第 5 章

别具慧眼：观影识图

随着人工智能技术的进步，计算机视觉在过去几年取得了长足的进步。在今天，配合深度学习，计算机视觉可以帮助汽车查明周围的行人和汽车，并避开它们；还可使人脸识别技术变得更加有效率和精准，如仅通过刷脸就能解锁手机或者门锁。有很多分享图片的手机应用。在上面，能看到美食，酒店或美丽风景的图片。有些公司在这些应用上使用了深度学习技术向用户展示最为生动美丽以及与用户最为相关的图片。这些新技术的应用甚至催生了新的艺术类型。计算机视觉的高速发展标志着新型应用产生的可能，这是几年前，人们所不敢想象的。人们对于计算机视觉的研究富有想象力和创造力，由此衍生出新的神经网络结构与算法，这实际上启发人们去创造计算机视觉与其他领域的交叉成果。期望有一天这项技术能被更广泛地应用，并带给人类更美好的生活。这一章就让我们一同学习计算机视觉相关技术，了解计算机是如何观影识图的。

- 计算机视觉中的图像
- 图像的几何变换
- 图像的特征提取与识别
- 结合深度学习的应用实例

5.1 计算机视觉

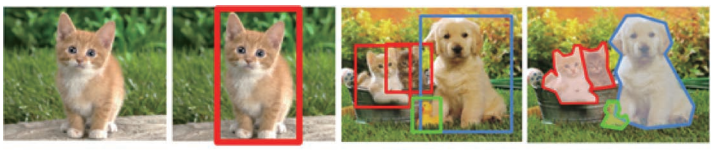
由于计算机视觉在智能安防、医疗健康、机器人以及自动驾驶等领域的广泛应用，因此其近年来变得越来越重要和有效。视觉任务识别是许多应用的核心模块。当前，基于计算机视觉的技术已经从感知模式转变为可以理解现实世界的智能的计算系统。因此，掌握计算机视觉和机器学习知识是现代创新企业所需的重要技能，并且在不久的将来可能变得更加重要。

那么，什么是计算机视觉呢？

计算机视觉是一个跨学科领域，专注于利用计算机技术实现对数字图像与视频的高层次理解。从工程角度来看，计算机视觉的目标是寻找一种能够与人类视觉系统实现相同功

能的自动化任务。

人类用眼睛和大脑观察、感知和理解周围的世界。例如，以图 5-1(a)所示的图像为例，人类很容易在图像中看到“猫”，从而实现对图像进行分类(分类任务)、在图像中定位猫[分类加定位任务，如图 5-1(b)所示]、定位并标记图像中存在的所有对象[目标检测任务，如图 5-1(c)所示]、分割图像中存在的各个对象[实例分割任务，如图 5-1(d)所示]。计算机视觉寻求开发方法以复制人类视觉系统中令人极为惊异的能力之一，即纯粹使用从各种物体反射到眼睛的光来推断真实世界的特征。



(a) 分类任务 (b) 分类加定位任务 (c) 目标检测任务 (d) 实例分割任务

图 5-1 计算机视觉任务(来源：斯坦福大学计算机视觉实验室课程)

由于计算机视觉和视觉传感器技术领域的重大进步，计算机视觉技术如今正在各种各样的现实应用中使用，如智能人机交互、机器人和多媒体。预计下一代计算机甚至可以与人类同水平地理解人类行为和语言，代表人类执行一些任务，并以智能方式响应人类命令。

5.2 认识图像

计算机视觉是人工智能的一个分支，也是基于图像的基础进行的研究。可以将图像处理视为计算机视觉的预处理步骤。更确切地说，图像处理的目标是提取基本图像基元，包括边缘和角点、滤波、形态学操作等，这些图像基元通常表示以像素形式表示。

人们看到的图像数据是以二维形式展现的，其中有的图像是缤纷多彩、富有表现力的，有的图像表现为沉郁顿挫的黑白风格，甚至有的图像只有纯黑和纯白两种颜色。诸如此类，都是图像的不同表现形式，本节即具体介绍它们的区别。

5.2.1 彩色图像

常用的颜色模型包括 RGB、XYZ、HSV/HSL、LAB、LCH、YIQ 和 YUV 等。本书仅以 RGB 颜色模型为例介绍色彩空间。

RGB 颜色模型即红、蓝、绿三原色模型，其将红、绿、蓝 3 种不同颜色根据不同的亮度配比进行混合，从而表现出不同的颜色。由于在实现上使用了 3 种颜色的定量配比，因此该模型也被称为加色混色模型。通过 3 种基本颜色的混合叠加来表现任意一种颜色的方法特别适用于显示器等主动发光的显示设备。值得一提的是，RGB 颜色的展现依赖设备的颜色空间，不同设备对 RGB 颜色值的检测不尽相同，表现出来的结果也存在差异，如一些手机屏

幕颜色特别逼真、绚丽，而另一些就难以令人满意。图 5-2 所示为 RGB 颜色模型的空间结构，这是一个立方体结构，3 个坐标轴分别代表 3 种颜色。从理论上讲，任何一种颜色都包含在该立方体结构中。

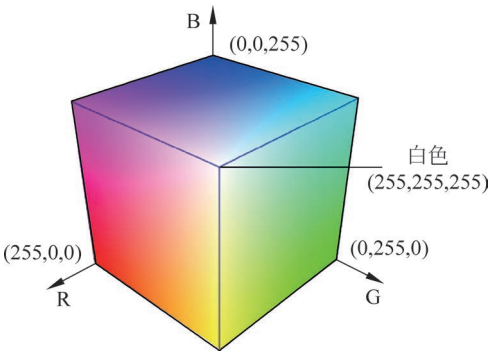


图 5-2　RGB 颜色模型的空间结构

有网页设计或开发经验的读者应该对 RGB 颜色模型有一些了解，如 #FFFFFF 代表纯白色，#FF0000 代表正红色。这是采用十六进制对 24bit 展示模式的一种表示方法，其中前两位十六进制数字表示红色，中间两位表示绿色，最后两位表示蓝色，每一种颜色采用 8bit 无符号整数（表示范围为[0,255]的整数区间）表示，3 种颜色共计占用 24bit。

例如，使用一个元组表示正红色，元组中元素的顺序为红、绿、蓝，则正红色可以表示为(255,0,0)；再如，黄色是由红色和绿色两种颜色叠加产生的，所以正黄色可以表示为(255,255,0)。如果想要减少某种黄色的亮度，只需把红、绿两种颜色同时按比例减少即可。改变它们的配比，可以使混合后的颜色向某种颜色偏移，如橘黄色会更加偏向红色。

以 RGB 颜色模型为例，可以认为一幅图像的颜色是由包含了红、绿、蓝 3 种不同通道的颜色进行叠加混合而产生的。从数学角度来看，对于一幅彩色图像，可以认为其是由 3 个二维矩阵进行叠加混合而产生的，每一个二维矩阵记录了某种颜色在不同位置的亮度值，因此 3 个二维矩阵就对应了 3 个最基本的颜色通道。有人说一幅图像就是一个矩阵，其实这样的表述是不严谨的。对于彩色图像来讲，一幅图像不仅包含了一个矩阵，而是包含了红、绿、蓝 3 种不同颜色信息的 3 个矩阵。那么，是否存在一幅图像就是一个矩阵的情况呢？当然有！我们下面介绍的灰度图像与二值图像就是如此。

5.2.2　灰度图像

如图 5-3 所示，只需用一个二维矩阵即可表示一个灰度图像。从图 5-3 中可以看到该 28×28 的图像表现的是一个数字 6。

我们在生活中也经常接触到灰度图像，如非彩色打印书籍中的图像就是灰度图像，黑白照片也是灰度图像。这类图像有一个特点，即这些图像虽然没有包含其他颜色的信息，但我们依然能够从这些图像中获取轮廓、纹理、形状等特征。

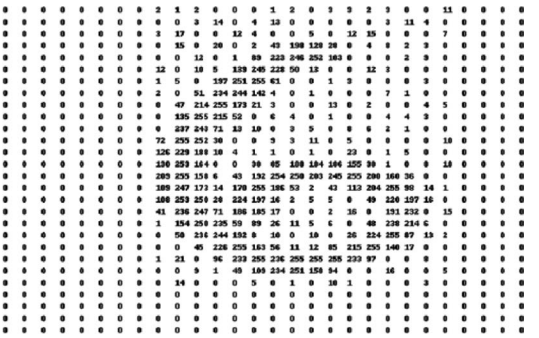


图 5-3 数字 6 图像的矩阵表示

我们的直观感觉是正确的，这也说明了灰度图像相对于彩色图像缺少了具体的颜色信息，但是，灰度图像依然能够完好地展示出图像中各个部分的轮廓、纹理、形状等关键特征，同时灰度图片的存储结构相对于彩色图片更为简单。这样便会产生一个优点，如果想要提取图像中的特征与颜色无太多关联，那么就可以选择将彩色图像处理成灰度图像的预处理方式。由于灰度图像的结构更为简单，同时关键信息又不大会损失，这样就可以极大地减少计算量。

