第5章 图像特征提取

本章学习目的与要求

本章学习基本的图像特征提取方法，了解并掌握图像变换、图像边缘检测、图像分割等常用的图像处理方法，了解数学形态学的基本原理。最后通过案例实现图像变换、形态学变换和图像特征提取。

本章主要内容

- 图像变换
- 图像边缘检测
- 图像分割
- 数学形态学
- 图像变换应用案例
- 形态学变换应用案例
- 图像特征提取应用案例

5.1 图像变换

图像变换是指对一幅图像进行不同的几何、颜色或灰度变换，以达到某种特定的目的。在图像处理中，图像变换是非常重要的技术之一。讨论图像变换时，经常提到仿射变换和透视变换这两个概念。图像仿射变换是一种可以保持图像形状的变换，其包括平移、旋转、缩放等操作。这些操作可以通过矩阵运算实现。在实际应用中，图像仿射变换广泛应用于图像的校正和匹配等领域。而图像透视变换则是指将三维场景映射到二维图像平面上的变换，它可以模拟出近似真实的三维效果。透视变换包括了仿射变换的所有操作，同时还包括了投影变换等更复杂的操作。在实际应用中，图像透视变换广泛应用于计算机视觉、虚拟现实等领域。

因此，讨论图像变换时，需要详细探讨仿射变换和透视变换的概念及其应用场景，以便更好地理解如何对图像进

行变换操作。

5.1.1 图像仿射变换

仿射变换，又称仿射映射，是指在几何中，一个向量空间进行一次线性变换并接上一个平移，变换为另一个向量空间。我们可以简单地把仿射变换理解为“线性变换”+“平移”。

仿射变换是一种二维坐标到二维坐标的线性变换，它保持了二维图形的“平直性”（直线经过变换后依然是直线）和“平行性”（二维图形之间的相对位置关系保持不变，平行线依然是平行线，且直线上点的位置顺序不变）。任意仿射变换都能表示为乘以一个矩阵（线性变换），再加上一个向量（平移）的形式，用公式表示如下。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = m_{11} \times x + m_{12} \times y + m_{13}$$

$$y' = m_{21} \times x + m_{22} \times y + m_{23}$$

以上公式将点(x, y)映射到(x', y')，在 OpenCV 中通过指定一个 2×3 的矩阵实现此功能（公式中的矩阵，是线性变换和平移的组合，m_{11}，m_{12}，m_{21}，m_{22}为线性变换参数，m_{13}，m_{23}为平移参数，其最后一行固定为 0，0，1，因此，将 3×3 的矩阵简化为 2×3 的矩阵）。

仿射变换在图像上应用十分广泛，常用的仿射变换有图像平移、图像旋转、图像缩放、图像翻转等。

OpenCV 中提供了 warpAffine 实现 2D 仿射变换，它可以对图像进行平移、旋转、缩放等操作，实现对图像的仿射变换。warpAffine 函数的原型如下。

```
dst = warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

其参数如下所示。

