
目录

第一章	图像及其表达与性质	2
1.1	图像表达的若干概念	2
1.2	图像数字化	3
1.3	数字图像性质	4
1.4	彩色图像	11
1.5	摄像机论述	12
第二章	图像及其数学与物理背景	13
2.1	概述	13
2.2	卷积的理解——来自广大的网友的理解	15
第三章	OpenCV 大数组类型	26
3.1	Mat 内存布局	26
3.2	Mat 数据访问方式	26
第四章	Mat 和 QImage 之间的转换	31
4.1	QImage 类	31

第 1 章 图像及其表达与性质

1.1 图像表达的若干概念

图像可以表述为两个变量的一个连续函数 $f(x, y)$ ，其中 (x, y) 是平面的坐标；或者可能是三个变量的连续函数 $f(x, y, t)$ ，其中 t 是时间。

当 3D 物体经透视投影映射到摄像机平面后，由于这样的变换不是一一对应的，因而大量的信息丢失了，通过一幅图像来识别和重构 3D 图像是个病态的问题，可以预料这不是一个简单的人物，涉及试图建立图像中点的**深度**这个中间表达层次。目标是恢复完整的 3D 表达，比如计算机图形学中的表达，及独立于视点的表达，表示在物体坐标系中而不是在观察者坐标系中。如果这样的表达可以恢复，则物体的任何视角的亮度图像都可以用标准的计算机图形学技术合成出来。

计算机化的图像处理使用的数字图像通常表示成矩阵的形式，因此其坐标是整数。图像函数的定义域是一个区域 R

$$R = \{(x, y), 1 \leq x \leq x_m, 1 \leq y \leq y_n\} \quad (1.1)$$

其中 x_m 和 y_n 表示最大的图像坐标。图像函数具有有限的作用域——由于假定图像函数在域 R 外的值为零，可以使用无限求和或者积分的形式。

图像函数的值域也是有限的，按照惯例，在单色图像中最低对应于黑，而最高对应于白。它们之间的亮度值是**灰阶**。

数字图像的品质随着空间、频谱、辐射计量、时间分辨率增长而提高。**空间分辨率**是由图像平面上图像采样点间的接近程度确定的，**频谱分辨率**是由传感器获得的光线频率带宽决定的，**辐射计量分辨率**对应于可区分的灰阶数量，**时间分辨率**取决于图像获得的时间采样间隔。

图像变换假定图像函数 $f(x, y)$ 是“良态的”，意思是指：该函数是可积的、具有可逆的傅里叶变换等。

1.2 图像数字化

图像数字化是指将 $f(x, y)$ 采样为一个 M 行 N 列的矩阵。图像量化给每个连续的样本数值一个整数数字，图像函数 $f(x, y)$ 的连续范围被划分为 K 个区间。采样及量化越精细（即 M 、 N 、 K 越大），对连续函数 $f(x, y)$ 的近似就越好。

图像函数采样有两个问题，其一是确定采样的间隔，即相邻两个采样图像点的距离；其二是设置采样点的几何排列（采样栅格）。

1.2.1 采样

一个连续图像在采样点处被数字化。这些采样点是在平面上排列的，称它们的几何关系为栅格。因此数字图像是一个数据结构，通常是矩阵。在实践中，栅格一般是方的或者是正六边形。

把栅格和光栅区别开是重要的。光栅是指在点之间定义了相邻关系的栅格¹。

栅格中一个无限小的采样点对应于数字化图像中的一个相元，也称作像素。从图像分析的角度看，像素是不能再分割的一个单元。

1.2.2 量化

在图像处理中，采样的图像数值 $f_s(j\Delta x, k\Delta y)$ 用域一个数字表示。将图像函数的连续数值（亮度）转化为其数字等价量的过程称为量化。为了使人们能够觉察图像的细微变化，量化的级别要足够的高。大部分数字图像处理仪器都采用 k 个等间隔的量化方式。如果用 b 位来表示像素亮度的数值，那么亮度阶就是 $k = 2^b$ 。通常采用每个像素每个通道（红、绿、蓝各一个）8 位的表示方式，尽管也有一些系统使用其他数字（比如 16）。数字图像中亮度值一种有效的计算机表示，需要 8 位、4 位或 1 位，也就是说计算机存储的每个字节分别相应地可以存下 1 个、2 个或 8 个像素的亮度。

在量化级别不够时，图像的主要问题是出现伪轮廓（false contours）。

¹即如果在方形的栅格上定义了 4-邻域，则得到方形的光栅

1.3 数字图像性质

数字图像具有一些度量和拓扑性质，与我们在基础微积分中所熟悉的连续二维函数的性质有所不同。另一个不同点在于人们对图像的感知，因为对图像质量的判断也是重要的。

1.3.1 数字图像的度量和拓扑性质

连续图像所具有的一些明显的直觉特性在数字图像领域中没有直接的类似推广。**距离**是一个重要的例子。满足下面三个条件的任何函数是一种“距离”(或度量):

$$\begin{aligned} D(\mathbf{p}, \mathbf{q}) &\geq 0, \text{ 当且仅当 } \mathbf{p} = \mathbf{q} \text{ 时 } D(\mathbf{p}, \mathbf{q}) = 0 && \text{同一性} \\ D(\mathbf{p}, \mathbf{q}) &= D(\mathbf{q}, \mathbf{p}) && \text{对称性} \\ D(\mathbf{p}, \mathbf{r}) &\leq D(\mathbf{p}, \mathbf{q}) + D(\mathbf{q}, \mathbf{r}) && \text{三角不等式} \end{aligned}$$

坐标为 (i, j) 和 (h, k) 的两点间的距离可以定义为几种形式。
经典的几何学和日常经验中的**欧氏距离** D_E 定义为:

$$D_E[(i, j), (h, k)] = \sqrt{(i - h)^2 + (j - k)^2} \quad (1.2)$$

两点之间的距离也可以表示为数字栅格上从起点移动到终点所需要的最少的基本步数。如果只允许横向和纵向的移动, 即是 D_4 距离。 D_4 也称为“**城市街区距离**”,

$$D_4[(i, j), (h, k)] = |i - h| + |j - k| \quad (1.3)$$

在数字栅格中如果允许对角线方向的移动, 我们就得到了距离 D_8 , 常常称为“**棋盘距离**”。距离 D_8 等于

$$D_8[(i, j), (h, k)] = \max\{|i - h|, |j - k|\} \quad (1.4)$$

像素**邻接性**是数字图像的另一个重要概念。任意两个像素如果它们之间的距离 $D_4(\mathbf{p}, \mathbf{q}) = 1$, 则称彼此**4-邻接**的。类似地, **8-邻接**是指两个像素之间的距离 $D_8(\mathbf{p}, \mathbf{q}) = 1$ 。

由一些彼此邻接的像素组成的重要几何, 我们称为“**区域**”, 是一个重要的概念。更具体描述性的说法是, 如果定义从像素 P 到像素 Q 的**路径**为一个点序列 A_1, A_2, \dots, A_n , 其中 $A_1 = P, A_n = Q$, 且 A_{i+1} 是 A_i 的临界点,