我们可以通过手机来拍摄彩色照片，同样也可以拍摄出黑白照片。在这个过程中，黑白照片和彩色照片是否存在转换关系呢？答案是肯定的。我们可以通过数学公式将 RGB 模型中的红、绿、蓝 3 个矩阵进行合并，合并成一个矩阵，这个矩阵就是代表了灰度图像的矩阵。

即便是黑色的程度也是可以量化的，介于黑色和白色之间的颜色就是灰色，那么直接量化的就是灰色的程度，这个程度就是灰度。一般的量化方法是将纯白色作为 255，纯黑色作为 0，在 0～255 中，使用对数的方法划分具体数值进行量化。当然这个数值可以是浮点数。

5.2.3 二值图像

二值图像中只有纯黑色和纯白色两种颜色，没有中间过渡的灰色，如图 5-4 所示。其数据结构也是一个二维矩阵，只不过其中的数值只有 0 和 1 两种。

二值图像是在灰度图像的基础上进一步计算的结果。其计算过程比较简单，指定一个阈值，判断图像中不同点处的灰度值，如果该点处的灰度值高于阈值，则该点值为 1，否则为 0，这样就实现了灰度图像的二值化。

可以看到，二值图像的空间占用量进一步减少，每一个像素点只需要 1bit 即可表示，这对于表示字符这类非黑即白形式的图像非常具有优势。由于二值图像是在灰度图像的基础上通过阈值判断产生的，因此会缺少细节部分，只能显示图像的大致轮廓。二值图像的这一特性在图像分割等场景中具有较高的利用价值。

图 5-4　二值图像示例

5.3　几何变换

为了让计算机更容易、清楚地理解图像的内容，通常会通过一些预处理程序对图像进行整理再放进神经网络，让计算机进行更有效地学习。有时会对图像以平移、旋转、缩放等方法进行预处理来增加照片数量，或者通过增加、减少图像的噪声或是特征提取来提高神经网络的精准度。本节即介绍图像的几何变换技术。

图像的几何变换就是指在不改变图像原有内容的基础上，改变图像的像素空间位置，以达到变换图像中像素点位置的目的。图像的几何变换一般包括图像空间变换和插值运算，常见的变换运算包括平移、旋转、缩放等。

5.3.1　平移

图像的平移比较容易理解，与人们在实际生活中将物体搬移类似。图像是由若干个像素点组成的，对于彩色图像来说，该像素点包含 R、G、B 3 种颜色；对于灰度图像来说，其就是一个简单的矩阵，该矩阵中某一个元素的数值就是图像中该像素点的灰度值。图像平移过程如图 5-5 所示。

255	255	255	255
0	0	255	0
0	255	0	0
255	255	255	255

(a)

	255	255	255	255
	0	0	255	0
	0	255	0	0
	255	255	255	255

(b)

0	0	0	0
0	255	255	255
0	0	0	255
0	0	255	0

(c)

图 5-5　图像平移过程

图 5-5 演示的是某一个 4 行 4 列共计 16 个像素点的灰度图像向右下角平移一个单位的过程。可以看到，图 5-5(a)是一个完整的字母 Z 的图形，在向右下角平移一个单位时，由

于图像尺寸的限制，在图 5-5(b)中位于阴影区域外部的像素点必然会被丢弃。在图 5-5(c)中，使用灰度值为 0 的像素点填补空白部分。

综上，在图像平移过程中必然会造成某些像素点的丢失，同时也会导致图像中产生空白区域，空白区域可以自己指定像素进行填充。当然，也可以选择先扩展图像的画布，然后进行平移，这样只会引入一些空白部分，而不会导致像素点丢失。图 5-6 所示为图像平移效果。

图 5-6　图像平移效果

可以看到，对图像进行平移操作其实就是对图像中的各个像素点进行平移操作，或者说对其坐标轴进行移动。

将该平移过程用矩阵的形式表示，如下：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{5-1}$$

由式(5-1)可以发现，该过程是一个非常简单的线性变换过程，只需进行矩阵的加法运算即可。

5.3.2　旋转

旋转也是一个线性变换过程。如图 5-7 所示，在平面直角坐标系中存在某一点 A，要想将点 A 移动到点 B，该如何操作呢？

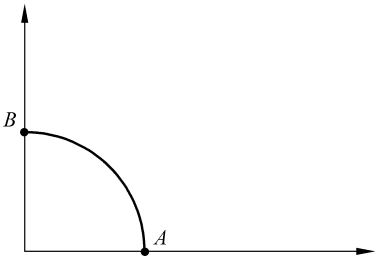


图 5-7　坐标点旋转

将点 A 旋转到点 B，该旋转过程可用如下公式表述：

$$\begin{cases} x_B = \cos\left(\frac{\pi}{2}\right) x_A - \sin\left(\frac{\pi}{2}\right) y_A \\ y_B = \sin\left(\frac{\pi}{2}\right) x_a + cos\ \left(\frac{\pi}{2}\right) y_A \end{cases} \tag{5-2}$$

更一般地，可以归纳出使点 A 旋转到点 B 的数学公式：

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_A \\ y_A \end{bmatrix} \tag{5-3}$$

可以看到，图像旋转是一个矩阵相乘的过程。

5.3.3　缩放

图像的缩放可通过矩阵相乘实现。

要将图像中某一个点的位置向中心移动若干倍，只需要将其横纵坐标值减小到若干分之一即可。由于图像是由无数个这样的点组成的，因此图像的缩放也是类似的。我们可以用矩阵乘法的形式来表示：

$$\begin{bmatrix} x_C \\ y_C \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \begin{bmatrix} x_B \\ y_B \end{bmatrix} = a \begin{bmatrix} x_B \\ y_B \end{bmatrix} \tag{5-4}$$

式中，a 为缩放的比例。

如果 $a<1$，则表示将图像缩小；如果 $a>1$，则表示将图像放大。将图像缩小必然会导致一些点的缺失；而将图像放大也会引入一些新的点，但新的点并不能用随便一个数值进行填充，而要通过一系列数学运算产生，该过程称为插值。常用的插值方法有最近邻插值、双线性插值等。

5.4　图像特征

可以通过一个人的面部来识别这个人的身份，虽然难以用直白的叙述来表示大脑是依据怎样的机制通过人脸识别人的身份的，但一定是通过某种机制提取一个人的面部特征，再通过这些面部特征进行身份识别的。这样也就能够解释为什么子女与父母长得比较像：这是因为他们的面部特征相似，只不过这些面部特征难以用语言显式地进行表述。图像的识别也是一样的道理，通过一系列的算法提取出图像的高级特征，这个特征可以通过数学手段进行描述，称为特征描述子。通过提取核心、有用的成分，摒弃无关成分，可对图像进行表示。

空间滤波是指利用滤波器与原图像进行卷积运算，得到能够突显出原图某方面视觉特征的描述图，使用不同的滤波器会得到不同效果。滤波器通常为正方形（如 3 × 3），又称为核。这一节我们来认识可以对影像进行预处理程序的空间滤波技术。在介绍常用的滤波

器之前，我们先来学习卷积运算。

5.4.1 卷积运算

卷积运算在图像处理中应用十分广泛，许多图像特征提取方法都会用到卷积。以灰度图像为例，其在计算机中被表示为一个整数矩阵。如果使用一个形状较小的矩阵和该图像矩阵进行卷积运算，就可以得到一个新的矩阵，这个新的矩阵可以看作一幅新的图像。也就是说，通过卷积运算，可以将原图像变换为一幅新的图像。这幅新图像有时比原图像可更清楚地表达某些性质，我们就可以将其作为原图像的一个特征。这里用到的小矩阵就称为卷积核。通常，图像矩阵中的元素都为 0～255 的整数，但是卷积核中的元素可以是任意实数，如图 5-8 所示。

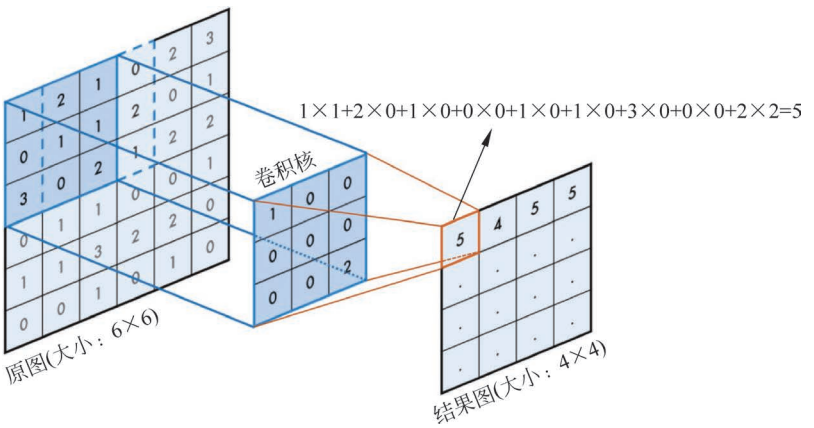


图 5-8 卷积核

图像与滤波器的卷积运算基本上就是在重复“移动—对齐—计算乘积和”这一过程，直到滤波器与图像的最后一格对齐为止。其中，滤波器与图像都可以视为向量或矩阵，最后得到的便是这两个向量卷积的结果。