```
#src:    输入图像
#M:    2×3 的变换矩阵
#dst:    目标图像
#dsize:    指定图像输出尺寸
#flags:    插值算法标识符，有默认值 INTER_LINEAR，还有 INTER_NEAREST(最近邻插值)，
#cv.INTER_AREA(区域插值)，cv.INTER_CUBIC(三次样条插值)，cv.INTER_LANCZOS4
#(Lanczos 插值)
#borderMode:    边界像素模式，有默认值 BORDER_CONSTANT
#borderValue:    边界取值，有默认值 Scalar()即 0
warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) -> dst
```

具体实现步骤如下。

(1) 定义仿射变换矩阵 $\boldsymbol{M}$。可以通过 cv::getRotationMatrix2D()函数获得旋转矩阵，cv::getAffineTransform()函数用于获得仿射变换矩阵。

(2) 调用 warpAffine()函数进行仿射变换。调用函数时，需要指定输入图像和输出图像，仿射变换矩阵，以及插值方式、边界处理方式和边界填充值等参数。

- 图像旋转

图像旋转是一种常见的图像处理技术，它可以改变图像的方向和角度。在旋转过程

中，图像像素沿着一个指定的轴或中心点进行旋转，从而呈现出新的方向和角度。旋转可以以任意角度进行，其中 90°、180°和 270°是最常用的选项。在计算机视觉领域，图像旋转有多种应用。例如，在数字照片中，拍摄角度不佳可能导致图像畸变。通过对图像进行旋转，可以纠正这些畸变并使图像更加自然。此外，图像旋转也可用于创建动画，例如旋转的标志或 3D 模型的旋转。除此之外，图像旋转还可用于增强计算机视觉系统的性能。例如，在人脸识别系统中，如果人物面部朝向与基准相差很大，则旋转图像可能会使系统更容易识别。

在旋转图像前，需要选择一个旋转角度和旋转中心点。旋转角度通常以度为单位表示，并可以是任何值，包括正值、负值和小数值。旋转中心点可以是图像的中心点、某个关键点或用户指定的点。

当图像被旋转时，原始图像上的像素不再与目标图像上的像素对齐。因此，需要重新采样目标图像上的像素值，并将其设置为源图像上最接近的像素值或插值计算的像素值。这个过程可以通过插值算法实现。最近邻插值算法是最简单的插值算法之一，它通过查找距离目标像素最近的源像素计算目标像素的值。虽然该算法速度非常快，但会导致图像锯齿状的边缘和失真。双线性插值算法是一种更平滑的插值算法，它考虑了目标像素周围 4 个最近的源像素，并使用这些像素的加权平均值计算目标像素的值。这个算法比最近邻插值更加平滑，并且在旋转角度不太大的情况下产生良好的结果。处理高分辨率图像时，双线性插值算法的速度已经足够快。双三次插值算法在计算新位置上的像素值时使用 16 个相邻像素。这个算法比双线性插值更加平滑，并且在旋转角度很大的情况下也会产生比较好的结果。但是，由于计算量较大，因此速度可能比较慢。Lanczos 插值算法通过使用 Lanczos()函数作为权重函数来计算新位置上的像素值。这个算法比双三次插值更加平滑，并且在旋转角度很大的情况下也会产生比较好的结果。然而，它可能比其他插值算法需要更长的计算时间。

在实际应用中，选择哪种插值算法取决于应用的具体需求，例如图像尺寸和计算速度等因素。除了重新采样像素值外，还可以使用仿射变换或透视变换实现图像旋转。

• 图像缩放

图像缩放是一种将原始图像按比例进行尺寸调整的操作，通常用于改变图像的大小或者适应不同大小的显示设备或场景。在图像处理中，缩放操作可以通过调整像素的数量和位置实现，从而使得图像在保持几何结构完整性的同时，调整其像素分辨率和视觉表现。

进行图像缩放操作时，需要将原始图像中的每个像素点重新映射到新的位置上，并通过插值计算获得新位置上的像素值。这个过程可以分为两个阶段：插值和重采样。在插值阶段中，系统会根据缩放比例和目标图像的大小等信息，重新计算每个像素的新位置，并对新位置上的像素进行插值计算，以获得更平滑、更真实的图像效果。插值计算的目的是填补新位置和原始位置之间的空白区域，从而使得缩放后的图像质量更加逼真。常用的插值算法包括最邻近插值、双线性插值、双三次插值等，选择不同的插值算法可以获得不同的缩放效果。在重采样阶段，系统会根据插值后的像素计算结果，重新组织像素矩阵结构，从而生成新的缩放后的图像。通常，像素的重采样分为两种方式：上采样和下采样。上采样是指将原始图像中的像素进行插值计算，得到更多的像素数据，在此基础上形成新的高分辨率图像。而下采样则是指将原始图像中的像素进行压缩，从而得到更少的像素数据，在

此基础上形成新的低分辨率图像。

在实际操作中，常用的图像缩放插值算法包括最邻近插值、双线性插值和双三次插值等。最邻近插值是一种速度较快但效果较差的插值算法。它通过找到与目标位置最近的一个像素点，将该点的像素值赋值给新位置上的像素，从而实现像素值计算。最邻近插值算法的优点是速度快，适用于对实时性要求比较高的图像处理场景。但是，由于它只采用了最近邻点的像素值，所以无法获得更精细的缩放效果，在某些情况下会出现锯齿状或者失真现象。双线性插值是一种速度适中且效果较好的插值算法。它通过对目标位置周围的 4 个像素点进行加权平均来计算新位置上的像素值。具体地，它首先在水平和竖直方向上分别进行线性插值，然后再将两个方向上的插值结果进行加权平均。由于双线性插值算法采用了周围多个像素点的像素值，因此能获得比最邻近插值更为平滑的缩放效果，能避免锯齿状和失真现象。双三次插值是一种速度较慢但效果最优的插值算法。它通过对目标位置周围的 16 个像素点进行双线性插值，并利用多项式拟合获得更加平滑的图像效果。具体地，它首先在水平和竖直方向上分别进行二次插值，然后再将两个方向上的插值结果进行加权平均，并利用多项式函数对结果进行拟合，从而获得更加平滑的缩放效果。由于双三次插值算法采用了更多的像素点进行插值和拟合计算，因此能获得最优的视觉效果，但同时也需要付出较高的计算代价。

