数字图像处理-作业 3

程铭 517021910750

1 canny 算子的计算步骤

Canny 算子的计算步骤如下：

1） 使用高斯平滑，抑制噪声；

2） 计算梯度的幅值和方向，达到边缘增强；

3） 非极大值抑制，排除非边缘像素，仅仅保留一些细线条；

4） 选取双阈值，高阈值的作用是判定该点属于边缘还是背景，低阈值的作用是连接边缘、进行平滑处理。

2 不同高斯函数的标准差选取时的结果

选取不同大小的高斯卷积模板（此处选取的值为 3，5，7，9），利用下述公式计算得到高斯函数的标准差的值为：

$$\sigma = 0.3 * [(kernel - 1) * 0.5 - 1] + 0.8$$

得到对应的标准差参数值分别为 0.8，1.1，1.4，1.7。

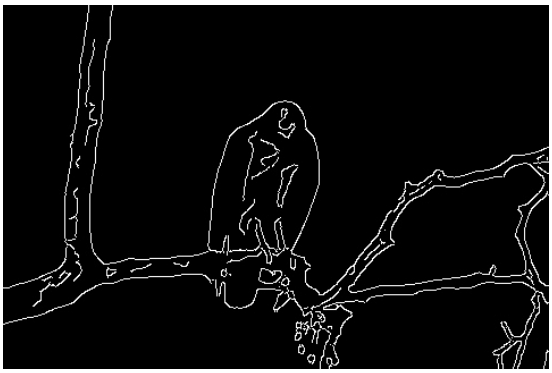
选取上述标准差参数进行处理的结果如下图所示：

简单图像：

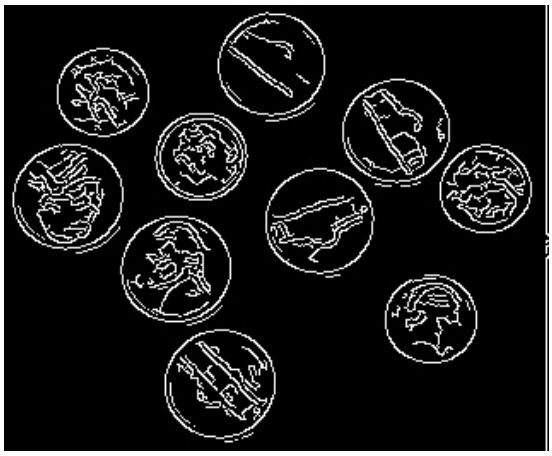
高阈值为 100，标准差为 0.8

高阈值为 100，标准差为 1.1

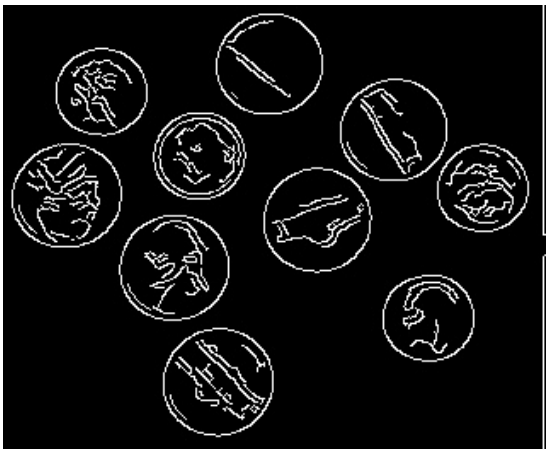
高阈值为 100，标准差为 1.4



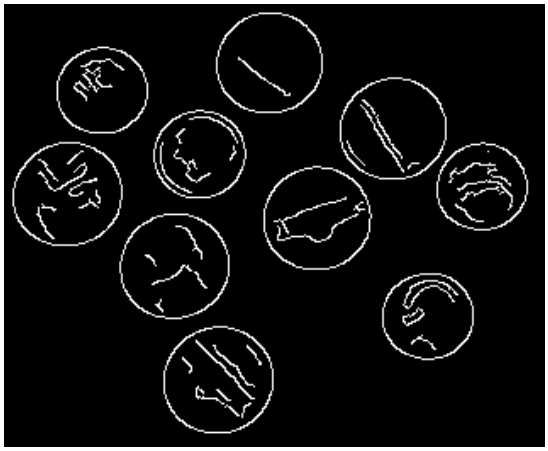