下面介绍一个向量卷积的例子。现在有 $\boldsymbol{A}$、$\boldsymbol{B}$ 两个向量，分别是(3 4 5)、(1 2 3 4 5)。其中，$\boldsymbol{A}$ 是短向量，也可以把它当作卷积核；而 $\boldsymbol{B}$ 为长向量，可视为图像。

第 1 步，将短向量与长向量对齐，进行第一次乘积和运算，便可以得到第一个结果向量，如图 5-9(a)所示。

第 2 步，移动短向量，在此以一次移动一格为例，此时短向量会对齐长向量的第二个元素。接着进行第二次乘积和运算，便可以得到第二个结果向量，如图 5-9(b)所示。

第 3 步，将短向量移动一格，此时短向量会对齐长向量的第三个元素，而短向量的结尾也对齐长向量的最后一个元素。完成这次乘积和运算之后，便可以得到两个向量的卷积运算和，如图 5-9(c)所示。

二维向量的运算是在进行图像处理时会用到的运算，其与一维向量运算的不同之处在

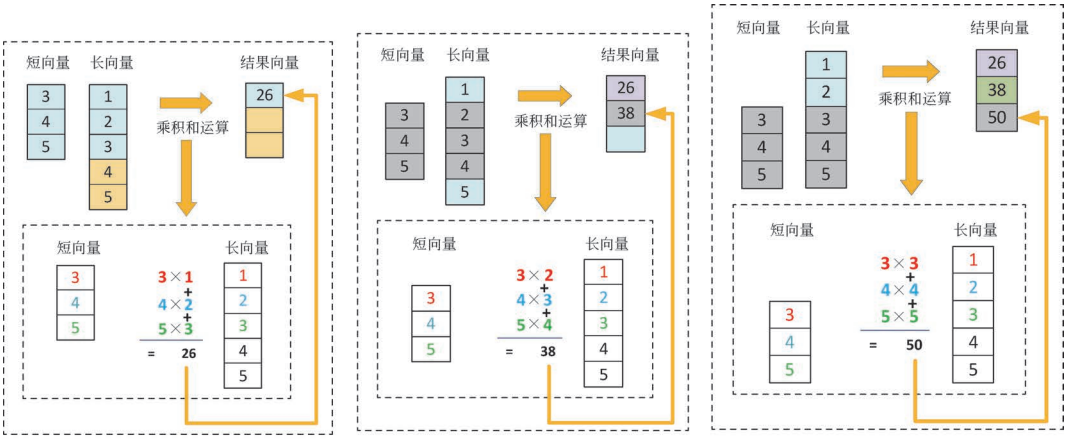


图 5-9　一维向量的卷积运算

于需要沿着横向与纵向两个方向进行移动。下面以 3×3 滤波器与 4×4 图像的卷积为例进行介绍。

第 1 步，将小矩阵与大矩阵对齐后，进行乘积求和，得到第一个卷积结果，如图 5-10(a)所示。

第 2 步，每次移动一格，如图 5-10(b)所示，大矩阵向右移动一格，小矩阵与之对齐后，进行乘积求和运算。

第 3 步，在卷积核横向到达最右端之后，将卷积核向下移动一格，即从最左端开始重复相同的步骤，对齐之后进行乘积求和运算，如图 5-10(c)所示。

第 4 步，将小矩阵与大矩阵的最后一个元素对齐后，进行乘积求和运算，如图 5-10(d)所示。

从上面的运算可以发现，卷积结果通常比原向量要小。有时为了使卷积后得到的结果与原向量大小保持一致，会先在原向量周围填补 0(Zero-Padding)再进行卷积运算，以维持相同大小。

5.4.2　认识滤波器

平滑滤波器又称为平均滤波器(average filter)，顾名思义，便是将卷积核所运算的区域取平均输出。平滑滤波器最主要的用途就是模糊化(blurring)以及减少噪声(noise reduction)。以下以 3×3 卷积核为例进行介绍。由于 3×3 的卷积核是每 9 格取一个值，因此通过 1 将每格的数值取出相加后，再除以 9，以得到平均数，如图 5-11 所示。根据每一格所占比例不同，将离中心较远的权重降低，就可以得到与原图最相近的结果，如图 5-12 所示。

中值滤波器(Median Filter)常用来减少噪声，但其并不是单纯地进行卷积，而是依照卷积的步骤将被卷积核运算的区块(如 3×3)取中间值(Median)来当作输出，而非直接进行乘

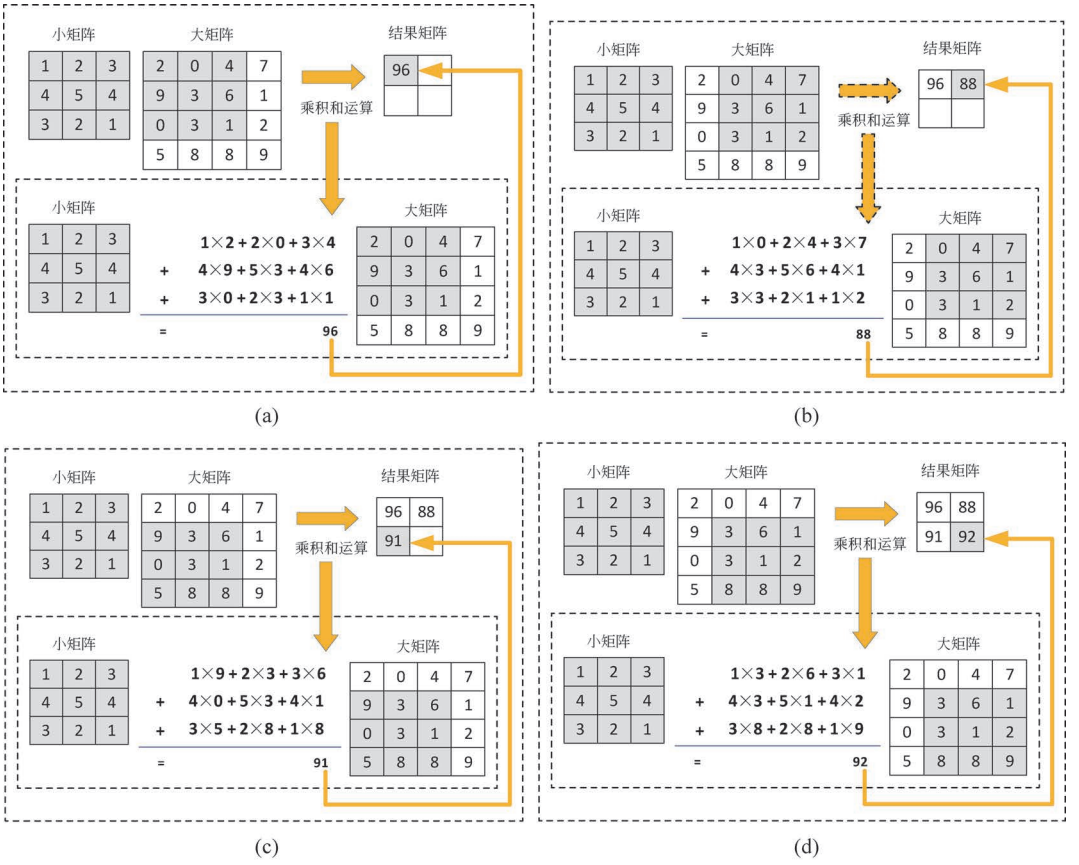


图 5-10 二维向量的卷积运算

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

图 5-11 平均滤波器示例

$$\frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

图 5-12 加权平均滤波器示例

积求和。由于噪声通常是与图像本身不相关的信息，与周围的像素亮度相差较大，因此在取中间值时通常不太会取到噪声，因而可以滤除噪声。对加入噪声图像使用不同滤波器后的效果如图 5-13 所示。

(a) 原始图像　(b) 加入强度20%椒盐噪声的图像

(c) 3×3窗的均值滤波图像　(d) 3×3窗的中值滤波图像

图 5-13　对加入噪声图像使用不同滤波器后的效果

索伯滤波器(Sobel Filter)也称为索伯算子(Sobel Operator)，经常应用于计算机视觉领域，其功能为边缘检测。索伯滤波器可以分为两个方向的边缘检测(图 5-14)，其数学意义是通过离散性差分运算计算图像亮度的梯度。

