线性变换，从而使得神经网络可以任意逼近任何非线性函数，然后将变换后的输出信息作为输入信息传给下一层神经元。

(4) 全连接层：全连接层用于对卷积神经网络提取到的特征进行汇总，将多维的特征映射为二维的输出。

3.2.1 卷积层

这一节将为读者介绍卷积算法的原理和实现方案，并通过具体的案例展示如何使用卷积对图片进行操作，主要包括卷积核、卷积计算、特征图、多输入通道、多输出通道。填充(Padding)、步幅(Stride)、感受野(Receptive Field)等概念在此不作介绍，读者可参考其他深度学习书籍。

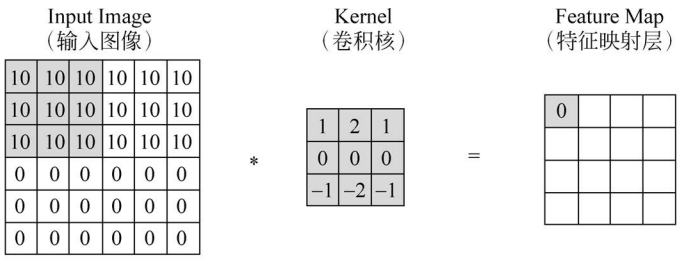
1. 卷积核/特征图/卷积计算

卷积核(Kernel)也被叫作滤波器(Filter)。假设卷积核的高和宽分别为 k_h 和 k_w，则将称为 $k_h \times k_w$ 卷积，比如 3×5 卷积，就是指卷积核的高为 3、宽为 5。卷积核中数值为对图像中与卷积核同样大小的子块像素点进行卷积计算时所采用的权重。卷积计算(Convolution)：图像中像素点具有很强的空间依赖性，卷积就是针对像素点的空间依赖性来对图像进行处理的一种技术。卷积滤波结果在卷积神经网络中被称为特征图(Feature Map)。

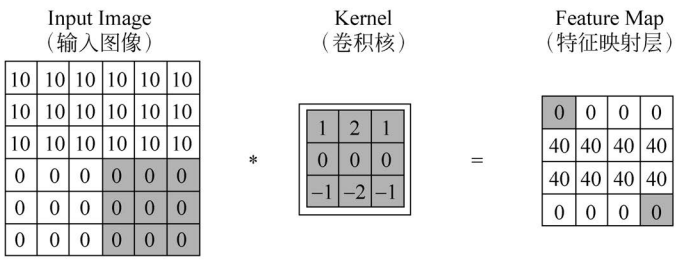
应用示例如下。

在卷积神经网络中，卷积层的实现方式实际上是数学中定义的互相关(Cross-correlation)运算，具体的计算过程如图 3-5 所示，每张图的左图表示输入数据是一个维度为 6×6 的二维数组，中间的图表示卷积核是一个维度为 3×3 的二维数组。

如图 3-5 所示，左边的图大小是 6×6，表示输入数据是一个维度为 6×6 的二维数组；中间的图大小是 3×3，表示一个维度为 3×3 的二维数组，这个二维数组称为卷积核。先将卷积核的左上角与输入数据的左上角(即输入数据的(0, 0)位置)对齐，把卷积核的每个元素跟其位置对应的输入数据中的元素相乘，再把所有乘积相加，如图 3-5(a)得到卷积输出的第一个结果 F[0,0]，以此类推，最终如图 3-5(b)所示得到结果特征图和结果 F[3,3]。



(a) F[0,0]=10×1+10×2+10×1+10×0+10×0+10×0+10×(–1)+10×(–2)+10×(–1)=0



(b) F[3,3]=0×1+0×2+0×1+0×0+0×0+0×0+0×(–1)+0×(–2)+0×(–1)=0

图 3-5　卷积计算过程

F[0,0]=10×1+10×2+10×1+10×0+10×0+10×0+10×(−1)+10×(−2)+10×(−1)=0

F[0,1]=10×1+10×2+10×1+10×0+10×0+10×0+10×(−1)+10×(−2)+10×(−1)=0

F[0,2]=10×1+10×2+10×1+10×0+10×0+10×0+10×(−1)+10×(−2)+10×(−1)=0

F[0,3]=10×1+10×2+10×1+10×0+10×0+10×0+10×(−1)+10×(−2)+10×(−1)=0

F[1,0]=10×1+10×2+10×1+10×0+10×0+10×0+0×(−1)+0×(−2)+0×(−1)=40

……

F[3,3]=0×1+0×2+0×1+0×0+0×0+0×0+0×(−1)+0×(−2)+0×(−1)=0

卷积核的计算过程可以用式(3-1)表示，其中 a 代表输入图片，b 代表输出特征图，w 是卷积核参数，它们都是二维数组，$\sum_{u,v}$ 表示对卷积核参数进行遍历并求和。

$$b[i,j]=\sum_{u,v}a[i+u,j+v]\cdot w[u,v] \tag{3-1}$$

举例说明，假如上图中卷积核大小是 2×2，则 u 可以取 0 和 1，v 也可以取 0 和 1，也就是说：

$$b[i,j]=a[i+0,j+0]\cdot w[0,0]+a[i+0,j+1]\cdot w[0,1]+a[i+1,j+0]\cdot w[1,0]+a[i+1,j+1]\cdot w[1,1]$$

读者可以自行验证，当[i,j]取不同值时，根据式(3-1)计算的结果与上图中的例子是否一致。

2. 多输入通道场景

前面介绍的卷积计算过程比较简单，实际应用时，处理的问题要复杂得多。例如，对于彩色图片有 RGB 三个通道，需要处理多输入通道的场景，相应的输出特征图往往也会具有多个通道，而且在神经网络的计算中常常把一个批次的样本放在一起计算，所以卷积算子需要具有批量处理多输入和多输出通道数据的功能。

当输入含有多个通道时，对应的卷积核也应该有相同的通道数。假设输入图片的通道数为 C_{in}，输入数据的形状是 $C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$。计算过程如下所示。

(1) 对每个通道分别设计一个二维数组作为卷积核，卷积核数组的形状是 $C_{\text{in}} \times k_h \times k_w$。

(2) 对任一通道 $C_{\text{in}} \in [0, C_{\text{in}})$，分别用大小为 $k_h \times k_w$ 的卷积核在大小为 $H_{\text{in}} \times W_{\text{in}}$ 的二维数组上做卷积。

(3) 将这 C_{in} 个通道的计算结果相加，得到的是一个形状为 $H_{\text{out}} \times W_{\text{out}}$ 的二维数组。

应用示例如下。

上面的例子中，卷积层的数据是一个二维数组，但实际上一张图片往往含有 RGB 三个通道，要计算卷积的输出结果，卷积核的形式也会发生变化。假设输入图片的通道数为 3，输入数据的形状是 $3 \times H_{\text{in}} \times W_{\text{in}}$，计算过程如图 3-6 所示。

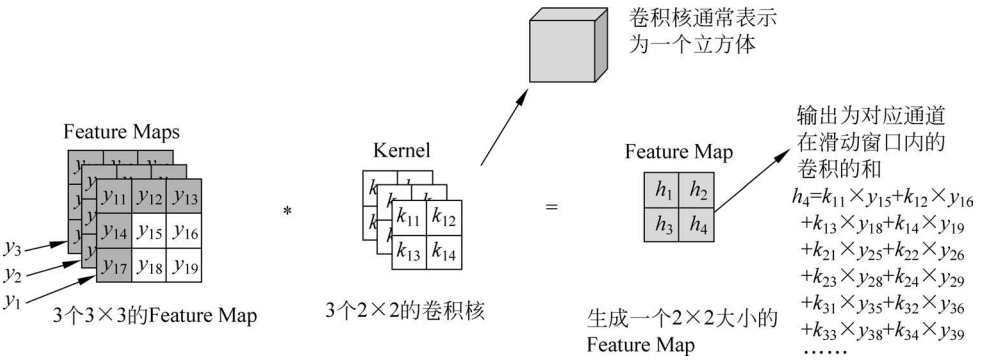


图 3-6　多输入通道计算过程

(1) 对每个通道分别设计一个二维数组作为卷积核，卷积核数组的形状是 $3 \times k_h \times k_w$。

(2) 对任一通道 $C_{\text{in}} \in [0, 3)$，分别用大小为 $k_h \times k_w$ 的卷积核在大小为 $H_{\text{in}} \times W_{\text{in}}$ 的二维数组上做卷积。