$i = 1, \dots, n - 1$; 那么**区域**是指这样的集合, 其中任意两个像素之间都存在完全属于该集合的路径。

如果两个像素之间存在一条路径, 那么这些像素就是**连通的**。因此, 可以说区域是彼此连通的像素的集合。“连通”关系是自反的, 对称的、且传递的, 因此它定义了集合的一个分解, 即等价类。

假设 R_i 是“连通”关系产生的不相交的区域, 进一步假设 (为了避免特殊的情况) 这些区域与图像边界不接触。设区域 R 是所有这些区域 R_i 的并集, R^C 是区域 R 相对于图像的补集合。我们称包含图像边界的 R^C 的连通集合为**背景**, 而称补集合 R^C 的其它部分称为**孔**。如果区域中没有孔, 则称为**简单连通**区域。等价地, 简单连通区域的补集合是连通的。有孔的区域称为**复连通**。

请注意, 区域概念只使用了“连通”性。我们可以给区域赋予第二属性, 这些源于对图像数据的理解。我们常称图像的一些区域为**物体**, 决定图像中哪些区域对应于世界中的物体的过程是图像**分解**。

像素的亮度是一种非常简单的性质, 在有些图像中可以用于寻找物体, 例如, 如果一个像素比给定的值 (阈值) 黑就属于物体。所有这样的点的连通集构成一个物体。一个孔由非物体的点组成且被物体所围, 所有其他的点就构成了背景。

距离变换, 也叫做**距离函数**或**斜切算法**或**简单地斜切**, 它是距离概念的一个简单应用。距离变化提供了像素与某个图像子集 (可能表示物体或某些特征) 的距离。所产生的图像在该子集元素位置处的像素值为 0, 邻接的像素具有较小的值, 而离它远的数值就大, 该技术的命名源于这个阵列的外观。理解的例子参考桑卡书籍第 15 页。

对于距离度量 D_4 和 D_8 , Rosenfeld 和 Pfaltz 提出了一个计算距离变换的两遍算法。其想法是用一个小的局部掩膜遍历图像。第一遍从左上角开始, 水平从左到右直至图像边界, 然后返回到下一行开始处继续。第二遍从右下角开始, 使用一个不同的局部掩膜, 从右到左, 从下到上。该算法的有效性源于以“波浪状”的方式传播前一步的勘测的数值。

Computer Vision (算法 1). 距离变换

1. 按照一种距离度量 D , D 是 D_4 或 D_8 , 对大小为 $M \times N$ 的图像的一个子集 S 计算距离变换, 建立一个 $M \times N$ 的数组 F 并做初始化, 子集 S 中的元素置为 0, 其他置为无穷。

2. 按行遍历图像，从上到下，从左到右。对于上方和左面的邻接像素，设

$$F(p) = \min_{q \in AL} [F(p), D(p, q) + F(q)]$$

3. 按行遍历图像，从下到上，从右到左。对于下方和右面的邻接像素，设

$$F(p) = \min_{q \in BR} [F(p), D(p, q) + F(q)]$$

4. 数组 F 中得到的是子集 S 的斜切。

边缘是另一个重要的概念。它是一个像素和其直接邻域的局部性质，它是一个有大小和方向的矢量。边缘告诉我们在一个像素的小邻域内图像亮度变化有多块。边缘计算的对象是具有很多亮度级别的图像，计算边缘的方式是计算图像函数的梯度。

区域的**边界**是图像分析中的另一个重要概念。区域 R 的边界是它自身的一个像素集合，其中的每个点具有一个或更多个 R 外的邻接点。该边界的定义与我们的直观理解相同，即边界是区域边界点的集合。有时我们称这样的边界为**内部边界**，同样地，我们可以定义类似的**外部边界**，**外部边界**是指区域的背景（即区域的补集）的边界。

“边界”和“边缘”虽然相关，但是它们不是同一个概念。边界是与区域有关的全局概念，而边缘表示图像函数的局部性质。一种可能的寻找边界的方法是链接显著的边缘。

一个区域是**凸**的是指如果区域内的任意两点连成一条线段，那么这条线段完整地处于区域内部。凸性将所有的区域划分成两个等价类：凸和非凸。

一个区域的**凸包**是指包含输入区域的一个最小凸区域。参看桑卡书籍的第17页。

拓扑性质不是基于距离的概念。相反，它们对于同胚的 (homeomorphic) 变换具有不变性，对图像而言，homeomorphic 变化可以解释为**橡皮面变换**。想像一下一个其表面上绘制了物体的小橡皮球，物体的拓扑性质是指在橡皮表面任意伸展时具有不变性的那部分性质。伸展不会改变物体各部分的连通性，也不会改变区域中孔的数目。

非规则形状的物体可以用一组它的拓扑分量来表示。凸包中非物体的部分

称为**凸损**；它可以分解为两个子集：**湖**，完全被物体所包围；**海湾**，与物体凸包的边界连通。

1.3.2 直方图

图像的**亮度直方图** $h_f(z)$ 给出图像中亮度值 z 出现的频率，一幅有 L 个灰阶的图像的直方图由具有 L 个元素的一维数组表示。

Computer Vision (算法 2). 计算亮度直方图

1. 数组 h_f 的所有元素赋值为 0。
2. 对于图像 f 的所有像素，做 $h_f[f(x, y)] + 1$ 。

直方图在图像和概率之间建立了一个自然的桥梁。我们可能需要考虑找一个一阶概率函数 $p_q(z; x, y)$ 来表示像素 (x, y) 的值 z 概率。在直方图中感兴趣的不是像素的位置，而是密度函数 $p_1(z)$ ，亮度直方图就是它的估计。直方图通常用柱状图来显示。

直方图通长是关于图像的唯一可得到的全局信息。在寻找最佳的照明条件以便抓取图像、进行灰阶变换以及将图像分割为物体和背景这些场合，都要用到直方图。

数字图像的直方图一般都有很多和极大值和极小值，这会使进一步的处理变得复杂。这个问题可以通过对直方图进行局部平滑来避免，比如，可以用相邻直方图元素的局部平均来做，因此新的直方图按下式来计算：

$$h'_f(z) = \frac{1}{2K+1} \sum_{j=-K}^K h_f(z+j) \quad (1.5)$$

其中 K 是一个常量，代表平滑所使用的邻域的大小。这个算法需要某种边界调整，也不能保证去除所有的局部极小值。还有一些其他平滑技术，重要的高斯模糊，在直方图的情况下，它是 $2D$ 高斯模糊的简化。

1.3.3 熵

如果知道概率密度 p ，用**熵** $H(\text{entropy})$ 就可以估计出图像的信息量，而与其解释无关。熵的信息论的形成源于香农，常称作**信息熵**。

信息熵的直观理解与关联于给定概率分布的事件的不确定性大小有关。熵可作为失调的度量。当失调水平上升时，熵就增加而事件就难于预测。

假设离散随机变量 X 的可能结果（也称作状态）为 x_1, \dots, x_n ，设 $p(x_k)$ 是出现 $x_k (k = 1, \dots, n)$ 的概率，熵定义为

$$H(X) = \sum_{k=1}^n p(x_k) \log_2 \left(\frac{1}{p(x_k)} \right) = - \sum_{k=1}^n p(x_k) \log_2 p(x_k) \quad (1.6)$$

