

# 第四章作业实验报告

## 实验环境

本信息检索作业采用基于Python3的Jupyter Notebook进行编程，需要第三方库 `OpenCV(cv2)`、`numpy`、`matplotlib`。原始工程代码为 `.ipynb` 格式，另有导出的 `.py` 源代码及 `.html` 格式的代码及运行结果。

## 实验设计

1. 通过 `OpenCV` 导入原始图片，以 `numpy` 数组形式，按照“B, G, R”的通道顺序将图片存在一个矩阵中
2. 用 `OpenCV.equalizeHist()` 方法实现直方图均衡化
3. 用 `OpenCV.threshold()` 方法实现二值化
4. 自主实现添加高斯白噪声
5. 用 `OpenCV.fastNlMeansDenoisingColored()` 方法实现平滑处理
6. 用 `OpenCV.Canny()` 方法实现边缘检测
7. 用 `matplotlib` 显示处理后的图片

## 实验过程

1. 导入图片
2. 分别实验直方图均衡化、二值化、添加噪声、平滑处理、边缘检测算法
3. 将处理后的图片与原始图片比对
4. 针对不同参数的处理效果进行比较
5. 保存最优结果

## 编程实现

### 直方图均衡化

由于目标图片为彩色，首先需要进行通道分离。

```
(b,g,r) = cv2.split(lena)
```

之后就可以对三个颜色通道分别进行均衡化。

```
bH = cv2.equalizeHist(b)
gH = cv2.equalizeHist(g)
rH = cv2.equalizeHist(r)
```

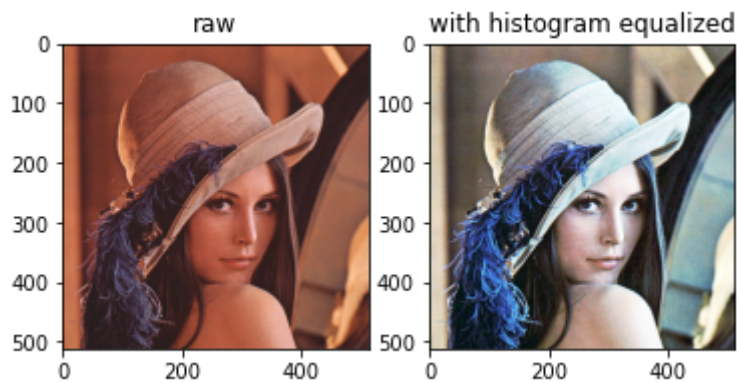
由于彩色图片本质上就是三个通道颜色的叠加，将均衡化后三个通道的颜色合并，就可以得到原始图片的均衡化结果。

```
lenah = cv2.merge((bH,gH,rH))
```

合并后，使用 `matplotlib` 显示处理结果。由于 `cv2` 将图片读取为(B, G, R)形式，而 `matplotlib` 按照(R, G, B)形式显示图片，需要对图片的表示矩阵进行处理。

```
plt.imshow(lenaH[:,:,:-1])
```