(3) 将这 3 个通道的计算结果相加，得到的是一个形状为 $H_{\text{out}} \times W_{\text{out}}$ 的二维数组。

3. 多输出通道场景

如果希望检测多种类型的特征，这需要采用多个卷积核进行计算。所以一般来说，卷积操作的输出特征图也会具有多个通道 C_{out}，这时需要设计 C_{out} 个维度为 $C_{\text{in}} \times k_h \times k_w$ 的卷积核数组，其维度是 $C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w$。

(1) 对任一输出通道 $C_{out} \in [0, C_{out})$，分别使用上面描述的形状为 $C_{in} \times k_h \times k_w$ 的卷积核对输入图片做卷积。

(2) 将这 C_{out} 个形状为 $H_{out} \times W_{out}$ 的二维数组拼接在一起，形成维度为 $C_{out} \times H_{out} \times W_{out}$ 的三维数组。

应用示例如下。

假设输入图片的通道数为 3 个，希望检测 n 个种类型的特征，这时需要设计 n 个维度为 $3 \times k_h \times k_w$ 的卷积核，如图 3-7 所示。

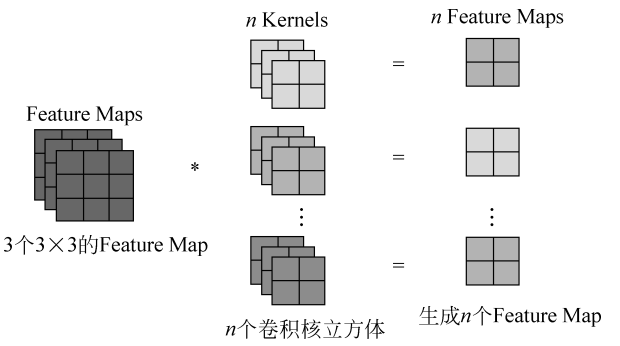


图 3-7　多输出通道计算过程

3.2.2　池化层

在图像处理中，由于图像中存在较多冗余信息，可用某一区域子块的统计信息(如最大值或均值等)来刻画该区域中所有像素点呈现的空间分布模式，以替代区域子块中所有像素点取值，这就是卷积神经网络中池化操作。池化操作对卷积结果特征图进行约减，实现了下采样，同时保留了特征图中主要信息。例如，当识别一张图像是否是人脸时，我们需要知道人脸左边有一只眼睛，右边也有一只眼睛，而不需要知道眼睛的精确位置，这时通过池化某一片区域的像素点来得到总体统计特征会显得很有用。池化常见方法有平均池化、最大池化。

(1) 平均池化：计算区域子块所包含所有像素点的均值，将均值作为平均池化结果。

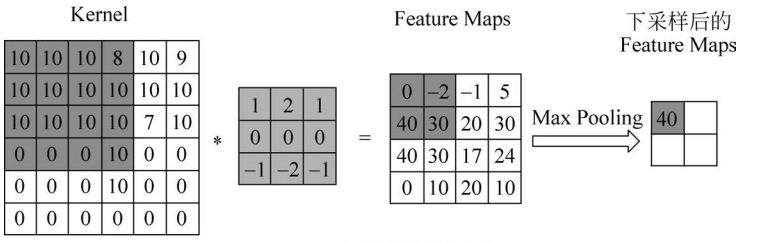
(2) 最大池化：从输入特征图的某个区域子块中选择值最大的像素点作为最大池化结果。

如图 3-8 所示，对池化窗口覆盖区域内的像素取最大值，得到输出特征图的像素值。当池化窗口在图片上滑动时，会得到整张输出特征图。

应用示例如下。

与卷积核类似，池化窗口在图片上滑动时，每次移动的步长称为步幅，当宽和高的移动大小不一样时，分别用 s_w 和 s_h 表示。也可以对需要进行池化的图片进行填充，填充方式与卷积类似，假设在第一行之前填充 p_{h1} 行，在最后一行后面填充 p_{h2} 行。在第一列之前填充 p_{w1} 列，在最后一列之后填充 p_{w2} 列，则池化层的输出特征图大小为：

$$H_{out} = \frac{H + p_{h1} + p_{h2} - k_h}{s_h} + 1$$



(a) 下采样为40 Ⅰ

(b) 下采样为30 Ⅱ

(c) 下采样为40 Ⅲ

(d) 下采样为40 Ⅳ

图 3-8　最大池化

$$W_{\text{out}}=\frac{W+p_{w1}+p_{w2}-k_w}{s_w}+1$$

在卷积神经网络中，通常使用 2×2 大小的池化窗口，步幅也使用 2，填充为 0，则输出特征图的尺寸为：

$$H_{\text{out}}=\frac{H}{2}$$

$$W_{\text{out}}=\frac{W}{2}$$

通过这种方式的池化，输出特征图的高和宽都减半，但通道数不会改变。

这里以图 3-8 中的两个池化运算为例，此时，输入大小是 4×4，使用大小为 2×2 的池化窗口进行运算，步幅为 2。此时，输出尺寸的计算方式为：

$$H_{\text{out}}=\frac{H+p_{h1}+p_{h2}-k_h}{s_h}+1=\frac{4+0+0-2}{2}+1=\frac{4}{2}=2$$

$$W_{\text{out}}=\frac{W+p_{w1}+p_{w2}-k_w}{s_w}+1=\frac{4+0+0-2}{2}+1=\frac{4}{2}=2$$

如图 3-8(a)所示，使用最大池化进行运算，则输出中的每一个像素均为池化窗口对应的 2×2 区域求最大值得到。其计算步骤如下。

(1) 池化窗口的初始位置为左上角，对应红色区域，此时输出为 40＝max{0，－2，40，30}；

(2) 由于步幅为 2，所以池化窗口向右移动两个像素，对应绿色区域，此时输出为 30＝max{－1，5，20，30}；

(3) 遍历完第一行后，再从第三行开始遍历，对应黄色区域，此时输出为 40＝max{40，30，0，10}；

(4) 池化窗口向右移动两个像素，对应蓝色区域，此时输出为 40＝max{17，24，20，10}。

3.2.3　卷积优势

卷积操作具有四大优势：保留空间信息、局部连接、权重共享、对不同层级卷积提取不同特征。具体如下。

1) 保留空间信息

在卷积运算中，计算范围是在像素点的空间邻域内进行的，它代表了对空间邻域内某种特征模式的提取。对比全连接层将输入展开成一维的计算方式，卷积运算可以有效地学习到输入数据的空间信息。

2) 局部连接

在卷积操作中，每个神经元只与局部的一块区域进行连接。对于二维图像，局部像素关联性较强，这种局部连接保证了训练后的滤波器能够对局部特征有最强的响应，使神经网络可以提取数据的局部特征。全连接与局部连接的对比如图 3-9 所示。

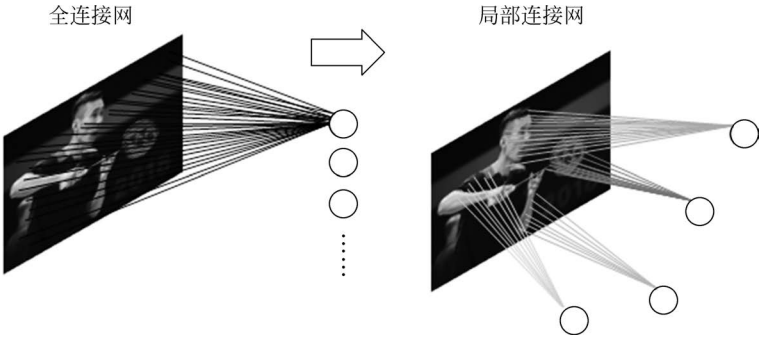


图 3-9　全连接与局部连接

同时，由于使用了局部连接，隐藏层的每个神经元仅与部分图像相连。例如，对于一幅 1000×1000 的输入图像而言，下一个隐藏层的神经元数目同样为 10^6 个，假设每个神经元只与大小为 10×10 的局部区域相连，那么此时的权重参数量仅为 $10\times10\times10^6=10^8$，相交密集链接的全连接层少了 4 个数量级。

3）权重共享

卷积计算实际上是使用一组卷积核在图片上进行滑动，实现计算乘加和。因此，对于同一个卷积核的计算过程而言，在与图像计算的过程中，它的权重是共享的。这大大降低了网络的训练难度，图 3-10 为权重共享的示意图。这里还使用上边的例子，对于一幅 1000×1000 的输入图像，下一个隐藏层的神经元数目为 10^6 个，隐藏层中的每个神经元与大小为 10×10 的局部区域相连，因此有 10×10 个权重参数。将这 10×10 个权重参数共享给其他位置对应的神经元，也就是 10^6 个神经元的权重参数保持一致，那么最终需要训练的参数就只有这 10×10 个权重参数。