高阈值为 100，标准差为 1.7



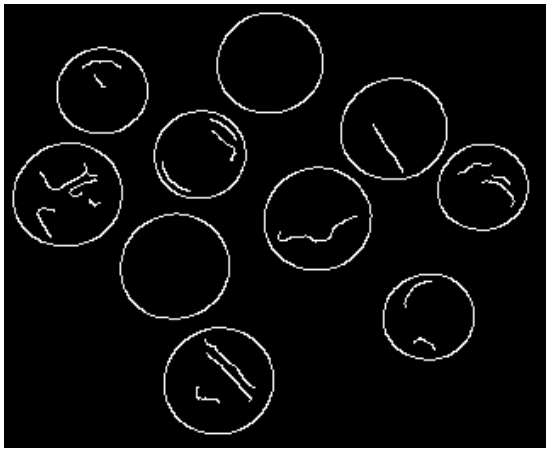
高阈值为 100，标准差为 0.8



高阈值为 100，标准差为 1.1



高阈值为 100，标准差为 1.4



高阈值为 100，标准差为 1.7

复杂图像：

高阈值为 100，标准差为 0.8

高阈值为 100，标准差为 1.1

高阈值为 100，标准差为 1.4

高阈值为 100，标准差为 1.7

无噪声图像：

高阈值为 100，标准差为 0.8

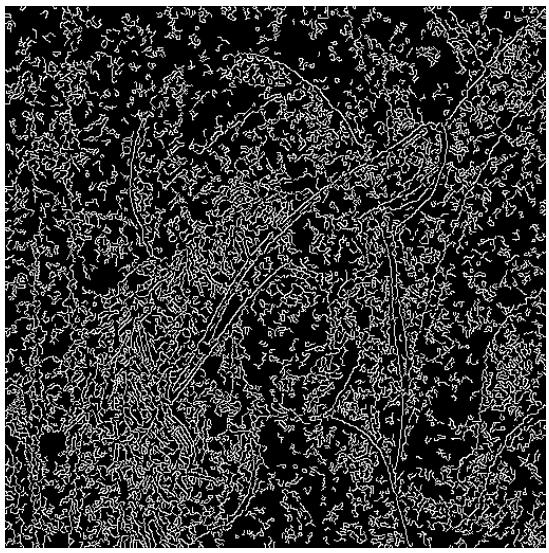
高阈值为 100，标准差为 1.1

高阈值为 100，标准差为 1.4

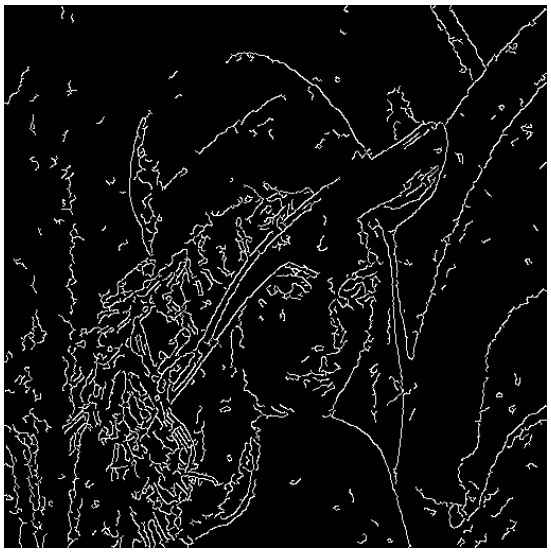
高阈值为 100，标准差为 1.7

高斯噪声图像：

高阈值为 100，标准差为 0.8



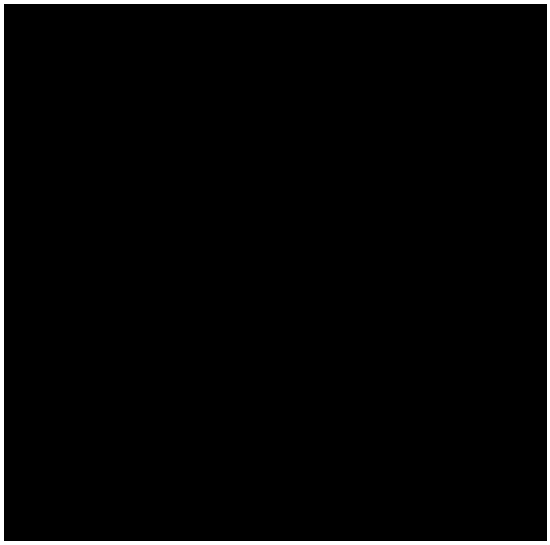
高阈值为 100，标准差为 1.1



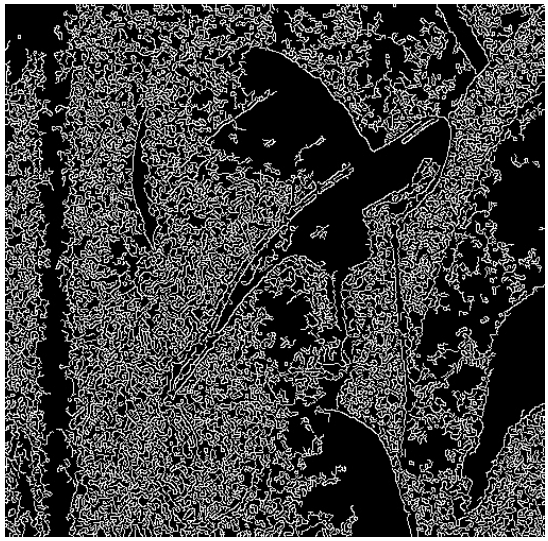
高阈值为 100，标准差为 1.4

高阈值为 100，标准差为 1.7

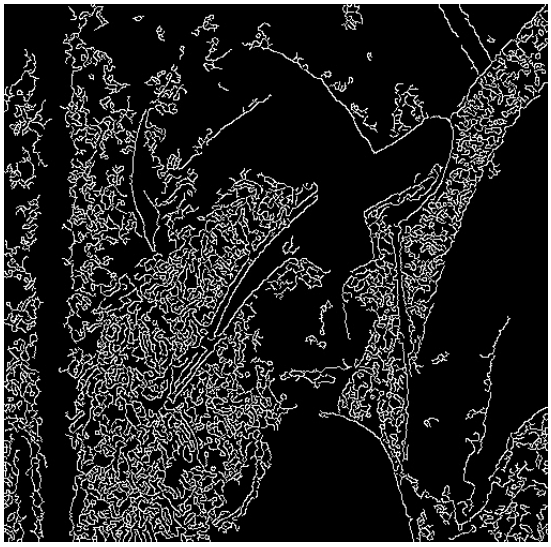
椒盐噪声图像：