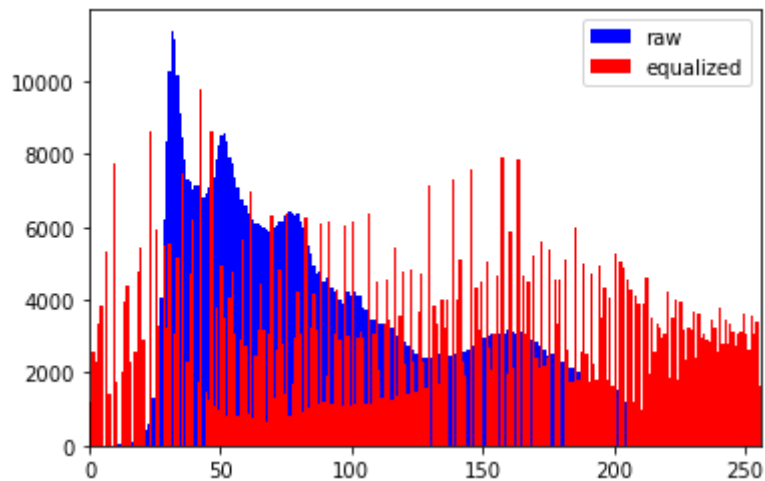
均衡化效果如下。



直观地看，原来的图片整体颜色较暗，经过处理后两部明显增多，明暗总体均衡。下面通过直方图定量比较处理前后的差异。

```
hist,bins = np.histogram(lena.flatten(),256,[0,256])
plt.hist(lena.flatten(),256,[0,256], color = 'b')
hist,bins = np.histogram(lenaH.flatten(),256,[0,256])
plt.hist(lenaH.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('raw','equalized'), loc = 'upper right')
plt.show()
```

直方图如下，可见均衡化处理效果显著。



## 二值化处理

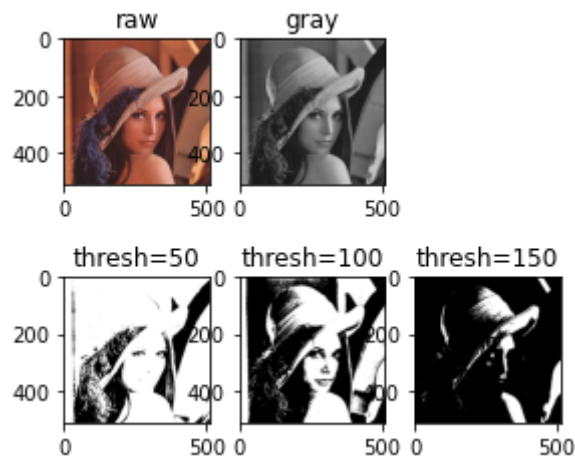
二值化处理只考虑像素的亮度，并按照阈值将其转为黑或白，而忽略颜色的因素，因此，在二值化处理前首先进行灰度处理。

```
lenaG = cv2.cvtColor(lena, cv2.COLOR_BGR2GRAY)
```

接下来的二值化处理中，分别尝试不同的阈值。

```
retval, lenaB_050 = cv2.threshold(lenaG, 50, 255, cv2.THRESH_BINARY)
retval, lenaB_100 = cv2.threshold(lenaG, 100, 255, cv2.THRESH_BINARY)
retval, lenaB_150 = cv2.threshold(lenaG, 150, 255, cv2.THRESH_BINARY)
```

对比处理效果。



可见阈值为100，即亮度低于100的转为黑色，高于100的转为白色时，效果比较好。

## 添加噪声与平滑处理

自主实现添加噪声，首先创建一个服从高斯分布的矩阵

```
mean = 0
var = 0.01
noise = np.random.normal(mean, var ** 0.5, lena.shape)
```

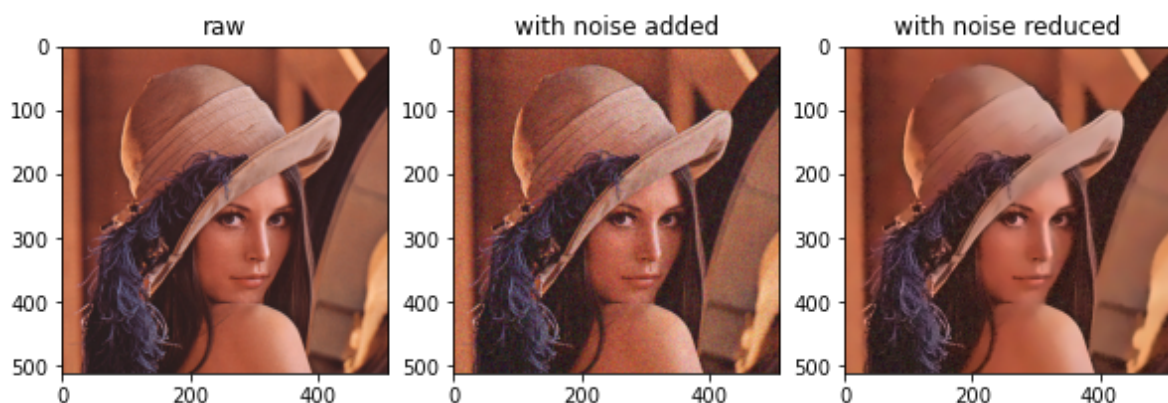
预处理后叠加原图片与噪声矩阵

```
noiseless = np.array(lena/255, dtype=float)
noisy = noiseless + noise
noisy = np.clip(noisy, 0, 1)
lenaNA = np.uint8(noisy*255)
```

