哈爾濱工業大學

视听觉与信号处理实验报告

实验 1

学　　院　计算机科学与技术学院

专　　业　视听觉信息处理

学　　号　１１７０３００５１

学　　生　易 亚 玲

任 课 教 师　姚 鸿 勋

哈尔滨工业大学计算机科学与技术学院

2019 年秋季

一、 实验内容（contents）

1. 实现给图像添加高斯噪声和椒盐噪声 ，要求显示添加噪声之后的结果图。
2. 实现图像的空域滤波：中值滤波和均值滤波算法，并选取适合的方法对 1 中的图像进行平滑处理，要求显示处理之后的结果。
3. 实现图像的边缘检测：Roberts 算子和 Sobel 算子。

选做：实现对 BMP 文件头的读取，并解析 BMP 图像文件。

PS：图像单通道形式读取

图 1

二、 实验目的（purposes）

1. 了解图像的文件结构和读写操作。
2. 掌握图像处理中常见的噪声来源。
3. 掌握图像处理中常见的空域滤波算法
4. 掌握图像处理中常见的边缘检测算子。

三、 实验设计、算法和流程 (Design, algorithm and procedure)

3.1 添加高斯噪声和椒盐噪声

对图像分布添加椒盐噪声和高斯噪声，然后在屏幕打印三幅图。

✧ **添加高斯噪声的部分代码如下**

生成一维高斯噪声，将噪声矩阵添加到原图像矩阵完成加噪

```
# input: img_array 二维数组        待添加噪声的图像
#         mean      一维数组    高斯噪声的均值（在这里为1维）
#         cov       二维矩阵    高斯噪声的协方差（这里的高斯噪声为一维的，所以这里也是方差）
# output: gauss_array   二维数组   经过高斯噪声处理后的数据

gauss_noise = np.random.multivariate_normal(mean, cov, img_array.shape)[:, :, 0]  # 高斯噪声
gauss_array = img_array + gauss_noise  # 添加噪声
return gauss_array
```

✧ **添加椒盐噪声的关键代码如下**

根

其中椒盐噪声的生成过程是：先计算椒盐噪声的个数，然后随机生成噪声点的位置，如果噪声点的位置是偶数行，令这一点为 0，否则为 255。

```
# input: img_array   二维数组    待处理的图片
#         SNR         double 信噪比
# output: impulse_array   二维数组   加了椒盐噪声处理后的图片

row = img_array.shape[0]  # 行数
line = img_array.shape[1]  # 列数
noise_size = int(img_array.shape[0] * img_array.shape[1] * (1 - SNR))
impulse_array = img_array
# 添加椒盐噪声
for i in range(noise_size):
    i = np.random.randint(0, row)  # 行标号
    j = np.random.randint(0, line)  # 列标号
    if i % 2 == 0:
        impulse_array[i][j] = 0
    else:
        impulse_array[i][j] = 255
return impulse_array
```

3.2 基于中值滤波和均值滤波的图像平滑处理

中值滤波是记录算子的中位数为新的图片中的点，而均值滤波是记录算子的均值作为新的图片的点。

3.3 实现图片的边缘检测

图像的边缘检测其实就是锐化处理，主要目的就是突出灰度的过度部分，即图像边缘。可以应用在众多领域，如电子印刷、医学成像、工业检测和军事系统的制导等。下面是几种关于图像一阶微分的算子

3.3.1 Sobel 算子实现边缘检测

Sobel 算子在 3x3 的模型中，相当于对 i，j 直接相邻的四个点赋予 2 的权值，其余 4 个点的权值赋值 1，这样将直接相邻的点与别的点区别开，更加有利于对边界的提取，然后再与原图像做卷积运算得到边缘图像。