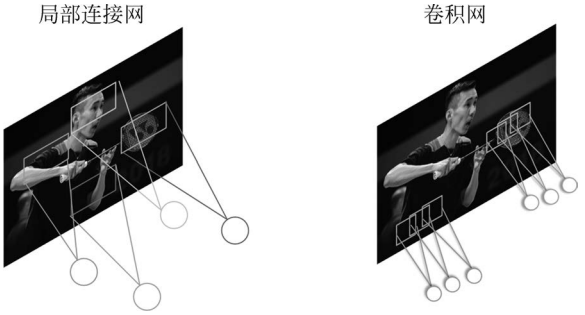


图 3-10　权重共享示意图

4）对不同层级卷积提取不同特征

在 CNN 网络中，通常使用多层卷积进行堆叠，从而实现提取不同类型特征的作用。例如，浅层卷积提取的是图像中的边缘等信息；中层卷积提取的是图像中的局部信息；深层卷积提取的则是图像中的全局信息。这样，通过加深网络层数，CNN 就可以有效地学习到图像从细节到全局的所有特征了。对一个简单的 5 层 CNN 进行特征图可视化后的结果如图 3-11 所示。

通过图 3-11 可以看到，Layer1 和 Layer2 中，网络学到的基本上是边缘、颜色等底层特征；Layer3 开始变得稍微复杂，学习到的是纹理特征；Layer4 学习到了更高维的特征，比如狗头、鸡脚等；Layer5 则学习到了更加具有辨识性的全局特征。

3.2.4　模型实现

PaddlePaddle 的 paddle.nn 包中使用 Conv1D、Conv2D 和 Conv3D 实现卷积层，它们分别表示一维卷积、二维卷积、三维卷积。

此处仅介绍自然语言处理中常用的一维卷积（Conv1D），其构造函数有三个参数：in_channels 为输入通道的个数，out_channels 为输出通道的个数，kernel_size 为卷积核宽度。当调用该 Conv1D 对象时，输入数据形状为（batch，in_channels，seq_len），输出数据形状为（batch，out_channels，seq_len），其中在输入数据和输出数据中，seq_len 表示输入的序列长度又表示输出的序列长度。卷积神经网络中卷积层代码如下所示。

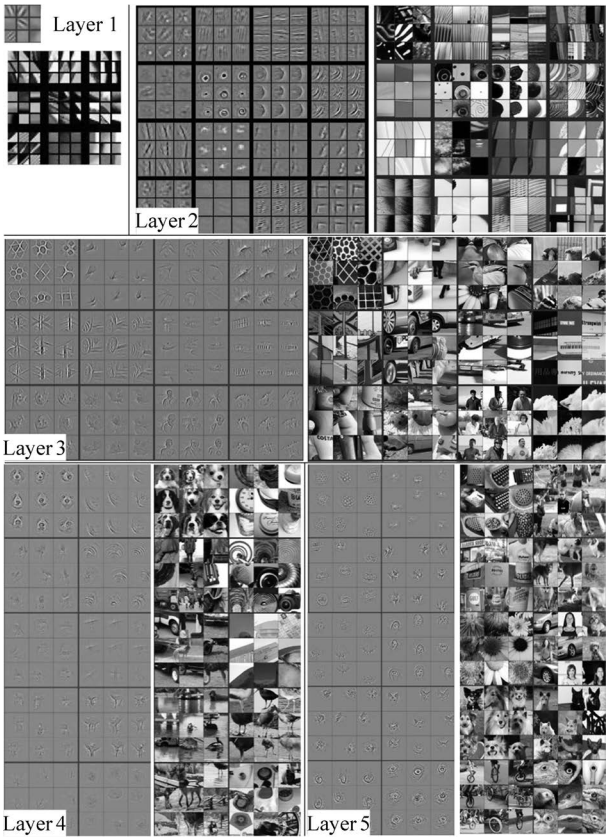


图 3-11　特征图可视化示意图

```
01. #1.定义卷积层
02. import paddle
03. from paddle.nn import Conv1D
04. #定义一个一维卷积,输入通道大小为 5,输出通道大小为 2,卷积核宽度为 4
05. conv1 = Conv1D(in_channels = 5, out_channels = 2, kernel_size = 4)
06. #定义一个一维卷积,输入通道大小为 5,输出通道大小为 2,卷积核宽度为 3
07. conv2 = Conv1D(in_channels = 5, out_channels = 2, kernel_size = 3)
08. #输入数据批次大小为 2,即输入两个序列,每个序列长为 6,每个输入的维度为 5
09. inputs = paddle.rand([2,5,6])
10. output1 = conv1(inputs)
11. print("output1:",output1)
12. output2 = conv2(inputs)
13. print("output2:",output2)
```

运行结果如下。

```
output1: Tensor(shape = [2, 2, 3], dtype = float32, place = CUDAPlace(0),
stop_gradient = False,     [[[ - 0.22635436, - 0.56034124, - 0.49246365],
       [ 0.50104654, 0.06747232, 0.71192616]],
```

```
      [[ - 0.41800800, - 0.27245334, - 1.22654486],
      [ 0.81338280, 0.46928132, 1.25316608]]])
output2: Tensor(shape = [2, 2, 4], dtype = float32, place = CUDAPlace(0),
stop_gradient = False,     [[[ 0.55459601, - 0.11513655, 0.83680284, 0.11368199],
        [ 0.07029381, - 0.14357673, 0.16201110, - 0.27607250]],
      [[ 0.19907981, 0.59146804, - 0.00985690, 0.49959880],
        [ - 0.28189474, 0.04756935, 0.50971091, 0.43118754]]])
```

接下来需要调用 paddle.nn 包中定义的池化层类,主要有最大池化、平均池化等。与卷积层类似,各种池化方法也分为一维、二维和三维三种。例如,MaxPool1D 是一维最大池化,其构造函数至少需要提供一个参数 kerne_size,即池化层核的大小,也就是对多大范围内的输入进行聚合。如果对整个输入序列进行池化,则其大小应为卷积层输出的序列长度。池化层代码如下所示。

```
01. #2.池化层
02. from paddle.nn import MaxPool1D
03. pool1 = MaxPool1D(3) #第一个池化层核的大小是 3
04. pool2 = MaxPool1D(4) #第一个池化层核的大小是 4
05. output_pool1 = pool1(output1) #执行一维最大化池化操作,即取每行输入的最大值
06. output_pool2 = pool1(output2)
07. print("output_pool1:",output_pool1)
08. print("output_pool2:",output_pool2)
```

运行结果如下。

```
output_pool1: Tensor(shape = [2, 2, 1], dtype = float32, place = CUDAPlace(0),
top_gradient = False,[[[ - 0.22635436],[ 0.71192616]], [[ - 0.27245334],
     [ 1.25316608]]])
output_pool2: Tensor(shape = [2, 2, 1], dtype = float32, place = CUDAPlace(0), stop_gradient
 = False,     [[[0.83680284],
     [0.16201110]],
    [[0.59146804],
     [0.50971091]]])
```

由于 output_pool1 和 output_pool2 是两个独立的张量,为了进行下一步操作,还需要调用 paddle.concat 函数将它们拼接起来。在此之前,还需要调用 squeeze 函数将最后一个为 1 的维度删除,即将 2 行 1 列的矩阵变为 1 个向量。代码如下所示。

```
01. output_pool_squeeze1 = paddle.squeeze(output_pool1,axis = 2)
02. print("output_pool_squeeze1 :",output_pool_squeeze1)
03. output_pool_squeeze2 = paddle.squeeze(output_pool2,axis = 2)
04. print("output_pool_squeeze2 :",output_pool_squeeze2)
05. outputs_pool = paddle.concat(x = [output_pool_squeeze1,output_pool_squeeze2],axis =
    1)
06. print("outputs_pool :",outputs_pool)
```

运行结果如下。

```
output_pool_squeeze1 : Tensor(shape = [2, 2], dtype = float32,
lace = CUDAPlace(0), stop_gradient = False,
    [[ - 0.22635436, 0.71192616],     [ - 0.27245334, 1.25316608]])
output_pool_squeeze2 : Tensor(shape = [2, 2], dtype = float32, place = CUDAPlace(0), stop_
gradient = False,     [[0.83680284, 0.16201110],     [0.59146804, 0.50971091]])
outputs_pool : Tensor(shape = [2, 4], dtype = float32, place = CUDAPlace(0), stop_gradient =
False,     [[ - 0.22635436, 0.71192616, 0.83680284, 0.16201110],
    [ - 0.27245334, 1.25316608, 0.59146804, 0.50971091]])
```

