

机器视觉-实践作业-2

国嘉通^{*}

2025 年 9 月 22 日

摘要

机器视觉第二次实验课程主要包含三大部分的练习，一是灰度变换基础，二是空域平滑滤波，三是空域锐化滤波。对于灰度变换基础部分，首先讲解如何使用 OpenCV 读取图像并显示其灰度直方图；在此基础上，还会讲解如何实现幂律（伽马）变换，并通过尝试不同的伽马值观察其对整体亮度与对比度的调控作用；进一步地，对灰度直方图进行均衡化并应用于低对比图像。对于空域平滑滤波部分，主要陈述了如何对图像进行加噪，并实现均值滤波、高斯滤波、中值滤波，讨论其对抑制高斯噪声和椒盐噪声方面的效果差异。对于空域锐化滤波部分，主要讲解如何进行拉普拉斯算子锐化操作，并实现高提升滤波，讨论锐化与噪声之间的平衡关系。希望本篇文章是一篇内容翔实、方便随时翻阅复习学习 OpenCV 的学习笔记。

Now the CODE is available at:

<https://github.com/GplasT0810/Machine-Vision-Experiment.git>

长沙理工大学

卓越工程师学院 绿色智慧交通 2401 班

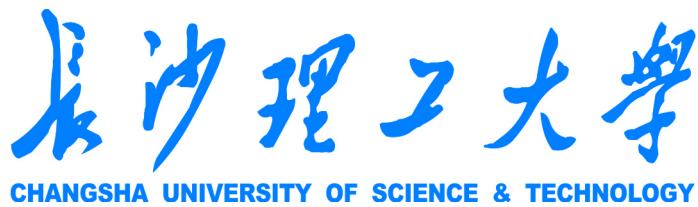
202404080608

国嘉通

*Tel.+86-166 8130 6085, E-mail:guojiatom2006@outlook.com

目录

1 灰度变换基础	3
1.1 直方图显示	3
1.2 幂律(伽马)变换	5
1.3 灰度直方图均衡化	7
2 空域平滑滤波	9
2.1 图像加噪	9
2.2 图像滤波	13
3 空域锐化滤波	17
3.1 拉普拉斯算子锐化操作	17
3.2 高提升滤波操作	19
3.3 锐化与噪声的权衡讨论	21



Complated Time: Sep.22th.2025

Guo Jiatong , Green & Intelligent Transportation 2401
Elite Engineering School , Changsha University of Science & Technology.
Changsha 410114 , Hunan , China