随机变量 X 的熵是 X 所有可能的出现 x_k 的图下乘积的累加和：出现 x_k 的概率于 x_k 概率的倒数之对数的乘积。 $\log_2(\frac{1}{p(x_k)})$ 也称作出现 x_k 的惊异 (surprisal)。随机变量 X 的熵是其出现惊异的期望。

1.3.4 图像的视觉感知

我们在设计或使用数字图像处理算法或设备时，应该考虑人的图像感知原理。

对比度 (contrast)

对比度是亮度的局部变化，定义为物体亮度的平均值与背景亮度的比值。严格地说，如果我们的目的是要在物理上精确，应该讲的是辐射率而非亮度。人的眼睛对亮度的敏感性称对数关系，意味着对于同样的感知，高亮度需要高的对比度。

表现上的亮度很大程度上取决于局部背景的亮度，这种现象被称为条件对比度。

敏锐度

敏锐度是觉察图像细节的能力。人的眼睛对于图像平面中的亮度的缓慢和快速变化敏感度差一些，而对于其间的中等变换较为敏感。敏锐度也随着离光轴的距离增加而降低。

1.3.5 图像品质

在图像的捕获、传输和处理过程中可能使图像退化，图像品质的度量可以用来估计退化的程度。

估计图像品质的方法可分为两类：主观的和客观的。度量图像品质的客观定量方法对我们更重要。理想的情况是，这样的方法同时也提供了主观的测试，且易于使用；这样我们就可以将标准用于参数优化。图像 $f(x, y)$ 的品质通常

通过与一个已知的参考图 $g(x, y)$ 作比较来估计。为这一目的，常常使用合成的图像作为参考图像。有一类方法使用简单的度量，比如均方差 $\Sigma(g - f)^2$ 。

1.3.6 图像中的噪声

实际的图像常受一些随机误差的影响而退化，我们通常称这个退化为**噪声**。噪声一般由其概率特征来刻画。理想的噪声，称作**白噪声**，经常会用到。白噪声具有是常量的功率谱，也就是说噪声在所有的频率上出现且强度相同。例如，白噪声的强度并不随着频率的增加而衰减，这在实际世界的信号中是典型的。白噪声是常用的模型，作为退化的最坏估计。

白噪声的一个特例是**高斯噪声**。服从高斯型的随机变量具有高斯曲线型的概率密度。在一维的情况下，密度函数是

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1.7)$$

其中 μ , σ 分别是随机变量的均值和标准差。在很多实际情况下，噪声可以很好地用高斯噪声来近似。

当图像通过信道传输时，噪声一般与其出现的图像信号无关。这种独立于信号的退化被称为**加性噪声**，可以用下列模型来表示

$$f(x, y) = g(x, y) + v(x, y) \quad (1.8)$$

其中，噪声 v 和传输图像 g 是相互独立的变量。

Computer Vision (算法 3). 产生加性零均值高斯噪声

1. 假定图像的灰阶范围是 $[0, G - 1]$ 。取 $\sigma > 0$ ；它的值小时，相应的噪声也小。
2. 针对每对水平相邻的像素 (x, y) , $(x, y + 1)$ 产生一对位于 $[0, 1]$ 范围的独立的随机数 r, ϕ 。
3. 计算

$$\begin{aligned} z_1 &= \sigma \cos(2\pi\phi) \sqrt{-2 \ln r} \\ z_2 &= \sigma \sin(2\pi\phi) \sqrt{-2 \ln r} \end{aligned} \quad (1.9)$$

(这是 Box-Muller 变换，假定 z_1, z_2 是独立的具有 0 均值和 σ 方差的正态分布。)

4. 置 $f'(x, y) = g(x, y) + z_1$ 和 $f'(x, y + 1) = g(x, y + 1) + z_2$, 其中 g 是输入图像。

5. 置

$$f(x, y) = \begin{cases} 0 & \text{当 } f'(x, y) < 0 \\ G - 1 & \text{当 } f'(x, y) > G - 1 \\ f'(x, y) & \text{其它} \end{cases} \quad (1.10)$$

$$f(x, y + 1) = \begin{cases} 0 & \text{当 } f'(x, y + 1) < 0 \\ G - 1 & \text{当 } f'(x, y + 1) > G - 1 \\ f'(x, y + 1) & \text{其它} \end{cases} \quad (1.11)$$

6. 跳转到步骤 3, 直到扫描完所有的像素。

式 (1.10) 和式 (1.11) 的截断会减弱噪声的高斯性质, 特别是当 σ 值与 G 比起来大时更加显著。

根据式 (1.9), 可以定义**信噪比 SNR**。计算噪声贡献的所有平方和

$$E = \sum_{(x,y)} v^2(x, y) \quad (1.12)$$

将它与观察到的信号的所有平方和作比较

$$F = \sum_{(x,y)} f^2(x, y) \quad (1.13)$$

信噪比就是

$$SNR = \frac{F}{E} \quad (1.14)$$

(严格地说, 我们将平均观测值相比于平均误差, 计算显然是一样的)。SNR 是图像品质的一个度量, 值大的就“好”。

信噪比常用对数尺度来表示, 单位为分贝:

$$SNR_{db} = 10 \log_{10} SNR \quad (1.15)$$

噪声的幅值在很多情况下与信号本身的幅值有关:

$$f = gv \quad (1.16)$$

这种模型表达的是**乘性噪声**。

量化噪声会在量化级别不足时出现，例如，仅有 50 个级别的单色图像。

冲击噪声是指一幅图像被个别噪声像素破坏，这些像素的亮度与其邻域显著不同。

胡椒盐噪声是指饱和的冲击在噪声，这是图像被一些白的或黑的像素所破坏，胡椒盐噪声会使二值图像退化。

1.4 彩色图像

人的色彩感知在电磁辐射的波长这一客观的物理性质上加了一主观层次。因此，色彩可以被认为是一种心里物理现象。

1.4.1 色彩物理学

只有狭窄的一段电磁波谱对人是可见的，大致对应于 380nm–740nm 的一段。色彩可以表达为基色的组合，基色即红、绿、蓝，为了标准化它们分别定义为 700nm、546.1nm 和 435.8nm。然而，这并不意味着所有的彩色都可以通过这三个基色的组合合成出来。

为什么我们看到的世界是带色彩的？当照射表面时，有两种占主导地位的物理机制来解释会发生什么。首先，**表面反射**以类似镜子的方式弹回进来的能量。反射光的光谱和入射光的光谱一模一样，与表面无关。其次，能量扩散进入材料内并随机地从其内部的颜料反射。这种机制称为**体反射**。色彩是颜料粒子的性质引起的，它们从入射光谱中吸收了某些波长。

1.4.2 人所感知的色彩

人类和一些动物在进化中发展了一种间接色彩感知机制。人建立起了对入射辐照光波长敏感的三种类型的传感器，即**三色觉**。人类视网膜上颜色敏感的感受器是**锥状体**。视网膜上另一种光敏感受器是**杆状体**，专注于在周边光照强度低的情况下的单色感知。锥状体按照感知的波长范围分为三类：S(短)，敏感度最大出现在 $\approx 430\text{nm}$ ；M(中) 在 $\approx 560\text{nm}$ ；L(长) 在 $\approx 610\text{nm}$ 。锥状体 S、M、L 偶尔也分别称作锥状体 B、G、R，但是这有点误导。当锥状体 L 激发时，我们看到的不单独时红。