池化后，再连接一个全连接层，实现其分类功能。全连接层实现代码如下。

```
01. from paddle.nn import Linear
02. #3.全连接层,输入维度为 4,即池化层输出的维度
03. linear = Linear(4,2)
04. outputs_linear = linear(outputs_pool)
05. print("outputs_linear:",outputs_linear)
```

运行结果如下。

```
outputs_linear: Tensor(shape = [2, 2], dtype = float32, place = CUDAPlace(0),
top_gradient = False,
     [[ - 1.26811194, - 0.16523767],
     [ - 1.84709907, 0.34843248]])
```

3.3　经典的卷积神经网络

前面介绍了卷积神经网络的基本组成和常见概念。本节将按如图 3-12 所示的方式介绍几种典型的卷积神经网络架构。读者在了解卷积神经网络历史发展的同时，也可以加深对卷积神经网络组成的认识。

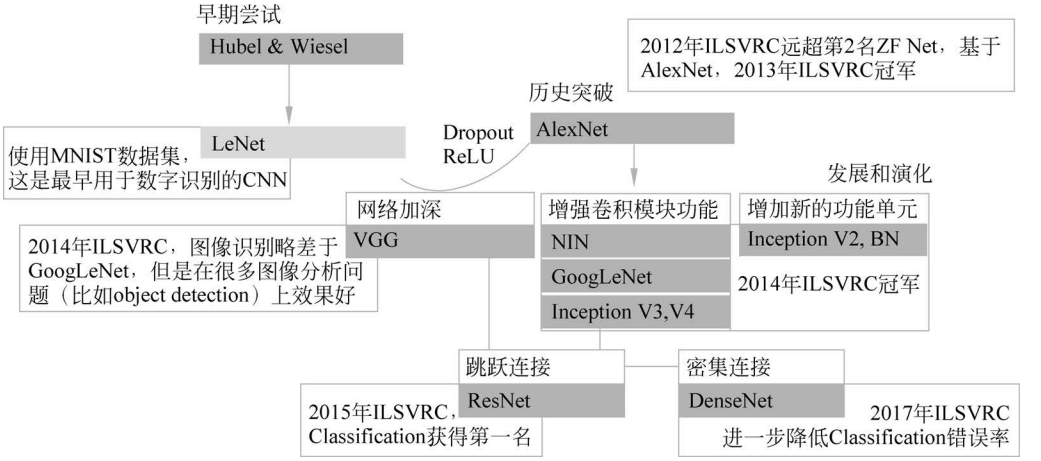


图 3-12　经典 CNN 发展概述

3.3.1 LeNet

LeNet是最早的卷积神经网络之一，其被提出用于识别手写数字和机器印刷字符。1998年，Yann LeCun第一次将LeNet卷积神经网络应用到图像分类上，在手写数字识别任务中取得了巨大成功。算法阐述了图像中像素特征之间的相关性，神经网络能够由参数共享的卷积操作所提取，同时也使用卷积、下采样（池化）和非线性映射这样的组合结构，是当前流行的大多数图像识别网络的基础。

LeNet通过连续使用卷积和池化层的组合提取图像特征，其架构如图3-13所示，这里展示的是MNIST手写体数字识别任务中的LeNet-5模型。

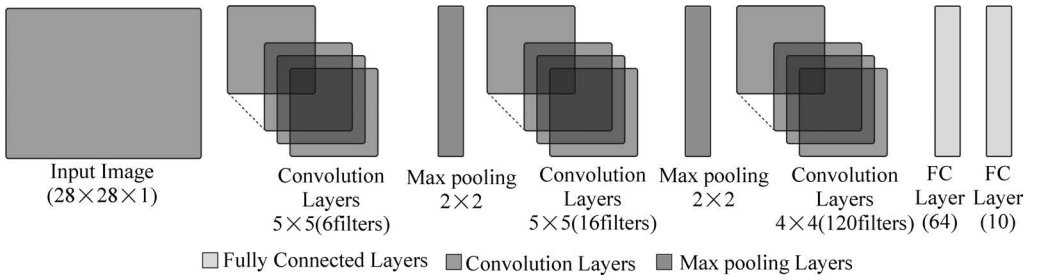


图3-13 LeNet模型网络结构示意图

第一模块：包含5×5的6通道卷积和2×2的池化。卷积提取图像中包含的特征模式（激活函数使用Sigmoid），图像尺寸从28减小到24。经过池化层可以降低输出特征图对空间位置的敏感性，图像尺寸减到12。

第二模块：和第一模块尺寸相同，通道数由6增加为16。卷积操作使图像尺寸减小到8，经过池化后变成4。

模块：包含4×4的120通道卷积。卷积之后的图像尺寸减小到1，但是通道数增加为120。将经过第3次卷积提取到的特征图输入到全连接层。第一个全连接层的输出神经元的个数是64，第二、第三个全连接层的输出神经元个数是分类标签的类别数，对于手写数字识别的类别数是10。然后使用Softmax激活函数即可计算出每个类别的预测概率。

3.3.2 AlexNet

AlexNet是2012年ImageNet竞赛的冠军模型，其作者是神经网络领域三巨头之一的Hinton，他的学生Alex Krizhevsky也参与了模型的编写。AlexNet以极大的优势领先2012年ImageNet竞赛的第二名，因此也给当时的学术界和工业界带来了很大的冲击。此后，更多更深的神经网络相继被提出，比如优秀的VGG、GoogLeNet、ResNet等。

AlexNet与此前的LeNet相比，具有更深的网络结构，包含5层卷积和3层全连接，具体结构如图3-14所示。

第一模块：对于224×224的彩色图像，先用96个11×11×3的卷积核对其进行卷积，提取图像中包含的特征模式（步长为4，填充为2，得到96个54×54的卷积结果（特征

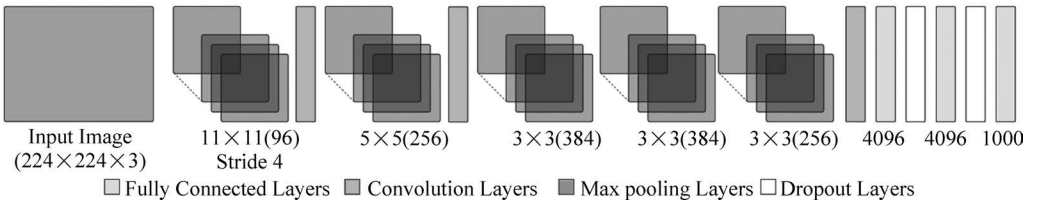


图 3-14　AlexNet 模型网络结构示意图

图))；然后以 2×2 大小进行池化，得到了 96 个 27×27 大小的特征图；

第二模块：包含 256 个 5×5 的卷积和 2×2 池化，卷积操作后图像尺寸不变，经过池化后，图像尺寸变成 13×13；

第三模块：包含 384 个 3×3 的卷积，卷积操作后图像尺寸不变；

第四模块：包含 384 个 3×3 的卷积，卷积操作后图像尺寸不变；

第五模块：包含 256 个 3×3 的卷积和 2×2 的池化，卷积操作后图像尺寸不变，经过池化后变成 256 个 6×6 大小的特征图。

将经过第 5 次卷积提取到的特征图输入到全连接层，得到原始图像的向量表达。前两个全连接层的输出神经元的个数是 4096，第三个全连接层的输出神经元个数是分类标签的类别数(ImageNet 比赛的分类类别数是 1000)，然后使用 Softmax 激活函数即可计算出每个类别的预测概率。

3.3.3　VGG

随着 AlexNet 在 2012 年的 ImageNet 大赛上大放异彩后，卷积神经网络进入了飞速发展的阶段。2014 年，由 Simonyan 和 Zisserman 提出的 VGG 网络在 ImageNet 上取得了亚军的成绩。VGG 的命名来源于论文作者所在的实验室 Visual Geometry Group。VGG 对卷积神经网络进行了改良，探索了网络深度与性能的关系，用更小的卷积核和更深的网络结构，取得了较好的效果，成为了 CNN 发展史上较为重要的一个网络。VGG 中使用了一系列大小为 3×3 的小尺寸卷积核和池化层构造深度卷积神经网络，因为其结构简单、应用性极强而广受研究者欢迎，尤其是它的网络结构设计方法，为构建深度神经网络提供了方向。

