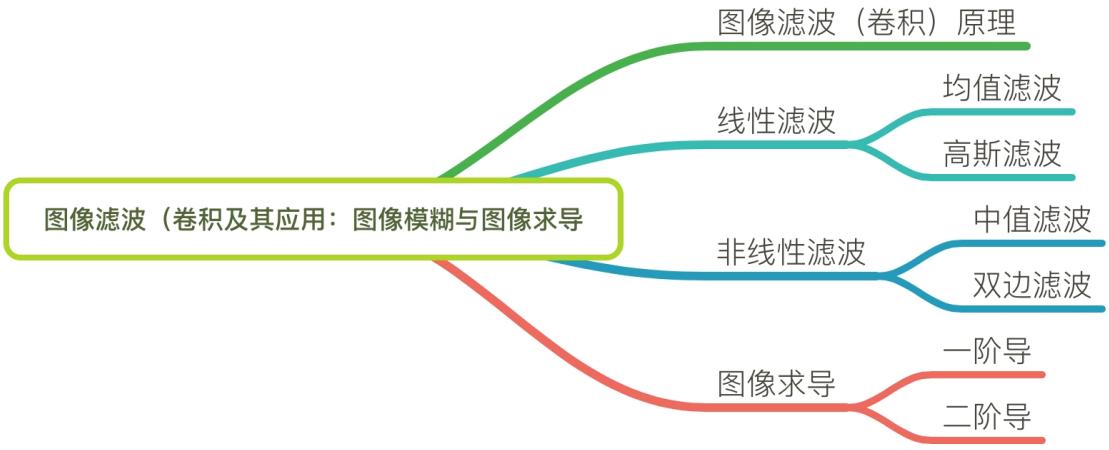


4.1 简介

图像的实质是一种二维信号，滤波是信号处理中的一个重要概念。在图像处理中，滤波是一种非常常见的技术，它们的原理非常简单，但是其思想却十分值得借鉴，滤波是很多图像算法的前置步骤或基础，掌握图像滤波对理解卷积神经网络也有一定帮助。

4.2 图像滤波原理

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

上图蓝色点可以解释为：

$$6.5+9.8+12.3+6.5+19.2+11.5+6.3+9.1+10.7=91.9\approx 92$$

线性滤波处理的输出像素值$g(i,j)$是输入像素值$f(i+k,j+l)$的加权和：

$$g(i,j)=\sum_{k,l}f(i+k,j+l)h(k,l)$$

其中的加权和为，我们称其为“核”，滤波器的加权系数，即滤波器的“滤波系数”。
上面的式子可以简单写作：

$$g=f\otimes h$$

其中f表示输入像素值，h表示加权系数“核”，g表示输出像素值

4.4.1卷积后图像大小问题

$$\lfloor (n+2\times p-k+s)/s \rfloor = \lfloor (n+2\times p-k)/s \rfloor + 1$$

其中n为图片长（宽），p为padding大小，k为卷积核长（宽），s为步长

4.4.2填充问题

在对图像应用滤波器进行过滤时，边界问题是一个需要处理的问题。一般来说，有4种处理的方法。

1. 填充0

对图像的边界做扩展，在扩展边界中填充0，对于边长为2k+1的方形滤波器，扩展的边界大小为k，若原来的图像为[m, n]，则扩展后图像变为[m+2k, n+2k]。

2. 填充固定值
   扩展与填充0的扩展类似,只不过将填充0改为填充固定值
3. 填充最近像素值

扩展与填充0的扩展类似，只不过填充0的扩展是在扩展部分填充0，而这个方法是填充距离最近的像素的值。

4. 镜像填充ReflectionPad2d

根据边缘进行镜像对称填充

4.3 线性滤波

4.3.1 均值滤波

均值滤波的应用场合：
根据冈萨雷斯书中的描述，均值模糊可以模糊图像以便得到感兴趣物体的粗略描述，也就是说，去除图像中的不相关细节，其中“不相关”是指与滤波器模板尺寸相比较小的像素区域，从而对图像有一个整体的认知。即为了对感兴趣的物体得到一个大致的整体的描述而模糊一幅图像，忽略细小的细节。