1 灰度变换基础

本章共有三个部分，它们分别为：

- 直方图显示

使用 OpenCV 读取一张典型的低对比度图像（如背光人像、X 光片、雾天场景）和一张正常对比度图像，显示原始图像及其灰度直方图。

- 幂律（伽马）变换

幂律（伽马）变换是一种简单而强大的灰度非线性映射，公式如下：

$$s = c \times r^\gamma$$

其中， r 为输入的已归一化后的像素值； γ 为核心参数，其值越小输出图像越亮，反之越暗； c 为常数，通常取为 1； s 为输出的像素值。

固定 $c=1$ ，尝试不同的 γ 值，观察其对整体亮度和对比度的调控作用。

- 灰度直方图均衡化

实现直方图均衡化，应用于低对比度图像，观察图像视觉效果和直方图形态的变化。

1.1 直方图显示

欲想要显示图像的直方图，要先将图像转换为单通道灰度图像。所有像素值均落在 0-255 之间。依旧采用 cvtColor 函数将 BGR 转换为 GRAY。

然后，可以调用 cv2 库中的 **calcHist**([image_name],[channels],mask,HistSize,PixelRanges) 来计算灰度直方图。值得注意的是，此函数返回的并不是直接的图像，而是一维数组。这意味着我们不能直接使用 imshow 来显示它，而应该用 plot 函数来显示它。

```
# 直方图显示
import cv2
import matplotlib.pyplot as plt
import numpy as np

# 读取照片
lh_img_path = '/Users/guo2006/myenv/Machine Vision Experiment/Machine Vision\u2022Experiment 2/lh.png'
low_contrast_background_img_path = '/Users/guo2006/myenv/Machine Vision\u2022Experiment/Machine Vision Experiment 2/low_contrast_background.jpg'
img_lh = cv2.imread(lh_img_path)
img_low_contrast_background = cv2.imread(low_contrast_background_img_path)
```

```

# 转换为灰度图像

img_lh_gray = cv2.cvtColor(img_lh, cv2.COLOR_BGR2GRAY)
img_low_contrast_background_gray = cv2.cvtColor(img_low_contrast_background, ↴
    ↪cv2.COLOR_BGR2GRAY)

# 计算并显示两个直方图的对比

# 计算图像灰度直方图的函数 cv2.calcHist(images, channels, mask, histSize, ↴
    ↪pixel_ranges).返回的是一维数组，不能用 imshow(二维图像数组)

hist_img_lh_gray = cv2.calcHist([img_lh_gray], [0], None, [256], [0, 256])
hist_img_low_contrast_background_gray = cv2. ↴
    ↪calcHist([img_low_contrast_background_gray], [0], None, [256], [0, 256])

img_low_contrast_background = cv2.cvtColor(img_low_contrast_background, cv2. ↴
    ↪COLOR_BGR2RGB)
img_lh = cv2.cvtColor(img_lh, cv2.COLOR_BGR2RGB)

# 显示图像

fig, ax = plt.subplots(2,3, figsize=(16,10))
ax[0,0].imshow(img_low_contrast_background)
ax[0,0].set_title('Low Contrast Background')

ax[0,1].imshow(img_low_contrast_background_gray, cmap = 'gray')
ax[0,1].set_title('Low Contrast Background Gray')

ax[0,2].plot(hist_img_low_contrast_background_gray, color='red', label = 'Low ↴
    ↪Contrast Background')
ax[0,2].set_xlabel('Grayscale Value')
ax[0,2].set_ylabel('Frequency')
ax[0,2].set_title('Grayscale Histogram(Low Contrast)')
ax[0,2].legend()

ax[1,0].imshow(img_lh)
ax[1,0].set_title('LH Original Image')

ax[1,1].imshow(img_lh_gray, cmap = 'gray')
ax[1,1].set_title('LH Gray Image')

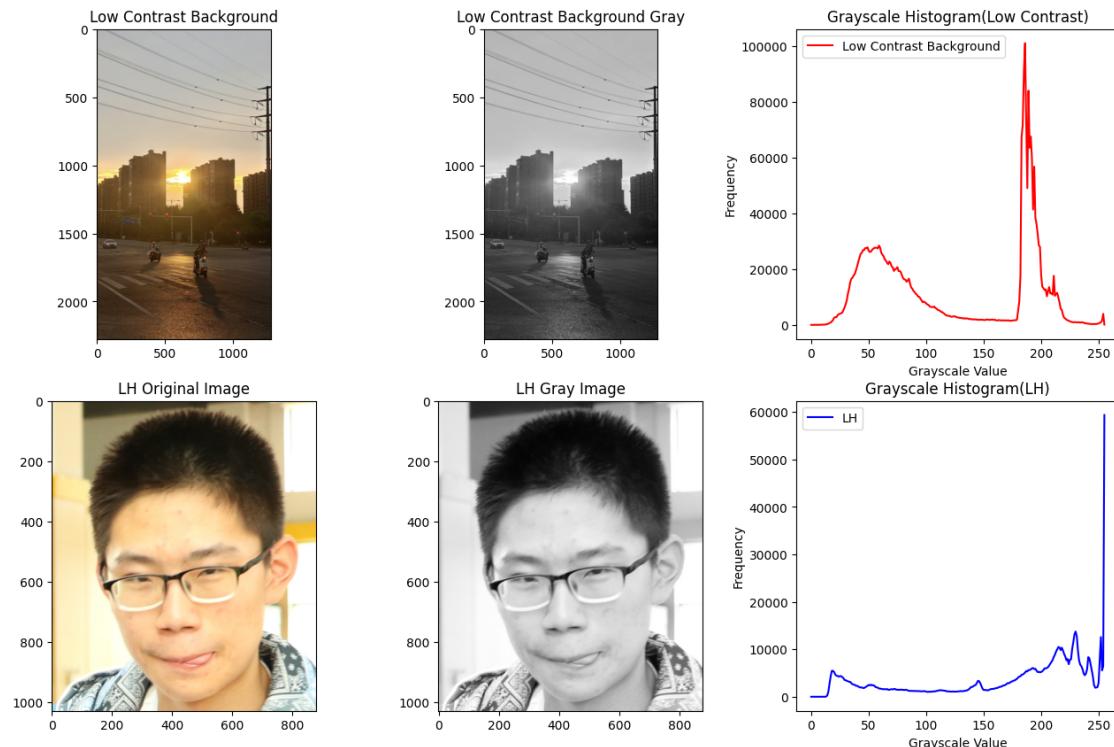
```

```

ax[1,2].plot(hist_img_lh_gray, color='blue', label = 'LH')
ax[1,2].set_xlabel('Grayscale Value')
ax[1,2].set_ylabel('Frequency')
ax[1,2].set_title('Grayscale Histogram(LH)')
ax[1,2].legend()

