

Canny Detect Projecr

朱启鹏 58119304

1. 实验内容

1.1 实验目标

- 实现对彩色图像的灰度处理，并使用高斯一阶导数滤波器计算图像梯度，进而执行非极大值抑制和阈值操作及连接，从而进行canny边缘检测。
- 具体实现任务：
 - 原图按灰度通道读取
 - 实现图像卷积函数
 - 高斯一阶偏导滤波实现
 - 非极大值抑制
 - 双边阈值

1.2 实验数据集

- 实验用图：
 - 数字图像处理经典图片 Lena 等。

1.3 编译环境

- python 3.7

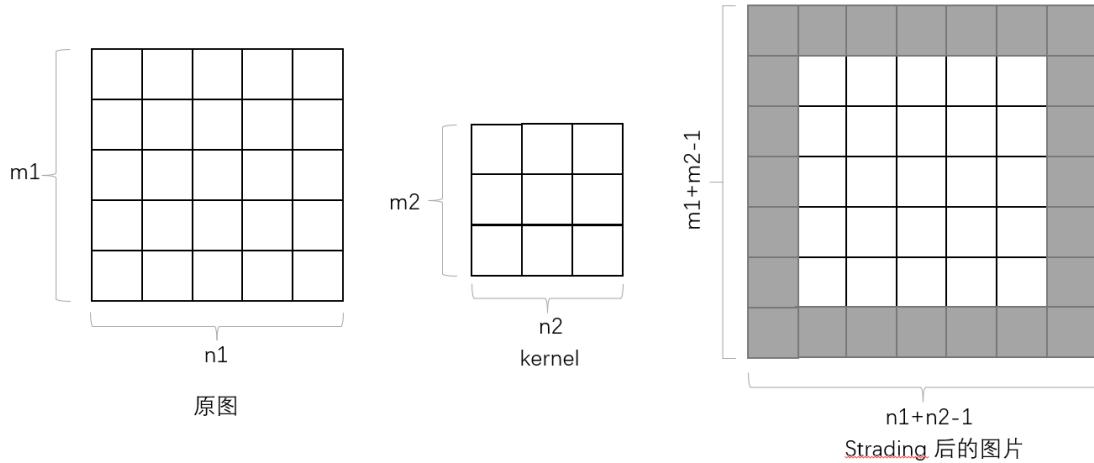
2 实验思路

2.1 二维图像卷积

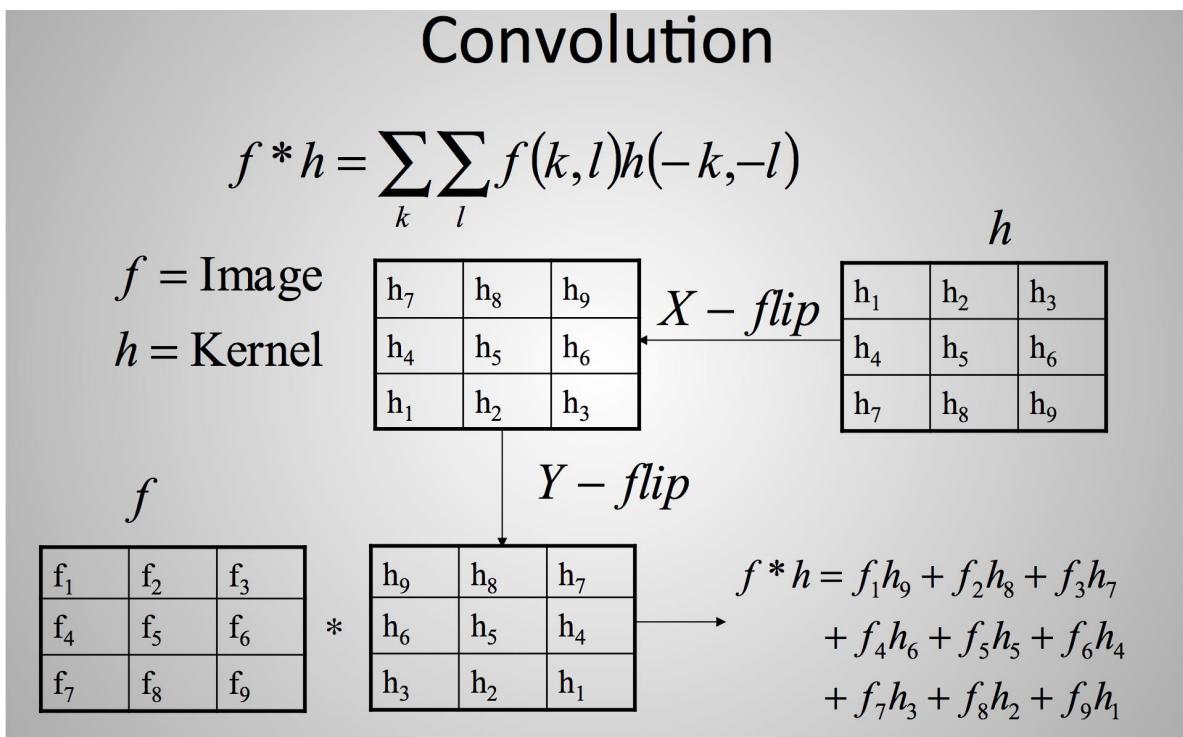
- 公式：

$$f[n, m] * h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l]h[n - k, m - l] \quad (1)$$

- Strading:(灰色部分为补零位置，白色部分为原图)



- Convolution:



- 让Kernel的中心遍历图像的每一个像素。

2.2 高斯一阶偏导滤波

2.2.1 Gaussian Kernel 实现

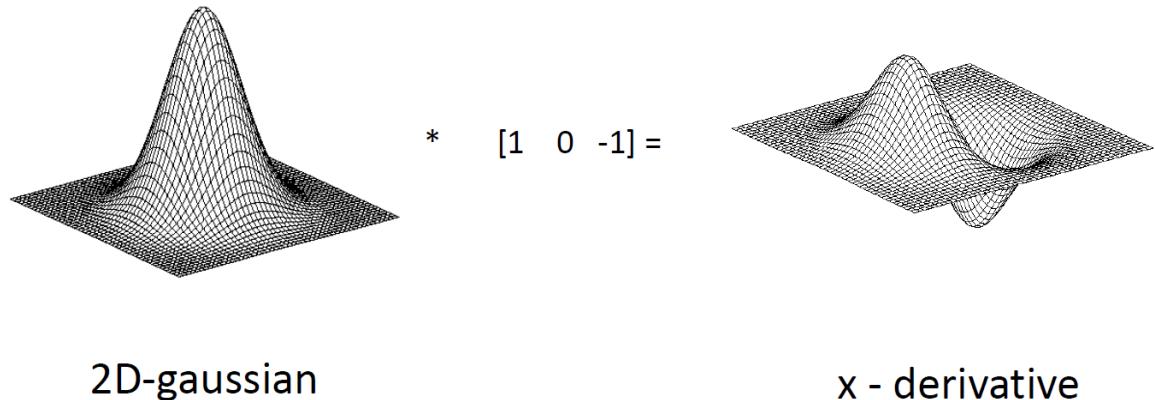
- 以核的中心为原点建立直角坐标系，利用二维高斯函数，计算图片上每一个像素值。
- 2-dim Gaussian Function:

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (2)$$

- 对kernel进行归一化处理

2.2.2 Gaussian Derive Kernel 实现

- 对得到的Gaussian Kernel分别在x方向和y方向求偏导
- 具体而言，即分别使用 $[1, 0, -1]$ 与 $[1, 0, -1]^t$ 分别高斯核进行卷积处理，从而分别得到x方向的高斯偏导核 G_x 以及y方向的高斯偏导核 G_y



- 分别使用 G_x 以及 G_y 对原图进行卷积，从而分别得到x方向核y方向的高斯偏导滤波图 img_Gx 和 img_Gy
- 特别的，处理完的图片需要将小于零的像素归为0，大于255的像素归为255，并将像素值转化为整型
- 接着再利用模值公式以及方向公式分别计算出 img_G 和 dir

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \quad (3)$$

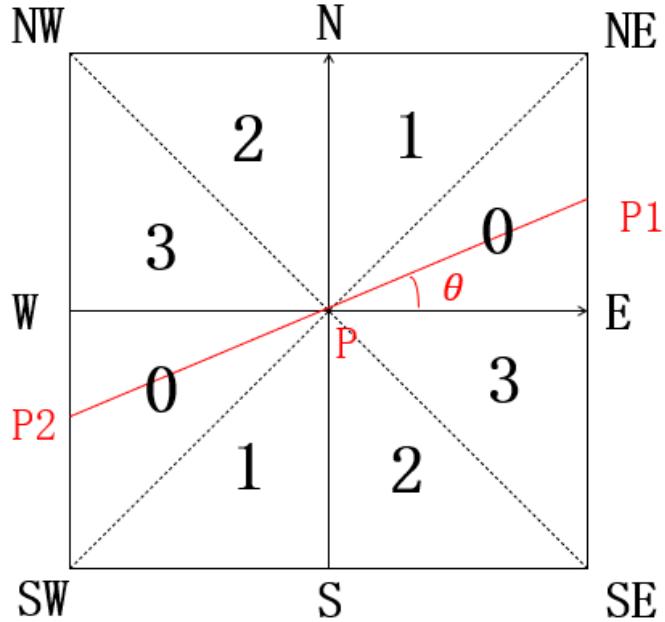
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2} \quad (4)$$

$$\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \quad (5)$$

- 其中，将模值作为此次滤波的结果图片

2.3 非极大值抑制

- 非极大值抑制是一种边缘稀疏技术，非极大值抑制的作用在于“瘦”边。对图像进行梯度计算后，仅基于梯度值提取的边缘仍然很模糊。对于标准3，对边缘有且应当只有一个准确的响应。而非极大值抑制则可以帮助将局部最大值之外的所有梯度值抑制为0，对梯度图像中每个像素进行非极大值抑制的算法是：
 - 将当前像素的梯度强度与沿正负梯度方向上的两个像素进行比较。
 - 如果当前像素的梯度强度与另外两个像素相比最大，则该像素点保留为边缘点，否则该像素点将被抑制。
- 通常为了更加精确的计算，在跨越梯度方向的两个相邻像素之间使用线性插值来得到要比较的像素梯度，现举例如下：



- 如图所示，将梯度分为8个方向，分别为E、NE、N、NW、W、SW、S、SE，其中0代表 $0^\circ \sim 45^\circ$ 或者 $-180^\circ \sim -135^\circ$, 1代表 $45^\circ \sim 90^\circ$ 或者 $-135^\circ \sim -90^\circ$, 2代表 $90^\circ \sim 135^\circ$ 或者 $-90^\circ \sim -45^\circ$, 3代表 $135^\circ \sim 180^\circ$ 或者 $45^\circ \sim 0^\circ$ 。如图，像素点P的梯度方向为 θ ，则像素点P1和P2的梯度线性插值为：

- 0区：

- 特别的，当 $\theta < 0$ 时， $\theta := \theta + \pi$

$$G_{p1} = (1 - \tan(\theta)) \times E + \tan(\theta) \times NE \quad (6)$$

$$G_{p2} = (1 - \tan(\theta)) \times W + \tan(\theta) \times SW \quad (7)$$

- 1区：

- 特别的，当 $\theta < 0$ 时， $\theta := -\theta - \frac{\pi}{2}$ ；当 $\theta \geq 0$ 时， $\theta := -\theta + \frac{\pi}{2}$ ；

$$G_{p1} = (1 - \tan(\theta)) \times N + \tan(\theta) \times NE \quad (8)$$

$$G_{p2} = (1 - \tan(\theta)) \times S + \tan(\theta) \times SW \quad (9)$$

- 2区：

- 特别的，当 $\theta < 0$ 时， $\theta := \theta + \frac{\pi}{2}$ ；当 $\theta \geq 0$ 时， $\theta := \theta - \frac{\pi}{2}$ ；

$$G_{p1} = (1 - \tan(\theta)) \times N + \tan(\theta) \times NW \quad (10)$$

$$G_{p2} = (1 - \tan(\theta)) \times S + \tan(\theta) \times SE \quad (11)$$

- 3区：

- 特别的，当 $\theta < 0$ 时， $\theta := -\theta + \pi$ ；当 $\theta \geq 0$ 时， $\theta := -\theta$ ；

$$G_{p1} = (1 - \tan(\theta)) \times W + \tan(\theta) \times NW \quad (12)$$

$$G_{p2} = (1 - \tan(\theta)) \times E + \tan(\theta) \times SE \quad (13)$$

- 因此非极大值抑制的伪代码描写如下：

$if \ G_p \geq G_{p1} \ and \ G_p \geq G_{p2}$
 G_p may be an edge
else
 G_p should be suppressed

2.4 双阈值检测

- 在施加非极大值抑制之后，剩余的像素可以更准确地表示图像中的实际边缘。然而，仍然存在由于噪声和颜色变化引起的一些边缘像素。为了解决这些杂散响应，必须用弱梯度值过滤边缘像素，并保留具有高梯度值的边缘像素，可以通过选择高低阈值来实现。如果边缘像素的梯度值高于高阈值，则将其标记为强边缘像素；如果边缘像素的梯度值小于高阈值并且大于低阈值，则将其标记为弱边缘像素；如果边缘像素的梯度值小于低阈值，则会被抑制。阈值的选择取决于给定输入图像的内容。
- 双阈值检测的伪代码描写如下：

$if \ G_p \geq HighThreshold$
 G_p is an strong edge
else if $G_p \geq LowThreshold$
 G_p is an weak edge
else
 G_p should be suppressed