感光器的反应或摄像机种的传感器的输出可以以数学建模。设 i 是某个传

传感器类型, $i = 1, 2, 3$ (在人的情况下是视网膜锥状体的类型 S、M、L)。设 $R_i(\lambda)$ 是传感器的光敏度, $I(\lambda)$ 是光照的谱密度, $S(\lambda)$ 表达表面元如何反射照明光的每个波长。第 i 个传感器的光谱响应可以用一定波长范围内的积分来建模:

$$q_i = \int_{\lambda_1}^{\lambda_2} I(\lambda) R_i(\lambda) S(\lambda) d\lambda \quad (1.17)$$

一种现象称为**条件等色**。一般而言, 条件等色是指两件物理上不同的事看起来却相同。红加绿产生黄就是一种条件等色, 因为黄也可以由一个光谱颜色产生。人的视觉系统受愚弄将绿感知成与黄一样。

由于彩色几乎可以用任意的基色集合来定义, 国际社会协商确定广泛应用的基色和色彩匹配函数。引进**色彩模型**作为数学抽象, 使我们可以将色彩表达为数字的元组, 通常是颜色分量的三或四数值的元组。受报刊和彩色电影发展的驱动, 于 1931 年 CIE 提出了一个技术标准, 称作**XYZ 色彩空间**。CIE 标准是绝对标准的一个例子, 它定义了色彩的无歧义的表达, 不依赖其它外部因素。

XYZ 色彩标准满足三个要求:

- 不同于色彩匹配实验中产生色彩匹配函数负波瓣的情况, XYZ 色彩空间的色彩匹配函数必须是非负的;
- $Y(\lambda)$ 的数值应该于亮度 (照度) 相等;
- 实施规范化以确保对应于三种色彩匹配函数的功率像等。

实际的色彩是如下的混合

$$c_X X + c_Y Y + c_Z Z \quad (1.18)$$

其中 $0 \leq c_X, c_Y, c_Z \leq 1$ 是混合权重。

1.5 摄像机论述

第2章 图像及其数学与物理背景

2.1 概述

在这里我们并不介绍所有必须的数学和物理基础，详细的内容请参看对应的物理和数学书籍。

2.1.1 线性

线性这一概念将在本书中频繁出现：这与**矢量空间**有关，其中常用矩阵代数。线性也与是矢量空间的更一般元素有关，比如，函数。在线性代数中**线性组合**是一个关键概念，允许矢量空间的一个元素可以表示为已有元素与系数乘积的和。两个矢量 x, y 的一般线性组合可以写成 $ax + by$ ，其中 a 和 b 是标量。

考虑两个线性空间的映射 L 。如果 $L(x, y) = L(x) + L(y)$ ，则称为加性的，如果对于任意的标量 a 有 $L(ax) = aL(x)$ ，则称为单一性的。从实际角度看，这意味着输入的和产生各自输出的和。这一性质也称为**叠加原理**。如果 L 是加性的且是单一性的，则称该映射是线性的。等价的，对于任意的矢量 x, y 和标量 a, b ，线性映射满足 $L(ax + by) = aL(x) + bL(y)$ ，即它保持线性组合。

2.1.2 狄拉克分布和卷积

理想的冲击是一个重要的输入信号，图像平面上的理想冲击是用**狄拉克分布**定义的， $\delta(x, y)$

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(x, y) dx dy = 1 \quad (2.1)$$

且对于所有的 $x, y \neq 0$ ，有 $\delta(x, y) = 0$ 。狄拉克函数具有的筛选性

$$\int_{-\infty}^{\infty} f(x, y) \delta(x - \lambda, y - \mu) dx dy = f(\lambda, \mu) \quad (2.2)$$

它提供了函数 $f(x, y)$ 在点 λ, μ 处的值。该筛选公式可以用来描述连续图像函数 $f(x, y)$ 的采样过程。我们可以将图像函数表示成覆盖整个图像平面的位于

各点 (a, b) 的狄拉克脉冲的线性组合，采样由图像函数 $f(x, y)$ 加权，

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(a, b) \delta(a - x, b - y) da db = f(x, y) \quad (2.3)$$

卷积在图像分析的方法中是一个重要的运算。卷积是一个积分，反映一个函数 $f(t)$ 在另一函数 $h(t)$ 移动时所重叠的量。函数 f 和 h 的在有限域 $[0, t]$ 上的 1D 卷积 $f * h$ 由下式给出：

$$(f * h)(t) = \int_0^t f(\tau) h(t - \tau) d\tau \quad (2.4)$$

为了准确起见，卷积积分的上下限是 $(-\infty, \infty)$ 。这里可以限定在 $[0, t]$ 区间，原因是我们假设负坐标部分的值为零。

$$(f * h)(t) = \int_0^t f(\tau) h(t - \tau) d\tau = \int_{-\infty}^{\infty} f(t - \tau) h(\tau) d\tau \quad (2.5)$$

设 f, g, h 为函数， a 是一个常数。则卷积具有如下的性质：

$$f * h = h * f \quad (2.6)$$

$$f * (g * h) = (f * g) * h \quad (2.7)$$

$$f * (g + h) = (f * g) + (f * h) \quad (2.8)$$

$$a(f * g) = (af) * g = f * (ag) \quad (2.9)$$

对卷积进行微分有

$$\frac{d}{dx}(f * h) = \frac{df}{dx} * h = f * \frac{dh}{dx} \quad (2.10)$$

卷积可以推广到更高维。2D 函数 f 和 h 的卷积 g 记为 $f * h$ ，通过积分定义为

$$\begin{aligned} (f * h)(x, y) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(a, b) h(x - a, y - b) da db \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x - a, y - b) h(a, b) da db \\ &= (h * f)(x, y) \end{aligned} \quad (2.11)$$

233 在数字图像分析中，**离散卷积**用求和来表达，而不是积分。数字图像在图像平
234 面上有有限的定义域。但是，有限的定义域并不妨碍我们使用卷积，因为在图
235 像定义域外它们的结果是零。

线性操作中输出图像像素 $g(x,y)$ 的计算结果是输入图像像素 $f(x,y)$ 的一个局部邻域 \mathcal{O} 的亮度的线性组合。邻域 \mathcal{O} 中像素的贡献用系数 h 进行加权：

$$f(i,j) = \sum_{(m,n) \in \mathcal{O}} h(i-m, j-n)g(m,n) \quad (2.12)$$

上式与以 h 为核的离散卷积等价，称 h 为**卷积掩膜**。一般使用具有为奇数的行和列的矩阵邻域 \mathcal{O} ，这样能够确定邻域的中心。

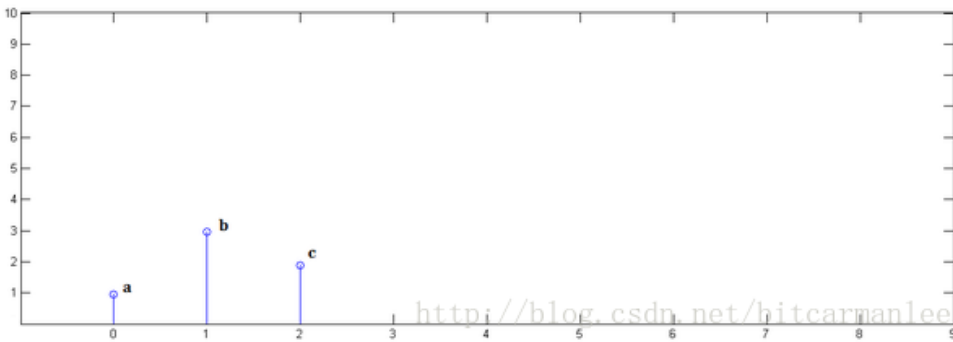
2.2 卷积的理解——来自广大的网友的理解

2.2.1 知乎上排名最高的解释

首先选取知乎上对卷积物理意义解答排名最靠前的回答。不推荐用“反转/翻转/反褶/对称”等解释卷积。好好的信号为什么要翻转？导致学生难以理解卷积的物理意义。这个其实非常简单的概念，国内的大多数教材却没有讲透。