```

<matplotlib.legend.Legend at 0x373ce3790>



1.2 幂律 (伽马) 变换

```

# 幂律 (伽马) 变换
import cv2
import numpy as np
import matplotlib.pyplot

low_contrast_background_img_path = '/Users/guo2006/myenv/Machine Vision'
    ↪Experiment/Machine Vision Experiment 2/low_contrast_background.jpg'
img_low_contrast_background = cv2.imread(low_contrast_background_img_path)
img_low_contrast_background_gray = cv2.cvtColor(img_low_contrast_background, ↪
    ↪cv2.COLOR_BGR2GRAY)

```

```

# 参数设置
gamma_list = [0.4, 0.7, 1.0, 1.5, 2.2]
c = 1.0
n_cols = len(gamma_list) #gamma 列表列数
# 归一化到 [0,1] 再做幂律变换
img_one = img_low_contrast_background_gray.astype(np.float32)/255.0
#uint8-->float255

fig, ax = plt.subplots(2, n_cols, figsize=(4*n_cols, 8))
# 进行幂律变换
for i, j in enumerate(gamma_list): #i 接收下标, j 接收对应列表值
    s = c * img_one ** j
    s_uint8 = np.clip(s * 255, 0, 255).astype(np.uint8) # 使用 clip 做截断后转换
    # 直方图
    hist = cv2.calcHist([s_uint8], [0], None, [256], [0,256])

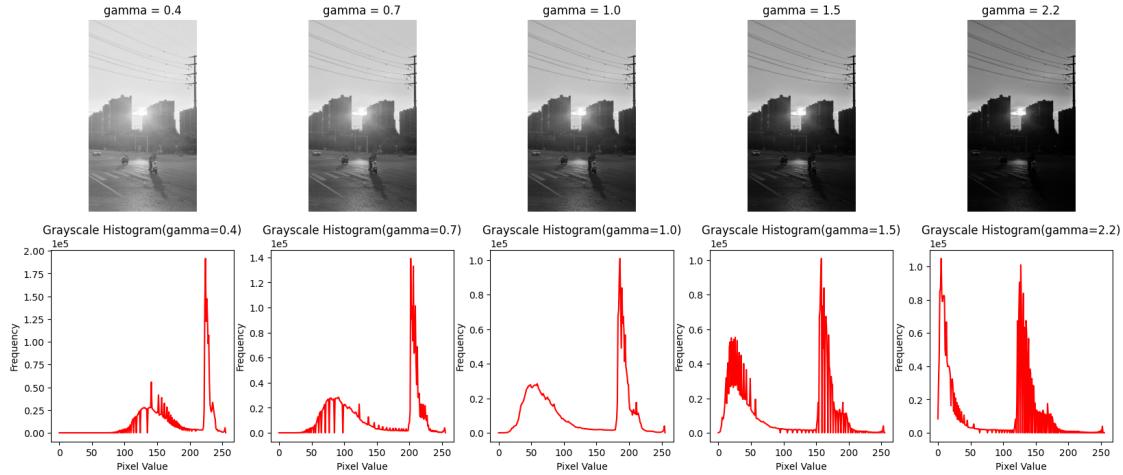
    ax[0, i].imshow(s_uint8, cmap='gray')
    ax[0, i].set_title(f'gamma = {j}')
    ax[0, i].axis('off')

    ax[1, i].plot(hist, color='red')
    ax[1, i].ticklabel_format(style='scientific', axis='y', scilimits=(0,0)) #
    使用科学计数法
    ax[1, i].set_xlabel('Pixel Value')
    ax[1, i].set_ylabel('Frequency')
    ax[1, i].set_title(f'Grayscale Histogram(gamma={j})')