2.5 抑制孤立低阈值点

- 到目前为止，被划分为强边缘的像素点已经被确定为边缘，因为它们是从图像中的真实边缘中提取出来的。然而，对于弱边缘像素，将会有一些争论，因为这些像素可以从真实边缘提取也可以是因噪声或颜色变化引起的。为了获得准确的结果，应该抑制由后者引起的弱边缘。通常，由真实边缘引起的弱边缘像素将连接到强边缘像素，而噪声响应未连接。为了跟踪边缘连接，通过查看弱边缘像素及其8个邻域像素，只要其中一个为强边缘像素，则该弱边缘点就可以保留为真实的边缘。
- 特别的，当出现弱边缘像素周围全为弱边缘像素，则无法判定，则先把该像素放入队列中
- 每次迭代结束，都判断一下是否有新的弱边缘变为强边缘，如果没有则跳出循环；否则，则将队列1赋值为队列2
- 抑制孤立边缘点的伪代码描述如下：

```

while(True):  

    Gp = queue_weakedge_1.pop()  

    flag = 0  

    initialize : queue_weakedge_2 ←  

    if Gp connected to a strong edge pixel  

        Gp is an strong edge  

        flag = 1  

    elif Gp only connected to weak edge pixels  

        Gp should be sup pressed  

    else  

        queue_weakedge_2.push(Gp)  

    if flag == 0 or queue_weakedge_2 is empty  

        break  

    else  

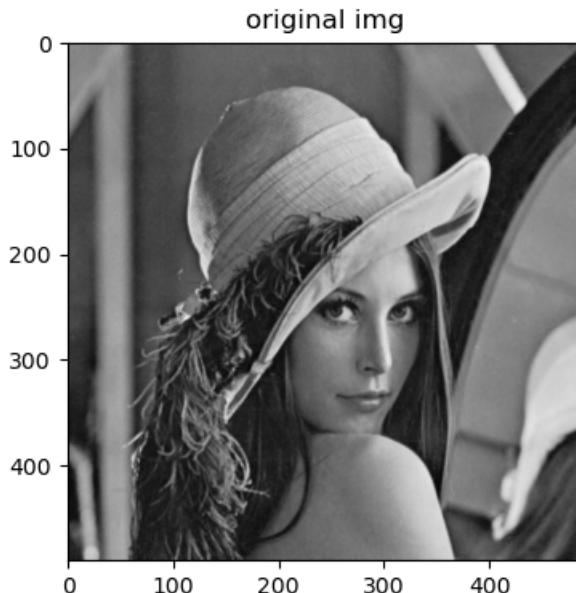
        queue_weakedge_1 = queue_weakedge_2

```

3. 实验过程

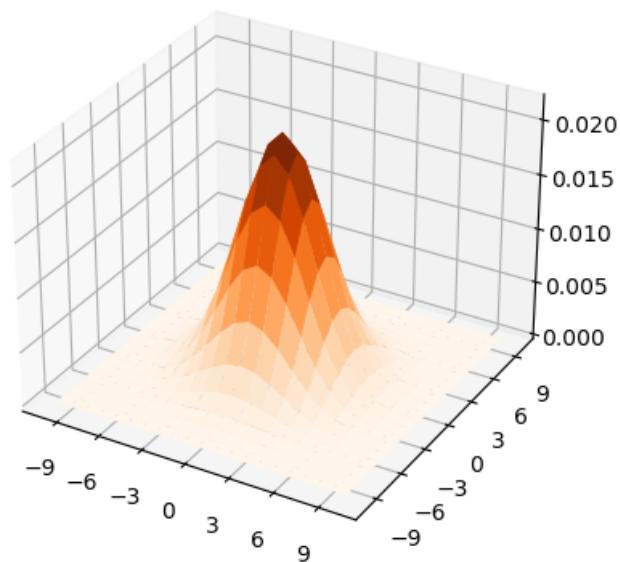
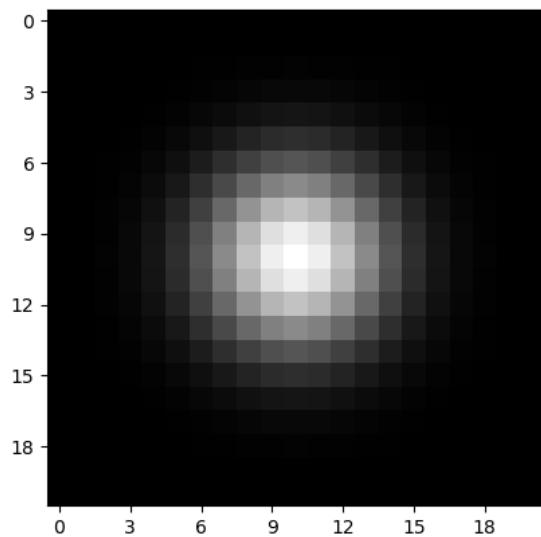
3.1 原图按灰度通道读取

- 本次读取使用 `cv2.imread` 函数读取图片，并用 `plt.show()` 展示，效果如下：



3.2 高斯一阶偏导滤波

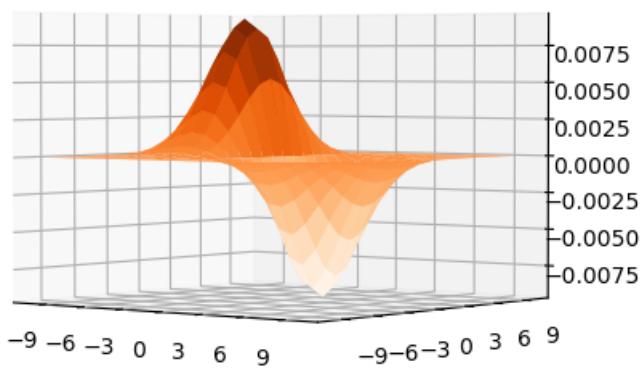
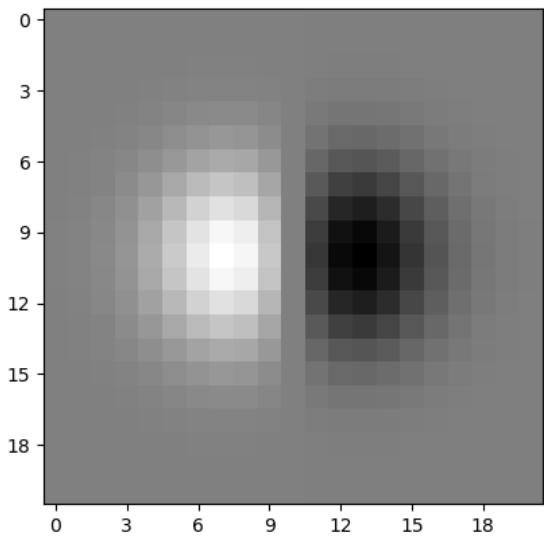
3.2.1 高斯核生成



