

第三章 二值图像分析

一幅数字图像是一个二维阵列，阵列元素值称为灰度值或强度值。实际上，图像在量化成数字图像前是一个连续强度函数的集合，场景信息就包含在这些强度值中。图像强度通常被量化成 256 个不同灰度级，对某些应用来说，也常有 32、64、128 或 512 个灰度级的情况，在医疗领域里甚至使用高达 4096(12bits) 个灰度级。很明显，灰度级越高，图像质量越好，但所需的内存也越大。

在机器视觉研究的早期，由于内存和计算能力非常有限，而且十分昂贵，因此视觉研究人员把精力主要集中在研究输入图像仅包含两个灰度值的二值视觉系统上。人们注意到，人类视觉在理解仅由两个灰度级组成的线条、轮廓影像或其它图像时没有任何困难，而且应用场合很多，这一点对研究二值视觉系统的研究人员是一个极大的鼓舞。

随着计算机计算能力的不断增强和计算成本的不断下降，人们普遍开始研究基于灰度图像、彩色图像和深度图像的视觉系统。尽管如此，二值视觉系统还是十分有用的，其原因如下：(1) 计算二值图像特性的算法非常简单，容易理解和实现，并且计算速度很快。(2) 二值视觉所需的内存小，对计算设备要求低。工作在 256 个灰度级的视觉系统所需内存是工作在相同大小二值图像视觉系统所需内存的八倍。如若利用游程长度编码等技术（见 3.4 节）还可使所需内存进一步减少。由于二值图像中的许多运算是逻辑运算而不是算术运算，所以所需的处理时间很短。(3) 许多二值视觉系统技术也可以用于灰度图像视觉系统上。在灰度或彩色图像中，表示一个目标或物体的一种简易方法就是使用物体模板(mask)，物体模板就是一幅二值图像，其中 1 表示目标上的点，0 表示其它点。在物体从背景中分离出来后，为了进行决策，还要求取物体的几何和拓扑特性，这些特性可以从它的二值图像计算出来。因此，尽管我们是在二值图像上讨论这些方法，但它们的应用并不限于二值图像。

一般来说，当物体轮廓足以用来识别物体且周围环境可以适当地控制时，二值视觉系统是非常有用的。当使用特殊的照明技术和背景并且场景中只有少数物体时，物体可以很容易地从背景中分离出来，并可得到较好的轮廓，比如，许多工业场合都属于这种情况。二值视觉系统的输入一般是灰度图像，通常使用阈值法首先将图像变成二值图像，以便把物体从背景中分离出来，其中的阈值取决于照明条件和物体的反射特性。二值图像可用来计算特定任务中物体的几何和拓扑特性，在许多应用中，这种特性对识别物体来说是足够的。二值视觉系统已经在光学字符识别、染色体分析和工业零件的识别中得到了广泛应用。

在下面的讨论中，假定二值图像大小为 $m \times n$ ，其中物体像素值为 1，背景像素值为 0。

3.1 阈值

视觉系统中的一个重要问题是从图像中识别代表物体的区域（或子图像），这种对人来说是件非常容易的事，对计算机来说却是令人吃惊的困难。为了将物体区域同图像其它区域分离出来，需要首先对图像进行分割。把图像划分成区域的过程称为分割，即把图像 $F[i, j]$ 划分成区域 p_1, p_2, \dots, p_k ，使得每一个区域对应一个候选的物体。下面给出分割的严格定义。

定义 分割是把像素聚合成区域的过程，使得：

- $\bigcup_{i=1}^k P_i = \text{整幅图像}$ ($\{P_i\}$ 是一个完备分割)。

- $P_i \cap P_j = \emptyset, i \neq j$, ($\{P_i\}$ 是一个完备分割).
- 每个区域 P_i 满足一个谓词, 即区域内的所有点有某种共同的性质.
- 不同区域的图像, 不满足这一谓词.

正如上面所表明的, 分割满足一个谓词, 这一谓词可能是简单的, 如分割灰度图像时用的均匀灰度分布、相同纹理等谓词, 但在大多数应用场合, 谓词十分复杂. 在图像理解过程中, 分割是一个非常重要的步骤.

二值图像可以通过适当地分割灰度图像得到. 如果物体的灰度值落在某一区间内, 并且背景的灰度值在这一区间之外, 则可以通过阈值运算得到物体的二值图像, 即把区间内的点置成 1, 区间外的点置成 0. 对于二值视觉, 分割和阈值化是同义的. 阈值化可以通过软件来实现, 也可以通过硬件直接完成.

通过阈值运算是否可以有效地进行图像分割, 取决于物体和背景之间是否有足够的对比度. 设一幅灰度图像 $F[i, j]$ 中物体的灰度分布在区间 $[T_1, T_2]$ 内, 经过阈值运算后的图像为二值图像 $F_T[i, j]$, 即:

$$F_T[i, j] = \begin{cases} 1 & \text{如果 } T_1 \leq F[i, j] \leq T_2 \\ 0 & \text{其它} \end{cases} \quad (3.1)$$

如果物体灰度值分布在几个不相邻区间内时, 阈值化方案可表示为:

$$F_T[i, j] = \begin{cases} 1 & \text{如果 } F[i, j] \in Z \\ 0 & \text{其它} \end{cases} \quad (3.2)$$

其中 Z 是组成物体各部分灰度值的集合. 图 3.1 是对一幅灰度图像使用不同阈值得到的二值图像输出结果.

阈值算法与应用领域密切相关. 事实上, 某一阈值运算常常是为某一应用专门设计的, 在其它应用领域可能无法工作. 阈值选择常常是基于在某一应用领域获取的先验知识, 因此在某些场合下, 前几轮运算通常采用交互式方式来分析图像, 以便确定合适的阈值. 但是, 在机器视觉系统中, 由于视觉系统的自主性能 (autonomy) 要求, 必须进行自动阈值选择. 现在已经研究出许多利用图像灰度分布和有关的物体知识来自动选择适当阈值的技术. 其中的一些方法将在 3.2 节介绍.

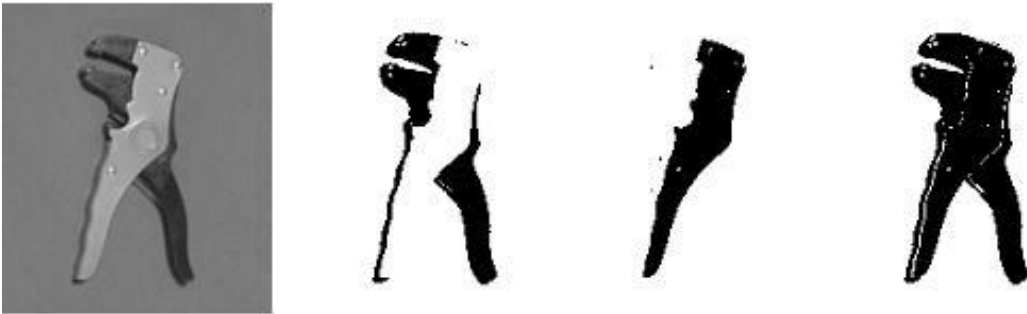


图 3.1 一幅灰度图像和使用不同阈值得到的二值图像结果. 上左: 原始灰度图像, 上右: 阈值 $T=100$; 左下: $T=128$. 右下: $T_1=100|T_2=128$.