```

根据观察不同 γ 值情况下的幂律变换，不难发现以下规律：

- $\gamma < 1$ 时
直方图右移，暗区被拉开 → 图像变亮，暗细节可见。
- $\gamma > 1$ 时
直方图左移，亮区被压缩 → 图像变暗，高光细节显现。
- $\gamma = 1$ 时
线性映射，原图不变。



1.3 灰度直方图均衡化

介绍直方图均衡化指令 `equalizeHist(Image Name)`。关于此函数有两点使用上的注意：

- 此为灰度直方图均衡化指令，作用对象必须为单通道的灰度图像。也就是说，必须在使用前对图像进行 `cvtColor` 转换。
- 此函数的输出为图像，可以使用 `imshow` 显示而非 `plot` 指令。

```
# 灰度直方图均衡化
import cv2
import matplotlib.pyplot as plt
import numpy as np

# 读取照片
low_contrast_background_img_path = '/Users/guo2006/myenv/Machine Vision\u2022Experiment/Machine Vision Experiment 2/low_contrast_background.jpg'
img_low_contrast_background = cv2.imread(low_contrast_background_img_path)

# 转换为灰度图像
img_low_contrast_background_gray = cv2.cvtColor(img_low_contrast_background, cv2.COLOR_BGR2GRAY)

# 计算原灰度图像的直方图
hist_img_low_contrast_background_gray = cv2.calcHist([img_low_contrast_background_gray], [0], None, [256], [0, 256])

# 直方图均衡化指令
```

```

equalized_img_low_contrast_background_gray = cv2.
    ↪equalizeHist(img_low_contrast_background_gray) # 输出为图像

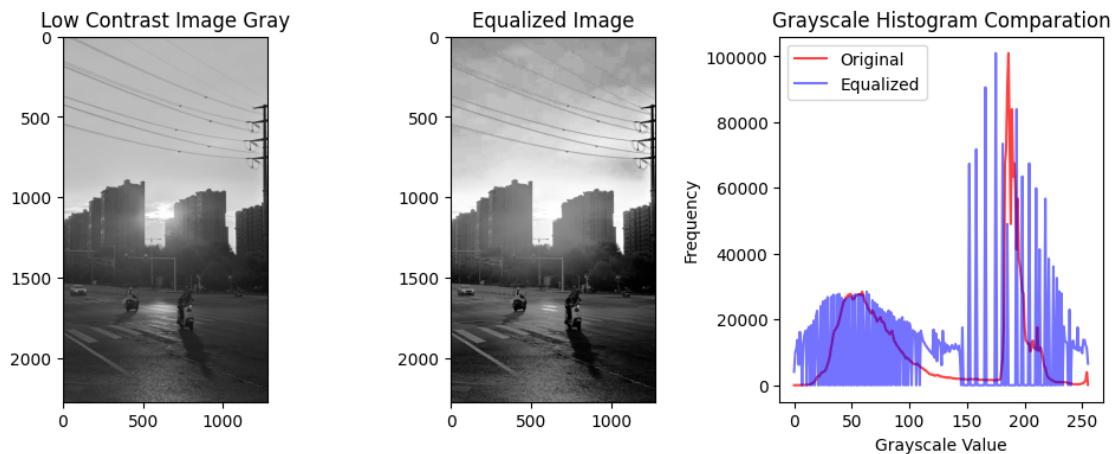
# 均衡化后直方图计算
hist_equalized_img_low_contrast_background_gray = cv2.
    ↪calcHist([equalized_img_low_contrast_background_gray], [0], None, [256], [0, ↪
    ↪256]) # 输出为数组

fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].imshow(img_low_contrast_background_gray, cmap='gray')
ax[0].set_title('Low Contrast Image Gray')
ax[1].imshow(equalized_img_low_contrast_background_gray, cmap='gray')
ax[1].set_title('Equalized Image')

ax[2].plot(hist_img_low_contrast_background_gray, color='red', ↪
    ↪label='Original', alpha=0.75)
ax[2].plot(hist_equalized_img_low_contrast_background_gray, color='blue', ↪
    ↪label='Equalized', alpha=0.55)
ax[2].set_title('Grayscale Histogram Comparation')
ax[2].set_xlabel('Grayscale Value')
ax[2].set_ylabel('Frequency')
ax[2].legend()