−1	0	+1
−2	0	+2
−1	0	+1

G_x

+1	+2	+1
0	0	0
+1	+2	−1

G_y

图 5-14　索伯滤波器两个方向的边缘检测

所谓的梯度计算，我们可以简单地把索伯滤波器理解为将索伯算子两侧相减，如图 5-14 中的 G_x 为 x 方向的索伯算子，将其叠加在图像上进行卷积运算时，便可以看成是用右侧的(+1，+2，+1)减掉左侧的(−1，−2，−1)。由于物体边缘通常会有明显的亮暗分界，通过相减，可以更加凸显边缘。图 5-15 展示了图像经过索伯滤波器凸显边缘的示例。

(a) 原始图像　　(b) 经过索伯滤波器处理的图像

图 5-15　图像经过索伯滤波器处理后的结果

5.5　图像识别

本节介绍利用深度学习让计算机分析经过处理的图像，以便从图像中得到更多有效且更便于人类解读的信息。目前图像识别常用的方法是利用人工神经网络识别图像中的对象，进而将图像进行分类或聚类。

事实上，要想得到一个相对来说健全、完整、准确率高的神经网络，仅靠三层神经元不太可能实现。其中间的隐藏层还可以加入更多层，实践中可能会高达数百、数千，甚至数万层以上。每加入新的一层，都会收到上一层输出的信息，每一层都对这些信息做进一步的处理，以得到更具代表性的信息表示，提升输出层判断结果的准确率。

像这样拥有一个输入层、多个隐藏层以及一个输出层，并且彼此之间层层相连的神经网络模型就称为深度神经网络(Deep Neural Network，DNN)。

有了基本的神经网络模型后，还需要经过一个学习过程，才能成功地识别目标对象，这个过程称为训练。深度神经网络模型中，一个神经元与自己上下层的每一个神经元都有连接，每个连接都由一个权重参数控制，这些参数决定了该神经元什么时候会被触发，进而向后传递信息。

然而，为了提升目标对象的识别准确率，人们会不断地增加其中的隐藏层，使得神经元的数量不断攀升，之前的连接也愈加错综，最终形成一个极其复杂的神经网络。这样的神经网络在训练上非常困难，因为每一次训练要调整变动的参数数量都非常庞大，不但很难找到最佳结果，在运算上也非常浪费计算资源。

因此，为了简化深度学习模型，人们引入了卷积运算，从而衍生出一种新的神经网络模型，即第 4 章介绍的卷积神经网络。在卷积神经网络中，一个神经元不会与其上下层的每一

个神经元都有连接，而是利用卷积核的概念找出图像的特征并向下传递。这些特征可以使人们更容易辨识出正确的对象，因此神经元个数相同的卷积神经网络的表现会比一般深度神经网络更加出色，大幅降低了需要训练的参数量。

经典的卷积神经网络架构 AlexNet 如图 5-16 所示，该架构在 2012 年度 ILSVRC 中获得了图像辨识的冠军，开创了深度学习的另一个时代。

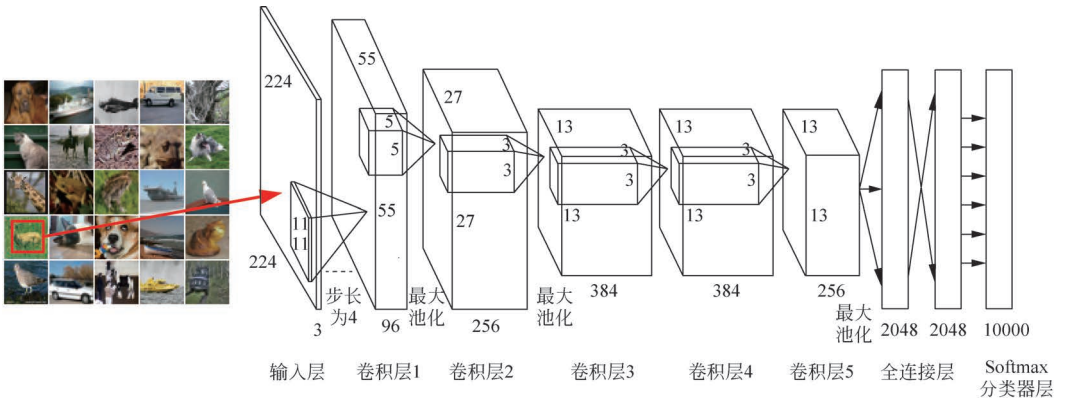


图 5-16　经典的卷积神经网络架构 AlexNet

5.5.1　卷积层

一个卷积神经网络大多是以卷积层为主体建构而成的。把输入的图像转换成 RGB 3 个通道的矩阵表示法，即一幅图像的每个像素都以 3 个数字表示，会得到 3 个与图像大小相符的矩阵，作为神经网络模型的输入。与单纯的深度神经网络模型不同，卷积神经网络中以卷积的核心作为基本单位，对模型的输入进行卷积运算，经过运算的矩阵可以视为一张新的图像。

多层卷积层相连时，新的输出图像就会通过一层层的滤镜在特征处愈发鲜明，称之为特征图(feature map)。除了输入的卷积层外，之后的卷积层都是用上层输出的特征图作为输入进行运算的。因此，一个拥有多个卷积层的神经网络还具备特征提取功能。利用这些特征图作为之后其他层的神经元输入，可以大大提升神经网络模型在辨识对象上的准确度。

5.5.2　池化层

一幅特征突出的图像在经过卷积层处理后，部分神经元可能会变得非常活跃，从而使得某个特定区域的计算量逐渐变大，难以处理。因此，在一个到数个卷积层后加入池化层，可以压缩卷积层输出的特征图，只留下主要特征，降低整个网络的计算复杂度，使训练网络的过程更加顺利。

池化层的功能主要是找出局部的特殊值。首先将一幅图像的 3 个信道(RGB)分开，在

每个信道中都将图像分成数个大小相同的区块，每个区块中的像素数量都相同，在每一个区块中只取一个值作为代表；然后将这些代表值组合起来形成一个新的通道；最后将 3 个通道的结果重叠，得到一张新的图片。

图 5-17 所示为最大池化运算示例，如果只看图片中的其中一个通道，假设一张图片的分辨率是 4×4，那么这 16 个像素值就可以表示成一个 4×4 的矩阵。将其切割成 4 个区块，每个区块都是一个 2×2 的矩阵，在每个区块的 4 个值中都挑选一个值，将这 4 个值组合成一个 2×2 的矩阵，其就是这个通道新的表示。一般而言，池化层分为平均池化层(Average Pooling Layer)和最大池化层(Max Pooling Layer)两种，其分别挑选每个区块的平均值和最大值作为代表。

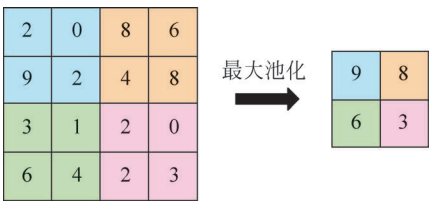


图 5-17 最大池化运算示例

虽然切割区块的大小及所选的代表值会影响池化层保留特征的效果，但每个区块中都保留了最具代表性的值，并且每个值之间的相对关系没有被改动，因此稍微降低的特征效果可以通过加深网络来改善。另外，从图 5-17 中可以看到矩阵的尺寸都从 4×4 缩小成了 2×2，计算量变为原来的 1/4，大大降低了计算资源的负担。

5.5.3 全连接层

全连接层中的每个神经元都会与上一层的所有神经元连接。在辨识对象的神经网络中，输出层就像一个分类器，其中每个神经元代表对象的类别，输出的值通常是辨识成这个对象的概率。而全连接层可以将该网络先前学习到的特征向量进行转换，以交给分类器进行分类。

不同于卷积层，经过全连接层计算的值不会受到学习到的特征所在位置的影响。例如，如果神经网络在一张图片的左上角学习到一只狗的特征，在另一张图片的右下角也学习到一只狗的特征，因为位置不同，所以其会表示成两个不同的特征向量。但是，经过全连接层的计算后，这两个特征向量会触发同一个神经元，使最后一层的分类器在狗的分类上有比较明显的响应。

在实践应用中，全连接层常利用卷积运算实现。以图 5-18 为例，AlexNet 的第一个全连接层输入是一个 6×6×256 的特征向量，输出为 4096×1 的向量，利用一个 6×6×256×4096 的卷积层进行运算，也就是执行了 4096 个 6×6×256 的卷积运算，从而得到每一个神经元的输出结果。