均值滤波的缺陷：
均值滤波本身存在着固有的缺陷，即它不能很好地保护图像细节，在图像去噪的同时也破坏了图像的细节部分，从而使图像变得模糊，不能很好地去除噪声点，特别是椒盐噪声。

均值滤波方法是：对待处理的当前像素，选择一个模板，该模板为其邻近的若干个像素组成，用模板的均值（方框滤波归一化）来替代原像素的值。公式表示为：

$$g(x,y) = 1/n \sum_{I \in Neighbour} I(x,y)$$

g(x,y)为该邻域的中心像素，n跟系数模版大小有关，一般3*3邻域的模板，n取为9，如：

$$\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

当然，模板是可变的，一般取奇数，如5 * 5 , 7 * 7等等。

```
import numpy as np
```

```
import cv2
from tensorbay import GAS
from tensorbay.dataset import Segment

# Authorize a GAS client.
gas = GAS('Accesskey-dac95e0d8e685ef4d5f8b80d51e38499')

# Get a dataset client.
dataset_client = gas.get_dataset("DogsVsCats-2")

# List dataset segments.
segments = dataset_client.list_segment_names()

# Get a segment by name
segment = Segment("train", dataset_client)
fp = segment[1].open()
with open("a.png", "wb") as f:
    f.write(fp.read())
```

```
def Conv(img, kernel, stride=1, padding=0):
    '''
    img: 输入图片
    kernel: 卷积核
    stride: 步长
    padding: 填充图片，这里用零来填充
    '''
    h, w = img.shape[0], img.shape[1]
    img = img.reshape(h, w, -1)
    c = img.shape[2]
    if padding > 0:
        img_p = np.zeros((h+2*padding, w+2*padding, c))
        img_p[padding:-padding, padding:-padding, :] = img
    else:
        img_p = img
    h_k, w_k = kernel.shape
    h, w = img_p.shape[0], img_p.shape[1]
    h_dst, w_dst = int((h - h_k + stride)/stride), int((w - w_k +
stride)/stride)
    img_dest = np.zeros((h_dst, w_dst, c))
    for i in range(h_dst):
        for j in range(w_dst):
            for k in range(c):
                img_dest[i, j, k] = np.sum(img_p[i*stride:i*stride+h_k,
j*stride:j*stride+w_k, k] * kernel)
    return img_dest
```

```
def noise(img,snr):
    h = img.shape[0]
    w = img.shape[1]
    img1 = img.copy()
    num = int(h * w * (1 - snr))
    for i in range (num):
        randx = np.random.randint(0, h)
        randy = np.random.randint(0, w)
        if np.random.random()<=0.5:
            img1[randx,randy]=0
```

```
        else:
            img1[randx,randy]=255
    return img1

img = cv2.imread('a.png', 1)
img_n = noise(img, 0.9)
cv2.imshow('n', img_n)
kernel = (1/25) * np.ones((5, 5))
img_blur = Conv(img_n, kernel).astype(np.uint8)
cv2.imshow('b', img_blur)
key = cv2.waitKey()
if key == 27:
    cv2.destroyAllWindows()
```

4.3.2 高斯滤波

应用： 高斯滤波是一种线性平滑滤波器，对于服从正态分布的噪声有很好的抑制作用。在实际场景中，我们通常会假定图像包含的噪声为高斯白噪声，所以在许多实际应用的预处理部分，都会采用高斯滤波抑制噪声，如传统车牌识别等。

高斯滤波和均值滤波一样，都是利用一个掩膜和图像进行卷积求解。不同之处在于：均值滤波器的模板系数都是相同的，而高斯滤波器的模板系数，则随着距离模板中心的增大而系数减小（服从二维高斯分布）。所以，高斯滤波器相比于均值滤波器对图像个模糊程度较小，更能够保持图像的整体细节。