```

<matplotlib.legend.Legend at 0x3738aa990>



观察均衡化前后的图像可以发现，对比十分明显，原图对比度相对比较弱，有“涂抹”感，层次弱且画面发灰，呈现效果差。对应的灰度直方图跨度较窄，不够均衡，有强烈的波峰、波谷对比。

而均衡化后的图像对比度增强，暗部细节与亮部细节对比明显，更加通透。灰度直方图表现的更为均衡，被“拉平”铺满整个 0-255 区域，不存在某一灰度值下无像素的情况。
可以见得灰度直方图的效果在“把像素数平均分配到整个灰度范围”。

2 空域平滑滤波

本章主要讲解两大部分，分别为图像的加噪处理与图像的滤波处理。

其中图像的加噪处理中，分为高斯噪声与椒盐噪声；图像滤波处理分为均值滤波、高斯滤波、中值滤波三种基本滤波器。在最后，我们还讨论了滤波器大小与其参数的选择策略。

2.1 图像加噪

在本节中，实现了对一张清晰图像人工添加不同类型的噪声：高斯噪声（尝试不同方差）、椒盐噪声（尝试不同噪声密度），并以直观的参数选择显著地展示了其不同之处，得出总结规律。

```
# 图像加噪
# 高斯噪声--> 调节方差，椒盐噪声--> 调节密度

import cv2
import numpy as np
import matplotlib.pyplot as plt
img_path_lh = '/Users/guo2006/myenv/Machine Vision Experiment/Machine Vision_」
↳Experiment 2/lh.png'
img_lh = cv2.imread(img_path_lh)

gauss_var_list = [0.04, 0.08, 0.12] # 高斯噪声方差
salt_pepper_density_list = [0.05, 0.10, 0.15] # 椒盐噪声密度
Height, Width, Channels = img_lh.shape

# 高斯加噪函数
def add_gauss_noise(img, var):
    #img= 图像, var= 方差
    sigma = np.sqrt(var)
    gauss = np.random.normal(0, sigma, img.shape) # 均值默认为 0
    noisy = img + gauss
    return (np.clip(noisy, 0, 1)*255).astype(np.uint8)

# 椒盐加噪函数
def add_salt_pepper_noise(img, density):
    #density= 椒盐点占像素点总比例
```

```

noisy = img.copy() # 在复制图上进行调整
# 随机数椒盐掩膜
mask = np.random.rand(Height, Width) # 生成 0-1 之间的随机矩阵
salt = mask < density / 2 # 白点
pepper = mask > 1 - density / 2 # 黑点
noisy[salt] = 1
noisy[pepper] = 0
return (np.clip(noisy, 0, 1)*255).astype(np.uint8)

# 对原图像加噪前要将灰度图像归一化
img_one = img_lh.astype(np.float32)/255.0
# 找出最大列数 方便进行图像绘制
n_gauss = len(gauss_var_list)
n_salt_pepper = len(salt_pepper_density_list)
cols = max(n_gauss, n_salt_pepper)

fig, ax = plt.subplots(4, cols, figsize=(6*cols, 20))
img_one = cv2.cvtColor(img_one, cv2.COLOR_BGR2RGB)
# 高斯噪声图像
for i, var in enumerate(gauss_var_list):
    noisy = add_gauss_noise(img_one, var)
    ax[0, i].imshow(noisy)
    ax[0, i].set_title(f'Gauss ^{var}')
    ax[0, i].axis('off')
# 高斯噪声直方图 (灰度)
for i, var in enumerate(gauss_var_list):
    noisy_rgb = add_gauss_noise(img_one, var)
    noisy_gray = cv2.cvtColor((noisy_rgb*255).astype(np.uint8), cv2.
    COLOR_RGB2GRAY)
    hist = cv2.calcHist([noisy_gray], [0], None, [256], [0, 256])
    ax[1, i].plot(hist, color='red')
    ax[1, i].ticklabel_format(style='scientific', axis='y', scilimits=(0,0)) #
使用科学计数法
    ax[1, i].set_xlabel('Pixel Value')
    ax[1, i].set_ylabel('Frequency')
    ax[1, i].set_title(f'Hist Gauss ^{var}')
    ax[1, i].set_xlim([0, 256])