5.1.2 图像透射变换

透视变换(Perspective Transformation)是一种将图像从一个透视投影空间映射到另一个透视投影空间的技术。在计算机视觉和图形学中，透视变换常用于校正或重构具有透视畸变的图像或场景。它可以使图像或场景看起来更自然、更精确，并能提高图像处理和模式识别的准确率。透视变换是计算机视觉和图形学领域中常用的技术之一，具有广泛的应用价值。

透视变换是将图片投影到一个新的视平面，也称作投影映射。它是二维(x,y)到三维(X,Y,Z)，再到另一个二维(x',y')空间的映射。

相对于仿射变换，它提供了更大的灵活性，将一个四边形区域映射到另一个四边形区域(不一定是平行四边形)。它不只是线性变换，也是通过矩阵乘法实现的，使用的是一个3×3的矩阵，矩阵的前两行与仿射矩阵相同$(m_{11},m_{12},m_{13},m_{21},m_{22},m_{23})$，也实现了线性变换和平移，第三行用于实现透视变换。

$$\begin{bmatrix}X\\Y\\Z\end{bmatrix}=\begin{bmatrix}m_{11}&m_{12}&m_{13}\\m_{21}&m_{22}&m_{23}\\m_{31}&m_{32}&m_{33}\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

$$X=m_{11}\times x+m_{12}\times y+m_{13}$$

$$Y=m_{21}\times x+m_{22}\times y+m_{23}$$

$$Z=m_{31}\times x+m_{32}\times y+m_{33}$$

$$x'=\frac{X}{Z}=\frac{m_{11}\times x+m_{12}\times y+m_{13}}{m_{31}\times x+m_{32}+y+m_{33}}$$

$$y'=\frac{Y}{Z}=\frac{m_{21}\times x+m_{22}\times y+m_{23}}{m_{31}\times x+m_{32}\times y+m_{33}}$$

以上公式设变换前的点是 z 值为 1 的点，它在三维空间的值是 x，y，1，在二维平面上的投影是 x，y，通过矩阵变换成三维空间中的点 X，Y，Z，再通过除以三维空间中 Z 轴的值，转换成二维平面中的点 x'，y'。从以上公式可知，仿射变换是透视变换的一种特殊情况.它把二维空间转到三维空间，变换后，再转回映射之前的二维空间（而不是另一个二维空间）。透视变换在图像上应用十分广泛，比如将二维矩阵图像变换成三维的空间显示效果，全景拼接。

透视变换的特点如下。

1）消除透视畸变

透视畸变是由于相机拍摄时角度问题导致图像出现的扭曲变形，例如拍摄建筑物时，远端会显得比近端小，这是因为相机的垂直方向与建筑物的法线角度不同，导致图像产生了透视畸变。透视变换可以消除这种畸变，使图像更加真实。

2）三维重构

透视变换可以通过对图像进行透视映射，将二维图像转换为三维场景，从而实现三维重构。例如，在机器人视觉领域中，可以使用透视变换将相机捕获的图像转换为三维点云，以便进行更准确的定位和检测。

3）场景校正

透视变换可以校正倾斜的场景，例如在图像处理中将书本或报纸上的文字校正为水平。此外，对于某些特定场景，例如安防监控等，需要对图像进行立体校正，以便获得更准确和完整的信息。

4）可视化效果

透视变换还可用于可视化效果，例如通过将图像映射到不同的三维形状上，实现特殊的视觉效果，如弯曲、扭曲、拉伸等。

cv.getPerspectiveTransform 是 OpenCV 中的一个函数，用于计算透视变换矩阵。该函数需要至少 4 个输入点和对应的 4 个输出点作为参数，返回一个 3×3 的转换矩阵，其中包含将输入点映射到输出点所需的所有信息。

透视变换的目标是将输入图像中的任意四边形区域（也可以是更多边形）映射为输出图像上的矩形区域。这样做的好处是可以消除或校正图像中的透视畸变，使其看起来更自然和准确。进行透视变换时，必须先明确原始图像和目标图像上的 4 个点，以便确定从原始图像到目标图像的映射关系。cv.getPerspectiveTransform()就是用于计算这种映射关系的函数，它能根据输入和输出点的坐标，计算出一个转换矩阵，以实现透视变换的目的。

计算转换矩阵的过程相当于解决一个线性方程组，并且具有唯一解。通过这个转换矩阵，可以使用 cv.warpPerspective()函数将原始图像进行透视变换，得到校正后的图像。在实际应用中，透视变换和计算转换矩阵通常用于图像处理、机器人视觉、摄影和计算机游戏等领域。在图像处理中，例如拍摄建筑物或文档时，透视变换和计算转换矩阵可以消除透视畸变，使其更加真实和准确；在机器人视觉中，透视变换和计算转换矩阵可以将二维图像转换为三维场景，从而实现三维重构；在摄影和计算机游戏中，透视变换和计算转换矩阵可以创建各种有趣的视觉效果。因此，计算转换矩阵是透视变换中至关重要的一步，通过它可以非常方便地进行透视变换操作。

OpenCV 中提供了函数 cv.getPerspectiveTransform()来计算转换矩阵，其函数原型为