高阈值为 150，标准差为 0.8



高阈值为 150，标准差为 1.1



高阈值为 150，标准差为 1.4

高阈值为 150，标准差为 1.7

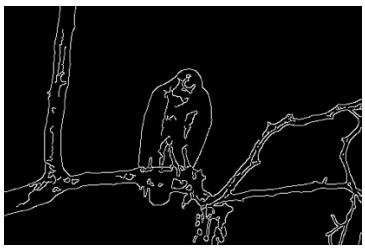
从上图对比可以看到，对于无噪声图像（简单图像、复杂图像、无噪声图像）而言，边缘检测的结果都较好，选取合适的σ值可以得到很好的检测效果。随着σ增加，图像的灰色区域变多，白色区域变少，所检测出来的边缘变少，模糊变多。这是因为 canny 算子中的第一步为进行高斯平滑，σ越大，高斯函数中心的权重越小，周围权重越大，因此周围点对中心点的影响效果越大，即中心点趋于与周围的其他像素点的灰度值相同，导致图像检测出的边缘细节少、较模糊。较小的σ值可以检测到更加锐利、细节的边缘。

对于有噪声的图像（高斯噪声、椒盐噪声）而言，当σ较小时图像黑色部分较多甚至全黑（当σ和高阈值参数都取值较小时），随着σ的增加，目标物的边缘逐渐显示。这是因为 canny 算子中的高斯平滑操作会抑制噪声，随着σ取值的增加，抑制噪声的程度加强，图像的细节得以显示。另外，对比高斯噪声图像和椒盐噪声图像可以发现，相比于椒盐噪声的图像，canny 算子对包含高斯噪声的图像的处理结果更好，这是因为其高斯平滑操作更有利于去除高斯噪声。另一方面，与无噪声图像中的分析同理，当σ取值更大时，会导致图像细节丢失。