对图片施加平滑降噪处理

```
lenaNR = cv2.fastNlMeansDenoisingColored(lenaNA, h=13, hColor=15)
```

比对处理效果



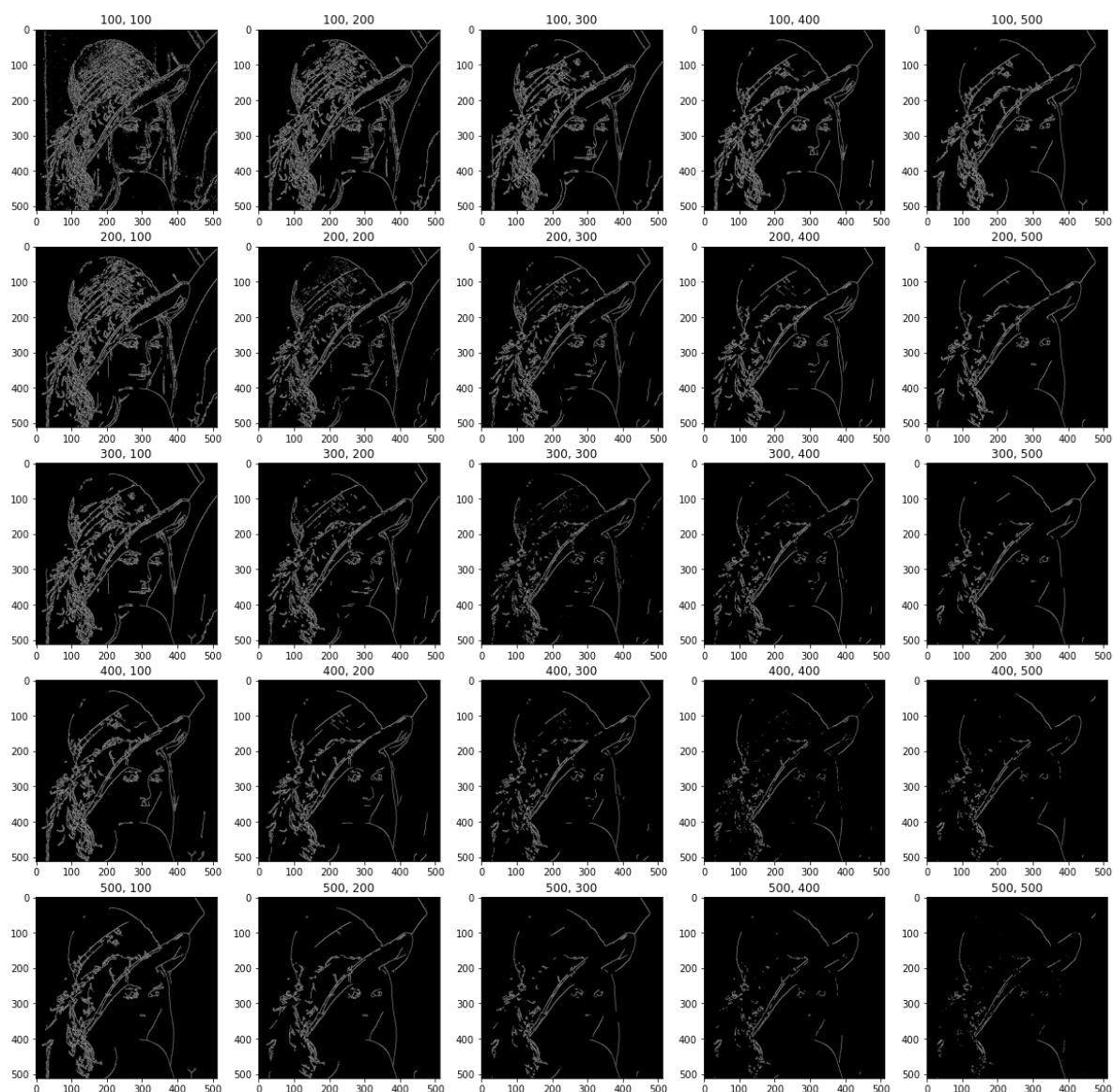
可见，添加高斯白噪声后图片上明显多了很多噪点，而在平滑处理后噪点基本被消除，但同时图片也损失了一些细节，尤其是帽子上的纹路有较多丢失，但面部、头发收到的影响不大。在降噪处理时，`h`、`hColor` 两个参数分别选取为13,15，如果低于该值，在图片左上方会残留很多噪点，高于该值基本上不能进一步降噪，而且细节丢失更多。