3.2 几何特性

通过阈值化方法从图像中检测出物体后, 下一步就要对物体进行识别和定位. 在大多数工业应用中, 摄像机的位置和环境是已知的, 因此通过简单的几何知识就可以从物体的二维图像确定出物体的三维位置. 在大多数应用中, 物体的数量不是很多, 如果物体的尺寸和形状完全不同, 则可以利用尺度和形状特征来识别这些物体. 实际上在许多工业应用中, 经常使用区域

的一些简单特征，如大小、位置和方向，来确定物体的位置并识别它们。

3. 2. 1 尺寸和位置

一幅二值图像区域的面积（或零阶矩）由下式给出：

$$A = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B[i, j] \quad (3. 3)$$

在许多应用中，物体的位置起着十分重要的作用。工业应用中，物体通常出现在已知表面（如工作台面）上，而且摄像机相对台面的位置也是已知的。在这种情况下，图像中的物体位置决定了它的空间位置。确定物体位置的方法有许多，比如用物体的外接矩形、物体矩心（区域中心）等来表示物体的位置。区域中心是通过图像进行“全局”运算得到的一个点，因此它对图像中的噪声相对来说是不敏感的。对于二值图像，物体的中心位置与物体的质心相同，因此可以使用下式求物体的中心位置：

$$\begin{aligned} \bar{x} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B[i, j] &= \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} j B[i, j] \\ \bar{y} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B[i, j] &= - \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} i B[i, j] \end{aligned} \quad (3. 4)$$

其中 \bar{x} 和 \bar{y} 是区域相对于左上角图像的中心坐标。物体的位置为：

$$\begin{aligned} \bar{x} &= \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} j B[i, j]}{A} \\ \bar{y} &= - \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} i B[i, j]}{A} \end{aligned} \quad (3. 5)$$

这些是一阶矩。注意，由于约定 y 轴向上，因此方程 3. 4 和 3. 5 的第二个式子的等号右边加上了负号。

3. 2. 2 方向

计算物体的方向比计算它的位置稍微复杂一点。某些形状（如圆）的方向不是唯一的，为了定义唯一的方向，一般假定物体是长形的，其长轴方向被定义为物体的方向。通常，二维平面上与最小惯量轴同方向的最小二阶矩轴被定为长轴。

图像中物体的二阶矩轴是这样一条线，物体上的全部点到该线的距离平方和最小。给出一幅二值图像 $B[i, j]$ ，计算物体点到直线的最小二乘方拟合，使所有物体点到直线的距离平方和最小：

$$\chi^2 = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} r_{ij}^2 B[i, j] \quad (3. 6)$$

其中 r_{ij} 是物体点 $[i, j]$ 到直线的距离。为了避免直线处于近似垂直时所出现的数值病态问题，人们一般把直线表示成极坐标形式：

$$\rho = x \cos \theta + y \sin \theta \quad (3.7)$$

如图 3.2 所示, θ 是直线的法线与 x 轴的夹角, ρ 是直线到原点的距离. 把点 (i, j) 坐标代入直线的极坐标方程得出距离 r :

$$r^2 = (x \cos \theta + y \sin \theta - \rho)^2 \quad (3.8)$$

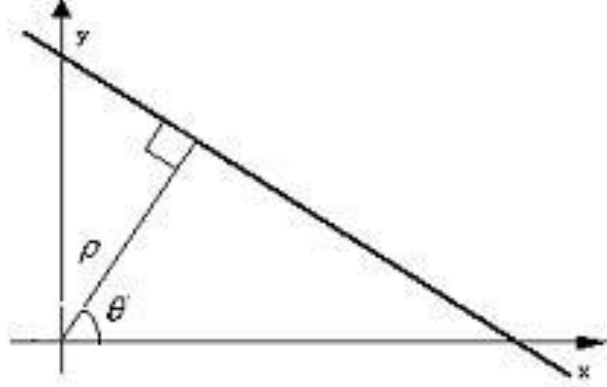


图 3.2 直线的极坐标表示

将方程 3.8 代入方程 3.6 并求极小化问题, 可以确定参数 ρ 和 θ :

$$\chi^2 = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (x_{ij} \cos \theta + y_{ij} \sin \theta - \rho)^2 B[i, j] \quad (3.9)$$

令 χ^2 对 ρ 的导数等于零求解 ρ 得:

$$\rho = (\bar{x} \cos \theta + \bar{y} \sin \theta) \quad (3.10)$$

它说明回归直线通过物体中心 (\bar{x}, \bar{y}) . 用这一 ρ 值代入上面的 χ^2 , 则极小化问题变为:

$$\chi^2 = a \cos^2 \theta + b \sin \theta \cos \theta + c \sin^2 \theta \quad (3.11)$$

其中的参数:

$$\begin{aligned} a &= \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (x_{ij} - \bar{x})^2 B[i, j] \\ b &= 2 \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (x_{ij} - \bar{x})(y_{ij} - \bar{y}) B[i, j] \\ c &= \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (y_{ij} - \bar{y})^2 B[i, j] \end{aligned} \quad (3.12)$$

是二阶矩. 表达式 χ^2 可重写为:

$$\chi^2 = \frac{1}{2}(a+c) + \frac{1}{2}(a-c) \cos 2\theta + \frac{1}{2}b \sin 2\theta \quad (3.13)$$

对 χ^2 微分, 并置微分结果为零, 求解 θ 值:

$$\tan 2\theta = \frac{b}{a-c} \quad (3.14)$$

因此, 惯性轴的方向由下式给出:

$$\begin{aligned}\sin 2\theta &= \pm \frac{b}{\sqrt{b^2 + (a-c)^2}} \\ \cos 2\theta &= \pm \frac{a-c}{\sqrt{b^2 + (a-c)^2}}\end{aligned}\quad (3.15)$$

所以由 χ^2 的最小值可以确定方向轴。注意，如果 $b=0, a=c$ ，那么物体就不会只有唯一的方向轴。物体的伸长率 E 是 χ^2 的最大值与最小值之比：

$$E = \frac{\chi_{\max}}{\chi_{\min}} \quad (3.16)$$

3. 2. 3 密集度和体态比

区域的密集度 (compact) 可用下面的式子来度量：

$$C = \frac{A}{p^2} \quad (3.17)$$

其中， p 和 A 分别为图形的周长和面积。根据这一衡量标准，圆是最密集的图形，其密集密度为最大值 $1/4\pi$ ，其它一些图形的比值要小一些。让我们来看一下圆，当圆后仰时，形状成了一椭圆，面积减小了而周长却不象面积减小的那么快，因此密集度降低了。在后仰到极限角时，椭圆被压缩成了一条无限长直线，椭圆的周长为无穷大，故密集度变成了零。对于数字图像， A/p^2 是指物体尺寸（像素点数量）除以边界长度的平方。这是一种很好的散布性或密集性度量方法。这一比值在许多应用中被用作为区域的一个特征。

密集度的另一层意义是：在给定周长的条件下，密集度越高，围成的面积就越大。注意在等周长的情况下，正方形密集度大于长方形密集度。

体态比定义为区域的最小外接矩形的长与宽之比，正方形和圆的体态比等于 1，细长形物体的体态比大于 1。图 3.3 所示的是几种形状的外接矩形。

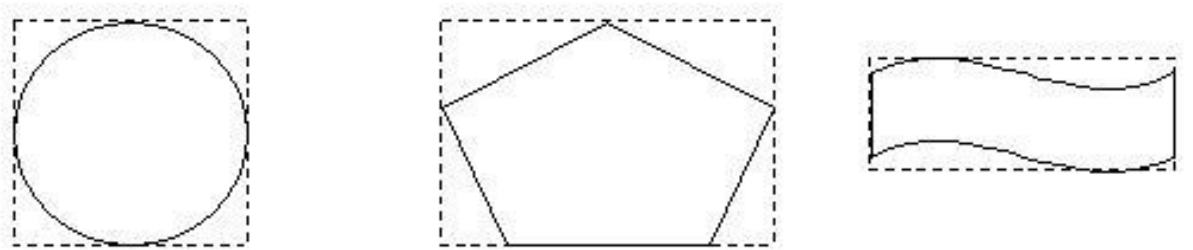


图 3.3 几种外接矩形示意图

3. 3 投影

给定一条直线，用垂直该直线的一簇等间距直线将一幅二值图像分割成若干条，每一条内像素值为 1 的像素个数为该条二值图像在给定直线上的投影 (projection)。当给定直线为水平或垂直直线时，计算二值图像每一列或每一行上像素值为 1 的像素数量，就得到了二值图像的水平或垂直投影，如图 3. 4 所示。由于投影包含了图像的许多信息，所以投影是二值图像的一种