全连接层的参数数量通常较大，在神经网络模型中，通常将全连接层放置在卷积层之后，用于对学习得到的特征向量进行最终的整合，而不再将全连接层用于图像的其他分析。

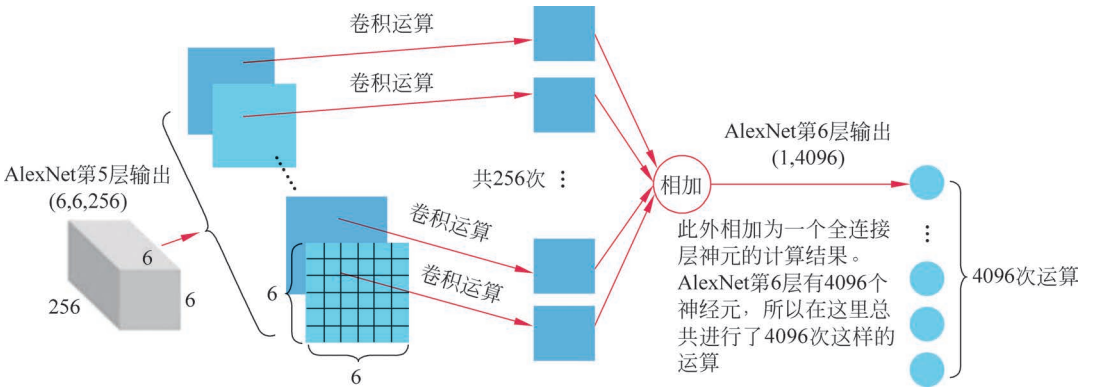


图 5-18　AlexNet 中的全连接层

5.5.4　激活层

前面介绍的每一种神经网络层所计算得到的输出都是输入的线性函数(Linear Function)，这会产生什么问题呢？我们知道，线性系统有迭加性质(Superposition Property)，而神经网络运算结果是层层相连的，上一层的输出会当作下一层的输入继续进行运算。倘若每一层运算都是一种线性函数，那么不管这些神经网络结构如何，最后的输出都只是输入的线性组合。也就是说，如果多层网络的运算都是线性的话，那么运算效果与只用一个全连接层网络相差无几。这样的线性分类器可以解决普通的直线二分类问题(图 5-19)。

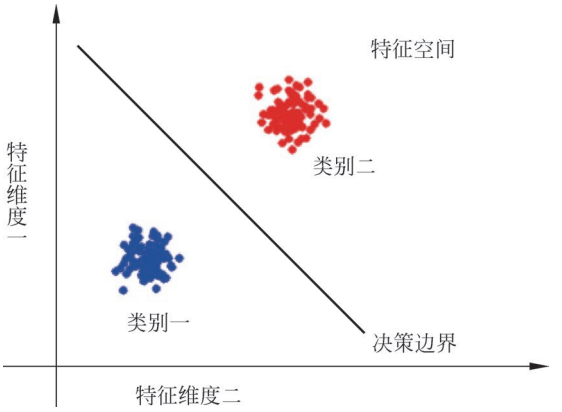


图 5-19　直线二分类问题

然而，现实生活中的分类问题复杂得多，往往不是线性分类可以解决的。为了解决这个问题，尝试在网络中加入非线性函数(Non-Linear Function)，即在卷积层、全连接层后加入激活层(Activation Layer)。

激活层的主要目的在于引进非线性因素到神经网络中，做法是在每次线性运算之后加上一个非线性函数运算。其中，ReLU 是目前最常被使用的非线性函数，其主要作用是将所有小于 0 的数值调整为 0，其余数值则维持不变，在计算上相对简单。实际上，ReLU 函数在神经网络模型中训练的效果也很不错，因此得以广泛应用。

运用在激活层中常见的非线性函数主要有以下 3 个。

(1) sigmoid(s 函数)：

$$s(x)=\frac{1}{1+e^{-x}}$$

sigmoid 函数如图 5-20(a)所示。

(2) tanh(双曲正切函数)：

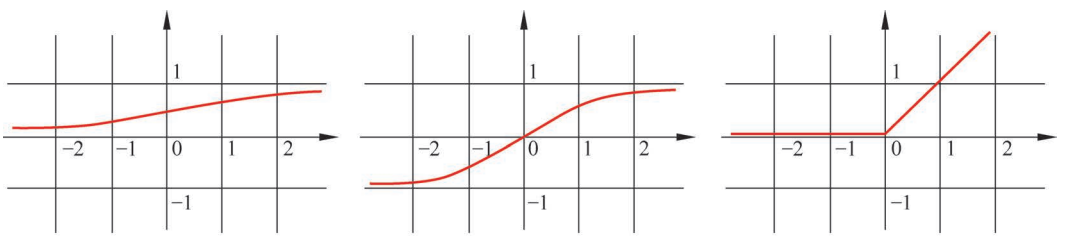
$$\tanh x=\frac{\sinh x}{\cosh x}=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

tanh 函数如图 5-20(b)所示。

(3) ReLU(线性整流函数)：

$$f(x)=\max(x,0)$$

ReLU 函数如图 5-20(c)所示。



(a) Sigmoid函数　(b) tanh函数　(c) ReLU函数

图 5-20　激活层中常见的非线性函数

这些非线性函数都会对训练效果产生一定的影响，应依据自己的神经网络模型选择合适的非线性函数。

5.5.5　标准化指数层

在一个功能为辨识对象的神经网络中，其输出层可以看作一个分类器。一般来说，输出层的神经元个数与要辨识的对象种类相对应。在输出层中，每个神经元的输出值可以作为一张图片辨识为该对象的概率，概率最高的对象会被视为预测结果。

为了让神经网络每一个神经元的输出都是一个合理的概率，即 0～1，一般使用标准化指数层(Softmax Layer)作为神经网络的输出层。标准化指数层的作用是利用一个标准化函数将先前的计算成果限定在我们希望的范围内，即 0～1。位于输出层的标准化指数层通常在前端与一个全连接层相接，全连接层将之前得到的特征向量加以运算，得到方便分类

的数值，然后加上标准化函数作为分类器，计算最终结果。

到此为止，本章已经介绍了在 AlexNet 中出现的所有神经网络层，该架构中一共运用了 5 层卷积层与 3 层全连接层。网络前端首先利用 5 层卷积层找出输入图片的特征，其中在第 1、2、5 个卷积层后接上了池化层，用来压缩提取到的特征，但会保留决定性的因素，以降低整个网络的运算量。然后，在网络后端接上 3 层全连接层及标准化函数，对提取到的特征进行调整；在每一个卷积层及部分全连接层后接上了激活层，以便每一层计算的结果得以保留。最后，将得到的特征输入分类器中，得到分类结果。

虽然随着计算机计算能力的进步，神经网络模型的结构逐渐变得又深又复杂，比 AlexNet 更快速、更准确的卷积神经网络结构相继出现，但不可否认的是，AlexNet 为后续的其他网络架构都奠定了基础，是深度学习发展的一个重要里程碑。

5.6　从图像到视频

这些年来，互联网视频（video）的数量日益增长，视频内容日渐丰富，视频技术的应用也日趋广泛。面对浩如烟海的视频资源，如何让计算机自动并准确地分析其内容，从而方便用户使用呢？视频理解（Video Understanding）为这一切的基础，因此其理所当然成为计算机视觉领域的热门方向。从光流特征到轨迹特征，从传统方法到深度学习，新方法的不断涌现推动着视频理解技术持续发展。如今，无论是在视频内容分析、视频监控领域，还是在人机交互、智能机器人等领域，视频理解技术都取得了令人振奋的成果。

人类眼中的视频本质上就是连续播放的图片。人们之所以能看到画面中的运动，是因为被人眼的视觉暂留机制“欺骗”了。一段足球射门视频实际上是由连续拍摄的数百张照片组成的序列，其中每张照片称为该视频的一帧（Frame）。这里选取其中 5 帧图像，如图 5-21 所示。当几百张图像以每秒 24 帧以上的速度播放时，在视觉暂留机制的作用下，原本静止的画面就可以运动起来。化静为动，一段极富张力的射门动作就这样呈现在了人们眼前。

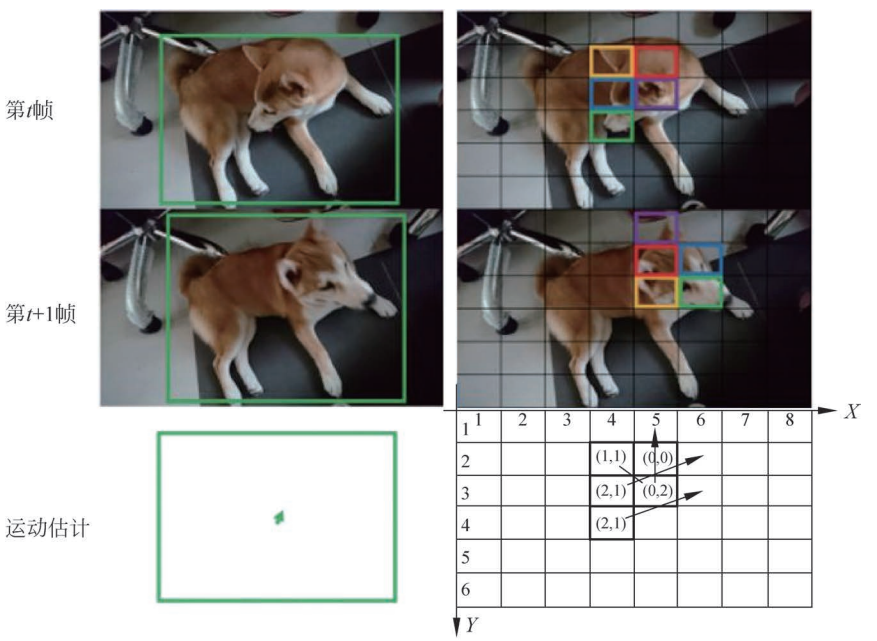
图 5-21　视频包含的 5 帧图像

我们在生活中处处都能接触到视频：电影院里放映的电影、电视上播放的节目、DVD 存储的影片，还有手机应用中可以在线播放的视频，等等。本章已经介绍了图像在计算机中的表示方式，那么视频又是如何在计算机上表示的呢？