## 边缘检测

在边缘检测中，需要设定两个阈值，这里分别对其取值进行遍历，寻找最优解。

```
for i in range(5):
    for j in range(5):
        lenaE = cv2.Canny(lena, threshold1=100+100*i, threshold2=100+100*j)
```

得到的结果如下图。





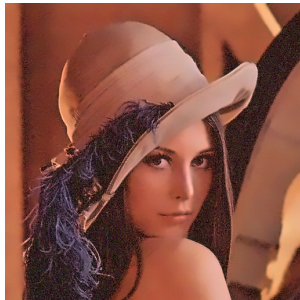




从图中可以总结出，两个临界值均对边缘检测的细节有很大影响，其值越大，检测出的边缘越简略，但这两个值之间的区别不大。

总体来说，在`threshold1=300`，`threshold2=200`时，边缘检测效果比较好，即比较充分地保留了脸部轮廓，也没有太多的帽子纹路的干扰。

## 实验结果



	原图	处理图-1	处理图-2
直方图均衡化			
二值化, 阈值为100			
添加高斯白噪声/平滑化			
边缘检测, 阈值为(300, 200)			

## 额外的思考与尝试

在前面的实验中发现，边缘检测中有一对矛盾的因素：帽子与面部。由于帽子纹路及其装饰物细节多，颜色变化剧烈，容易被识别成边缘，因此在面部轮廓基本完整时，头顶上就会有很乱的线条（如threshold为100时）。反之，帽子的线条简化了，面部轮廓丢失就很严重（如threshold为500时）。而在平滑化处理时，帽子的纹理被极大地抹平了，细节比原图少。

**基于以上发现，可以得出一个猜想：在进行边缘检测前，可以先对图片平滑化处理，去除一些容易被误判为轮廓的线条。**

```
lenaE = cv2.Canny(cv2.fastNlMeansDenoisingColored(lenaNA, h=30, hColor=100),
threshold1=50, threshold2=50)
plt.imshow(lenaE, cmap="gray")
plt.show()
```

这里首先对图片进行较强的平滑化处理，再施加较弱的边缘检测，效果如下。



可见，在保留了丰富的面部细节的同时，帽子纹路及装饰品的线条被极大地简化了。相比于前面权衡之下直接进行边缘检测，尤其是右脸颊的轮廓非常清晰、完整。该结果证明上边提到的猜想是正确的、有意义的。

## 实验总结

---

本实验完成了对于直方图均衡化、二值化处理、添加噪声与平滑化和边缘检测的图像处理方法的应用，找到了一些效果比较好的模型参数。此外，本实验还创造性地结合了图像平滑化与边缘检测，找到了降低边缘误判率的有效手段。