- 我们可以发现该高斯核呈现类似高斯函数的分布，中间数值高，周围数值低，所以实验成功。

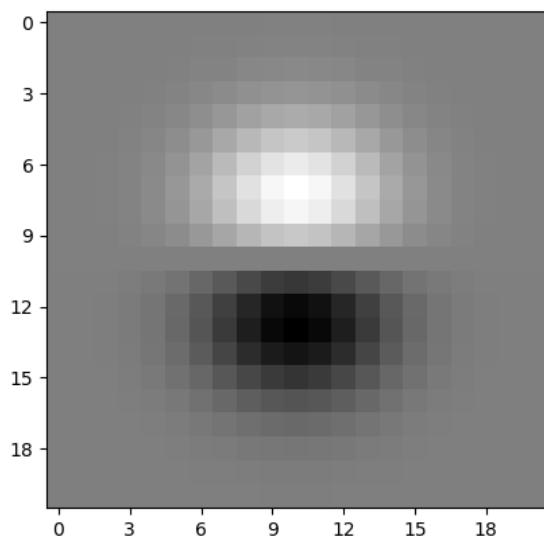
3.2.2 高斯偏导核生成

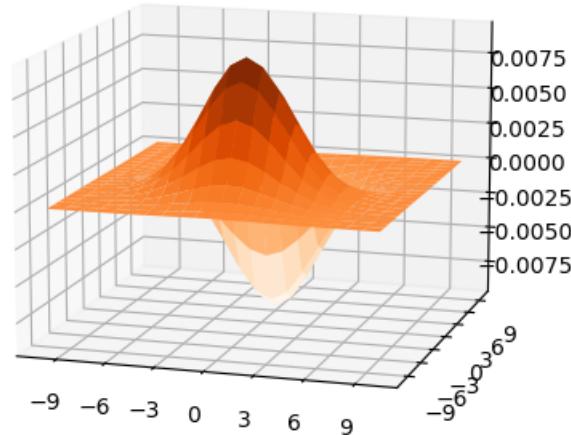
3.2.2.1 x方向



- 我们可以发现该高斯核在x方向呈现类似高斯x方向偏导函数的分布，所以实验成功。

3.2.2.2 y方向



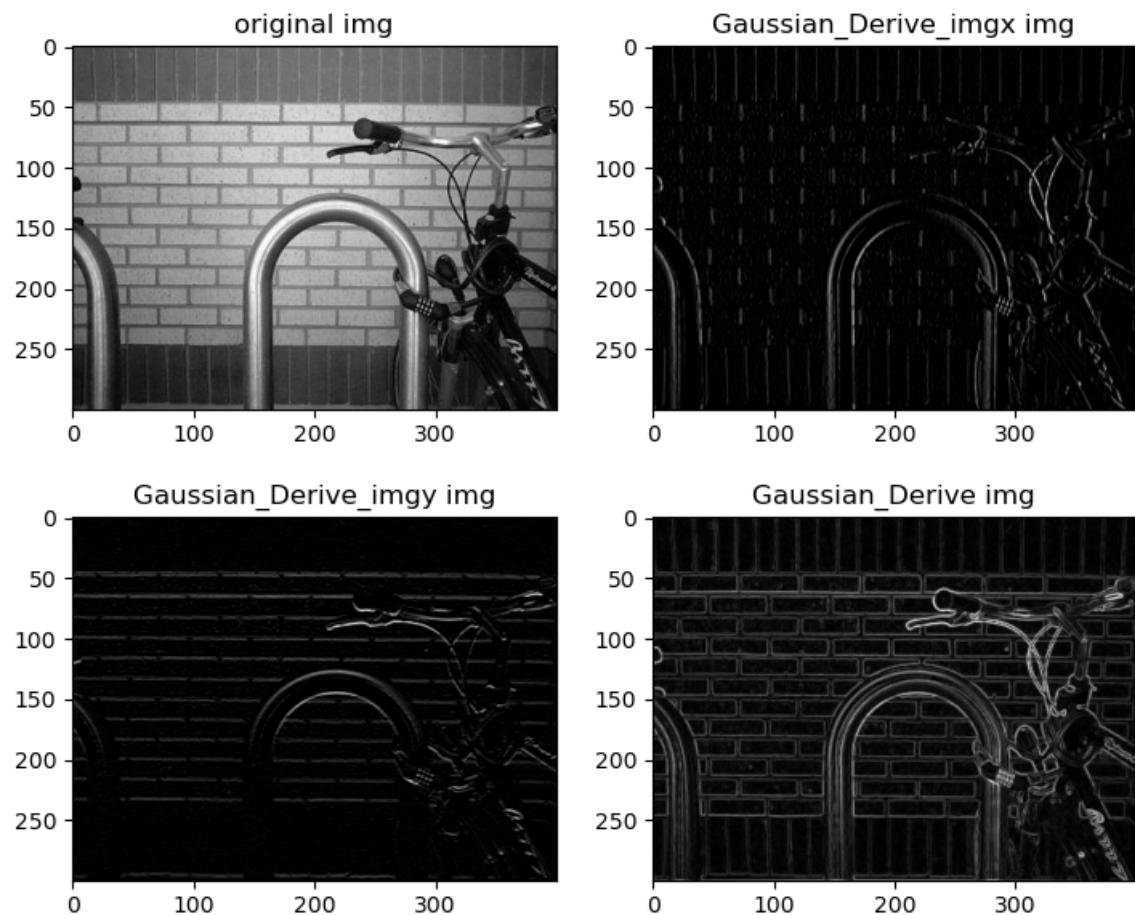


- 我们可以发现该高斯核在y方向呈现类似高斯y方向偏导函数的分布，所以实验成功。

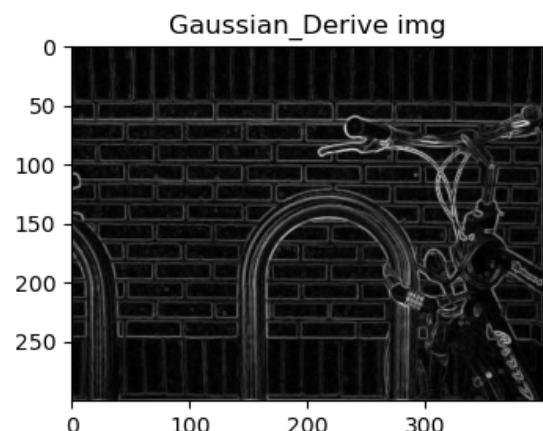
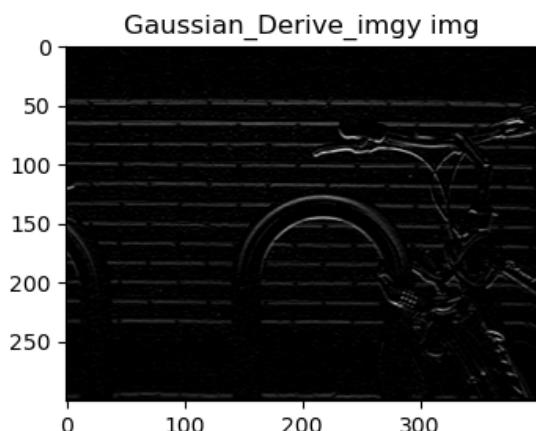
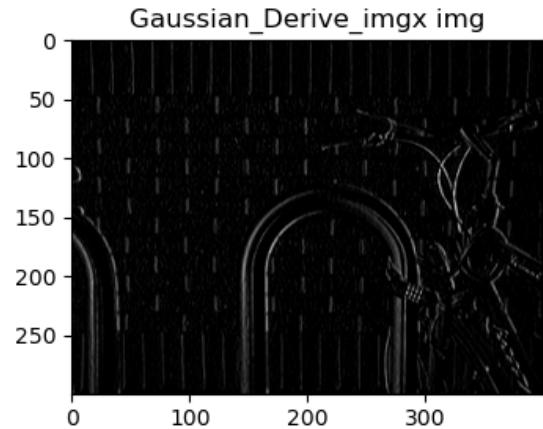
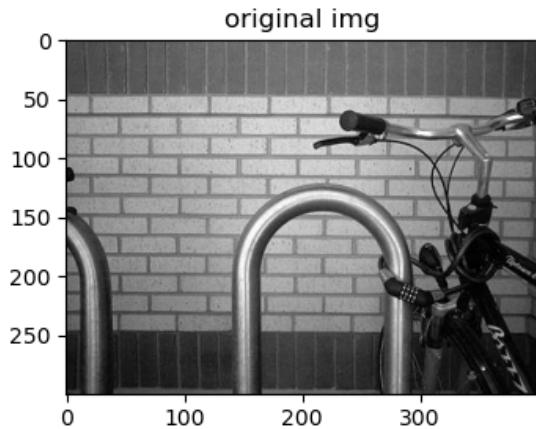
3.2.2.3 滤波