直接看图，不信看不懂。以离散信号为例，连续信号同理。

已知 $X[0] = a, x[1] = b, x[2] = c$



已知 $y[0] = i, y[1] = j, y[2] = k$

下面通过演示求 $x[n] * y[n]$ 的过程，揭示卷积的物理意义。

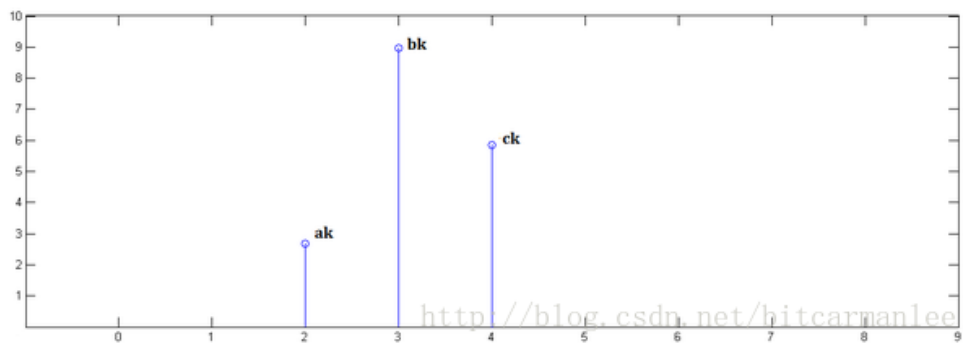
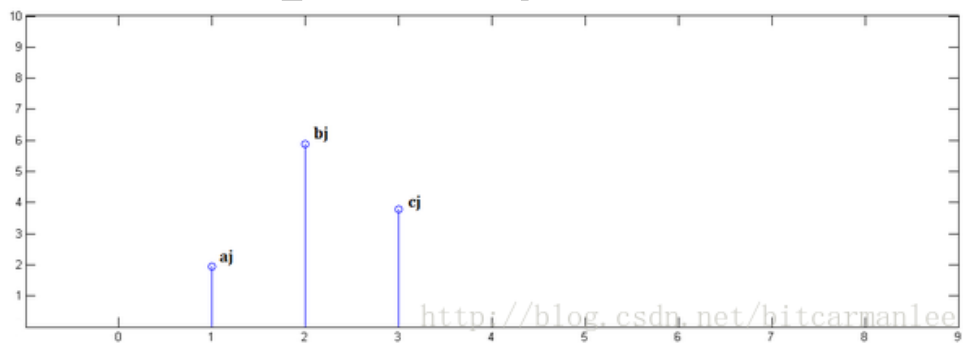
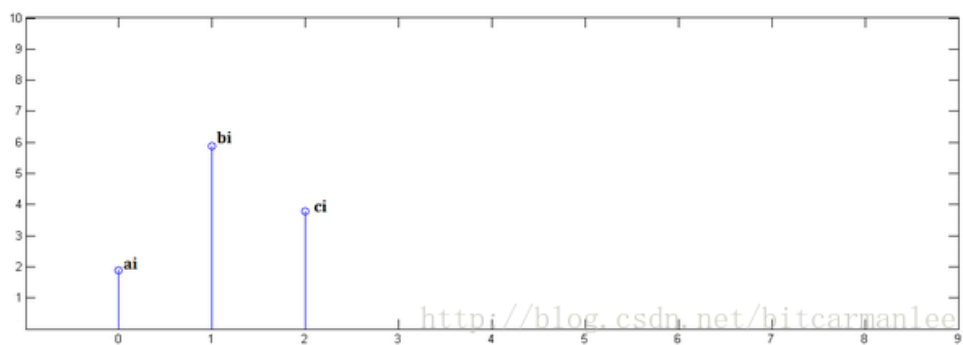
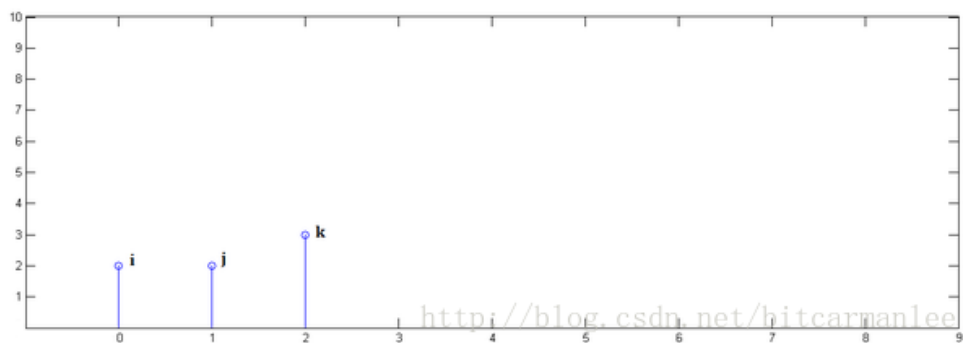
第一步， $x[n]$ 乘以 $y[0]$ 并平移到位置 0：