简洁表示方式，显然，投影不是唯一的，同样的投影可能对应不同的图像。

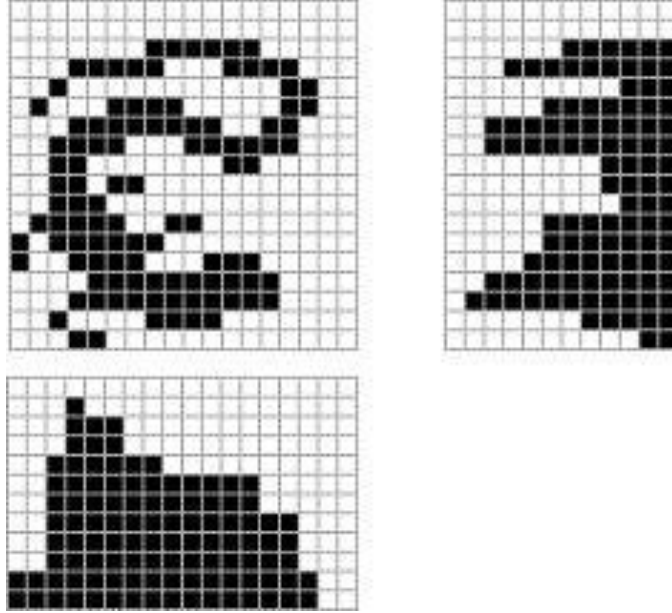


图 3.4 一幅二值图像及其水平投影图

在某些应用中，投影可以作为物体识别的一个特征。投影既是一种简洁的图像表示，又可以实现快速算法。下面介绍对角线投影的求解方法。对角线投影的关键是计算当前行和列对应的投影分布图位置标号。设行和列的标号分别用 i 和 j 表示。若图像矩阵为 n 行 m 列，则 i 和 j 的范围分别为 0 到 $n-1$ 和 0 到 $m-1$ 。假设对角线的标号 d 用行和列的仿射变换（线性组合加上常数）计算，即：

$$d = ai + bj + c \quad (3.18)$$

对角线投影共对应 $n+m-1$ 个条，其中仿射变换把右上角像素映射成对角线投影的第一个位置，把左下角像素映射成最后一个位置，如图 3.5 所示，则当前行列对应的标号 d 的公式为：

$$d = i - j + m - 1 \quad (3.19)$$

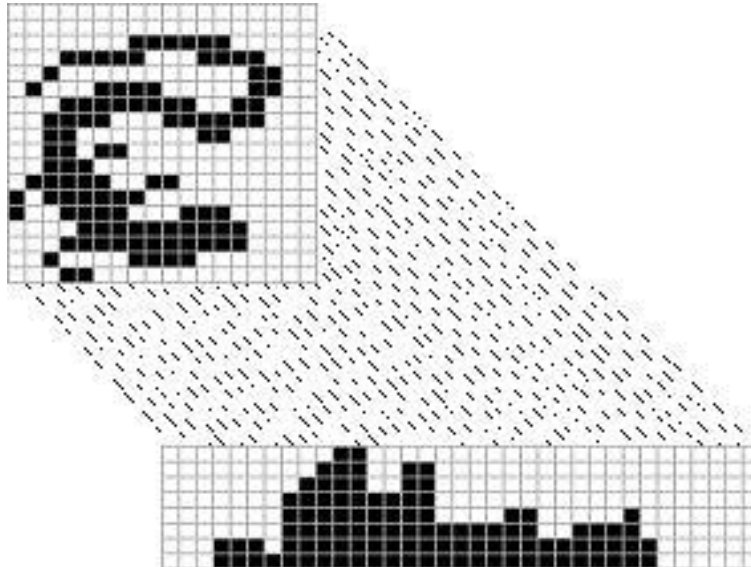


图 3.5 二值图像及其对角线上的投影图

3. 4 游程长度编码

游程长度编码(run-length encoding)是另一种二值图像的简洁表示方法，它是用图像像素值连续为 1 的个数（像素 1 的长度）来描述图像。这种编码已被用于图像传输。另外，图像的某些性质，如物体区域面积，也可以从游程长度编码直接计算出来。

在游程长度编码中经常运用两种方法，一种是使用 1 的起始位置和 1 的游程长度，另一种是仅仅使用游程长度，但须从 1 的游程长度开始描述，如图 3.6 所示。

0	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	1	0	1
0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1

1 的游程 (2,2) (6,3) (13,6) (20,1)

(4,6) (11,10)

(1,5) (11, 1) (17, 4)

1 和 0 的游程长度: 0, 2, 2, 3, 4, 6, 1, 1

0, 3, 6, 1, 10

5, 5, 1, 5, 4

图 3. 6 一幅简单二值图像的游程长度编码。

如果用第二种方法来表示图像每行的游程长度，并用 $r_{i,k}$ 代表图像第 i 行的第 k 个游程长度，则全部 1 的游程长度之和就是所求物体的面积。

$$A = \sum_{i=0}^{n-1} \sum_{k=0}^{\left\lfloor \frac{m_i-1}{2} \right\rfloor} r_{i,2k+1} \quad (3. 20)$$

其中 m_i 是第 i 行游程个数， $(m_i - 1)/2$ 取整，表示 1 的游程个数。

由游程长度编码能很容易地计算水平投影而无需变成原来的图像。使用更巧妙的方法也能从游程长度编码计算出垂直和对角线投影。

3. 5 二值图像算法

从背景中分离出物体是一个困难的问题，在此将不讨论这个问题。这里假设物体可以从背景中分离，并且使用某一谓词，可以对图像中属于物体的点进行标记。因此，问题就变为如何将一幅图像中所有被标记的点组合成物体图像。这里还假设物体点在空间上是非常接近的。利用空间接近概念可以严格定义，利用此定义研究的算法可以把空间上非常接近的点聚合在一起，构成图像的一个成分 (component)。下面首先引进一些定义，然后讨论有关算法。

3. 5. 1 定义

(1) 近邻

在数字图像中，一个像素在空间上可能非常接近其它一些像素。在用方格表示的数字图像中，一个像素与其它四个像素有公共边界，并与另外四个像素共享顶角。如果两个像素有公共边界，则把它们称为 4-近邻(4-neighbors)。同样，如果两个像素至少共享一个顶角，则称它们为 8-近邻。例如，位于 $[i, j]$ 的像素有四个 4-近邻： $[i-1, j]$ ， $[i+1, j]$ ， $[i, j-1]$ ， $[i, j+1]$ 。它的 8-近邻包括这四个 4-近邻，再加上 $[i-1, j-1]$ ， $[i+1, j-1]$ ， $[i-1, j+1]$ ， $[i+1, j+1]$ 。一个像素被认为与它的 4-近邻是 4-连通(4-connected)关系，与它的 8-近邻是 8-连通关系(如图 3. 7)。

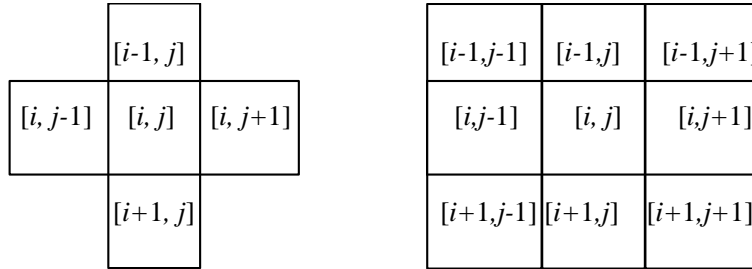


图 3. 7 矩形像素网格的 4-近邻和 8-近邻示意图。像素 $[i, j]$ 位于图的中心。

(2) 路径

从像素 $[i_0, j_0]$ 到像素 $[i_n, j_n]$ 的路径(path)是指一个像素序列 $[i_0, j_0]$ ， $[i_1, j_1]$ ， \dots ， $[i_n, j_n]$ ，其中像素 $[i_k, j_k]$ 是像素 $[i_{k+1}, j_{k+1}]$ 的近邻像素， $0 \leq k \leq n-1$ 。如果近邻关系是 4-连通的，则路径是 4-路径；如果是 8-连通的，则称为 8-路径。图 3. 8 即为路径的两个简单例子。

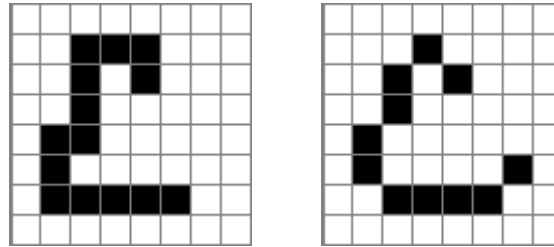


图 3. 8 4-路径和 8-路径示意图

(3) 前景

图像中值为 1 的全部像素的集合称为前景(foreground)，用 S 表示。

(4) 连通性

已知像素 $p, q \in S$ ，如果存在一条从 p 到 q 的路径，且路径上的全部像素都包含在 S 中，则称 p 与 q 是连通的。

注意，连通性(connectivity)是等价关系。对属于 S 的任意三个像素 p 、 q 和 r ，有下列性质：

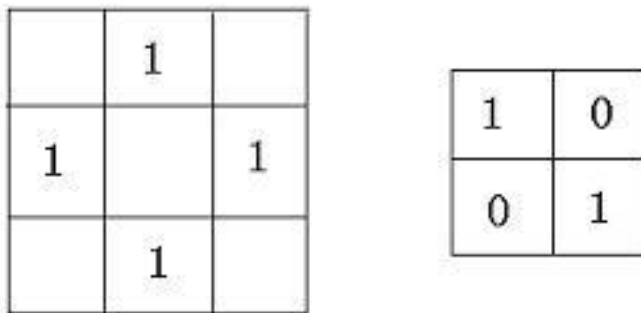
1. 像素 p 与 p 本身连通（自反性）。
2. 如果 p 与 q 连通，则 q 与 p 连通（互换性）。
3. 如果 p 与 q 连通且 q 与 r 连通，则 p 与 r 连通（传递性）。

(5) 连通成份

一个像素集合，如果集合内的每一个像素与集合内其它像素连通，则称该集合为一个连通成份(connected component)。

(6) 背景

\bar{S} (S 的补集) 中包含图像边界点的所有连通成份的集合称为背景(background). \bar{S} 中所有其它元称为洞. 考虑下面的两个图像.