```
getPerspectiveTransform(src, dst)
```

参数如下所示。

src：源图像中待测矩形的四点坐标。

dst：目标图像中矩形的四点坐标。

OpenCV 中提供了函数对图像进行透视变换，函数原型为

```
warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

其参数如下所示。

src：输入图像。

M：变换矩阵。

dsize：目标图像尺寸。

flags：插值方式，interpolation 方法 INTER_LINEAR 或 INTER_NEAREST。

borderMode：边界补偿方式，BORDER_CONSTANT 或 BORDER_REPLICATE。

borderValue：边界补偿大小，常值，默认为 0。

5.1.3 图像重映射变换

图像重映射变换(Image Remapping)是将一幅图像从一个坐标系映射到另一个坐标系的过程。在计算机视觉中，它通常用于纠正图像畸变，校准摄像头，实现鱼眼矫正、图像缩放等任务。图像重映射变换可以看作一种像素级别的操作，其基本思想是将源图像中的每个像素位置，按照指定的映射关系，对应到目标图像中的新位置，并从源图像中取出该位置的像素值，复制到目标图像的对应位置中。这个过程可以通过插值算法实现，以保证目标图像中像素的连续性和平滑性。图像重映射变换的核心是建立源图像与目标图像之间的映射关系，这通常可以通过计算某些变换参数实现，例如旋转角度、平移向量、缩放因子、透视变换矩阵等。此外，还可以通过使用标定板等专用工具对摄像头进行标定，获取相机内参和畸变参数，从而实现对图像进行畸变矫正。OpenCV 中的 remap()函数是一种基于映射表进行图像重定向的方法。它可用于对图像进行几何变换，如旋转、缩放、平移等，并且还可实现其他的变换操作。remap()函数的主要作用是将源图像的每个像素位置映射到目标图像的另一个位置上。

remap()函数需要提供两幅图像——源图像和目标图像，以及一个由源图像和目标图像的关系定义的映射表。这个映射表可以通过某些算法或自定义的规则生成。

remap()函数的语法如下。

```
remap(
    InputArray src,                     //输入图像
    OutputArray dst,                    //输出图像
    InputArray map1,                    //第一维映射表
    InputArray map2,                    //第二维映射表
    int interpolation = INTER_LINEAR,   //插值方法
    int borderMode = BORDER_CONSTANT,   //处理边界的方式
    const Scalar& borderValue = Scalar()
                                        //边界值(如果 borderMode=BORDER_CONSTANT)
)
```