图 3-15 是 VGG-16 的网络结构示意图，有 13 层卷积和 3 层全连接层。VGG 网络的设计严格使用 3×3 的卷积层和池化层来提取特征，并在网络的最后面使用三层全连接层，将最后一层全连接层的输出作为分类的预测。VGG 中还有一个显著特点：每次经过池化层(Max pooling)后特征图的尺寸减小一半，而通道数增加一倍(最后一个池化层除外)。在 VGG 中每层卷积将使用 ReLU 作为激活函数，在全连接层之后添加 Dropout 来抑制过拟合。使用小的卷积核能够有效地减少参数的个数，使得训练和测试变得更加有效。比如使用两层 3×3 卷积层，可以得到感受野为 5 的特征图，而比使用 5×5 的卷积层需要更少的参数。由于卷积核比较小，可以堆叠更多的卷积层，加深网络的深度，这对于图像分类任务来说是有利的。VGG 模型的成功证明了增加网络的深度，可以更好地学习图像中的特征模式。

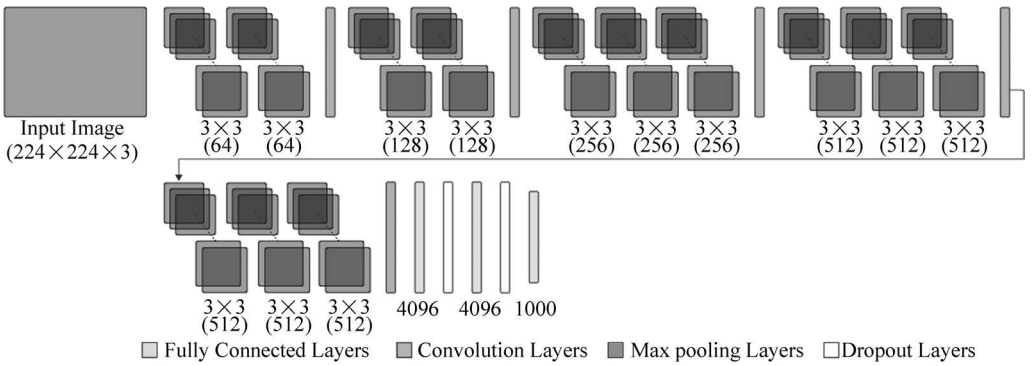


图 3-15　VGG 模型网络结构示意图

3.3.4　GoogLeNet

GoogLeNet 是 2014 年 ImageNet 比赛的冠军，它的主要特点是网络不仅有深度，还在横向上具有“宽度”。从名字 GoogLeNet 可以知道这是谷歌工程师设计的网络结构，而名字 GoogLeNet 更是致敬了 LeNet。GoogLeNet 中最核心的部分是其内部子网络结构 Inception，该结构灵感来源于 NIN(Network In Network)。

由于图像信息在空间尺寸上的巨大差异，如何选择合适的卷积核来提取特征就显得比较困难了。空间分布范围更广的图像信息适合用较大的卷积核来提取其特征；而空间分布范围较小的图像信息则适合用较小的卷积核来提取其特征。为了解决这个问题，GoogLeNet 提出了一种被称为 Inception 模块的方案，如图 3-16 所示。

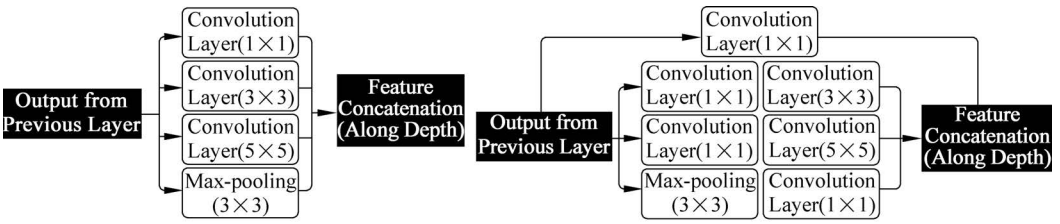


图 3-16　Inception 模块结构示意图

图 3-16(a)是 Inception 模块的设计思想，使用 3 个不同大小的卷积核对输入图片进行卷积操作，并附加最大池化，将这 4 个操作的输出沿着通道这一维度进行拼接，构成的输出特征图将会包含经过不同大小的卷积核提取出来的特征，从而达到捕捉不同尺度信息的效果。Inception 模块采用多通路(Multi-path)的设计形式，每个支路使用不同大小的卷积核，最终输出特征图的通道数是每个支路输出通道数的总和，这将会导致输出通道数变得很大，尤其是使用多个 Inception 模块串联操作的时候，模型参数量会变得非常大。

为了减小参数量，Inception 模块使用了图 3-16(b)中的设计方式，在每个 3×3 和 5×5 的卷积层之前，增加 1×1 的卷积层来控制输出通道数；在最大池化层后面增加 1×1 卷积层减小输出通道数。基于这一设计思想，形成了图 3-16(b)中所示的结构。

3.3.5　ResNet

相较于 VGG 的 19 层和 GoogLeNet 的 22 层，ResNet 可以提供 18、34、50、101、152 甚至更多层的网络，同时获得更好的精度。但是为什么要使用更深层次的网络呢？同时，如果只是网络层数的堆叠，那么为什么前人没有获得 ResNet 一样的成功呢？ResNet 是 2015 年 ImageNet 比赛的冠军，将识别错误率降低到了 3.6%，这个结果甚至超出了正常人眼识别。通过前面几个经典模型学习，我们可以发现随着深度学习的不断发展，模型的层数越来越多，网络结构也越来越复杂。那么是否加深网络结构，就一定会得到更好的效果呢？从理论上来说，假设新增加的层都是恒等映射，只要原有的层学出跟原模型一样的参数，那么深模型结构就能达到原模型结构的效果。换句话说，原模型的解只是新模型的解的子空间，在新模型的解的空间里应该能找到比原模型的解对应的子空间更好的结果。但是实践表明，增加网络的层数之后，训练误差往往不降反升。

He Kaiming 等提出了残差网络 ResNet 来解决上述问题，其基本思想如图 3-17 所示。如图 3-17(a)表示增加网络的时候，将 x 映射成 $y=F(x)$输出。如图 3-17(b)对图 3-17(a)作了改进，输出 $y=F(x)+x$。这时不是直接学习输出特征 y 的表示，而是学习 $y-x$。如果想学习出原模型的表示，只需要将 $F(x)$的参数全部设置为 0，则 $y=x$ 是恒等映射。

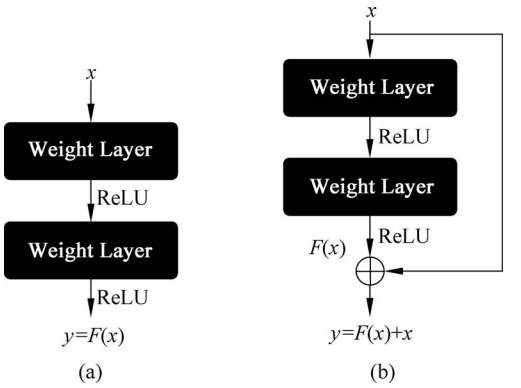


图 3-17　残差块设计思想

图 3-18 表示出了 ResNet-50 的结构，一共包含 49 层卷积和 1 层全连接，所以被称为 ResNet-50。

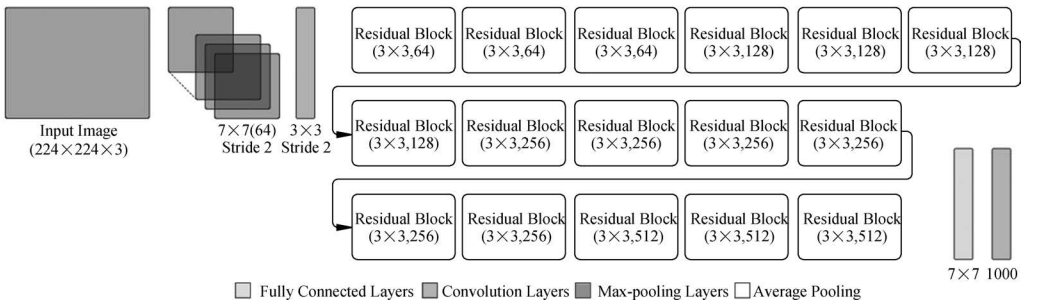


图 3-18　ResNet-50 模型网络结构示意图

视频讲解

3.4 案例：图像分类网络 VGG 在中草药识别任务中的应用

本节利用 PaddlePaddle 框架搭建 VGG 网络，实现中草药识别，如图 3-19 所示。本案例旨在通过中草药识别来让读者对图像分类问题初步了解，同时理解和掌握如何使用 PaddlePaddle 搭建一个经典的卷积神经网络。本案例支持在实训平台或本地环境操作，建议使用 AI Studio 实训平台。