首先看左图中有几个洞和几个物体. 如果从前景和背景来考虑 4-连通, 有四个大小为一个像素的物体和一个洞. 如果考虑 8-连通, 那么有一个物体而没有洞. 直观地, 在这两种情况下出现了不确定性情况. 右图为另一个类似的不确定问题. 其中如果 1 是连通的, 那么 0 就应该是不连通的.

为了避免这种难以处理的情况, 对物体和背景应使用不同的连通. 如果我们对 S 使用 8-连通, 那么对 \bar{S} 就应使用 4-连通.

(7) 边界

S 的边界(boundary)是 S 中与 \bar{S} 中有 4-连通关系的像素集合. 边界通常记为 S' .

(8) 内部

内部(interior)是 S 中不属于它的边界的像素集合. S 的内部等于 $S - S'$.

(9) 包围

如果从 S 中任意一点到图像边界的 4-路径必须与区域 T 相交, 则区域 T 包围(surrounds)区域 S (或 S 在 T 内). 图 3. 9 即为一幅简单二值图像和它的边界、内部、包围示意图.

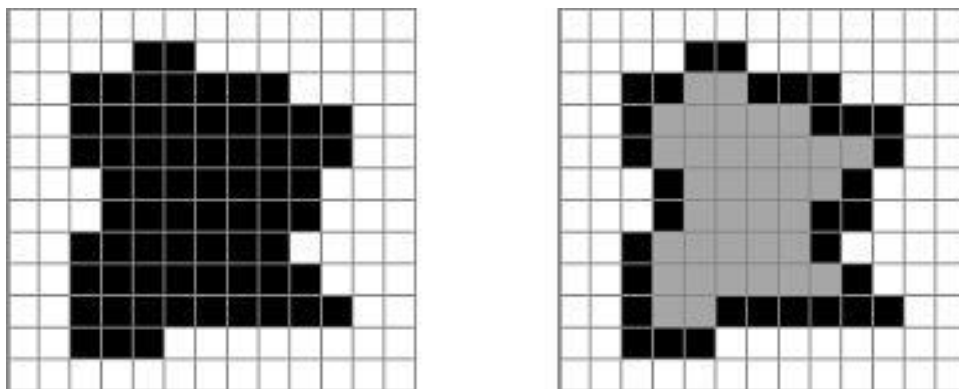





图 3. 9 一幅二值图像与它的边界  , 内部  和包围 

3. 5. 2 连通成份标记

在一幅图像中找出连通成份是机器视觉中最常见的运算之一. 连通区域内的点构成表示物体的候选区域. 机器视觉中的大多数物体都有表面, 显然, 物体表面点投影到图像平面上会形

成空间上密集的点集。这里应该指出，连通成份算法常常会在二值视觉系统中形成瓶颈效应，原因是连通成份运算是一个全局性的运算，这种算法在本质上是序贯的。如果图像中仅有一个物体，那么找连通成份就没有必要；如果图像中有许多物体，且要求出物体的特性与位置，则必须确定连通成份。

连通标记算法可以找到图像中的所有连通成份，并对同一连通成份中的所有点分配同一标记。图 3.10 表示的是一幅图像和已标记的连通成份。在很多应用中，要求在标记连通成份的同时算出连通成份的特征，如尺寸、位置、方向和外接矩形。下面介绍两种连通成份标记算法：递归算法和序贯算法[Jain 1995]。

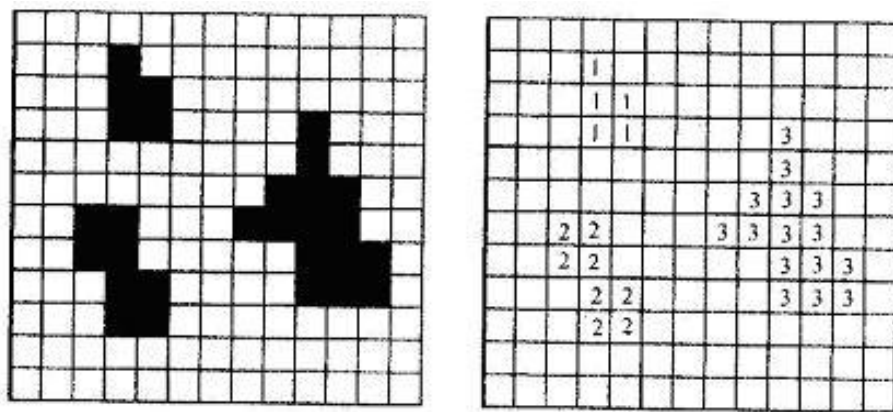


图 3.10 一副图像及其连通成分图像

(1) 递归算法

递归算法在串行处理器上的计算效率是很低的，因此，这一算法主要用于并行机上。

算法 3.1 连通成份递归算法

1. 扫描图像，找到没有标记的 1 点，给它分配一个新的标记 L。
3. 递归分配标记 L 给 1 点的邻点。
3. 如果不存在没标记的点，则停止。
4. 返回第一步。

(2) 序贯算法

序贯算法通常要求对图像进行二次处理。由于这一算法一次仅运算图像的两行，因此当图像以文件形式存贮且空间不允许把整幅图像载入内存时也能使用这一算法。这一算法(见算法 3.2)可以查看某一点的邻点，并且可以给像素值为 1 的邻点分配一个已经使用过的标记。如果图像的邻点有两种不同的标记，则用一个等价表(equivalent table)来记录所有的等价标记。在第二次处理过程中，使用这一等价表来给某一连通成份中所有像素点分配唯一的标记。

本算法在从左到右、从上到下扫描图像时，算法仅能查询到某一像素点的 4-近邻中的两个近邻点，即上点与左点。设算法已经查到了该像素的这两个近邻点，此时出现三种情况：(1) 如果这两个近邻点中没有一点为 1，则该像素点需要一个新的标记。(2) 如果这两个近邻点中只有一点为 1，且分配了标记 L，那么该像素点的标记也为 L。(3) 如果这两个邻点都为 1，且已分配了标记 L，则该像素点的标记还是 L；但是当近邻点被分配了不同标记 M 与 N，则这两个标记被用于了同一组元，应该把它们合并。在这种情况下，应把其中的一个标记（一般选用最小的那个标记）分配给该像素点，并在等价表中登记为等价标记。

等价表包含了给每一连通成份分配唯一标记的信息。在第一次扫描中，所有属于同一连通成份的标记被视为是等价的。在第二次扫描中，从一个等价集(equivalent set)中选择一个标记并分配给连通成份中所有像素点。通常将最小的标记分配给一个连通成份。第二次扫描将给每一连通成份分配唯一的标记。

在找到所有的连通成份后，应该统计等价表，以便删除其中的空格；然后将等价表作为查

找表对图像重新进行扫描，以便重新统计图像中的标记。

计算每一连通成份的面积、一阶矩、二阶矩是序贯连通成份算法的一个部分。当然，必须使用分离变量来累加每一区域的矩信息。当区域合并后，每一区域的矩累计值也应加到一起。

算法 3. 2 4-连通序贯连通成份算法

1. 从左至右、从上到下扫描图像。
2. 如果像素点为 1，则：
 - (a) 如果上面点和左面点有一个标记，则复制这一标记。
 - (b) 如果两点有相同的标记，复制这一标记。
 - (c) 如果两点有不同的标记，则复制上点的标记且将两个标记输入等价表中作为等价标记。
 - (d) 否则给这一个像素点分配一新的标记并将这一标记输入等价表。
3. 如果需考虑更多的点，则回到第二步。
4. 在等价表的每一等价集中找到最低的标记。
5. 扫描图像，用等价表中的最低标记取代每一标记。