在计算机中，视频就是按照时间顺序排列起来的图像。在播放时，只需要按照一定的速度依次将图像显示出来，就能呈现运动的视频画面。相比于图像，我们可以认为视频多了一个维度——时间维。因此，可以用一个函数 $I(x,y,t)$ 表示一个视频的信息，其中 t 是某一视频帧对应的时刻，(x,y) 是该视频帧中某个像素对应的位置（二维坐标）。这种表示方法将视频和图像紧密地联系起来，使用户可以运用图像领域的很多技术进行视频方面的研究。

运动估计（Motion Estimation）是提取视频中重要信息常使用的方法，除了用于视频压缩（Video Compression）之外，也能用于视频识别等任务。运动估计的目的在于估测视频中像素、区块或是对象随时间推移在空间中的位置变化。因此，运动估计的结果可以使用该像素、区块或是对象在两相邻时间取样点间的水平位移量与垂直位移量表示，这些成对的水平与垂直位移量称为动作向量（Motion Vector）。

如图 5-22(a)所示，把图像中的狗当作对象，选取对象范围（如图中的方框），对从第 t 帧到第 $t+1$ 帧狗的动作进行运动估计，即为对象层级的运动估计。如图 5-22(b)所示，将狗的图像分为不同区块，并分别标以不同颜色代表各区块的特征，如黄色代表右耳，紫色代表左耳，蓝色代表额头等。从第 t 帧到第 $t+1$ 帧，将各区块的水平位移和垂直位移记录下来，即为区块层级的运动估计。



(a) 对象层级　　(b) 区块层级

图 5-22　目标运动估计

在估测动作向量时，通常对视频中的移动物体有如下几项假设。

（1）物体与相邻两个帧间的位移量不会太大。

（2）物体不会随着时间改变颜色。

（3）物体形状不会随着时间改变。

如此一来，在估计时间点 t 某个物体的动作向量时，便可以将搜寻范围确定为时间点 $t+1$ 的图像中，以该物体于时间点 t 时的位置向周围延伸边长为 $2r$ 的正方形区域，并于此区域中搜寻与目标物体最相似的新位置。时间点 $t+1$ 时的新位置$(x_{\text{new}}, y_{\text{new}})$与时间点 t 时的原位置$(x_{\text{original}}, y_{\text{original}})$的坐标差$(x_{\text{new}}-x_{\text{original}}, y_{\text{new}}-y_{\text{original}})$即为动作向量的估测结果 v。

动作向量估测如图 5-23 所示，时间点 t 的帧（第 t 帧）与时间点 $t+1$ 的帧（第 $t+1$ 帧）被分割成 $m \times n$ 个 $d \times d$ 大小的区块。假设时间点 t 时，帧位置于(x, y)的区块为估测动作向量的目标区块（图 5-23 中的黄色方块）；在时间点 $t+1$ 时，帧以(x, y)为中心，周围的区块则为搜寻范围（图 5-23 中的绿色范围）。

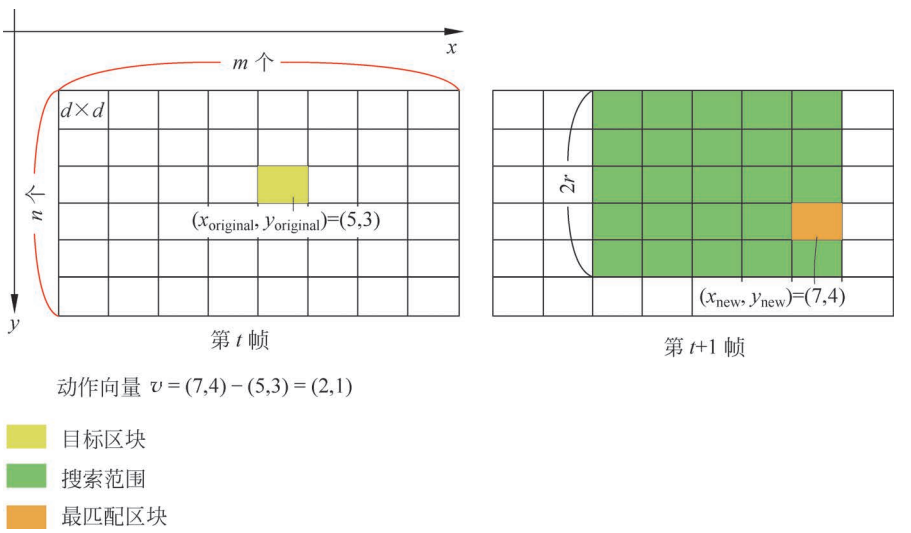


图 5-23　动作向量估测

首先，将目标区块的数值取出，假设 $d=2$，目标区块的数值即有 2×2 共 4 个。

然后，依次取出搜寻范围中每个区块的数值与目标区块的数值，计算绝对值误差总和，如图 5-24 所示。

最后，选取与目标区块绝对值误差总和最小的区块位置作为估计的新位置，新位置与原位置的坐标差即为动作向量估测结果，如图 5-25 所示。

对整张影像的所有区块进行上述运算后，即可获得动作向量图。假设区块大小为 8×8，那么在一部分辨率为 1920 像素 $\times$ 1080 像素的视频中，动作向量图的数据量为原始视频的 1/96，可以大幅减少后续处理所需的运算量。

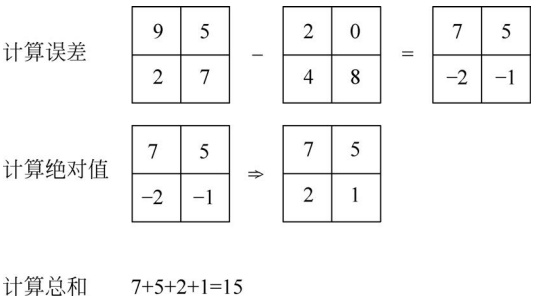


图 5-24 绝对值误差总和

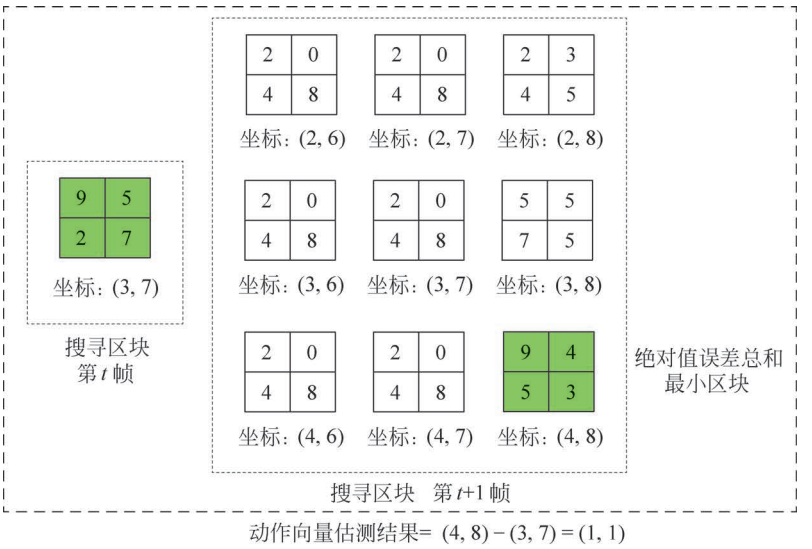


图 5-25 动作向量估测结果

5.7 应用实例

如果需要根据大量已归类的鲜花图片建立一个能识别鲜花的模型，给未归类的图片自动贴标签，这就是一个典型的分类问题，同时也是一个计算机视觉领域的问题。

5.7.1 数据收集和预处理

在 Kaggle 网站有一个公开的数据集(https://www.kaggle.com/alxmamaev/flowers-recognition)，里面收纳了各类花朵图片，可以将其作为本项目的数据集。由于该花朵图片的数据集较大，因此不必把它下载下来，可以直接在网站中创建 Notebook，以完成对花朵图片进行归类的工作。下面指定 4 个花朵目录，并通过 OpenCV(开源计算机视觉库)工具箱