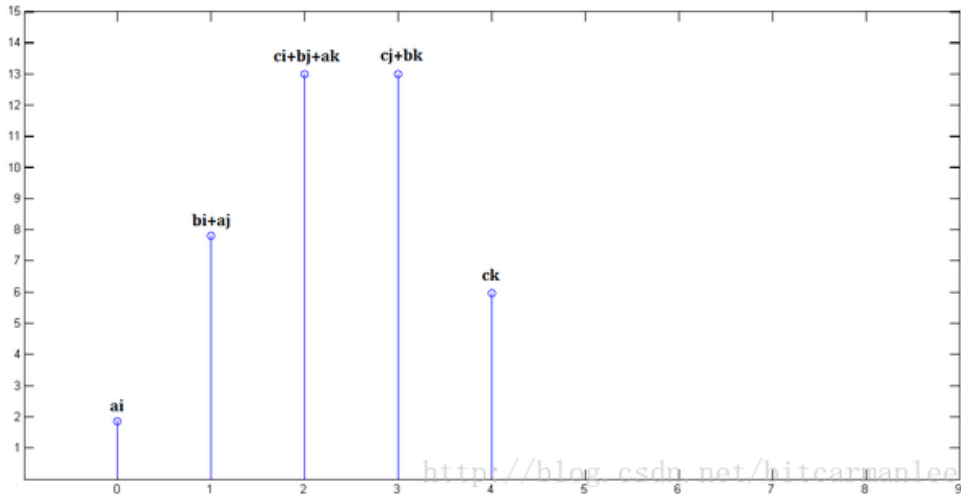
第二步， $x[n]$ 乘以 $y[1]$ 并平移到位置 1

第三步， $x[n]$ 乘以 $y[2]$ 并平移到位置 2

最后，把上面三个图叠加，就得到了 $x[n] * y[n]$ ：

简单吧？无非是平移（没有反褶！）、叠加。从这里，可以看到卷积的重要的物理意义是：一个函数（如：单位响应）在另一个函数（如：输入信号）上





的加权叠加。重复一遍，这就是卷积的意义：加权叠加。

对于线性时不变系统，如果知道该系统的单位响应，那么将单位响应和输入信号求卷积，就相当于把输入信号的各个时间点的单位响应加权叠加，就直接得到了输出信号。

通俗的说：在输入信号的每个位置，叠加一个单位响应，就得到了输出信号。这正是单位响应是如此重要的原因。

在输入信号的每个位置，叠加一个单位响应，就得到了输出信号。这正是单位响应是如此重要的原因。

在输入信号的每个位置，叠加一个单位响应，就得到了输出信号。这正是单位响应是如此重要的原因。

以上是知乎上排名最高的回答。比较简单易懂。

2.2.2 卷积的另外解释

卷积表示为 $y(n) = x(n) * h(n)$ ，使用离散数列来理解卷积会更形象一点，我们把 $y(n)$ 的序列表示成 $y(0), y(1), y(2), \dots$ ，这是系统响应出来的信号。

同理， $x(n)$ 的对应时刻的序列为 $x(0), x(1), x(2), \dots$ ，其实我们如果没有学过信号与系统，就常识来讲，系统的响应不仅与当前时刻系统的输入有关，也跟之前若干时刻的输入有关，因为我们可以理解为这是之前时刻的输入信号经过一种过程（这种过程可以是递减，削弱，或其他）对现在时刻系统输出的影响，那么显然，我们计算系统输出时就必须考虑现在时刻的信号输入的响应以及之前若干时刻信号输入的响应之“残留”影响的一个叠加效果。假设 0 时

刻系统响应为 $y(0)$ ，若其在 1 时刻时，此种响应未改变，则 1 时刻的响应就变
成了 $y(0) + y(1)$ ，叫序列的累加和（与序列的和不一样）。但常常系统中不是这
样的，因为 0 时刻的响应不太可能在 1 时刻仍旧未变化，那么怎么表述这种变
化呢，就通过 $h(t)$ 这个响应函数与 $x(0)$ 相乘来表述，表述为 $x(m) \times h(m-n)$ ，
具体表达式不用多管，只要记着有大概这种关系，引入这个函数就能够表述
 $y(0)$ 在 1 时刻究竟削弱了多少，然后削弱后的值才是 $y(0)$ 在 1 时刻的真实
值，再通过累加和运算，才得到真实的系统响应。

再拓展点，某时刻的系统响应往往不一定是由当前时刻和前一时刻这两个
响应决定的，也可能是再加上前前时刻，前前前时刻，前前前前时刻，等
等，那么怎么约束这个范围呢，就是通过对 $h(n)$ 这个函数在表达式中变化后
的 $h(m-n)$ 中的 m 的范围来约束的。即说白了，就是当前时刻的系统响应与
多少个之前时刻的响应的“残留影响”有关。当考虑这些因素后，就可以描述
成一个系统响应了，而这些因素通过一个表达式（卷积）即描述出来不得不说是
数学的巧妙和迷人之处了。

2.2.3 卷积的应用

用一个模板和一幅图像进行卷积，对于图像上的一个点，让模板的原点和
该点重合，然后模板上的点和图像上对应的点相乘，然后各点的积相加，就得
到了该点的卷积值。对图像上的每个点都这样处理。由于大多数模板都是对称
的，所以模板不旋转。卷积是一种积分运算，用来求两个曲线重叠区域面积。
可以看作加权求和，可以用来消除噪声、特征增强。

把一个点的像素值用它周围的点的像素值的加权平均代替。卷积是一种线
性运算，图像处理中常见的 mask 运算都是卷积，广泛应用于图像滤波。

卷积关系最重要的一种情况，就是在信号与线性系统或数字信号处理中的
卷积定理。利用该定理，可以将时间域或空间域中的卷积运算等价为频率域的
相乘运算，从而利用 FFT 等快速算法，实现有效的计算，节省运算代价。

2.2.4 补充

另外在知乎上看到非常好也非常生动形象的解释，特意复制粘贴过来。（知
乎马同学的解释）

从数学上讲，卷积就是一种运算。

某种运算，能被定义出来，至少有以下特征：

1. 首先是抽象的、符号化的

2. 其次，在生活、科研中，有着广泛的作用

比如加法：

1. $a+b$ ，是抽象的，本身只是一个数学符号

2. 在现实中，有非常多的意义，比如增加、合成、旋转等等

卷积，是我们学习高等数学之后，新接触的一种运算，因为涉及到积分、级数，所以看起来觉得很复杂。

1 卷积的定义

我们称 $(f * g)(n)$ 为 f, g 的卷积

其连续的定义为：

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n - \tau)d\tau$$

其离散的定义为：

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n - \tau)$$

这两个式子有一个共同的特征：

这个特征有什么意义？

只看数学符号，卷积是抽象的，不好理解的，但是，我们可以通过现实中的意义，来习惯卷积这种运算，正如我们小学的时候，学习加减乘除需要各种苹果、糖果来帮助我们习惯一样。

我们来看看现实中，这样的定义有什么意义。

离散卷积的例子：丢骰子我有两枚骰子：把这两枚骰子都抛出去：求：两枚骰子点数加起来为 4 的概率是多少？这里问题的关键是，两个骰子加起来要等于 4，这正是卷积的应用场景。

我们把骰子各个点数出现的概率表示出来：

那么，两枚骰子点数加起来为 4 的情况有：

因此，两枚骰子点数加起来为 4 的概率为：


$$f(1)g(3) + f(2)g(2) + f(3)g(1) \quad (2.13)$$

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n - \tau)d\tau$$

$n = \tau + (n - \tau)$

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n - \tau)$$

<http://blog.csdn.net/bitcarmanlee>



<http://blog.csdn.net/bitcarmanlee>

f

1	2	3	4	5	6
---	---	---	---	---	---

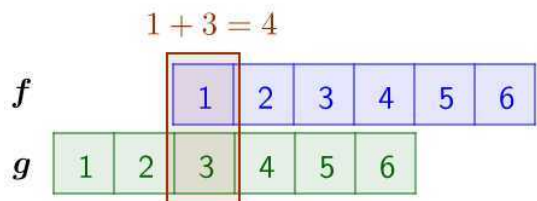
 f 表示第一枚骰子
 $f(1)$ 表示投出1的概率
 $f(2)$ 、 $f(3)$ 、...以此类推