3. 5. 3 欧拉数

在许多应用中，亏格数(genus)或欧拉数可作为识别物体的特征。亏格数定义为连通成份数减去空洞数，

$$E = C - H \quad (3. 21)$$

其中， E ， C 和 H 分别是欧拉数、连通成份数与空洞数。这个式子给出了一个简单的拓扑特征，这种拓扑特征具有平稳、旋转和比例不变特性。图 3. 11 给出了一些例子及其对应的欧拉数。

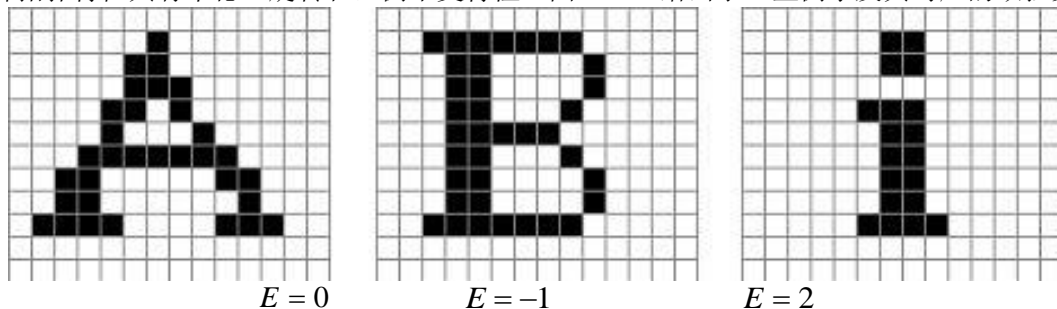


图 3. 11 字母“A”、“B”、“i”及它们的欧拉数。注意前景用了 8-连通，而背景用了 4-连通。

3. 5. 4 区域边界

连通成份 S 的边界是那些属于 S 且与 \bar{S} 邻接的点集。使用简单的局部运算就可找到边界点。在大多数应用中，我们都想用一特定的顺序跟踪边界点。一般的算法是按顺时针方向跟踪区域的所有点。此处讨论一个简单的边界跟踪算法。

假定物体边界不在图像的边界上（即物体完全在图像内部），边界跟踪算法先选择

一起始点 $s \in S$ ，然后跟踪边界直到回到起始点。这种算法概括在算法 3.3 中。这种算法对尺寸大于 1 个像素的所有区域都是有效的。用这种算法求区域 8-邻点的边界如图 3.12(a) 所示。为了得到平滑的图像边界，可以在检测和跟踪图像边界后，利用边界点的方向信息来平滑边界。显然，图像边界噪声越大，图像边界点变化越剧烈，图像边界相邻点的方向变化数（与差分链码有一点区别，链码见第七章）也越大。根据这一特点，设置一个边界点方向变化数阈值，把方向变化数大于这一阈值的图像边界点滤除，由此可得到平滑的图像边界。图 3.12 (b) 所示的是一个经过平滑过的区域边界示意图，其中的方向变化数阈值为 1。注意，由于采用 8-邻点边界跟踪，因此方向变化数的最大值为 4。如果阈值设成 4，则对原始边界没有平滑。边界跟踪和平滑常常结合在一起使用，见计算机作业 3.5。

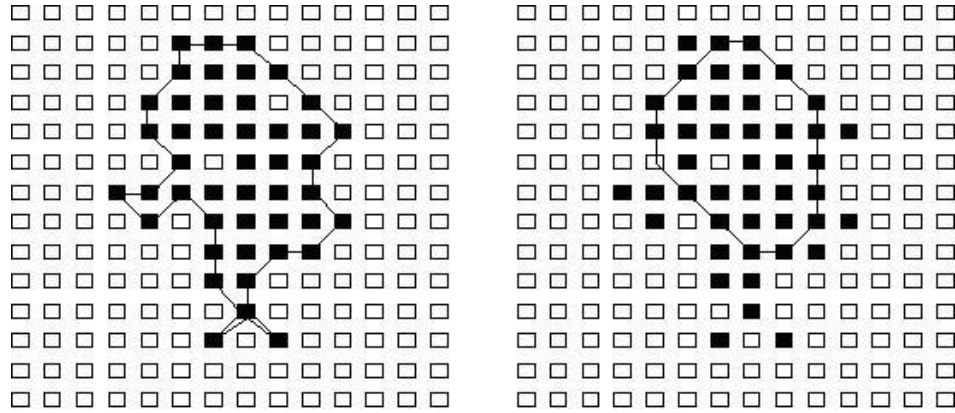


图 3.12 边界跟踪算法结果，(a) 图像边界跟踪结果；(b) 边界跟踪与平滑结果。

算法 3.3 边界跟踪算法

- ① 从左到右、从上到下扫描图像，求区域 S 的起始点 $s(k) = (x(k), y(k)), k = 0$ 。
- ② 用 c 表示当前边界上被跟踪的像素点。置 $c = s(k)$ ，记 c 左 4-邻点为 b ， $b \in \bar{S}$ 。
- ③ 按逆时针方向从 b 开始将 c 的 8 个 8-邻点分别记为 n_1, n_2, \dots, n_8 ， $k = k + 1$ ，
- ④ 从 b 开始，沿逆时针方向找到第一个 $n_i \in S$ ，
- ⑤ 置 $c = s(k) = n_i$ ， $b = n_{i-1}$ ，
- ⑥ 重复步骤③、④、⑤，直到 $s(k) = s(0)$ 。

3.5.5 距离测量

在许多应用中，找到一幅图像中两个像素点或两个连通成份之间的距离是很有必要的。目前还没有定义数字图像距离的唯一方法，但对所有的像素点 p 、 q 和 r ，任何距离度量都必须满足下列性质：

1. $d(p, q) \geq 0$ ，当且仅当 $p = q$ 时， $d(p, q) = 0$
2. $d(p, q) = d(q, p)$
3. $d(p, r) \leq d(p, q) + d(q, r)$

下面是一些常用的距离函数

欧几里德距离：

$$d_{\text{Euclidean}}([i_1, j_1], [i_2, j_2]) = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2} \quad (3.22)$$

街区距离:

$$d_{\text{Block}} = |i_1 - i_2| + |j_1 - j_2|. \quad (3.23)$$

棋盘距离:

$$d_{\text{Chess}} = \max(|i_1 - i_2|, |j_1 - j_2|). \quad (3.24)$$

3.5.6 中轴

如果对 S 中像素 $[i, j]$ 的所有邻点 $[u, v]$ 有下式成立:

$$d([i, j], \bar{S}) \geq d([u, v], \bar{S}) \quad (3.25)$$

则 S 中像素 $[i, j]$ 到 \bar{S} 的距离 $d([i, j], \bar{S})$ 是局部最大值. S 中所有到 \bar{S} 的距离是局部最大值的像素点集合称为对称轴或中轴, 通常记为 S^* . 使用 $[u, v]$ 4-近邻的中轴变换的一些例子见图 3.13. 图 3.13b 表明少量噪声会使中轴变换结果产生显著的差异.

由 S^* 和 S^* 中每一点到 \bar{S} 的距离能重构原始像素集 S . S^* 是 S 的简洁表示. S^* 可用来表示一个区域的形状. 通过去除 S^* 中与 \bar{S} 距离较小的像素点, 可以生成一个简化的 S^* 集.

中轴可作为物体的一种简洁表示. 但是, 二值图像中的区域也可用其边界来表示. 边界跟踪算法可用来获得表示边界的序列点. 在第七章还将讨论用链码来简洁地表示边界的方法. 对任意物体, 边界将是区域的简洁表示. 但要明确给定像素点是否在某一区域内, 中轴则是更好的表示, 因为使用中轴上的像素点和每一个给定像素点的最大距离圆盘(中轴距离变换), 可以很容易地检测出给定像素是否在中轴定义的区域中.