读入图片的数据。OpenCV 是一个跨平台的开源计算机视觉方面的 API 库，这里应用其中的 imread 和 resize 功能读入并裁剪图片到 150×150 像素。

```
import numpy as np                                          #导入 Numpy
import pandas as pd                                         #导入 Pandas
import os                                                   #导入 OS
import cv2                                                  #导入 OpenCV 工具箱
print(os.listdir('../input/flowers - recognition/flowers'))          #输出目录结构
daisy_dir = '../input/flowers - recognition/flowers/daisy'           #雏菊目录
rose_dir = '../input/flowers - recognition/flowers/rose'             #玫瑰目录
sunflower_dir = '../input/flowers - recognition/flowers/sunflower'   #向日葵目录
tulip_dir = '../input/flowers - recognition/flowers/tulip'           #郁金香目录
X = []                                                      #初始化
y_label = []                                                #初始化
imgsize = 150                                               #图片大小
#定义一个函数,读入花朵图片
def training_data(label,data_dir):
    print ("正在读入:", data_dir)
    for img in os.listdir(data_dir):                        #目录
        path = os.path.join(data_dir,img)                   #目录 + 文件名
        img = cv2.imread(path,cv2.IMREAD_COLOR)             #读入图片
        img = cv2.resize(img,(imgsize,imgsize))             #设定图片像素维度
        X.append(np.array(img))                             #X 特征集
        y_label.append(str(label))                          #y 标签,即花朵的类别
#读入目录中的图片
training_data('daisy',daisy_dir)                            #读入雏菊
training_data('rose',rose_dir)                              #读入玫瑰
training_data('sunflower',sunflower_dir)                    #读入向日葵
training_data('tulip',tulip_dir)                            #读入郁金香
```

图片数据导入程序之后，随机用 imshow 功能显示几张花朵图片，以确认已经成功读入图片。

```
import matplotlib.pyplot as plt                             #导入 matplotlib
import random as rdm                                        #导入随机数工具
#随机显示几张花朵图片
fig,ax = plt.subplots(5,2)                                  #画布
fig.set_size_inches(15,15)                                  #大小
for i in range(5):
    for j in range (2):
        r = rdm.randint(0,len(X))                           #随机选择图片
        ax[i,j].imshow(X[r])                                #显示图片
        ax[i,j].set_title('Flower: ' + y_label[r])          #花朵的类别
plt.tight_layout()                                          #绘图
```

数据集随机输出结果如图 5-26 所示。

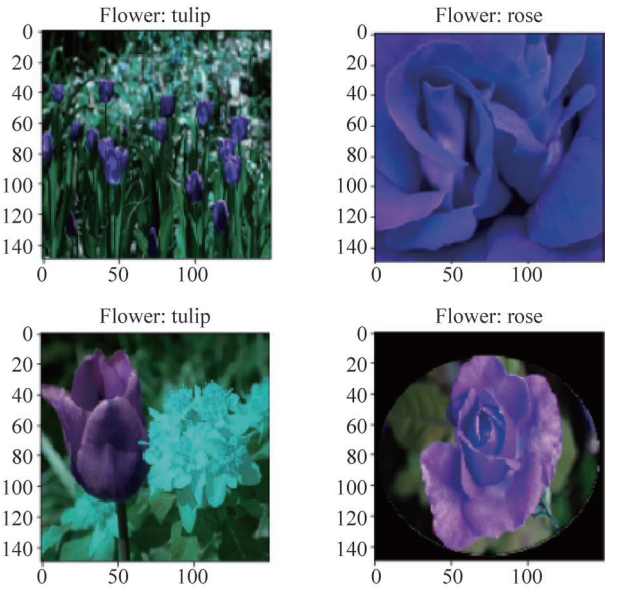


图 5-26 数据集随机输出结果

5.7.2 构建特征集和标签集

用 LabelEncoder 为标签 y 编码，并把特征集 X 转换为张量数组，代码如下：

```
from sklearn.preprocessing import LabelEncoder          # 导入标签编码工具
from tensorflow.keras.utils import to_categorical       # 导入 One-hot 编码工具
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y_label)                # 标签编码
y = to_categorical(y,4)                                 # 将标签转换为 One-hot 编码
X = np.array(X)                                         # 将 X 从列表转换为张量数组
```

输出如下：

```
(3265, 150, 150, 3)
```

该输出结果表示，在当前数据集中一共有 3265 张 150×150 像素的图片，且所有图片的颜色通道数为 RGB 3。

5.7.3 特征工程和数据集拆分

首先进行归一化处理，把 0～255 的 RGB 像素值压缩到 0～1。然后，对数据集进行拆分。

```
X = X/255                                                    #将 X 张量归一化
from sklearn.model_selection import train_test_split         #导入拆分工具
X_train, X_test, y_train, y_test = train_test_split(X, y,#拆分数据集
                                test_size = 0.2,random_state = 1)
```

到这里，就完成了数据集的所有准备工作，下面进入建立模型环节。

5.7.4　建立模型

这里用 Keras 建立卷积神经网络模型。因为 TensorFlow 和 Keras 完全集成在 Kaggle 的 Notebook 中，所以不需要 pip install。直接调用其中的 API，就能够搭建网络模型。下面这段不到 20 行的代码就搭建了一个能够为花朵图片分类的卷积神经网络：

```
from tensorflow.keras import layers                          #导入所有层 行 1
from tensorflow.keras import models                          #导入所有模型 行 2
cnn = models.Sequential()                                    #贯序模型 行 3
cnn.add(layers.Conv2D(32, (3, 3), activation = 'relu',       #输入卷积层 行 4
                      input_shape = (150, 150, 3)))
cnn.add(layers.MaxPooling2D((2, 2)))                         #最大池化层 行 5
cnn.add(layers.Conv2D(64, (3, 3), activation = 'relu'))      #卷积层 行 6
cnn.add(layers.MaxPooling2D((2, 2)))                         #最大池化层 行 7
cnn.add(layers.Conv2D(128, (3, 3), activation = 'relu'))     #卷积层 行 8
cnn.add(layers.MaxPooling2D((2, 2)))                         #最大池化层 行 9
cnn.add(layers.Conv2D(128, (3, 3), activation = 'relu'))     #卷积层 行 10
cnn.add(layers.MaxPooling2D((2, 2)))                         #最大池化层 行 11
cnn.add(layers.Flatten())                                    #展平层 行 12
cnn.add(layers.Dense(512, activation = 'relu'))              #全连接层 行 13
cnn.add(layers.Dense(4, activation = 'softmax'))             #分类输出层 行 14
cnn.compile(loss = 'categorical_crossentropy',               #损失函数 行 15
            optimizer = 'RMSprop',                           #优化器
            metrics = ['acc'])                               #评估指标
```

这段代码的结构并不复杂。神经网络中最主要的结构就是“层”，各种不同的层像拼积木一样组合起来，就形成了不同的神经网络。可以用下面的方法图形化显示整个 CNN 模型：

```
from IPython.display import SVG                              #实现神经网络结构的图形化显示
from tensorflow.keras.utils import model_to_dot              #导入 model_to_dot 工具
SVG(model_to_dot(cnn).create(prog = 'dot', format = 'svg'))#绘图
```