图 3-19 中草药

- 实训平台：如果选择在实训平台上操作，无须安装实验环境。实训平台集成了实验必需的相关环境，代码可在线运行，同时还提供了免费算力，可以做到即使实践复杂模型也无算力之忧。
- 本地环境：如果选择在本地环境上操作，需要安装 Python3.7、PaddlePaddle 开源框架等实验必需的环境，具体要求及实现代码请参见百度 PaddlePaddle 官方网站。

3.4.1 方案设计

本案例的实现方案如图 3-20 所示，对于一幅输入的中草药图像，首先使用卷积神经网络 VGG 提取特征，获取特征表示；然后使用分类器(3 层全连接＋Softmax)获取属于每个中草药类别的概率值。在训练阶段，通过模型输出的概率值与样本的真实标签构建损失函数，从而进行模型训练；在推理阶段，选出概率最大的类别作为最终的输出。

3.4.2 整体流程

中草药识别流程如图 3-21 所示，包含如下 7 个步骤。

(1) 数据处理：根据网络接收的数据格式，完成相应的数据集准备及数据预处理操作，保证模型正常读取。

(2) 模型构建：设计卷积网络结构(模型的假设空间)。

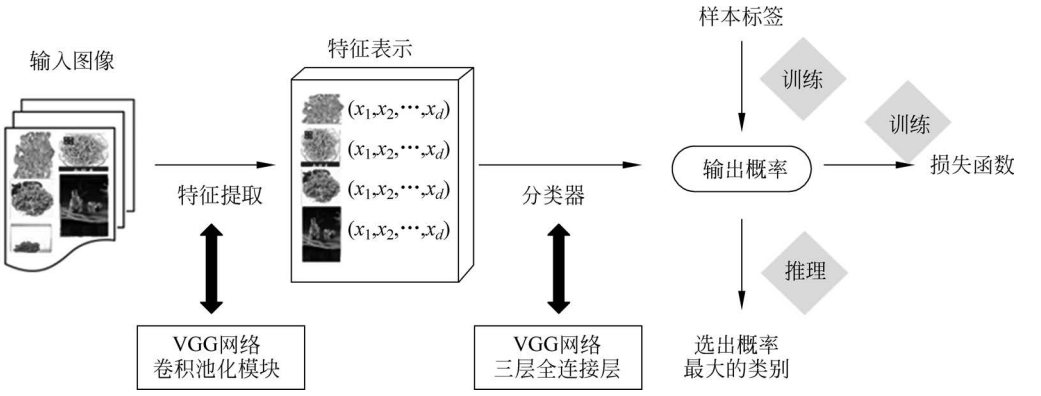


图 3-20　方案设计

（3）训练配置：声明模型实例，加载模型参数，指定模型采用的寻解算法（定义优化器）并加载数据。

（4）模型训练：执行多轮训练不断调整参数，以达到较好的效果。

（5）模型保存：将模型参数保存到指定位置，便于后续推理或继续训练使用。

（6）模型评估：对训练好的模型进行评估测试，观察准确率和 loss。

（7）模型推理及可视化：使用一张中草药图片来验证模型识别的效果，并可视化推理结果。

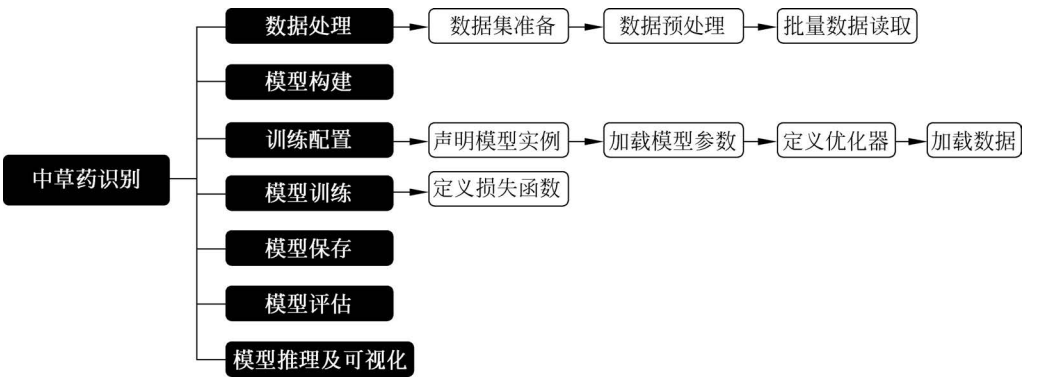
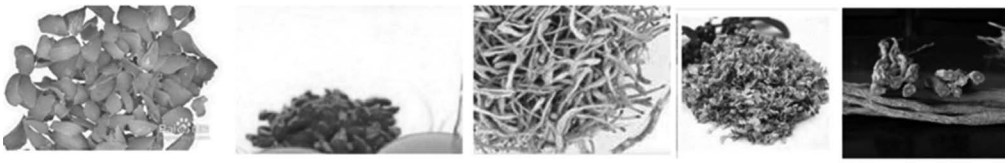


图 3-21　中草药识别流程

3.4.3　数据处理

1. 数据集介绍

本案例数据集 data/data105575/Chinese Medicine.zip 来源于互联网，分为五个类别，共 902 张图片，其中，百合 180 张图片，枸杞 185 张图片，金银花 180 张图片，槐花 167 张图片，党参 190 张图片，部分图片如图 3-22 所示。



百合　　枸杞　　金银花　　槐花　　党参

图 3-22　五类中草药

2. 数据预处理

图像分类网络对输入图片的格式、大小有一定的要求。数据预处理指将数据是输入到模型前，需要对数据进行预处理操作，使图片满足网络训练以及预测的需要。本案例主要应用了如下方法：

(1) 图像解码：将图像转为 NumPy 格式。

(2) 调整图片大小：将原图片中短边尺寸统一缩放到 256。

(3) 图像裁剪：将图像的长宽统一裁剪为 224×224，确保模型读入的图片数据大小统一。

(4) 归一化(Normalization)：通过规范化手段，把输入图像的分布改变成均值为 0，方差为 1 的标准正态分布，使得最优解的寻优过程明显会变得平缓，训练过程更容易收敛。

(5) 通道变换：图像的数据格式为[H,W,C](即高度、宽度和通道数)，而神经网络使用的训练数据的格式为[C,H,W]，因此需要对图像数据重新排列，例如[224,224,3]变为[3,224,224]。

对于图像分类问题，除了以上对图片数据进行处理外，还需要对数据作以下的处理：解压原始数据集；按照比例划分训练集与验证集，乱序生成数据列表；定义数据读取器和转换图片；加载数据集。

图 3-23　数据集文件目录结构

1) 解压原始数据集

首先使用 zipfile 模块来解压原始数据集，将 src_path 路径下的 zip 包解压至 target_path 目录下，解压后可以在 AI Studio 观察到数据集文件目录结构如图 3-23 所示。代码如下所示。

```
01. # 引入需要的模块
02. import os
03. import zipfile
04. import random
05. import json
06. import paddle
07. import sys
08. import numpy as np
09. from PIL import Image
```

```
import matplotlib.pyplot as plt
from paddle.io import Dataset
random.seed(200)

def unzip_data(src_path,target_path):
    if(not os.path.isdir(target_path + "Chinese Medicine")):
        z = zipfile.ZipFile(src_path, 'r')
        z.extractall(path = target_path)
        z.close()
```

2）按照比例划分训练集与验证集

本案例定义 get_data_list()遍历文件目录和图片，按照 7∶1 的比例划分训练集与验证集，之后打乱数据集的顺序并生成数据列表，生成的训练集数据的格式如图 3-24 所示。代码如下所示。

```
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_185.jpg  1
/home/aistudio/data/Chinese Medicine/gouqi/u=4169302532,3747596770&fm=26&gp=0.jpg   4
/home/aistudio/data/Chinese Medicine/gouqi/cgcjyfyj (98).jpg    4
/home/aistudio/data/Chinese Medicine/huaihua/huaihua_146.jpg    3
/home/aistudio/data/Chinese Medicine/huaihua/huaihua_50.jpg 3
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_47.jpg   1
/home/aistudio/data/Chinese Medicine/huaihua/huaihua_160.jpg    3
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_117.jpg  1
/home/aistudio/data/Chinese Medicine/dangshen/dangshen_97.jpg   1
/home/aistudio/data/Chinese Medicine/gouqi/u=228093977,1203529990&fm=26&gp=0.jpg    4
```