其中，src 参数为输入的源图像，dst 参数为输出的目标图像，map1 和 map2 分别为源图像和目标图像之间的映射表，interpolation 参数用于指定插值方法，borderMode 参数用于指定处理边界的方式，borderValue 参数用于指定边界的填充值。

使用 remap()函数时，需要先生成一个映射表。常见的映射表生成方式包括：

① 通过调用 cv::initUndistortRectifyMap()函数生成校正映射表；

② 通过调用 cv::getRotationMatrix2D()函数生成旋转矩阵，并使用该矩阵生成映射表；

③ 通过自定义规则生成映射表；

④ 映射表生成后，将其传递给 remap()函数进行图像重定向操作。

1. 复制

要实现图像复制，可以将原始图像作为输入，并使用单位矩阵作为转换矩阵。这将导致输出图像与输入图像完全相同。

2. 绕 x 轴翻转

为了实现图像绕 x 轴翻转，需要先定义一个变换矩阵，然后传递给 remap()函数。

以下是具体步骤。

(1) 定义变换矩阵 $\boldsymbol{M}$，用于将图像沿 x 轴翻转，可以使用如下代码：

```
M = np.array([[-1, 0, img.shape[1]], [0, 1, 0]], dtype=np.float32)
```

其中，img.shape[1]表示图像的宽度。

(2) 使用 cv2.initUndistortRectifyMap()函数计算重映射矩阵 map1 和 map2，如下所示。

```
map1, map2 = cv2.initUndistortRectifyMap(K, D, R, P, (w, h), cv2.CV_32FC1)
```

其中，$\boldsymbol{K}$、$\boldsymbol{D}$、$\boldsymbol{R}$、$\boldsymbol{P}$ 分别为相机内部参数、畸变系数、旋转矩阵和投影矩阵，(w,h)表示输出图像的大小。

(3) 将 map1 和 map2 作为参数传递给 remap()函数，完成图像重映射：

```
dst = cv2.remap(src, map1, map2, interpolation=cv2.INTER_LINEAR)
```

其中，src 表示输入图像，dst 表示输出图像，interpolation 表示插值方法，这里使用线性插值。

通过以上步骤，可以实现将图像绕 x 轴翻转的效果。

5.1.4 Hough(霍夫)变换

霍夫变换是一种图像处理技术，最初由保罗·霍夫(Paul Hough)在 1962 年提出。它的作用是在图像中检测出特定形状的对象，如直线、圆、曲线等，并且可以通过检测到的这些对象进行后续的图像分析和处理。

霍夫变换的基本思路是将图像中的每个像素点转换为参数空间上的一个曲线或者点。比如，对于直线检测来说，每个像素点被转换为在参数空间上表示一条直线的某个点。然后，在参数空间上搜索特定形状的模式，也就是在参数空间上寻找曲线或者点的交叉点，从

而实现对象的检测。

直线霍夫变换的原理包括以下几个知识点。

1. 直线方程

在介绍直线霍夫变换之前，先了解一下直线的数学表示方法。平面上的直线可以用一般式、斜截式和截距式等形式表示，其中最常见的是斜截式：

$$y=kx+b$$

式中，k 表示直线的斜率，b 表示直线在 y 轴上的截距。

2. 霍夫空间

直线霍夫变换将平面上的直线表示为霍夫空间中的点，从而使直线的检测问题转换为在霍夫空间中寻找点的问题。霍夫空间是一个二维坐标系，在该坐标系中，每个点代表一条直线。具体地，如果直线的斜率为 k，截距为 b，则其在霍夫空间中的坐标为(k,b)。