神经网络结构的图形化结果如图 5-27 所示。

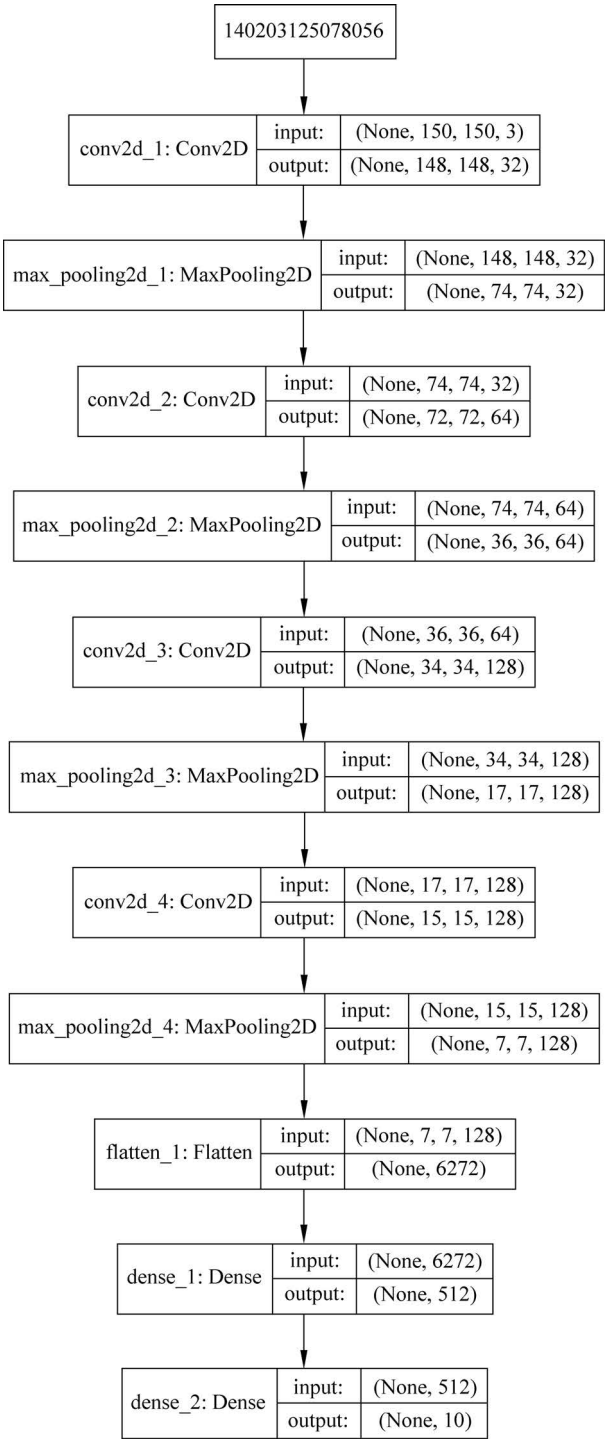


图 5-27　神经网络结构的图形化结果

模型构建好后，可以用 fit 语句进行训练。

```
#训练网络并把训练过程信息存入 history 对象
history = cnn.fit(X_train,y_train,              #训练数据
                  epochs = 10,                  #训练轮次(梯度下降)
                  validation_split = 0.2)       #训练的同时进行验证
```

在训练过程中指定了 validation_split，其可以在训练的同时自动把训练集部分拆出来进行验证，在每一个训练轮次中求出该轮次在训练集和验证集中的损失和预测准确率。训练输出如下：

```
Train on 2089 samples, validate on 523 samples
Epoch 1/5
2089/2089 [ ============================== ] - 86s 41ms/step - loss: 1.3523 -
acc: 0.3978 - val_loss: 1.0567 - val_acc: 0.5411
Epoch 2/5
2089/2089 [ ============================== ] - 85s 41ms/step - loss: 1.0167 -
acc: 0.5692 - val_loss: 1.0336 - val_acc: 0.5526
Epoch 3/5
2089/2089 [ ============================== ] - 85s 41ms/step - loss: 0.8912 -
acc: 0.6343 - val_loss: 0.9183 - val_acc: 0.6310
Epoch 4/5
2089/2089 [ ============================== ] - 84s 40ms/step - loss: 0.8295 -
acc: 0.6596 - val_loss: 0.9289 - val_acc: 0.6138
Epoch 5/5
2089/2089 [ ============================== ] - 85s 41ms/step - loss: 0.7228 -
acc: 0.7056 - val_loss: 1.0086 - val_acc: 0.5736
... ...
```

输出信息包括训练轮次（梯度下降次数）、每轮次训练时长、每轮次训练过程中的平均损失，以及分类的准确度。这里的轮次，就是神经网络对其中的一个神经元自动调参、通过梯度下降进行最优化的过程。

上述训练过程已经包含了验证环节。但是，为了更好地体现训练过程中的损失变化情况，这里对每轮次的损失和准确率进行可视化，绘制损失曲线，以展示模型在训练集中评估分数和损失的变化过程。

```
def show_history(history):                                  #显示训练过程中的学习曲线
    loss = history.history['loss']                          #训练损失
    val_loss = history.history['val_loss']                  #验证损失
    epochs = range(1, len(loss) + 1)                        #训练轮次
    plt.figure(figsize = (12,4))                            #图片大小
    plt.subplot(1, 2, 1)                                    #子图 1
    plt.plot(epochs, loss, 'bo', label = 'Training loss')   #训练损失
```

```
    plt.plot(epochs, val_loss, 'b', label = 'Validation loss')      # 验证损失
    plt.title('Training and validation loss')                       # 图题
    plt.xlabel('Epochs')                                            # X 轴文字
    plt.ylabel('Loss')                                              # Y 轴文字
    plt.legend()                                                    # 图例
    acc = history.history['acc']                                    # 训练准确率
    val_acc = history.history['val_acc']                            # 验证准确率
    plt.subplot(1, 2, 2)                                            # 子图 2
    plt.plot(epochs, acc, 'bo', label = 'Training acc')             # 训练准确率
    plt.plot(epochs, val_acc, 'b', label = 'Validation acc')        # 验证准确率
    plt.title('Training and validation accuracy')                   # 图题
    plt.xlabel('Epochs')                                            # X 轴文字
    plt.ylabel('Accuracy')                                          # Y 轴文字
    plt.legend()                                                    # 图例
    plt.show()                                                      # 绘图
show_history(history)                                               # 调用该函数
```

损失曲线如图 5-28 所示。

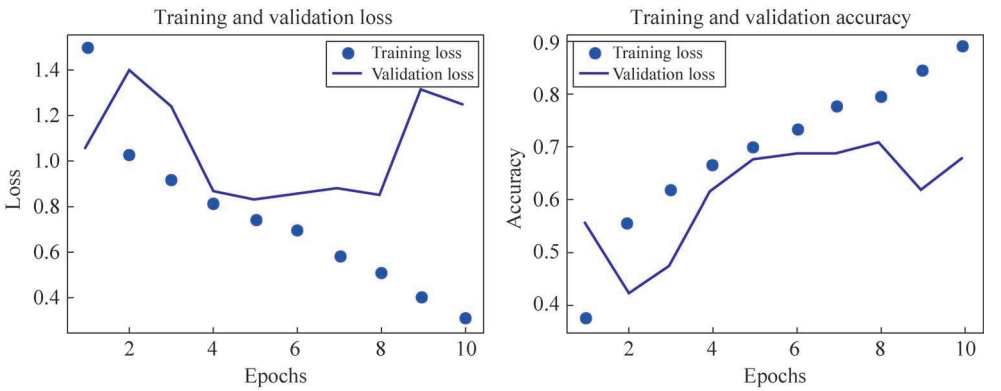


图 5-28 训练集和验证集的损失值和准确率

可以看到，训练集的损失呈现下降趋势，但是测试集上的损失则呈现跳跃现象，这说明该神经网络性能不是很稳定，可能有过拟合现象。可以在此基础上继续优化神经网络，让神经网络的损失值更低，准确率更高。

用该训练好的模型在测试集中进行分类结果的评分。

```
result = cnn.evaluate(X_test, y_test)          # 评估测试集中的准确率
print('CNN 的测试准确率为',"{0:.2f} % ".format(result[1]))
```

输出如下：

```
653/653 [ ============================== ] - 10s 15ms/step
CNN 的测试准确率为 0.69 %
```

输出显示，在 653 张测试集的图片中进行测试，模型的分类准确率达到了 0.69 以上。

可以应用模型的 predict 属性传入 X 特征集，进行花朵图片的分类。

```
prediction = cnn.predict(X_test)          # 预测测试集的图片分类
```

如下代码输出第一个图片（Python 的索引从 0 开始，所以下标 0 就是第一张图片）的分类预测值：

```
prediction[0]      # 第一张图片的分类
```

输出如下：

```
array([0.0030566 , 0.13018326, 0.00846946, 0.8582906 ], dtype = float32)
```

需要注意，此时输出的是分类概率。上述输出结果表示，第一类花 Daisy（雏菊）的概率为 0.03，第二类花 Rose（玫瑰）的概率为 0.13，第三类花 Sunflower（向日葵）的概率为 0.008，第四类花 Tulip（郁金香）的概率为 0.858。如下代码选出最大的概率，并把它当作 CNN 的分类结果：

```
print('第一张测试图片的分类结果为:', np.argmax(prediction[0]))
```

输出如下：

```
第一张测试图片的分类结果为: 3
```

输出结果显示第一张图片被 CNN 网络模型分类为第 4 种花（索引从 0 开始，所以类别 3 就是第 4 种花），即 Tulip（郁金香）。到此神经网络完成了分类功能，准确率达到 69%。

本章小结

本章介绍了图像识别的几个主要概念，说明了图像与视频的差异，并讨论了深度学习如何将空间数据推广到时空数据进行处理。首先是图片的预处理，介绍了空间滤波器的功能及特质、卷积运算方法；接着，在深度学习架构中探讨了深度神经网络，以及卷积层、池化层、全连接层、激活层、标准化指数层的作用。本章列举了一些当前最常见的图像识别应

用，希望能更好地诠释图片识别。相信在看完本章后，读者对计算机视觉会有一个初步的了解和认识。

习题

1. 简述颜色的三要素(three elements of color)。
2. 什么是计算机视觉？
3. 卷积核的作用是什么？
4. 简述卷积核的层数对模型泛化能力的影响。