图 3-24　训练集数据

```
def get_data_list(target_path,train_list_path,eval_list_path):
    '''
    生成数据列表
    '''
    #存放所有类别的信息
    class_detail = []
    #获取所有类别保存的文件夹名称
    data_list_path = target_path + "Chinese Medicine/"
    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,['baihe', 'gouqi','jinyinhua','huaihua','dangshen']
    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张图片
        class_sum = 0
        #获取类别路径
        path = data_list_path + class_dir
        # 获取所有图片
        img_paths = os.listdir(path)
        for img_path in img_paths:                  # 遍历文件夹下的每张图片
            name_path = path + '/' + img_path       # 每张图片的路径
            if class_sum % 8 == 0:                  # 每8张图片取一个作验证数据
               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          #每类图片的数目
            all_class_images += 1   #所有类图片的数目

        # 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 #该类数据的测试集数目
        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_parameters = {
    "src_path":"/home/aistudio/data/data124873/Chinese Medicine.zip", # 原始数据集路径
    "target_path":"/home/aistudio/data/",                    # 要解压的路径
    "train_list_path": "/home/aistudio/data/train.txt",      # train.txt 路径
    "eval_list_path": "/home/aistudio/data/eval.txt",        # eval.txt 路径
    "label_dict":{},                                         # 标签字典
    "readme_path": "/home/aistudio/data/readme.json",        # readme.json 路径
    "class_dim": -1,                                         # 分类数
}
src_path = train_parameters['src_path']
target_path = train_parameters['target_path']
train_list_path = train_parameters['train_list_path']
eval_list_path = train_parameters['eval_list_path']

# 调用解压函数解压数据集
unzip_data(src_path,target_path)

# 划分训练集与验证集,乱序,生成数据列表
# 每次生成数据列表前,首先清空 train.txt 和 eval.txt
with open(train_list_path, 'w') as f:
    f.seek(0)
    f.truncate()
with open(eval_list_path, 'w') as f:
    f.seek(0)
    f.truncate()
#生成数据列表
get_data_list(target_path,train_list_path,eval_list_path)
```

3）定义数据读取器

接下来，定义数据读取器类 dataset，用来加载训练和验证时要使用的数据，也包括图片格式的修改：图片转为 RGB 格式、数据维度由（H，W，C）转为（C，H，W）、图片大小 resize 为 224×224，其过程与 2.6 节定义方式相同。代码如下所示。

```
# 定义数据读取器
class dataset(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((224, 224), Image.BILINEAR)
      #img = rand_flip_image(img)
      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])

```

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

4）加载数据集

最后，使用 paddle.io.DataLoader 模块实现数据加载，并且指定训练用参数：训练集批次大 batch_size 为 32，乱序读入；验证集批次大小为 8，不打乱顺序。通过大量实验发现，模型对最后出现的数据印象更加深刻。训练数据导入后，越接近模型训练结束，最后几个批次数据对模型参数的影响越大。为了避免模型记忆影响训练效果，需要进行样本乱序操作。如果数据预处理耗时较长，推荐使用 paddle.io.DataLoader API 中的 num_workers 参数，设置进程数量，实现多进程读取数据。代码如下所示。

```
01. #训练数据加载
02. train_dataset = dataset('/home/aistudio/data',mode = 'train')
03. train_loader = paddle.io.DataLoader(train_dataset, batch_size = 32, shuffle = True)
04. #评估数据加载
05. eval_dataset = dataset('/home/aistudio/data',mode = 'eval')
06. eval_loader = paddle.io.DataLoader(eval_dataset, batch_size = 8, shuffle = False)
```

至此，完成了数据读取、提取数据标签信息、批量读取和加速等过程，接下来将处理好的数据输入到神经网络。

3.4.4　模型构建

本案例使用 VGG 网络进行中草药识别。VGG 是当前最流行的 CNN 模型之一，于 2014 年由 Simonyan 和 Zisserman 在 ICLR 2015 会议上的论文 *Very Deep Convolutional Networks for Large-scale Image Recognition* 提出。VGG 命名于论文作者所在的实验室 Visual Geometry Group。VGG 设计了一种大小为 3×3 的小尺寸卷积核和池化层组成的基础模块，通过堆叠上述基础模块构造出深度卷积神经网络，该网络在图像分类领域取得了不错的效果，在大型分类数据集 ILSVRC 上，VGG 模型仅有 6.8% 的 top-5 test error。VGG 模型一经推出就很受研究者们的欢迎，因为其网络结构的设计合理，总体结构简明，且可以适用于多个领域。VGG 的设计为后续研究者设计模型结构提供了思路。

如图 3-25VGG 网络所示，VGG 网络所有的 3×3 卷积都是等长卷积，包括步长和填充为 1，因此特征图的尺寸在每个模块内大小不变。特征图每经过一次池化，其高度和宽度减少一半(1/2 Pool)，作为弥补通道数增加一倍，最后通过三层全连接层和 Softmax 层输出结果。

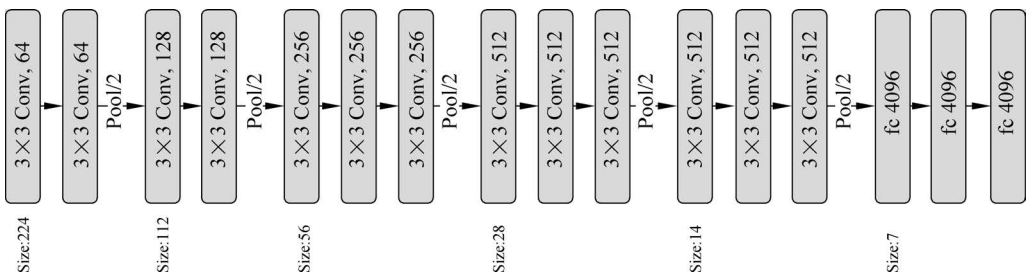


图 3-25　VGG 网络结构

VGG网络引入“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，而不是以“层”为单位组装网络。定义类ConvPool实现“模块化”，通过类函数add_sublayer创建网络层列表，其中包括卷积、ReLU、池化，并在前向计算函数forward中通过named_children()实现模块内网络层的先后顺序，代码如下所示。

```
# 定义卷积池化网络
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__()

        # groups代表卷积层的数量
        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():
            # print(prefix,sub_layer)
            x = sub_layer(x)
        return x
```

接下来，根据上述模块 ConvPool 定义网络 VGGNet，在构造函数 init()中多次调用 ConvPool 模块，生成 VGGNet 的每一个模块实例，再通过前向计算函数 forward 完成计算图，代码如下所示。注意：输出由三个全连接层组成，全连接层之间使用 Dropout 层防止过拟合。

```
# VGG 网络
class VGGNet(paddle.nn.Layer):
    def __init__(self):
        super(VGGNet, self).__init__()
        # 5 个卷积池化操作
        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
        # 三个全连接层
        self.fc01 = paddle.nn.Linear(self.pool_5_shape, 4096)
        self.drop1 = paddle.nn.Dropout(p = 0.5)
        self.fc02 = paddle.nn.Linear(4096, 4096)
        self.drop2 = paddle.nn.Dropout(p = 0.5)
        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)
        # print('convpool01_shape:', out.shape)       #[8, 64, 112, 112]
        out = self.convpool02(out)
        # print('convpool02_shape:', out.shape)       #[8, 128, 56, 56]
        out = self.convpool03(out)
        # print('convpool03_shape:', out.shape)       #[8, 256, 28, 28]
        out = self.convpool04(out)
        # print('convpool04_shape:', out.shape)       #[8, 512, 14, 14]
```

```
        out = self.convpool05(out)
        # print('convpool05_shape:', out.shape)        #[8, 512, 7, 7]

        out = paddle.reshape(out, shape=[-1, 512*7*7])
        out = self.fc01(out)
        out = self.drop1(out)
        out = self.fc02(out)
        out = self.drop2(out)
        out = self.fc03(out)

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

3.4.5 训练配置

本案例使用 Adam 优化器。2014 年 12 月，Kingma 和 Lei Ba 提出了 Adam 优化器。该优化器对梯度的均值(即一阶矩估计，First Moment Estimation)和梯度的未中心化的方差(即二阶矩估计，Second Moment Estimation)进行综合计算，获得更新步长。Adam 优化器实现起来较为简单，且计算效率高，需要的内存更少，梯度的伸缩变换不会影响更新梯度的过程，超参数的可解释性强，且通常超参数无须调整或仅需微调。如下代码通过 train_parameters.update 更新参数字典，即：

- 输入图片的 shape。
- 训练轮数。
- 训练时输出日志的迭代间隔。
- 训练时保存模型参数的迭代间隔。
- 优化函数的学习率。
- 保存的路径。