关键代码如下

其中 k 可以调节强度，一般取 2

```
"""Sobel算子边缘检测,3x3"""
# input:img_array   数组   待检测轮廓的图像
#           k       int    表示与像素点临近点的加权值

kernel1 = np.array([[1, 0, -1], [k, 0, -k], [1, 0, -1]])  # 左边减右边
kernel2 = np.array([[1, k, 1], [0, 0, 0], [-1, -k, -1]])  # 上边减下边
m = img_array.shape[0]
n = img_array.shape[1]
rel_array = np.zeros((m - 2, n - 2))
array1 = con(kernel1, img_array)
array2 = con(kernel2, img_array)
for i in range(m - 2):
    for j in range(n - 2):
        rel_array[i][j] = np.sqrt(array1[i][j] ** 2 + array2[i][j] ** 2)
return rel_array
```

3.3.2 Premitt 算子实现图片的边缘检测

是一种边缘样板算子，利用像素点左右邻点灰度差，在边缘处达到极值检验边缘，对噪声具有平滑作用。

关键代码如下

```
"""prewitt算子边缘检测"""
# input:img_array  数组  待检测轮廓的图像

kernel1 = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  # 右边减左边
kernel2 = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  # 下边减上边
m = img_array.shape[0]
n = img_array.shape[1]
rel_array = np.zeros((m - 2, n - 2))
array1 = con(kernel1, img_array)
array2 = con(kernel2, img_array)
for i in range(m - 2):
    for j in range(n - 2):
        rel_array[i][j] = np.sqrt(array1[i][j] ** 2 + array2[i][j] ** 2)
return rel_array
```

3.3.3 Roberts 算子实现图片的边缘检测

Roberts 算子是针对 2x2 的，斜对角的差值和，算子精度高，对水平和垂直方向的效果好，但是对噪声敏感。

```
m = img_array.shape[0]
n = img_array.shape[1]
rel_array = np.zeros((m - 1, n - 1))
for i in range(m - 1):
    for j in range(n - 1):
        rel_array[i][j] = np.sqrt(
            (img_array[i][j] - img_array[i + 1][j + 1]) ** 2 + (img_array[i + 1][j] - img_array[i][j + 1])
```

3.3.4 Canny 算子实现图片的边缘检测

Canny 算子把边缘检测问题转换为检测单位函数极大的问题来考虑。它利用高斯模型，借助图像滤波的概念址储一个好的边沿检测算子应该具有 3 个指标：

1. 低失误率，既要少将真的边缘丢弃，也要少将非边缘判为边缘
2. 高位置精度，检测出的边缘应在真正的边界上
3. 单像素边缘，每个边缘有唯一的相应，得到的边界为单像素宽

Canny 提出了判定边缘检测算子的三个准则：信噪比准则、定位精度准则和单位边缘相应准则。

Canny 算子进行边缘检测的步骤：

1） 用高斯滤波平滑图像，去除一些噪声的影响

```
new_gray = con(kernel, img_array) / sum_kernel  # 高斯平滑
```

2） 求一阶偏导，用一阶偏导的有限差分来计算梯度的幅值和方向

```
for i in range(W1 - 1):
    for j in range(H1 - 1):
        dx[i, j] = new_gray[i, j + 1] - new_gray[i, j]
        dy[i, j] = new_gray[i + 1, j] - new_gray[i, j]
        d[i, j] = np.sqrt(np.square(dx[i, j]) + np.square(dy[i, j]))
```