- 我们知道， σ 越小对越能提取边缘信息，但是有可能把不必要的噪音也保留； σ 越大，虽然越容易消除噪音的影响，但是可能造成边缘确实的影响，所以需要合适的 σ 才能保证滤波成功。
- 而一般而言，kernel的大小取 3σ 的最靠近奇数。
- 下图利用自行车的图片进行实验，便于寻找合适的 σ 和kernel大小

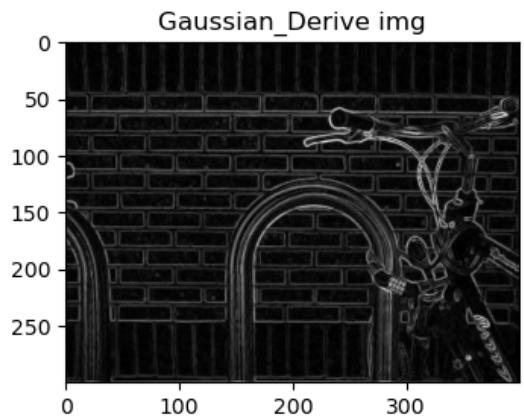
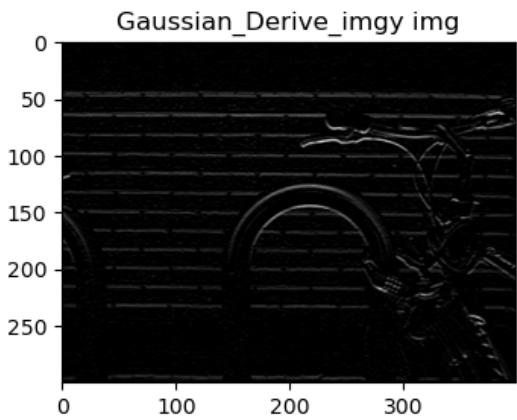
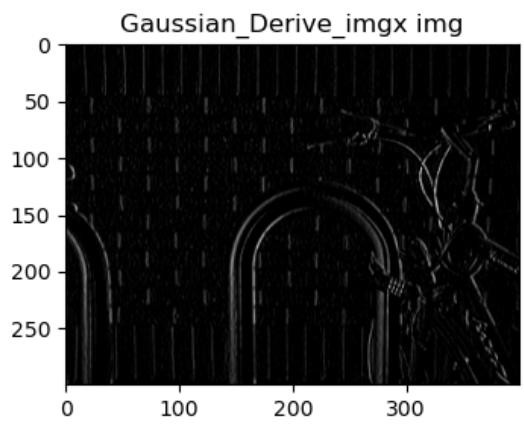
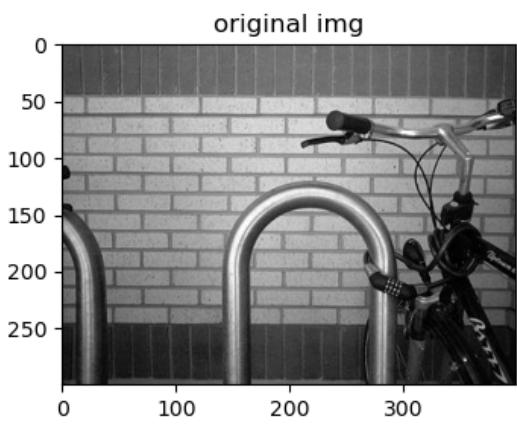
`sigma = 1, kernelsize = 3`



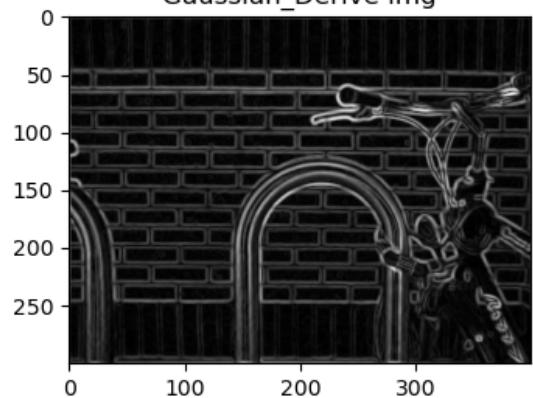
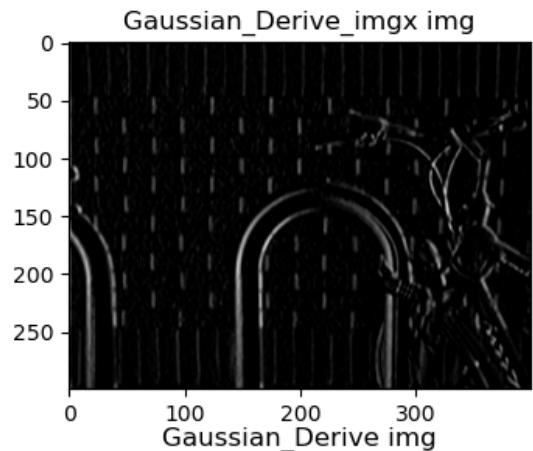
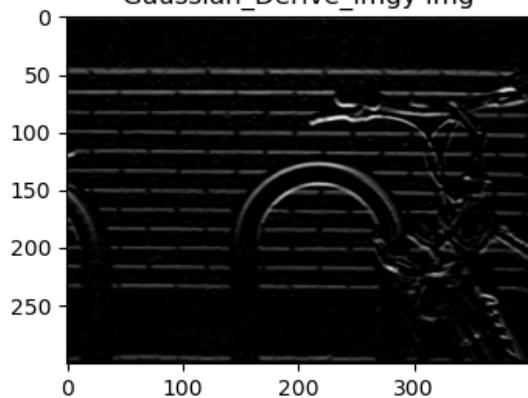
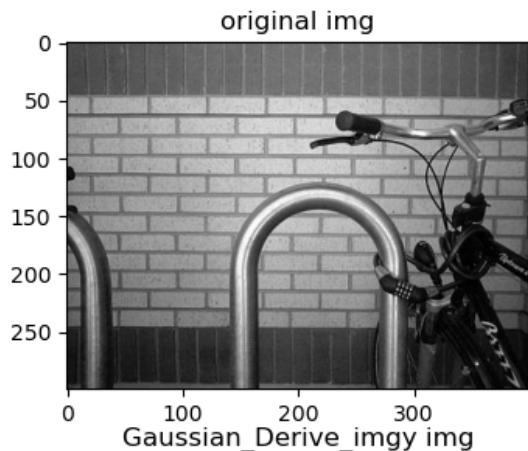
sigma = 1.3, kernelsize = 3