3. 直线到霍夫空间的映射

将平面上的直线表示为霍夫空间中的点，需要进行一个映射过程。假设直线在平面坐标系中的表达式为 $y=kx+b$，则可以通过对 k 和 b 进行变换，得到它们在霍夫空间中的坐标(k,b)。

具体地，可以以截距 b 为横坐标轴，以斜率 k 为纵坐标轴，构建一个二维数组，通常称为累加器，用于记录每个(k,b)坐标出现的次数。

从而，对于平面空间中的一条直线来说，其在霍夫空间中的映射可以简述为：平面上的一条直线对应霍夫空间中的一个点，这个点代表了所有可能与该直线共线的直线。

4. 霍夫空间中的直线检测

在霍夫空间中，由于平面上的同一条直线可以映射成霍夫空间中的多个点，因此需要采取一定的策略确定哪些点对应的是同一条直线。

假设霍夫空间中的一条直线对应一组参数(k_0,b_0)，那么哪些点对应的是同一条直线呢？

从平面空间的角度看，一条直线可以通过两个参数确定，因此，如果这些点在霍夫空间中位于同一个(k_0,b_0)处，则它们对应的就是同一条直线。

因此，在霍夫空间中进行直线检测的方法是：遍历累加器中的每个元素，找出其中出现次数最多的一组参数(k_0,b_0)，然后以此为基准，在平面空间中检测出所有与之共线的直线。

直线霍夫变换的实现：

1）将图像转换为二值图像

首先，将原始图像转换为二值图像，这是因为直线霍夫变换是基于边缘检测的，只有二值图像才能进行准确的边缘检测。

常用的二值化方法有全局阈值法、自适应阈值法等，可以根据实际情况选择合适的方法。

2）边缘检测

接着，对二值图像进行边缘检测，以便将图像中的直线提取出来。常用的边缘检测算

法有 Sobel 算子、Canny 算子等。

3）构建霍夫空间累加器

在构建霍夫空间累加器之前，需要确定一些参数，如霍夫空间的大小和步长、斜率和截距的范围等。

然后，遍历所有的边缘点，在霍夫空间中为每个边缘点投票，即将经过该点的所有直线在霍夫空间中对应的位置上加 1，这样就可以得到一个累加器数组，其中每个元素对应一个空间位置，记录了在该位置上通过多少个边缘点。

4）检测直线

在霍夫空间中，找到累加器数组中的最大值，即出现次数最多的位置。然后根据该位置确定所有可能的直线，以及它们在平面空间中的位置和角度。

具体地，可以通过以下方式将霍夫空间中的点转换为平面空间中的直线：

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

式中，θ 为直线与 x 轴的夹角，ρ 为直线到原点的距离。

这样就可以在图像上标记出检测出的直线了。

圆形霍夫变换（Circular Hough Transform）是一种基于霍夫变换（Hough Transform）的圆形检测方法，用于从图像中检测出圆形。

圆形霍夫变换的原理是将圆形在图像空间中表示为其心点坐标和半径大小。然后，在霍夫空间中，将每个圆形的参数组合都表示为一个点，即通过心点坐标和半径大小 3 个参数表示一个圆形，并且在霍夫空间中以这个参数组合为中心建立一个以半径为轴的圆形区域，然后扫描图像上所有可能的圆形，将其转换为霍夫空间上相应的点集。

实现过程如下。

预处理：将原始图像进行边缘检测，获得边缘点集。常见的边缘检测算法有 Canny 边缘检测、Sobel 算子等。

在霍夫空间创建累加器数组 A，并初始化为 0。数组 A 的大小和分辨率需要根据实际情况进行选择。通常，可以对圆心位置和半径值离散化，然后选择一个合适的步长表示，这样可以减少霍夫空间的存储空间和计算量。

对于每个边缘点(x,y)，计算其对应的在霍夫空间中可能的圆心坐标(a,b)和半径 r 的值。具体来说，对于给定的圆心坐标(a,b)，可以使用勾股定理计算出该圆心到该边缘点的距离，进而计算出对应的半径 r 值，其中 $r=\mathrm{sqrt}((x-a)^2+(y-b)^2)$。