g

1	2	3	4	5	6
---	---	---	---	---	---

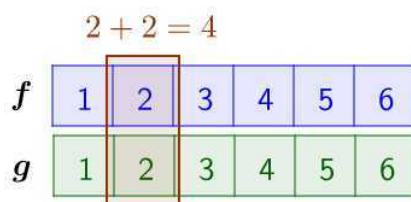
 g 表示第二枚骰子

<http://blog.csdn.net/bitcarmanlee>



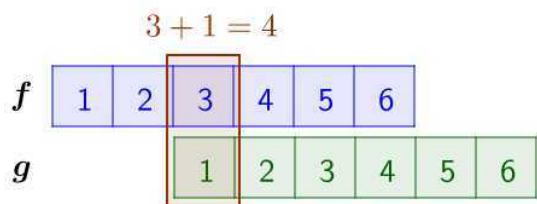
出现概率为: $f(1)g(3)$

<http://blog.csdn.net/bitcarmanlee>



出现概率为: $f(2)g(2)$

<http://blog.csdn.net/bitcarmanlee>



出现概率为: $f(3)g(1)$

<http://blog.csdn.net/bitcarmanlee>

符合卷积的定义，把它写成标准的形式就是：

$$(f * g)(4) = \sum_{m=1}^3 f(4-m)g(m) \quad (2.14)$$

2.2.5 连续卷积的例子：做馒头

连续卷积的例子：做馒头

楼下早点铺子生意太好了，供不应求，就买了一台机器，不断的生产馒头。假设馒头的生产速度是 $f(t)$ ，那么一天后生产出来的馒头总量为：

$$\int_0^{24} f(t) dt \quad (2.15)$$

馒头生产出来之后，就会慢慢腐败，假设腐败函数为 $g(t)$ ，比如，10 个馒头，24 小时会腐败：10 * $g(t)$ 想想就知道，第一个小时生产出来的馒头，一天后会经历 24 小时的腐败，第二个小时生产出来的馒头，一天后会经历 23 小时的腐败。如此，我们可以知道，一天后，馒头总共腐败了：

$$\int_0^{24} f(t)g(24-t)dt \quad (2.16)$$

这就是连续的卷积。

2.2.6 图像处理

2.2.6.1 原理

有这么一副图像，可以看到，图像上有很多噪点：



这些噪点，属于高频信号

<http://blog.csdn.net/bitcarmanlee>

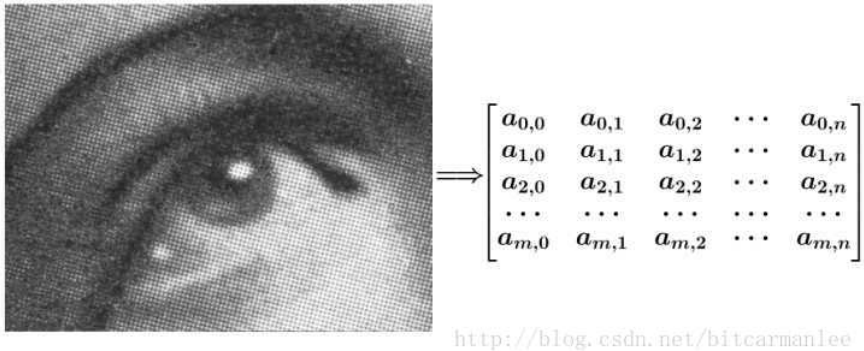
高频信号，就好像平地耸立的山峰：



329 看起来很显眼。平滑这座山峰的办法之一就是，把山峰刨掉一些土，填到
330 山峰周围去。用数学的话来说，就是把山峰周围的高度平均一下。平滑后得到：



331
332 卷积可以帮助实现这个平滑算法。有噪点的原图，可以把它转为一个矩阵：



333 然后用下面这个平均矩阵（说明下，原图的处理实际上用的是正态分布矩

阵，这里为了简单，就用了算术平均矩阵) 来平滑图像：

$$g = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \quad (2.17)$$

记得刚才说过的算法，把高频信号与周围的数值平均一下就可以平滑山峰。

比如我要平滑 $a_{1,1}$ 点，就在矩阵中，取出 $a_{1,1}$ 点附近的点组成矩阵 f ，和 g 进行卷积计算后，再填回去

$$f = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix}$$

$$c = \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & \cdots & c_{1,n} \\ c_{1,0} & c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,0} & c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{m,0} & c_{m,1} & c_{m,2} & \cdots & c_{m,n} \end{bmatrix}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m,0} & a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \Rightarrow c_{1,1} = f * g$$

<http://blog.csdn.net/bitcarmanlee>

要注意一点，为了运用卷积， g 虽然和 f 同维度，但下标有点不一样：

注意两者的下标**不一样**

$$f = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \quad g = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

<http://blog.csdn.net/bitcarmanlee>

我用一个动图来说明下计算过程：

a, b 的下标相加都为1, 1

$$f = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \quad g = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

$$c_{1,1} = a_{0,0}b_{1,1} + a_{0,1}b_{1,0} + a_{0,2}b_{1,-1} + a_{1,0}b_{0,1} \\ + a_{1,1}b_{0,0} + a_{1,2}b_{0,-1} + a_{2,0}b_{-1,1} \\ + a_{2,1}b_{-1,0} + a_{2,2}b_{-1,-1}$$

blog.csdn.net/bitcarmanlee

写成卷积公式就是：

$$(f * g)(1, 1) = \sum_{k=0}^2 \sum_{h=0}^2 f(h, k) g(1-h, 1-k) \quad (2.18)$$

339 要求其它的 $c_{4,5}$ ，一样可以套用上面的公式。这样相当于实现了 g 这个矩阵在
 340 原来图像上的划动（准确来说，下面这幅图把 g 矩阵旋转了 π 角度）。

第3章 OpenCV 大数组类型

在 OpenCV 中, `cv::Mat` 类被用来表示任何维度的密度矩阵, 其中 `cv::Mat` 针对的是密集连续型的存储, 大多数的图型数据都是被保存为这种类型, 即使数据为空, 预留的存储空间仍然存在。

3.1 Mat 内存布局

对于一维的矩阵, 其元素可以被认为是连续存储的。对于二维的矩阵, 其数据是按行进行存储的, 对于三维的矩阵, 其中每个平面是按行进行排列, 然后每个平面进行排列。

`Mat` 类包含一个公有的 `flags` 数据成员标识数据的内容, `dims` 来标识矩阵的维度, `rows` 和 `cols` 标识矩阵的行数和列数 (对于 `dims > 2` 的矩阵, 这两个数据成员不可用), `data` 用来表示数据存储的首地址, `refcount` 表示其指针 `cv::Ptr<>` 引用的次数, 其中 `cv::Ptr<>` 的行为类似于 C++ 中的智能指针。对于 `data` 中的数据布局可以通过矩阵的 `step[]` 进行描述, 对于索引为 $(i_0, i_1, \dots, i_{N_d-1})$ 的元素, 其地址可以表示为:

$$\begin{aligned} \&(mtx_{i_0, i_1, \dots, i_{N_d-1}, N_d}) = &mtx.data + mtx.step[0] * i_0 + mtx.step[1] * i_1 \\ &+ \dots + mtx.step[N_d - 1] * i_{N_d-1} \end{aligned} \quad (3.1)$$

对于二维的简单情形可以表示为:

$$\&(mtx_{i,j}) = mtx.data + mtx.step[0] * i + mtx.step[1] * j \quad (3.2)$$

3.2 Mat 数据访问方式

对于 `Mat` 中的数据, 最直接的访问方式是使用模板函数 `at<>()`。例如:

```
1 cv::Mat m = cv::Mat::eye(10, 10, 32FC1);
2 std::cout << "Element (3,3) is "
3     << m.at<float>(3,3) << std::endl;
```