二维高斯分布

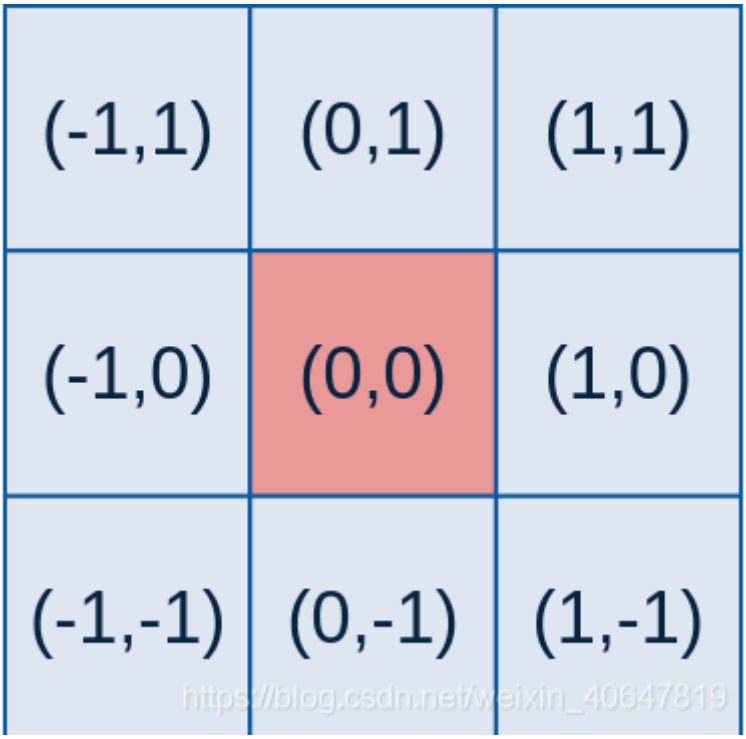
$$f(x,y)=\frac{1}{(\sqrt{2\pi}\sigma)^2}e^{-((x-ux)^2+(y-uy)^2)/2\sigma^2}$$

其中不必纠结于系数，因为它只是一个常数！并不会影响互相之间的比例关系，并且最终都要进行归一化，所以在实际计算时我们是忽略它而只计算后半部分:

$$f(x,y)=e^{-((x-ux)^2+(y-uy)^2)/2\sigma^2}$$

其中(x,y)为掩膜内任一点的坐标，(ux,uy)为掩膜内中心点的坐标，在图像处理中可认为是整数；σ是标准差。

例如：要产生一个3×3的高斯滤波器模板，以模板的中心位置为坐标原点进行取样。模板在各个位置的坐标，如下所示（x轴水平向右，y轴竖直向下）。



这样，将各个位置的坐标带入到高斯函数中，得到的值就是模板的系数。

生成高斯掩膜（小数形式）
知道了高斯分布原理，实现起来也就不困难了。

首先我们要确定我们生产掩模的尺寸wsize，然后设定高斯分布的标准差。生成的过程，我们首先根据模板的大小，找到模板的中心位置center。 然后就是遍历，根据高斯分布的函数，计算模板中每个系数的值。

最后模板的每个系数要除以所有系数的和。这样就得到了小数形式的模板。由于中心点为（0, 0）点，故公式变为：

$$f(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

3×3,σ=0.8的小数型模板：

```
[0.05711825900067253, 0.1247577476216454, 0.05711825900067253;
 0.1247577476216454, 0.2724959735107281, 0.1247577476216454;
 0.05711825900067253, 0.1247577476216454, 0.05711825900067253]
```

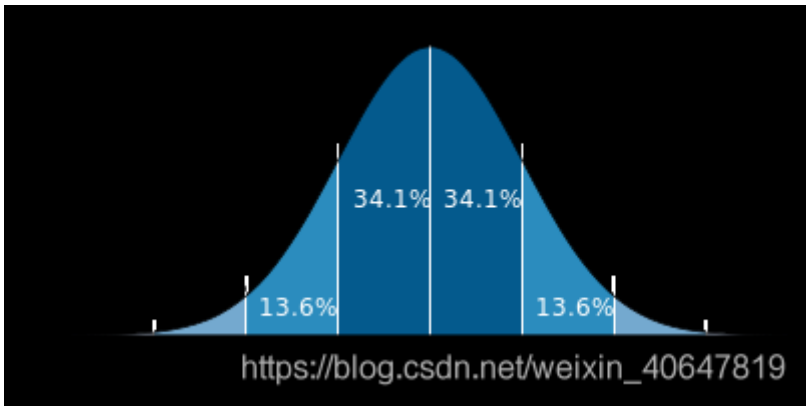
```python
import numpy as np

def imgGaussian(ksize, sigma):
    '''
    :ksize: 卷积核大小，必须是奇数
    :param sigma: σ标准差
    :return: 高斯滤波器的模板
    '''
    assert ksize%2 == 1
    gaussian_mat = np.zeros((ksize, ksize))
    k = int(ksize/2)
    for x in range(-k, k+1):
        for y in range(-k, k + 1):
            gaussian_mat[x + k][y + k] = np.exp(- (x ** 2 + y ** 2) / (2 * sigma
** 2))
    return gaussian_mat / gaussian_mat.sum()
```