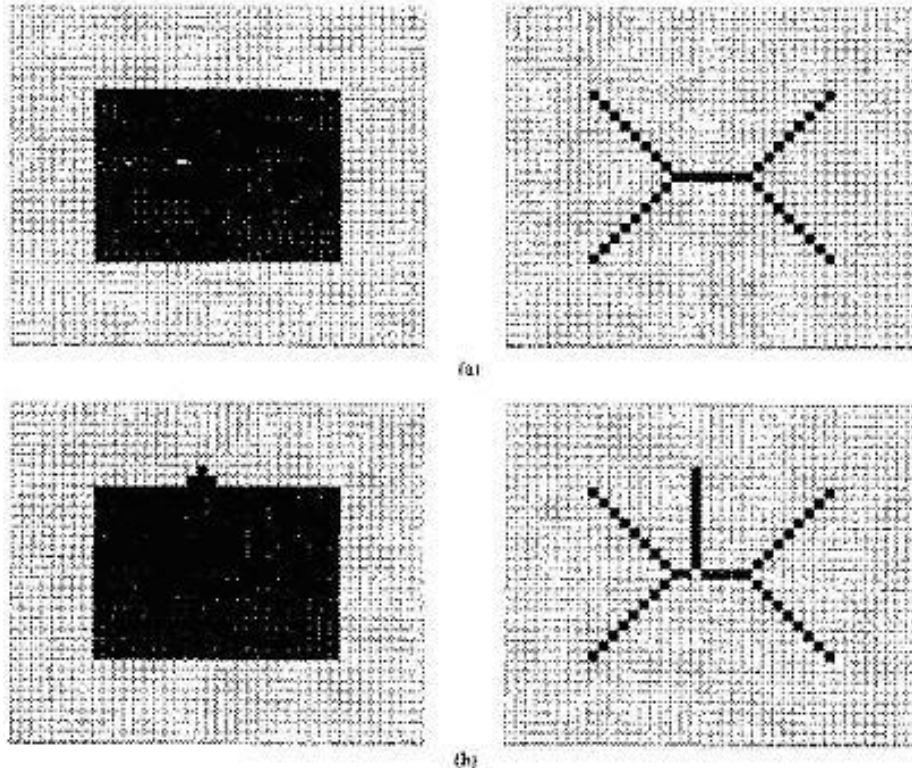


图 3.13 中轴变换举例

3. 5. 7 细化

细化(thinning)是一种图像处理运算,可以把二值图像区域缩成线条,以逼近区域的中心线,也称之为骨架或核线.细化的目的是减少图像成份,直到只留下区域的最基本信息,以便进一步分析和识别.虽然细化可以用在包含任何区域形状的二值图像,但它主要对细长形(而不是凸圆形或水滴状)区域有效.细化一般用于文本分析预处理阶段,以便将文本图像中线条图画或字符笔画表示成单像素线条.细化要求如下:

- (1) 连通图像区域必须细化成连通线结构.
- (2) 细化结果最少应该是 8-连通 (下面将要解释).
- (3) 保留近似终止线的位置.
- (4) 细化结果应该近似于中轴线.
- (5) 由细化引起的附加突刺(短分支)应该是最小的.

细化结果应该保证第一条要求中所定义的连通性,这一点是最基本的要求,它保证了连通线结构的数量等于原始图像中连通区域的数量.第二条要求保证所得到的线条总是含有 8-连通图像的最小数量.第三条要求说明终止线位置应该保持不变.细化可以通过迭代方式不断去除边界点来实现,重要的是在迭代过程中不要去端点像素,因为这样不仅会缩短细化线,丢掉结构信息,而且不能保持其位置不变.第四条要求说明所得线段应能最好地逼近原始区域的中线,如两个像素点宽的竖线或水平线的真正中线应该位于这两个像素之间半个像素间距的位置.在数字图像中表示半个像素间距是不可能的,因此得到的结果是一条位于原直线一侧的直线.第五条要求没有明确指出噪声的影响控制到最低程度,因为判断噪声本身是一件很难的事.一般不希望原始区域含有会引起突刺的隆起,但当某些较大隆起是区域特征时,却必须识别它们.应该指出,某些细化算法有去除突刺的参数,不过最好将细化和去除噪声分开进行,这是由于某些情况下不需要的突刺,可能是另一些情况下所需要的短线.因此,最好的办法是先进行细化,然后单独去除长度低于某一特定最小值的任何突刺.

一种常用的细化手段是在至少 3×3 邻域内检查图像的每一点,剥去区域边界.一次剥去一层图像,直至区域被细化成一条线.这一过程是用迭代法实现的,如算法 3.4.在每次迭代时,每一个像素点用 $n \times n$ 窗函数检查,为了保持连通性或线末端位置,将单像素厚的边界擦除.在图 3.14 中将会看到,在每次迭代中,值为 1 的外层区域就是用这种方式削掉的.当迭代结果没有变化时,迭代过程结束,图像得到细化

算法 3.4 4-近邻细化迭代算法

- 对于每一个像素,如果
 - (1) 没有上近邻(下近邻\左近邻\右近邻)
 - (2) 不是孤立点或终止线
 - (3) 去除该像素点不会断开区域则去除该像素点.
- 重复这一步骤直到没有像素点可以去除.

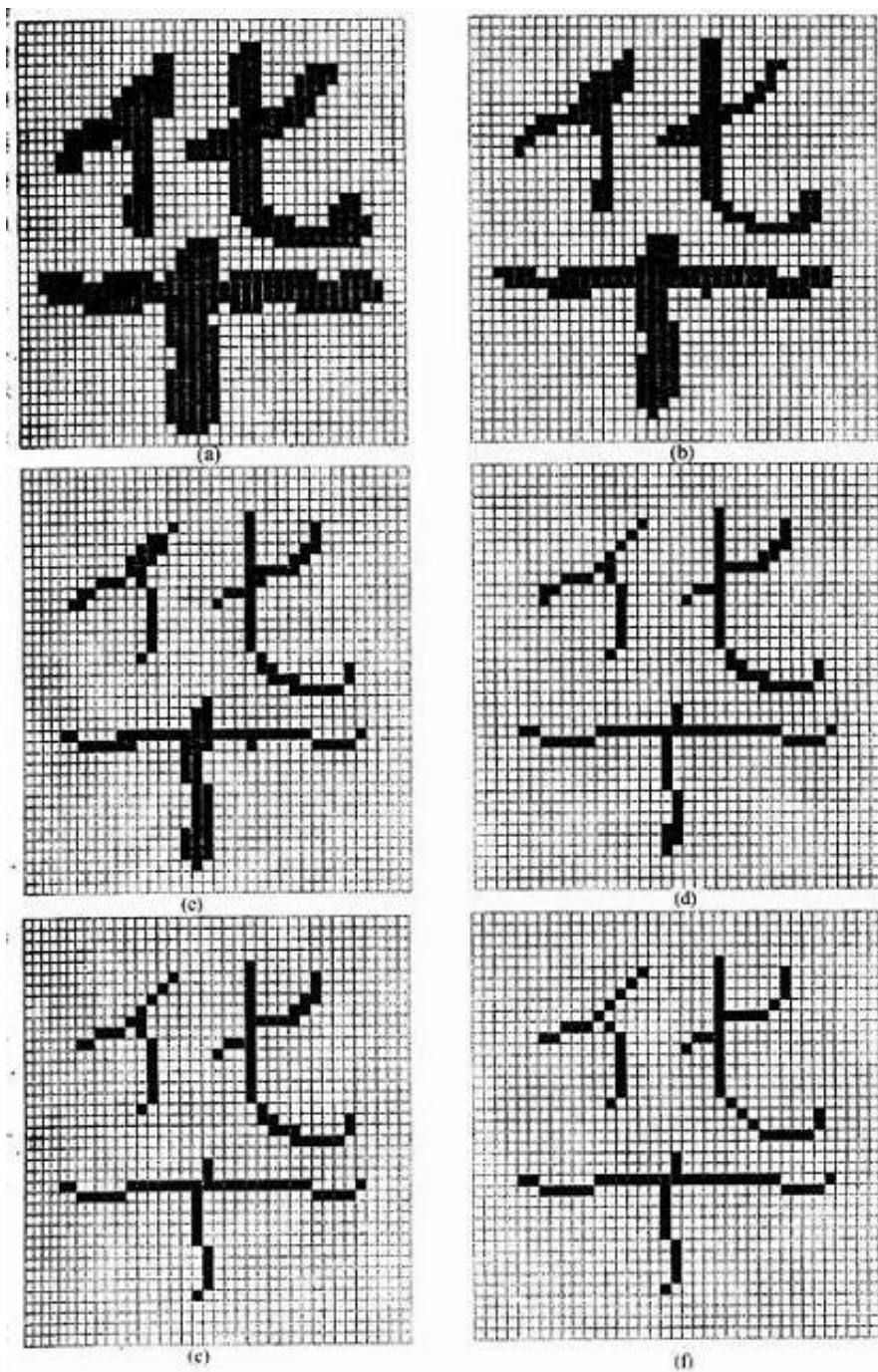


图 3. 14 细化手写体“华”的迭代过程. (a) 原图像, (b)—(f)为五次迭代过程, 每次迭代削去一层边界.