3） 对梯度幅值进行非极大值抑制

```
for i in range(1, W2 - 1):
    for j in range(1, H2 - 1):
        if d[i, j] == 0:
            NMS[i, j] = 0
        else:
            gradX = dx[i, j]
            gradY = dy[i, j]
            gradTemp = d[i, j]

            # 如果Y方向幅度值较大
            if np.abs(gradY) > np.abs(gradX):
                weight = np.abs(gradX) / np.abs(gradY)
                grad2 = d[i - 1, j]
                grad4 = d[i + 1, j]
                # 如果x,y方向梯度符号相同
                if gradX * gradY > 0:
                    grad1 = d[i - 1, j - 1]
                    grad3 = d[i + 1, j + 1]
                # 如果x,y方向梯度符号相反
                else:
                    grad1 = d[i - 1, j + 1]
                    grad3 = d[i + 1, j - 1]
```

```
            # 如果X方向幅度值较大
            else:
                weight = np.abs(gradY) / np.abs(gradX)
                grad2 = d[i, j - 1]
                grad4 = d[i, j + 1]
                # 如果x,y方向梯度符号相同
                if gradX * gradY > 0:
                    grad1 = d[i + 1, j - 1]
                    grad3 = d[i - 1, j + 1]
                # 如果x,y方向梯度符号相反
                else:
                    grad1 = d[i - 1, j - 1]
                    grad3 = d[i + 1, j + 1]

            gradTemp1 = weight * grad1 + (1 - weight) * grad2
            gradTemp2 = weight * grad3 + (1 - weight) * grad4
            if gradTemp >= gradTemp1 and gradTemp >= gradTemp2:
                NMS[i, j] = gradTemp
            else:
                NMS[i, j] = 0
```

4） 用双阈值算法进行检测和边缘连接

```
W3, H3 = NMS.shape
DT = np.zeros([W3, H3])
# 定义高低阈值
TL = 0.15 * np.max(NMS)
TH = 0.3 * np.max(NMS)
for i in range(1, W3 - 1):
    for j in range(1, H3 - 1):
        if (NMS[i, j] < TL):
            DT[i, j] = 0
        elif (NMS[i, j] > TH):
            DT[i, j] = 1
        elif ((NMS[i - 1, j - 1:j + 1] < TH).any() or (NMS[i + 1, j - 1:j + 1]).any()
              or (NMS[i, [j - 1, j + 1]] < TH).any()):
            DT[i, j] = 1
```

3.4 BMP 文件读取

BMP 文件的数据从文件头开始的先后顺序分为 4 个部分：bmp 文件头、位图信息头、调色板和维图数据。具体字节数如下

数据段名称	对应的Windows结构体定义	大小(Byte)
bmp文件头	BITMAPFILEHEADER	14
位图信息头	BITMAPINFOHEADER	40
调色板		由颜色索引数决定
位图数据		由图像尺寸决定

Window 中 bmp 文件头信息如下

变量名	地址偏移	大小	作用
bfType	0000h	2 bytes	说明文件的类型，可取值为： • BM - Windows 3.1x, 95, NT, ... • BA - OS/2 Bitmap Array • CI - OS/2 Color Icon • CP - OS/2 Color Pointer • IC - OS/2 Icon • PT - OS/2 Pointer
bfSize	0002h	4 bytes	说明该位图文件的大小，用字节为单位
bfReserved1	0006h	2 bytes	保留，必须设置为0
bfReserved2	0008h	2 bytes	保留，必须设置为0
bfOffBits	000Ah	4 bytes	说明从文件头开始到实际的图象数据之间的字节的偏移量。 这个参数是非常有用的，因为位图信息头和调色板的长度会根据不同情况而变化， 所以我们可以用这个偏移值迅速的从文件中读取到位图数据。

位图信息头定义如下

变量名	地址偏移	大小	作用
biSize	000Eh	4 bytes	BITMAPINFOHEADER结构所需要的字数。
biWidth	0012h	4 bytes	说明图像的宽度，用像素为单位
biHeight	0016h	4 bytes	说明图像的高度，以像素为单位。 注：这个值除了用于描述图像的高度之外，它还有另一个用处，就是指明该图像是倒向的位图，还是正向的位图。 如果该值是一个正数，说明图像是倒向的，如果该值是一个负数，则说明图像是正向的。 大多数的BMP文件都是倒向的位图，也就是高度值是一个正数。
biPlanes	001Ah	2 bytes	为目标设备说明颜色平面数， 其值将总是被设为1。
biBitCount	001Ch	2 bytes	说明比特数/像素，其值为1、4、8、16、24或32。
biCompression	001Eh	4 bytes	说明图像数据压缩的类型。取值范围： 0 BI_RGB 不压缩(最常用) 1 BI_RLE8 8比特游程编码(RLE)，只用于8位位图 2 BI_RLE4 4比特游程编码(RLE)，只用于4位位图 3 BI_BITFIELDS 比特域，用于16/32位位图 4 BI_JPEG JPEG 位图含JPEG图像(仅用于打印机) 5 BI_PNG PNG 位图含PNG图像(仅用于打印机)
biSizeImage	0022h	4 bytes	说明图像的大小， 以字节为单位。当用BI_RGB格式时，可设置为0。
biXPelsPerMeter	0026h	4 bytes	说明水平分辨率，用像素/米表示，有符号整数
biYPelsPerMeter	002Ah	4 bytes	说明垂直分辨率，用像素/米表示，有符号整数
biClrUsed	002Eh	4 bytes	说明位图实际使用的彩色表中的颜色索引数 (设为0的话，则说明使用所有调色板项)
biClrImportant	0032h	4 bytes	说明对图像显示有重要影响的颜色索引的数目 如果是0，表示都重要。