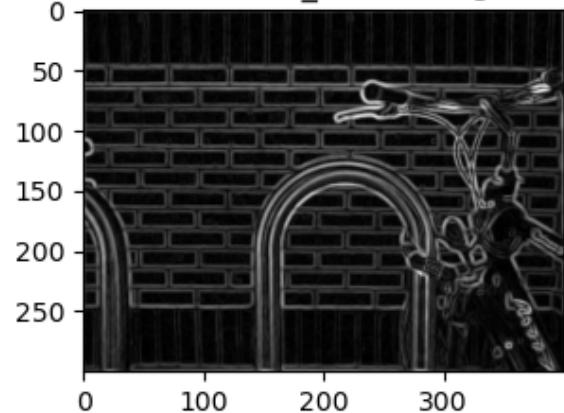
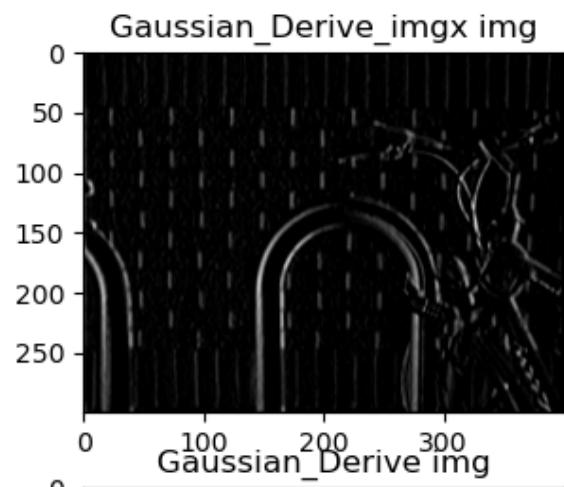
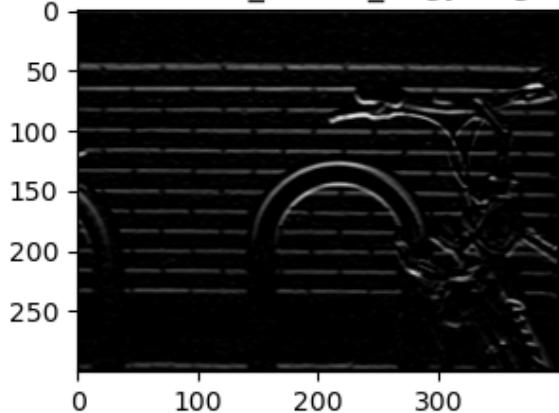
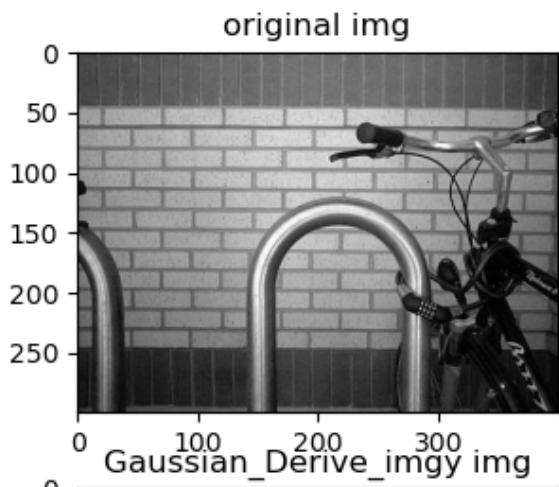
sigma = 2.1, kernelsize = 3



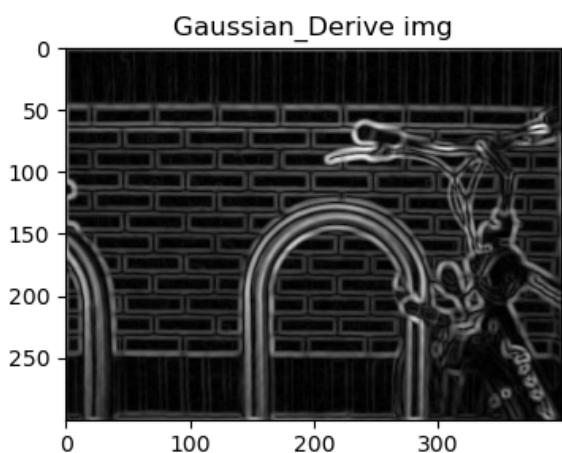
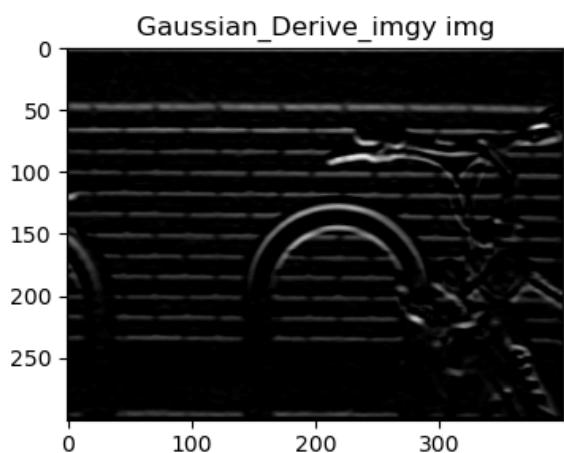
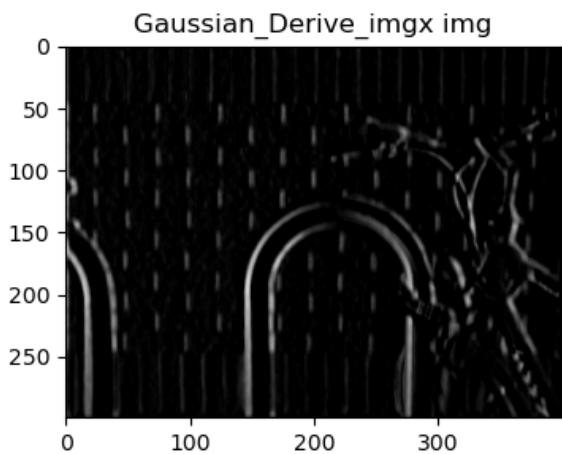
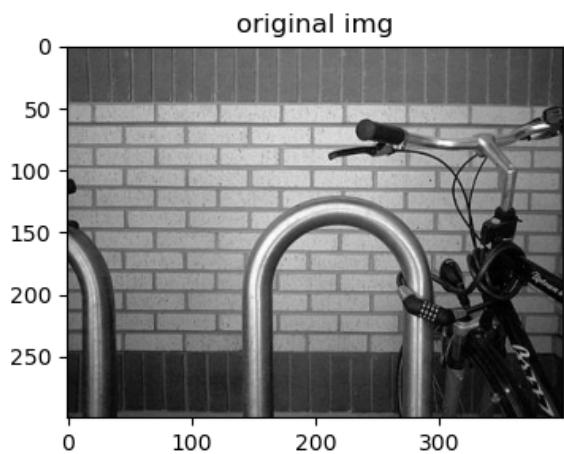
$\text{sigma} = 1.7$, $\text{kernelsize} = 5$