遍历图像上所有可能的圆心坐标和半径 r 的组合，在霍夫空间中相应的位置加 1。具体来说，对于给定的圆心坐标(a,b)和半径 r，可以通过以下公式计算其在霍夫空间中的位置：

$$A(a,b,r)+=1$$

在累加器数组 A 中找到所有值大于设定阈值的点，这些点表示的参数组合即检测出的圆形的参数，包括圆心坐标和半径大小。可以根据这些参数在原始图像上绘制相应的圆形。

可选：引入非极大值抑制算法，以避免检测出重复的圆形。该算法通常是在霍夫空间中实现的，具体来说，就是对于每个检测到的圆形，在相邻的圆形区域内将投票次数最高的圆形保留，其他圆形则被丢弃。

霍夫变换是一种图像处理算法，用于检测图像中的特定形状，如直线、圆等。该算法基于参数空间投票的思想，通过将每个点在参数空间中投票来识别目标形状。

对于直线的检测，霍夫变换将直线表示为极坐标系中的两个参数，即距离和角度。对于给定图像中的每个点，在参数空间中与其相应的曲线上增加一个投票。最终，投票最多的曲线即待检测直线。

对于圆的检测，霍夫变换将圆表示为 3 个参数，即圆心坐标和半径。对于给定图像中的每个点，在参数空间中与其相应的圆上增加一个投票。最终，投票最多的圆即待检测圆。

虽然霍夫变换具有很好的通用性，但也存在一些缺点。对于大尺寸图像，计算量可能非常大，因为需要搜索一个大的参数空间。而且，对于一些复杂的目标，如弧形、椭圆等，需要更复杂的参数空间才能描述，这样会增加计算的复杂度。同时，当输入图像存在噪声时，霍夫变换的结果可能会受到影响，因为噪声点也会在参数空间中投票。

尽管如此，霍夫变换仍然是一种非常有用的图像处理算法，在计算机视觉领域中应用广泛。它可用于物体识别和跟踪等方面，以提高图像处理的准确性和效率。此外，该算法还可用于特征提取和匹配等任务。

5.2 图像边缘检测

边缘就是图像中包含的对象的边界对应的位置。物体的边缘是以图像局部特性的不连续性的形式出现的，例如，灰度值的突变，颜色的突变，纹理结构的突变等。本质上，边缘意味着一个区域的终结和另外一个区域的开始。图像边缘信息在图像分析和人的视觉中十分重要，是图像识别中提取图像特征的一个重要属性。

边缘检测在图像处理和对象识别领域中都是一个重要的问题。由于边缘的灰度不连续性，因此可以使用求导数的方法检测。最早的边缘检测方法都是基于像素的数值导数的运算。目前已经有非常多的边缘检测算法，现有的边缘检测方法主要分为基于梯度的方法和基于机器学习的方法两类。基于梯度的方法通常使用一些特定的滤波器（如 Sobel、Prewitt、Canny 等）检测不同方向上的梯度变化，并根据一些预设的阈值将梯度强度高于某一阈值的像素划分为边缘像素。这类方法简单易实现，但对噪声敏感，而且在边缘连接处可能产生断裂或者多余的边缘。

相比之下，基于机器学习的方法可以通过深度学习网络自动学习边缘特征并进行边缘检测。这类方法相对于传统方法的优点在于具有更好的鲁棒性和准确性，缺点是需要大量的训练数据和运算资源。

5.2.1 Prewitt 边缘检测

Prewitt 算子是一种经典的图像边缘检测算法，它利用特定区域内像素灰度值的差分实现边缘检测。该算法采用一个 3×3 的模板对图像进行卷积运算，以检测像素之间的变化。计算时，将模板中的元素与图像的像素值进行乘法运算，并将结果相加，从而得到中心像素的梯度值。

由于 Prewitt 算子采用了比 Robert 算子更大的模板，在水平和垂直方向上都能更好地检测边缘，因此，Prewitt 算子适用于识别噪声较多、灰度渐变的图像。在这种情况下，