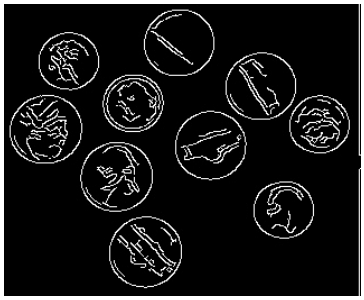
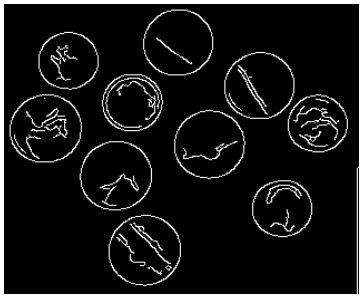
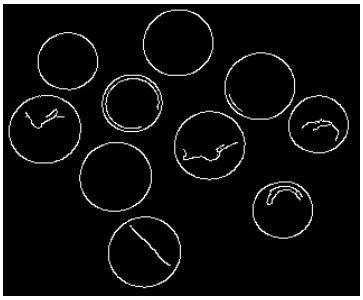
3 不同高阈值选择时的结果

选取不同的高阈值参数（此作业中选取的值为 100，150，200），对简单图像、复杂图像、无噪声图像、高斯噪声图像、椒盐噪声图像进行处理，结果如下所示：

简单图像：


高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1

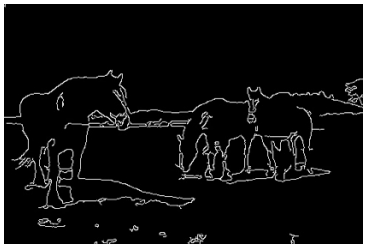
高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1

复杂图像：



高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1





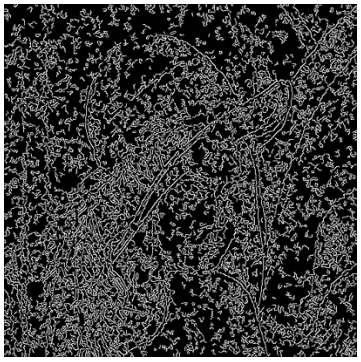
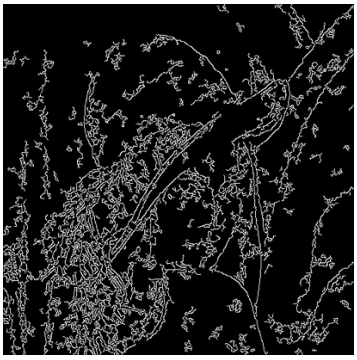
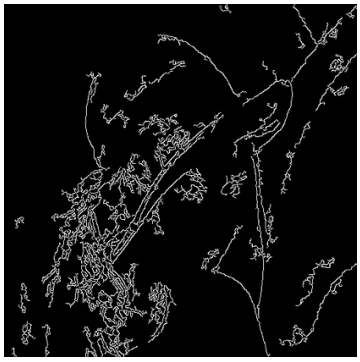
高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1

无噪声图像：



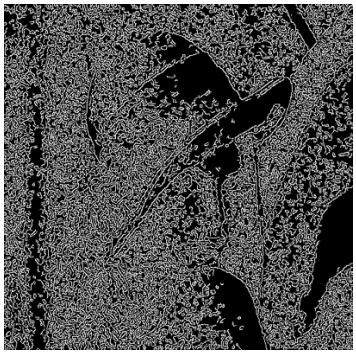
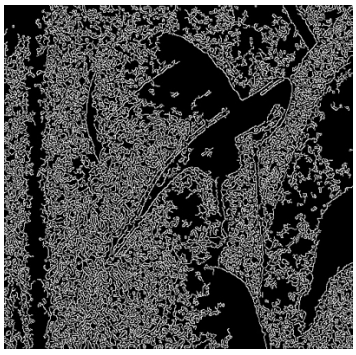
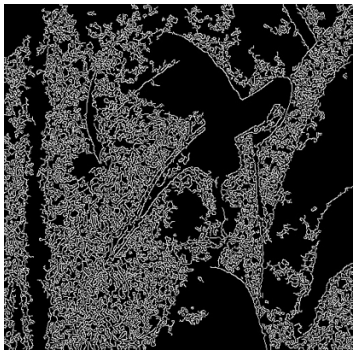
高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1

高斯噪声图像：

高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1

椒盐噪声图像：

高阈值为 100，σ为 1.1　　高阈值为 150，σ为 1.1　　高阈值为 200，σ为 1.1

Canny 算子中的其中一个步骤为：设置双阈值进行图像边缘的检测和连接，具体步骤为：设置两个阈值 low 和 high，若该点的幅值小于 low，说明不是边缘点，因此置为黑色；若该点的幅值大于 high，说明是边缘点，置为白色；若位于 low 和 high 之间，若该点与幅值大于 high 的点相邻或者间接相邻，则认为属于边缘，置为白色，反之置为黑色。因此可以得出，这两个参数定义了三段区间，来区分是边缘还是背景。