3. 5. 8 扩展与收缩

图像中的一个连通成份可以进行全方位的扩展(expanding)或收缩(shrinking). 如果某一连通

成份可以变化，使得一些背景像素点变成 1，这一运算就称为扩展。如果物体像素点全方位地消减或变为 0 时，则称为收缩。一种简单的扩展与收缩实现方法如下：

扩展：如果近邻点是 1，则将该点从 0 变为 1。

收缩：如果近邻点是 0，则将该点从 1 变为 0。

这样，收缩可以看作是扩展背景。这类运算的例子见图 3. 15。

需要指出，扩展与收缩这样简单的运算可以完成非常有用而又貌似很复杂的运算。下面引进符号

$S^{(k)}$ ： S 扩展 k 倍。

$S^{(-k)}$ ： S 收缩 k 倍。

其中下列性质必须满足：

$$(S^m)^{-n} \neq (S^{-n})^m \neq S^{(m-n)}$$

$$S \subset (S^k)^{-k}$$

$$S \supset (S^{-k})^k$$

先扩展后收缩算法能补上不希望存在的洞，如图 3.15 (b) (d) 所示；先收缩后扩展算法则能去除孤立的噪声点，见图 3. 15 (c) (e)。请注意，扩展与收缩可用来确定孤立组元或簇。注意，扩展后收缩有效地填满了空洞却没有去除噪声；相反，收缩后扩展能去除噪声却没有填满空洞。在地形图像处理 and 膨胀与腐蚀运算中，扩展与收缩算法的一般形式被广泛地用于许多任务中。

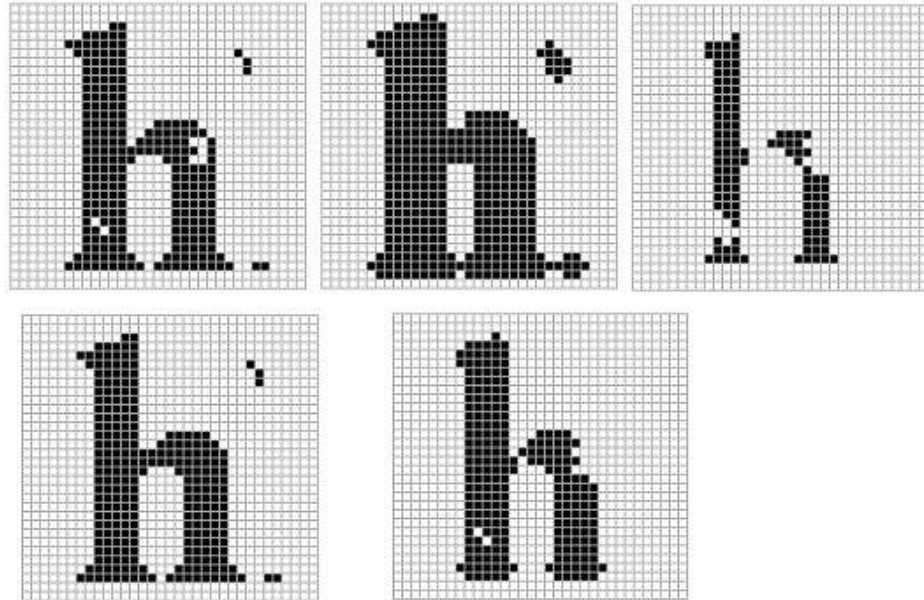


图 3. 15 对字母“h”收缩与扩展算法实验结果。(a) 原始噪声图像；(b) 扩展运算；(c) 收缩运算；(d) 扩展后收缩运算；(e) 收缩后扩展运算。

3. 6 形态算子

数学形态 (morphology) 这一名称是从形状研究得来的。这种方法也说明了一个事实，即在许多机器视觉算法设计中，根据形状来思考问题是最自然，也是最容易的。形态方法有助于进行基于形状或图形思考。形态方法中图像信息的基本单元是二值像素。

任意两个二值图像 A 和 B 的交是一个二值图像，记为 $A \cap B$ ，即 A 与 B 中皆为 1 的图像点

p 的集合:

$$A \cap B = \{p \mid p \in A, p \in B\} \quad (3.26)$$

A 和 B 的并, 记为 $A \cup B$, 是一个二值图像, 是 A 或 B 或两者中为 1 的所有图像点 p 的集合, 用符号表示为:

$$A \cup B = \{p \mid p \in A \text{ or } p \in B\} \quad (3.27)$$

设 Ω 是全值二值图像 (所有的像素值皆为 1), A 是二值图像. A 的补集是 A 中 1 与 0 互相交换后的二值图像, 即:

$$\overline{A} = \{p \mid p \in \Omega, p \notin A\} \quad (3.28)$$

标号为 $[i, j]$ 与 $[k, l]$ 的两像素点 p 和 q , 其向量和是标号为 $[i+k, j+l]$ 的像素点 $p+q$. 向量差是标号为 $[i-k, j-l]$ 的像素点 $p-q$.

若 A 是二值图像, p 是二值图像 B 中的一个像素点, 则 A 被 p 平移后的二值图像由下式表示:

$$A_p = \{a+p \mid a \in A\} \quad (3.29)$$

即二值图像 A 被一个像素点 p 平移是指将 A 的原点移到 p

(1) 膨胀

已知二值图像 A , 如果 $A_{b_1}, A_{b_2}, \dots, A_{b_n}$ 是由二值图像 $B = \{b_1, b_2, \dots, b_n\}$ 中像素值为 1 的点平移得到的, 则 A 由 B 平移的并称为 A 被 B 膨胀, 即:

$$A \oplus B = \bigcup_{b_i} A_{b_i} \quad (3.30)$$

膨胀具有结合性、交换性. 这样, 在进行膨胀的步骤序列中, 完成运算的顺序就不重要了. 这就允许我们将一个复杂的形状拆成几个简单的形状, 然后重新组合成为膨胀序列.

(2) 腐蚀

腐蚀是膨胀的相反过程. 二值图像 A 经二值图像 B 腐蚀后在 p 点仍为 1 的充分必要条件是: B 平移到 p 后, B 中的 1 像素也是 A 中的 1 像素. A 被 B 腐蚀可用下式表示:

$$A \ominus B = \{p \mid B_p \subseteq A\} \quad (3.31)$$

二值图像 B 常常是规则图像, 是作用于图像中的一种探针, 也称为结构元. 腐蚀在许多应用中起着十分重要的作用. 结构元对一幅图像进行腐蚀会生成一幅包含结构元所有位置的图像.

图 3.16 到 3.17 是一个倒 T 形结构元对一个简单物体进行膨胀与腐蚀运算的示意图. 用结构元进行膨胀或腐蚀运算也可以描述为: 结构元的原点像素经过待膨胀的二值图像中所有 1 像素点时, 对应结构元所有 1 像素的待膨胀二值图像像素置为 1 像素; 在腐蚀运算过程中, 结构元的原点像素经过待腐蚀的二值图像中所有 1 像素点时, 如果结构元中有一个 1 像素没有对应待腐蚀二值图像的 1 像素, 则对应结构元原点的待腐蚀二值图像 1 像素置为 0.

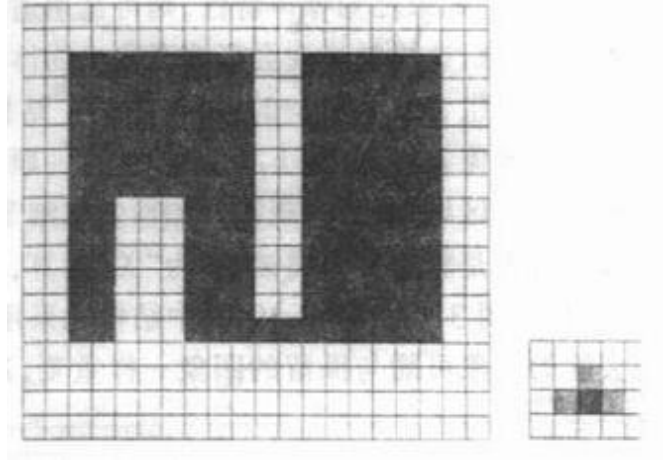


图 3. 16 原始测试图像 A (左)与结构元 B (右). 注意结构元 B 的原点比 B 中的其它像素点要黑一些.