351 对于多通道矩阵，对于每个像素 (element) 的访问形式最简单就是使用
352 cv::Vec<>，如：

```
1 cv::Mat m = cv::Mat::eye(10, 10, 32FC2);
2 printf(
3     "Element (3,3) is (%f,%f)\n",
4     m.at<cv::Vec2f>(3,3)[0],
5     m.at<cv::Vec2f>(3,3)[1]
6 );
```

353 完整的例子如下：

```
1 #include <iostream>
2 #include "opencv2/opencv.hpp"
3
4 int main(int argc, char *argv[]){
5     cv::Mat grayimg(600, 800, CV_8UC1);
6     cv::Mat colorimg(600, 800, CV_8UC3);
7
8     for (int i = 0; i != grayimg.rows; ++i){
9         for(int j = 0; j != grayimg.cols; ++j){
10             grayimg.at<uchar>(i, j) = (i+j)%255;
11         }
12     }
13
14     for (int i = 0; i != colorimg.rows; ++i){
15         for (int j = 0; j != colorimg.cols; ++j){
16             cv::Vec3b pixel;
17             pixel[0] = i % 255;
18             pixel[1] = j % 255;
19             pixel[2] = 0;
20             colorimg.at<cv::Vec3b>(i, j) = pixel;
21         }
```

```
22 }
23
24 cv::namedWindow("grayimg", cv::WINDOW_AUTOSIZE);
25 cv::imshow("grayimg", grayimg);
26 cv::namedWindow("colorimg", cv::WINDOW_AUTOSIZE);
27 cv::imshow("colorimg", colorimg);
28 cv::waitKey(0);
29
30 return 0;
31 }
```

同样的我们也可以使用指针形式的访问，注意因为 Mat 中的数据并不一定是连续存储的，所以我们只能获取每行的指针。如：

```
1 #include <cmath>
2 #include <iostream>
3 #include "opencv2/opencv.hpp"
4
5 int main(int argc, char *argv[]){
6     cv::Mat grayimg(600, 800, CV_8UC1);
7     cv::Mat colorimg(600, 800, CV_8UC3);
8
9     for (int i = 0; i != grayimg.rows; ++i){
10         /*! 获取第i行的首地址 */
11         uchar *p = grayimg.ptr<uchar>(i);
12         for (int j = 0; j != grayimg.cols; ++j){
13             p[j] = (i+j)%255;
14         }
15     }
16
17     for(int i = 0; i != colorimg.rows; ++i){
18         /*! 获取第i行的首地址 */
19         cv::Vec3b *p = colorimg.ptr<cv::Vec3b>(i);
```

```
20     for (int j = 0; j != coloring.cols; ++j){
21         p[j][0] = i % 255; /*!< Blue */
22         p[j][1] = j % 255; /*!< Green */
23         p[j][2] = std::abs(i-j) % 255; /*!< Red */
24     }
25 }
26
27 cv::imshow("grayimg", grayimg);
28 cv::imshow("colorimg", coloring);
29
30 cv::waitKey(0);
31
32 return 0;
33 }
```

356 在 OpenCV 中同时提供了类似于 C++ 标准库中的迭代器，我们同样的
357 使用的这种方式进行访问元素：

```
1 #include <iostream>
2 #include "opencv2/opencv.hpp"
3
4 int main(int argc, char *argv[]){
5     cv::Mat grayimg(600, 800, CV_8UC1);
6     cv::Mat coloring(600, 800, CV_8UC3);
7
8     /*! loop gray image */
9     for (cv::MatIterator_<uchar> iter = grayimg.begin<uchar>();
10         iter != grayimg.end<uchar>();
11         ++iter){
12         *iter = rand() % 255;
13     }
14
15     /*! loop color image */
```

```
16  for (cv::MatIterator_<cv::Vec3b> iter =  
    ↪  coloring.begin<cv::Vec3b>();  
17  iter != coloring.end<cv::Vec3b>();  
18  ++iter){  
19      (*iter)[0] = rand() % 123;  
20      (*iter)[1] = rand() % 255;  
21      (*iter)[2] = rand() % 255;  
22  }  
23  
24  cv::imshow("grayimg", grayimg);  
25  cv::imshow("colorimg", coloring);  
26  
27  cv::waitKey(0);  
28  return 0;  
29 }
```

第 4 章 Mat 和 QImage 之间的转换

4.1 QImage 类

在 Qt 包含的类中，其中 **QImage** 类可能是最和计算机视觉最有关的类了，QImage 提供了像素访问和图像操作的众多函数，我们可以很容易的完成 QImage 对象和 OpenCV 中 Mat 对象之间的转换。

4.1.1 QImage 的构造函数

QImage 类包含了众多的构造函数允许我们创建 QImage 对象，例如从文件中，从数据中，或者简单先创建一个空的 QImage 对象，然后我们在后面再对每个像素进行操作赋值。我们可以创建一个空 QImage 对象通过给定图像的大小和图片的格式，

```
1 QImage image(320, 240, QImage::Format_RGB888);
```

这条语句创建了一个空的 RGB 图像，其中拥有 320×240 个像素。

4.1.2 QImage 从 Mat 中创建

QImage 类提供了可以直接从 Mat 对象创建图片的构造函数，但是在使用该构造函数之前我们必须对图像进行一丁点的转换，这是因为对于 Mat 中的图像，其格式是采用的 BGR，而不是我们通常采用的 RGB，因此要进行格式的转换。如下面的例子：

```
1 cv::Mat mat = cv::imread("/home/xiaohai/Pictures/test.jpeg");
2 cv::cvtColor(mat, mat, cv::COLOR_BGR2RGB);
3 QImage image((const uchar *)mat.data, mat.cols, mat.rows,
  ↳ QImage::Format_RGB888);
```

其中 `cv::cvtColor()` 函数可以完成图像彩色空间的转换。

然而上面的方式并不是推荐的方式，我们可以采用下面的构造函数去创建。该构造函数需要一个 **bytesPerLine** 参数，而该参数的值就是 `Mat` 对象中的 **step** 值，完整的例子如下：

```
1 QImage Mat2QImage(const cv::Mat &src)
2 {
3     //! make the same cv::Mat
4     cv::Mat temp;
5     //! cvtColor Makes a copy, that's what I need
6     cv::cvtColor(src, temp, cv::COLOR_BGR2RGB);
7     QImage dest((const uchar *)temp.data, temp.cols, temp.rows,
8         ↪ temp.step, QImage::Format_RGB888);
9     //! enforce deep copy, see documentation
10    dest.bits();
11
12    return dest;
13 }
```

4.1.3 QImage 对象转 Mat 对象

`QImage` 类包含大量的成员函数，这些大量的成员函数可以完成图像的操作和处理，如将 `QImage` 对象转换称 `Mat` 对象，

```
1 cv::Mat QImage2Mat(const QImage &src)
2 {
3     cv::Mat tmp(src.height(), src.width(), CV_8UC3, (uchar
4         ↪ *)src.bits(), src.bytesPerLine());
5     //! deep copy just in case (my lack of knowledge with open cv)
6     cv::Mat result;
7     cv::cvtColor(tmp, result, cv::COLOR_RGB2BGR);
8     return result;
9 }
```