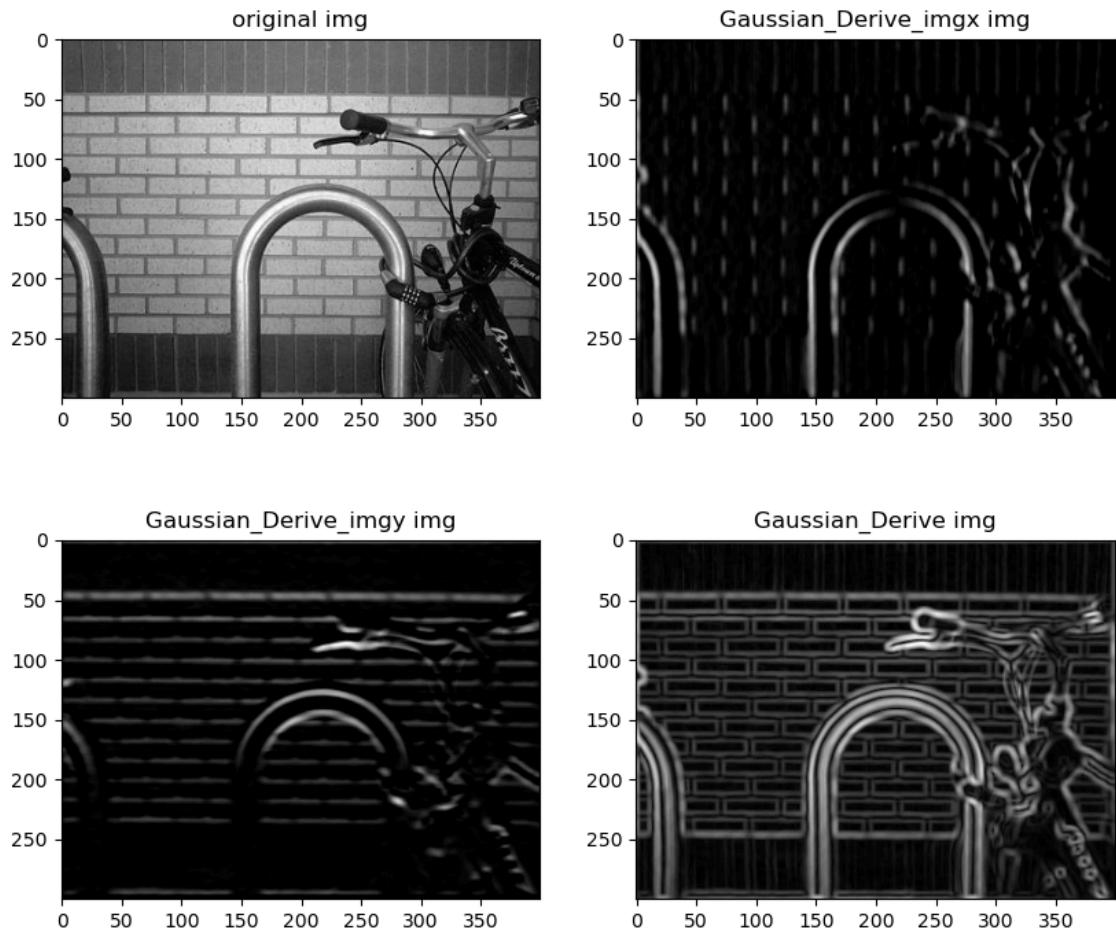
$\text{sigma} = 2.1$, $\text{kernelsize} = 5$



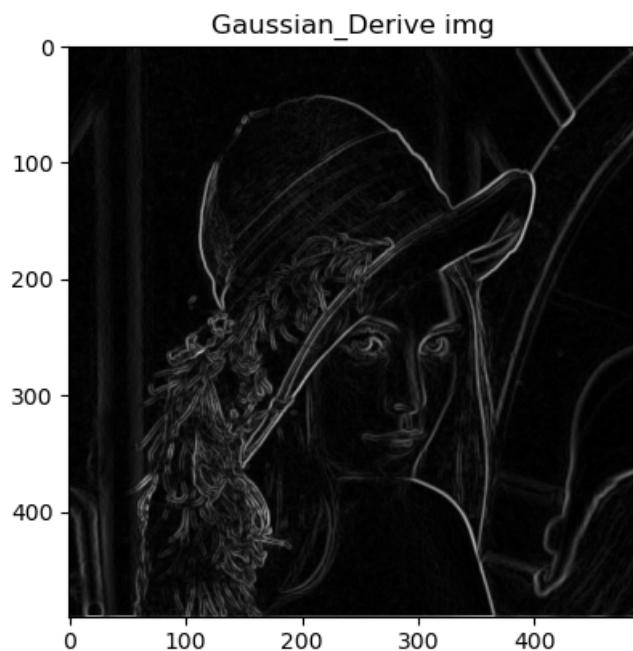
sigma = 2.5, kernelsize = 7



sigma = 3, kernelsize = 9

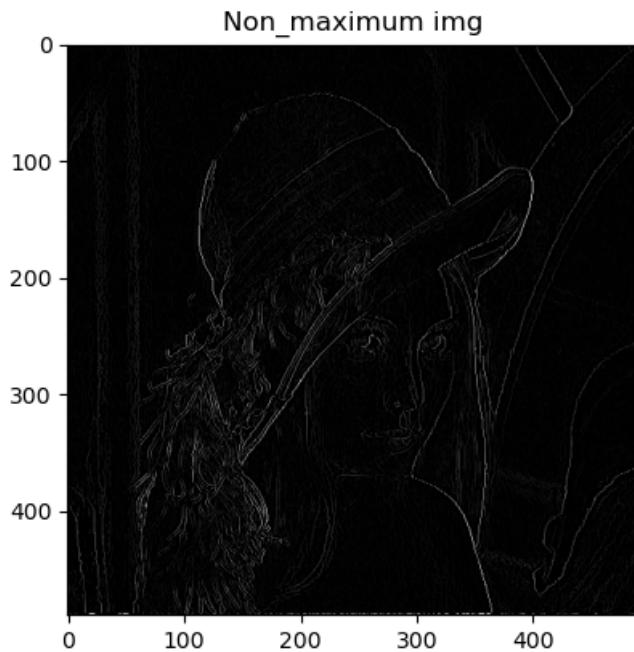


- 我们可以发现当 σ 取到3时，边缘已经过粗，并没有很好的提取出边缘
- 从实验结果看， σ 取1~1.7就可以很好的完成任务
- 所以之后实验取 $\sigma = 1.3, kernel_size = 3$
- 得到下图：