σ的意义及选取

通过上述的实现过程，不难发现，高斯滤波器模板的生成最重要的参数就是高斯分布的标准差σ。标准差代表着数据的离散程度，如果σ较小，那么生成的模板的中心系数较大，而周围的系数较小，这样对图像的平滑效果就不是很明显；反之，σ较大，则生成的模板的各个系数相差就不是很大，比较类似均值模板，对图像的平滑效果比较明显。

来看下一维高斯分布的概率分布密度图：



于是我们有如下结论：σ越小分布越瘦高，σ越大分布越矮胖。

- σ越大，分布越分散，各部分比重差别不大，于是生成的模板各元素值差别不大，类似于平均模板；
- σ越小，分布越集中，中间部分所占比重远远高于其他部分，反映到高斯模板上就是中心元素值远远大于其他元素值，于是自然而然就相当于中间值得点运算。

```
kernel = imgGaussian(5, 7)
img_b = Conv(img_n, kernel, padding=0).astype(np.uint8)  # 高斯模糊
cv2.imshow('b', img_b)
key = cv2.waitKey()
if key == 27:
    cv2.destroyAllWindows()
```

4.4 非线性滤波

4.4.1 中值滤波

中值滤波是一种典型的非线性滤波，是基于排序统计理论的一种能够有效抑制噪声的非线性信号处理技术，基本思想是用像素点邻域灰度值的中值来代替该像素点的灰度值，让周围的像素值接近真实的值从而消除孤立的噪声点。该方法在取出脉冲噪声、椒盐噪声的同时能保留图像的边缘细节。这些优良特性是线性滤波所不具备的。

中值滤波首先也得生成一个滤波模板，将该模板内的各像素值进行排序，生成单调上升或单调下降的二维数据序列，二维中值滤波输出为

$$\text{mid}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

10	9	4	7	2
4	31	12	23	30
5	43	1	32	8
54	7	3	43	4
23	76	56	45	9

10	9	4	7	2
4	31	12	23	30
5	43	23	32	8
54	7	3	43	4
23	76	56	45	9

```
img_b = cv2.medianBlur(img_n, 3)
cv2.imshow('b', img_b)
key = cv2.waitKey()
if key == 27:
    cv2.destroyAllWindows()
```

4.4.2 双边滤波

双边滤波器的好处是可以做边缘保存（edge preserving），一般过去用的维纳滤波或者高斯滤波去降噪，都会较明显地模糊边缘，对于高频细节的保护效果并不明显。双边滤波器顾名思义比高斯滤波多了一个高斯方差sigma - d，它是基于空间分布的高斯滤波函数，所以在边缘附近，离的较远的像素不会太多影响到边缘上的像素值，这样就保证了边缘附近像素值的保存。

在双边滤波器中，输出像素的值依赖于邻域像素值的加权值组合：

$$g(i,j)=\frac{\sum_{k,l} f(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

而加权系数w(i,j,k,l)取决于定义域核和值域核的乘积。

其中定义域核表示如下（如图）：

$$d(i,j,k,l)=\exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}\right)$$

值域核表示为：

$$r(i,j,k,l)=\exp\left(-\frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}\right)$$

两者相乘后，就会产生依赖于数据的双边滤波权重函数：

$$w(i,j,k,l)=\exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}-\frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}\right)$$

```
img_b = cv2.bilateralFilter(img ,9,75,75)
cv2.imshow('b', img_b)
key = cv2.waitKey()
if key == 27:
    cv2.destroyAllWindows()
```

参考资料：https://github.com/datawhalechina/team-learning-cv/blob/master/ImageProcessingFundamentals/04%20%E5%9B%BE%E5%83%8F%E6%BB%A4%E6%B3%A2.md