根据位图信息头的信息可以逐个读出调色盘的内容，最后读取位图数据。如果图像是 24 位数据的维图，则是按照 BGR 来存储每个像素的个颜色通道的值；如果是 32 位数据，则是按照 BGRA 的顺序存储。分离出 RGB 三个通道可以供后续图像处理操作。

四、实验结果

4.1 生成高斯噪声和椒盐噪声的图像为：

图 1 从这张图可以看出，高斯噪声类似于加深了纹理，而椒盐噪声则是直接以颗粒的形式存在。

图 2 高斯噪声会协方差的影响，如果协方差增大，纹理会变得更明显

图 3 高斯噪声也会受均值的影响，如果均值增大，图片的整体亮度会提升。当均值为 500 时，图片全部变成白色，完全没有原图的痕迹。

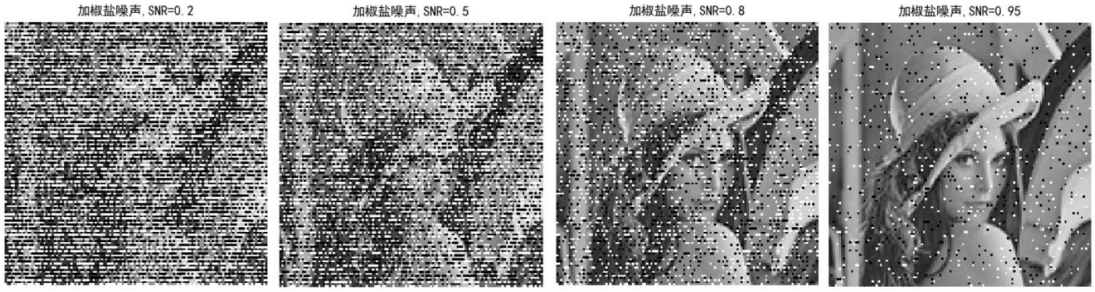

图 4 椒盐噪声受信噪比的影响，信噪比越大，噪声的影响越小。当信噪比低于 0.5 时，图片的轮廓几乎无法辨别。

4.2 基于中值滤波和均值滤波的图像平滑

图 5 分别使用中值滤波和均值滤波对高斯噪声和椒盐噪声做去噪处理。从图中可以看出，中值滤波更擅长处理椒盐噪声，而均值滤波更擅长去除高斯噪声。其中均值滤波是完全不能用于椒盐噪声的去除，因为会出现很多点去不掉。而中值滤波是可以勉强去除高斯噪声的，只是会有些纹理去除不了。

图 6 随意找了一张有噪声的图，从这张图片难以分辨两种方法的优劣。在实际运用当中，如果噪声主要以点的形式存在，中值滤波表现会更好；如果噪声是块状的，则均值滤波表现更好。