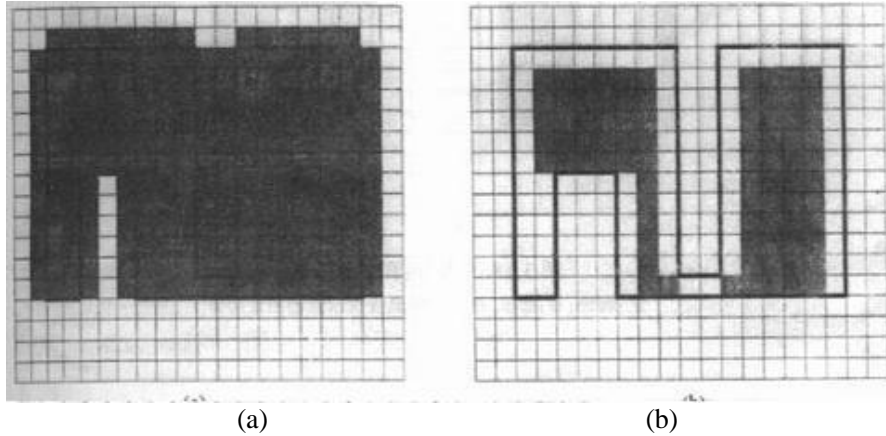


图 3. 17 膨胀与腐蚀实验结果. (a) A 被 B 膨胀, 其中原始像素 A 的边界用粗黑线表示. (b) A 被 B 腐蚀, 其中原始像素 A 的边界用粗黑线表示.

膨胀和腐蚀展示了几何的而不是逻辑的对偶特性, 这种特性也包含了几何互补性与逻辑互补性. 二值图像的几何互补称为它的反射. 二值图像 B 的反射 B' 是与 B 关于原点对称的二值图像, 即:

$$B' = \{-p \mid p \in B\} \quad (3.32)$$

膨胀与腐蚀的对偶性由下面关系式表示:

$$\overline{A \oplus B} = \overline{A} \ominus B' \quad (3.33)$$

$$\overline{A \ominus B} = \overline{A} \oplus B' \quad (3.34)$$

与几何对偶性相比, 逻辑对偶性的关系式为:

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \quad (3.35)$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B} \quad (3.36)$$

上式也称为 DeMorgan 定律.

腐蚀与膨胀常常用于图像滤波. 如果已知噪声特性, 则可选用适当的结构元和一系列腐蚀与膨胀运算来去除噪声. 请注意, 这样的滤波会影响图像中物体的形状.

数学形态中的基本运算可以组合成很复杂的运算. 用同一结构元(探针)腐蚀后再膨胀可去除比结构元小的所有区域像素点, 而留下其余部分, 这一顺序称为“开”运算. 如, 用一圆形

的探针将所有比探针小的区域删除，实现抑制加性空域细节的滤波。

与上述处理顺序相反的过程是膨胀后再腐蚀，称为“关”运算，这种顺序会填满比探针小的孔洞和凹状区。这些运算见图 3. 18 和图 3. 17，其中结构元采用了相同的 T 形结构。这里仍然存在一个问题是去除的图像可能与保留的图像一样重要。这样的滤波器可用来抑制空域特征或区分基于尺寸的物体类型。所用的结构元不一定是简洁的或规则的，可以是任意模式的像素点集合。这样可以探测到具有一定分布的图像特征。

关于数学形态的详细内容参见[Dougherty 1988, Haralick 1992]，关于数学形态在字符识别中的应用见[Mitchell 1989]。

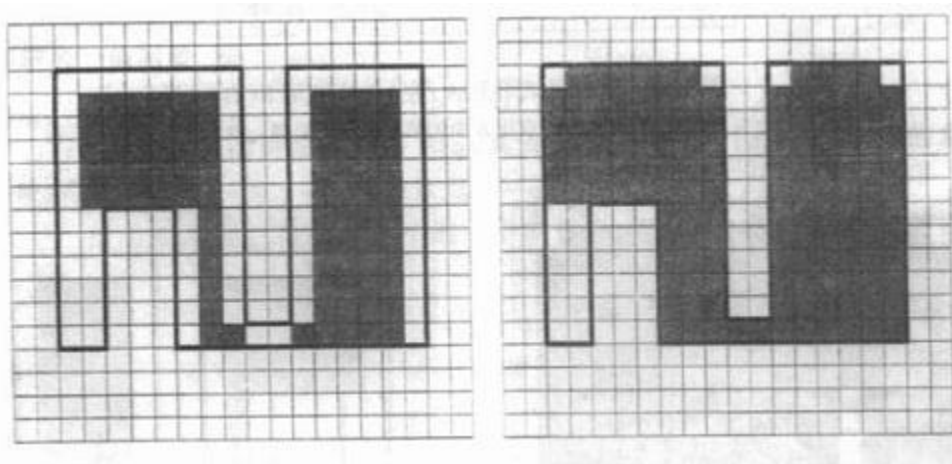


图 3. 18 “开”运算。左：腐蚀；右：膨胀。图中的粗黑线表示原始图像边界。

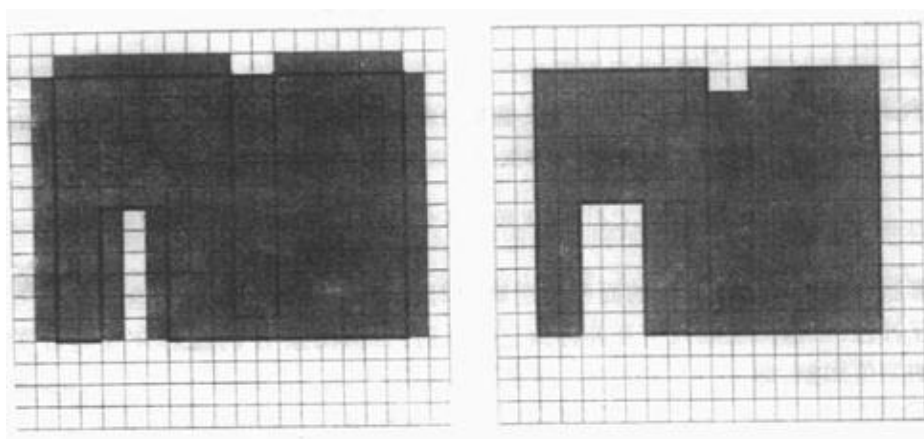


图 3. 19 “关”运算。左：膨胀；右：腐蚀。图中的粗黑线表示原始图像边界。

思考题

3. 1 考虑一幅二进制图像如图 3.20 所示。

- (a) 分别使用 4-连通和 8-连通，求黑像素集合中包含有多少个区域。
- (b) 分别使用 4-连通和 8-连通，求白像素集合中包含有多少个区域。

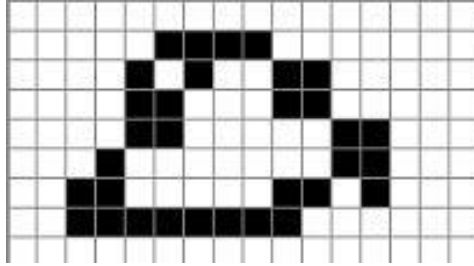


图 3.20 题 3.1 图

3. 2 假定图 3.21 的白色像素是边缘点. 请分别使用 4-连通和 8-连通细化算法去除多余的白点 (把去除的白点用阴影线表示).

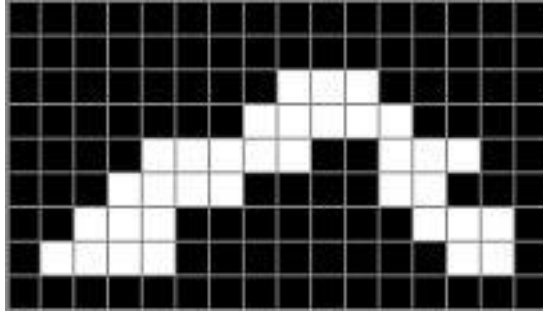


图 3.21 题 3.2 图

3. 3 连通域标记算法是许多应用中的“瓶颈”问题. 连通域标记可以看作是视觉系统中连接高层和低层算法的桥梁. 请你给出一种连通标记快速算法设计的构思? 请问能否开发出一种并行算法, 为什么?

计算机练习题

3. 1 开发一个算法来计算区域面积和游程代码的第一、二阶矩.
3. 2 开发一个中轴算法. 将这一算法作用于具有不同规则物体的二值图像上, 并研究这种技术在表示物体形状时的优点与缺点.
3. 3 构造一种算法来实现膨胀与收缩运算. 使用这些算法来实现二值图像中不同噪声的滤波.
3. 4 设计一个机器视觉系统, 该系统可以获取场景的二值图像并识别物体. 考虑一些常用物体, 如, 硬币、钢笔、笔记本和其它的办公用品. 根据你在本章中学到的物体特征来建立物体识别策略, 完成所有的运算并测试你的系统.