具体而言，高阈值的作用是将要提取出的轮廓和背景分离开来，即类似于阈值分割算法中阈值的作用；低阈值的作用是平滑边缘，有时高阈值设置过大，会导致边缘不连续、不平滑，因此通过低阈值来平滑轮廓线、连接边缘。因此，如果设置的阈值较高，说明过滤区间位于灰度值较高的位置，因此会将边缘判别为是背景，导致部分边缘无法检测出来。

经过在无噪声图像(简单图像、复杂图像、无噪声图像)中的对比，可以发现，固定低阈值，随着高阈值的取值增加，图像判别出来的边缘变少，从而导致图像边缘细节减少。

对于含有噪声的图像（高斯噪声、椒盐噪声），对比结果图可以发现，固定低阈值，随着高阈值的取值增加，图像的噪声减少，可以更清晰地看到边缘。这是因为随着高阈值的增加，算法将噪声点判别为是背景，置为黑色;而高阈值选取较低时，算法会认为噪声点也是边缘，因此去噪的效果不好。然而，高阈值也不能选取过高，否则会导致图像边缘的细节丢失过多（原因同上）。

对比高斯噪声和椒盐噪声的图像，可以发现对于含有高斯噪声图像的处理效果好于椒盐噪声的图像。这是因为 canny 算法的步骤中包含高斯平滑操作，这一操作对于去除高斯噪声效果较好，因此可以更有效地滤除高斯噪声。而对于椒盐噪声的图像，尽管高阈值选取的较高，仍然有较多的噪声点存在。

```
import cv2
import numpy as np
import os

def getFname(path):
    img_name = []
    for _, _, fname in os.walk(path):
        for f in fname:
            str_ = path + f
            img_name.append(str_)
            print(img_name)

    return img_name

def canny(img_, sigma, kernel, high_thresh):
    img = cv2.imread(img_, cv2.IMREAD_GRAYSCALE)
    img_ = cv2.GaussianBlur(img, (kernel, kernel), sigma)
    img_canny = cv2.Canny(img_, 50, high_thresh)

    return img_canny

def canny_interface(kernel_list, high_thresh_list, path):
    for kernel in kernel_list:
        sigma = 0.3 * ((kernel - 1) * 0.5 - 1) + 0.8
        for high_thresh in high_thresh_list:
            for easy in easy_img:
                img_name = easy.split('/')[-1].split('.')[0]
                canny_ = canny(easy, sigma, kernel, high_thresh)
                binary_ = binary(canny_)
                cv2.imwrite('./easy/' + str(img_name) + '_sigma_' +
str(int(sigma * 10)) \
                    + '_highThresh_' + str(high_thresh) + '.jpg', binary_)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
            for complex_ in complex_img:
                img_name = complex_.split('/')[-1].split('.')[0]
                canny_ = canny(complex_, sigma, kernel, high_thresh)
                binary_ = binary(canny_)
                cv2.imwrite('./complex/' + str(img_name) + '_sigma_' +
str(int(sigma * 10)) \
                    + '_highThresh_' + str(high_thresh) + '.jpg', binary_)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
            for noise in noise_img:
                img_name = noise.split('/')[-1].split('.')[0]
                canny_ = canny(noise, sigma, kernel, high_thresh)
```

```
                binary_ = binary(canny_)
                cv2.imwrite('./noise/' + str(img_name) + '_sigma_' +
str(int(sigma * 10)) \
                    + '_highThresh_' + str(high_thresh) + '.jpg', binary_)
                cv2.waitKey(0)
                cv2.destroyAllWindows()


def binary(img):
    mean_ = np.mean(img)
    _, img = cv2.threshold(img, mean_ * 3, 255, cv2.THRESH_BINARY)
    return img

if __name__ == "__main__":
    easy_path = '../easy/'
    complex_path = '../complex/'
    noise_path = '../noise/'

    ## get list of image name
    easy_img = getFname(easy_path)
    complex_img = getFname(complex_path)
    noise_img = getFname(noise_path)

    kernel_list = [3, 5, 7, 9]
    high_thresh_list = [100, 150, 200]

    canny_interface(kernel_list, high_thresh_list, easy_img)
    canny_interface(kernel_list, high_thresh_list, complex_img)
    canny_interface(kernel_list, high_thresh_list, noise_img)
```