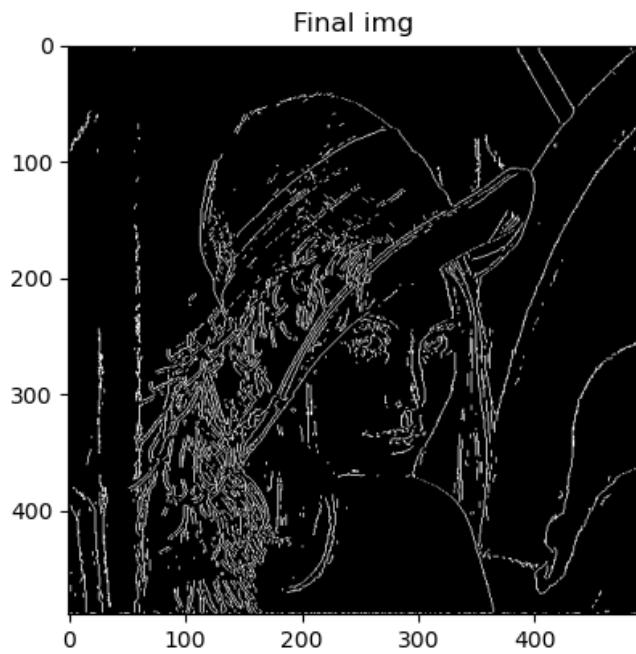
3.3 非极大值抑制

- 为解决边缘过粗问题，接着进行非极大值抑制。

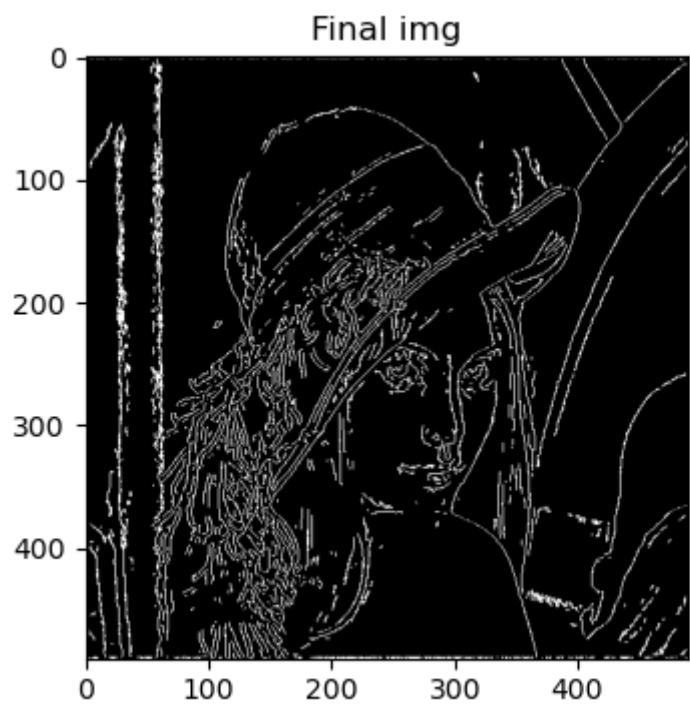
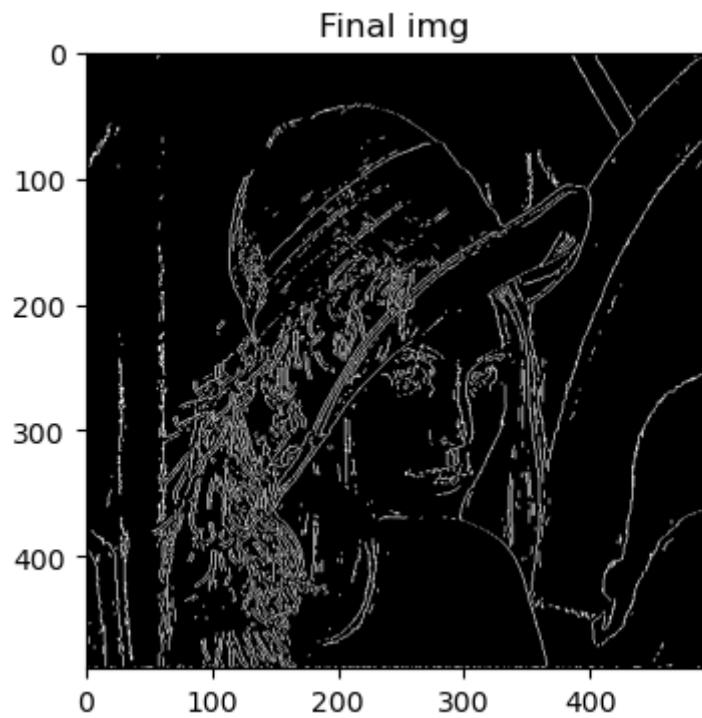


- 我们可以发现，边缘确实细化很多，但好像有点看不清，这是因为图片的对比度不够高，这在之后转换为二值图像就可以解决。

3.4 双边阈值



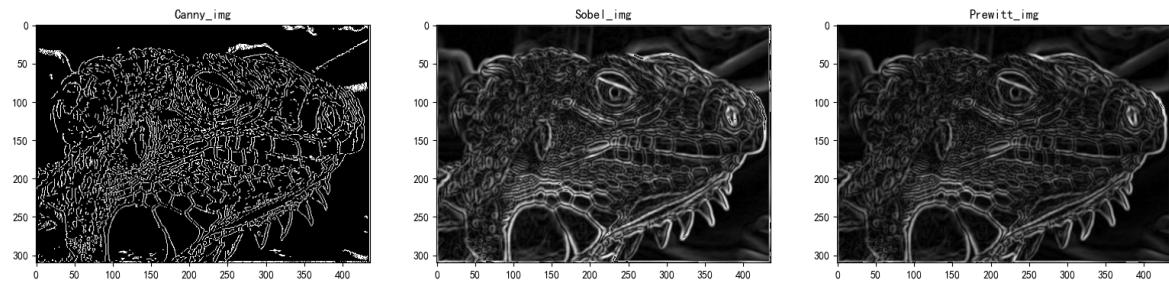
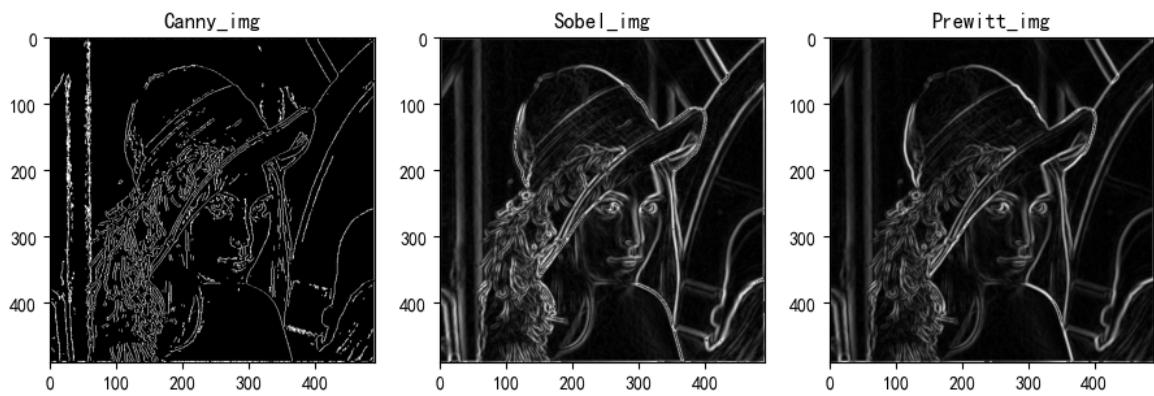
- 进行双边阈值后，我们可以明显发现，图像效果好了很多，并且没有出现边缘过粗的情况。
- 另外我还分别实验了 $\sigma = 1$, $kernel_size = 3$ 与 $\sigma = 1.7$, $kernel_size = 5$



- 我们可以发现 $\sigma = 1.7$, $kernel_size = 5$ 的效果最好。

4. 实验对比

- 与课上提过的Sobel算子, Prewitt算子与本次实验的Sobel算子进行对比



- 我们可以发现，由于没有边缘细化以及双边阈值，另外梁总算子提取的边缘都出现了过粗的现象，其中对于sobel算子面部等受到光照影响的非边缘部分也被保留，而Prewitt算子则出现了一些部位较为模糊的现象。

- 一些其他图片：