```
# 参数配置,要保留之前数据集准备阶段配置的参数,所以使用 update 更新字典
train_parameters.update({
    "input_size": [3, 224, 224],
    "num_epochs": 35,
    "skip_steps": 10,
    "save_steps": 100,
    "learning_strategy": {
        "lr": 0.0001
    },
    "checkpoints": "/home/aistudio/work/checkpoints"
})
```

为了更直观地看到训练过程中的 loss 和 acc 变化趋势，需要实现画出折线图的函数，代码如下所示。

```
# 折线图,用于观察训练过程中 loss 和 acc 的走势
def draw_process(title,color,iters,data,label):
    plt.title(title, fontsize = 24)
    plt.xlabel("iter", fontsize = 20)
    plt.ylabel(label, fontsize = 20)
    plt.plot(iters, data,color = color,label = label)
    plt.legend()
    plt.grid()
    plt.show()
```

3.4.6　模型训练

训练模型并调整参数的过程,观察模型学习的过程是否正常,如损失函数值是否在降低。本案例考虑到时长因素,只训练了 35 个 epoch,每个 epoch 都需要在训练集与验证集上运行,并打印出相应的 loss、准确率以及变化图,如图 3-26 所示。训练步骤如下:

(1) 模型实例化。

(2) 配置 loss 函数。

(3) 配置参数优化器。

(4) 开始训练,每经过 skip_step 打印一次日志,每经过 save_step 保存一次模型。

(5) 训练完成后画出 acc 和 loss 变化图。

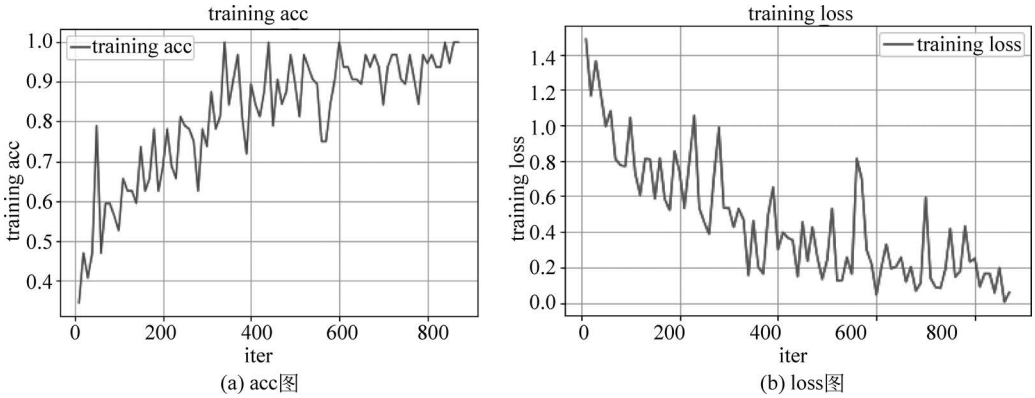


图 3-26　acc 图和 loss 图

代码如下所示。

```
model = VGGNet()
model.train()
# 配置 loss 函数
cross_entropy = paddle.nn.CrossEntropyLoss()
# 配置参数优化器
optimizer = paddle.optimizer.Adam(learning_rate = train_parameters['learning_
    strategy']['lr'], parameters = model.parameters())
```

```
07.
08. steps = 0
09. Iters, total_loss, total_acc = [], [], []
10.
11. for epo in range(train_parameters['num_epochs']):
12.    for _, data in enumerate(train_loader()):
13.       steps += 1
14.       x_data = data[0]
15.       y_data = data[1]
16.       predicts, acc = model(x_data, y_data)
17.       loss = cross_entropy(predicts, y_data)
18.       loss.backward()
19.       optimizer.step()
20.       optimizer.clear_grad()
21.       if steps % train_parameters["skip_steps"] == 0:
22.         Iters.append(steps)
23.         total_loss.append(loss.numpy()[0])
24.         total_acc.append(acc.numpy()[0])
25.         #打印中间过程
26.         print('epo: {}, step: {}, loss is: {}, acc is: {}'\
27.             .format(epo, steps, loss.numpy(), acc.numpy()))
28.       #保存模型参数
29.       if steps % train_parameters["save_steps"] == 0:
30.           save_path = train_parameters["checkpoints"] + "/" + "save_dir_" + str
    (steps) + '.pdparams'
31.           print('save model to: ' + save_path)
32.           paddle.save(model.state_dict(),save_path)
33. paddle.save(model.state_dict(),train_parameters["checkpoints"] + "/" + "save_dir_
    final.pdparams")
34. draw_process("training loss","red",Iters,total_loss,"training loss")
35. draw_process("training acc","green",Iters,total_acc,"training acc")
```

本节尝试改变 batch_size 优化模型。batch_size 指的是一次训练所选取的样本数。在网络训练过程中，batch_size 过大或者过小都会影响训练的性能和速度，如果 batch_size 过小，那么花费时间多，同时梯度振荡严重，不利于收敛；如果 batch_size 过大，那么不同 batch 的梯度方向没有任何变化，容易陷入局部极小值。例如，在本案例中，我们直接使用神经网络通常设置的 batch_size＝16，训练 35 轮之后模型在验证集上的准确率为 0.825。在合理范围内，增大 batch_size 会提高显存的利用率，提高大矩阵乘法的并行化效率，减少每轮需要训练的迭代次数。在一定范围内，batch size 越大，其确定的下降方向越准，引起训练时准确率振荡越小。在本案例中，我们设置 batch_size＝32，同样训练 35 轮，模型在验证集上的准确率为 0.842。当然，过大的 batch_size 同样会降低模型性能。在本案例中，我们设置 batch_size＝48，训练 35 轮之后模型在验证集上的准确率为 0.817。从以上的实验结果，可以清楚地了解到，在模型优化的过程中，找到合适的 batch_size 是很重要的。

3.4.7　模型评估和推理

1. 模型评估

使用验证集来评估训练过程保存的最后一个模型，首先加载模型参数，之后遍历验证集进行预测并输出平均准确率。与训练部分的代码不同，模型评估不需要参数优化，因此使用验证模式 model_eval.eval()。代码如下所示。

```
# 模型评估
# 加载训练过程保存的最后一个模型
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 = []
# 开始评估
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))
```

2. 模型推理

模型推理阶段首先采用与训练过程同样的图片转换方式对测试集图片进行预处理，然后使用训练过程保存的最后一个模型预测测试集中的图片。代码如下所示。

```
import time
def load_image(img_path):
    '''
    预测图片预处理
    '''
    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 # HWC to CHW 及归一化
    return img

label_dic = train_parameters['label_dict']

# 加载训练过程保存的最后一个模型
model__state_dict = paddle.load('work/checkpoints/save_dir_final.pdparams')
```

```
model_predict = VGGNet()
model_predict.set_state_dict(model__state_dict)
model_predict.eval()
infer_imgs_path = os.listdir("infer")
# print(infer_imgs_path)

# 预测所有图片
for infer_img_path in infer_imgs_path:
    infer_img = load_image("infer/" + infer_img_path)
    infer_img = infer_img[np.newaxis,:, : ,:] #reshape(-1,3,224,224)
    infer_img = paddle.to_tensor(infer_img)
    result = model_predict(infer_img)
    lab = np.argmax(result.numpy())
    print("样本: {},被预测为:{}".format(infer_img_path,label_dic[str(lab)]))
    img = Image.open("infer/" + infer_img_path)
    plt.imshow(img)
    plt.axis('off')
    plt.show()
    sys.stdout.flush()
    time.sleep(0.5)
```

3.5 本章小结

传统图像分类由多个阶段构成,框架较为复杂。基于 CNN 的深度学习模型采用端到端一步到位的方式,大幅提高了分类准确度。本章首先介绍了 CNN 的基本理论,包括卷积层、池化层等,并以代码实现了一个基本的 CNN 模型。接着,讲述了卷积神经网络的历史发展过程,介绍了多款经典的 CNN 模型。最后,使用 PaddlePaddle 构建了经典的 VGG 图像分类网络,并在中草药数据集上实现了中草药识别。通过本案例,读者将不仅会掌握卷积神经网络的相关原理,而且会进一步熟悉通过开源框架求解深度学习任务的实践过程。读者可以在此案例的基础上,尝试开发自己感兴趣的图像分类任务。