4.3 实现图像的边缘检测

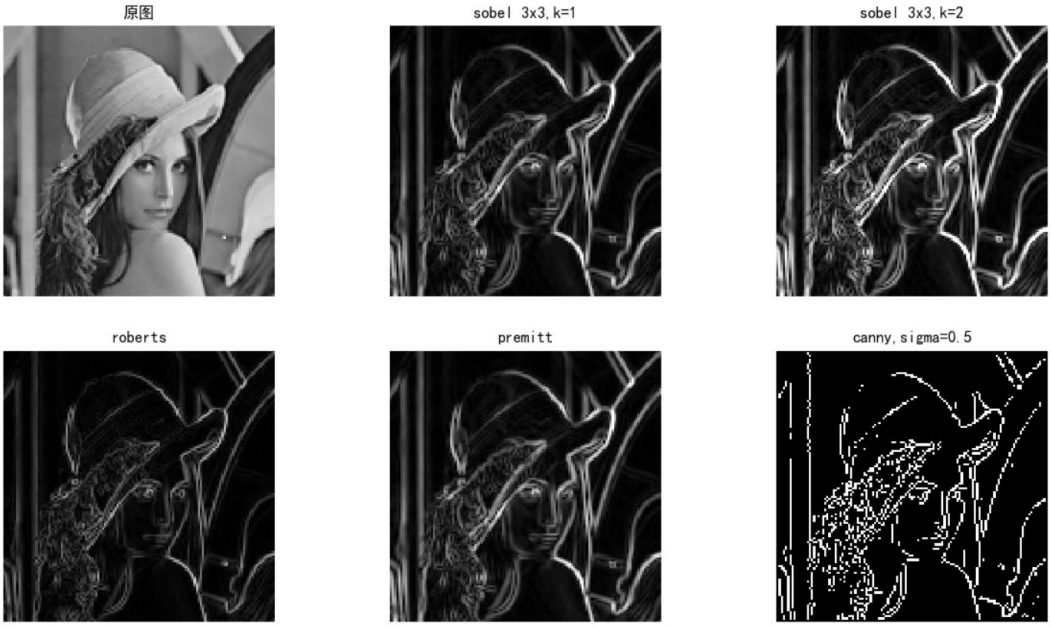

图 7 由 4 中图片边缘提取算子的对比可以看出，由于 canny 算子做了二值化，所以整体只有黑和白，没有灰色，图片颜色更单一，而且有明显的不连续现象（可以依靠算法后期修复，这里不展开研究）。Canny 算子与其他算子相比，canny 算子检测出的边界相对更细一些，这也符合前面 Canny 算子的定义。而其他三个算子，Roberts 算子检测出的边界画面更精细一些，但是由于收到噪声的干扰，将一些非边界的判为边界；Permitt 算子检测出的边界比 Roberts 算子的边界更粗一些；Sobel 算子检测出的边界更宽保留的细节更多，但是还需要细化处理。

四、 结论(conclusion)

1. 高斯噪声形成的噪声主要表现在整体亮度（对应高斯噪声中的均值，均值越大，整体亮度越高）和纹理（对应高斯噪声中的协方差，协方差越大，纹理越显著）
2. 椒盐噪声是直接产生黑白点，使图片直接产生颗粒感，对于图片的影响非常大。
3. 图片中的噪声有很多种，从离散性和随机性来分有高斯噪声、椒盐噪声和泊松噪声；从成像系统来分有暗电流噪声、散射噪声、量化误差、ccd 死点带来的脉冲噪声、热噪声，还有外界对成像系统的干扰引入的噪声，压缩存储编码引入的噪声。
4. 由于高斯噪声是针对大面积产生噪声，所以去除高斯噪声效果比较好的是均值滤波和维纳滤波，使用中值滤波不能很好地去除高斯噪声；由于椒盐噪声往往是一些离散的“坏点”，去除椒盐噪声效果比较好的是中值滤波，而且使用中值滤波去噪效果非常不好，只会引入大的“坏点”。
5. Robert 算子、Sobel 算子、Permitt 算子以及 Canny 算子在 lena 图的边缘检测这个问题上表现都不错，但是综合评价来看，Roberts 模型简单，而且效果非常好。Canny 算子较复杂，但是提升空间更大，尤其是针对边界不连

续问题可以做进一步优化。