```

```

ax[1, i].grid(True)
# 椒盐噪声图像

for i, density in enumerate(salt_pepper_density_list):
    noisy = add_salt_pepper_noise(img_one, density)
    ax[2, i].imshow(noisy)
    ax[2, i].set_title(f'Salt Pepper Noisy d={density}')
    ax[2, i].axis('off')

# 椒盐噪声直方图

for i, density in enumerate(salt_pepper_density_list):
    noisy_rgb = add_salt_pepper_noise(img_one, density)
    noisy_gray = cv2.cvtColor(noisy_rgb, cv2.COLOR_RGB2GRAY)
    hist = cv2.calcHist([noisy_gray], [0], None, [256], [0,256])
    ax[3, i].plot(hist, color='blue')
    ax[3, i].ticklabel_format(style='scientific', axis='y', scilimits=(0,0))
    ax[3, i].set_xlabel('Pixel Value')
    ax[3, i].set_ylabel('Frequency')
    ax[3, i].set_title(f'Hist Salt Pepper d={density}')
    ax[3, i].grid(True)

```

强度等级	高斯噪声图像 & 直方图	椒盐噪声图像 & 直方图
低	图像：轻微颗粒，整体灰度仍在原范围 直方图：整体形状基本保持，两侧出现低幅“拖尾”	图像：零星黑白点 直方图：在 0 和 255 处出现两个小尖峰
中	图像：颗粒感明显，边缘被“毛刺”包围 直方图：整体变胖（方差增大），两侧拖尾加高	图像：黑白点增多，像“雪花” 直方图：0/255 两峰显著增高，中间区域轻微下降
高	图像：出现“雾状”纹理，细节被掩盖 直方图：整体展宽，近似高斯鼓包，原峰值上升	图像：大片盐粒与胡椒点，视觉阻塞 直方图：两端尖峰极高，中间灰度被“挖空”

- 图像特性差异

1. 高斯噪声直接的作用效果为连续的随机偏差，即每个像素都变，使得图像整体呈“雾状/颗粒状”感，整体图像表现较为模糊，且有毛刺感。
2. 椒盐噪声直接作用效果为离散的孤点污染，即只有部分像素被替换成纯黑或纯白，使得图像呈“雪花”或“盐粒”状，图像模糊视觉效果突兀但作用效果局限在局部范围内。这是使得其灰度直方图在 0/255 值处出现双尖峰的原因。

- 直方图特性差异

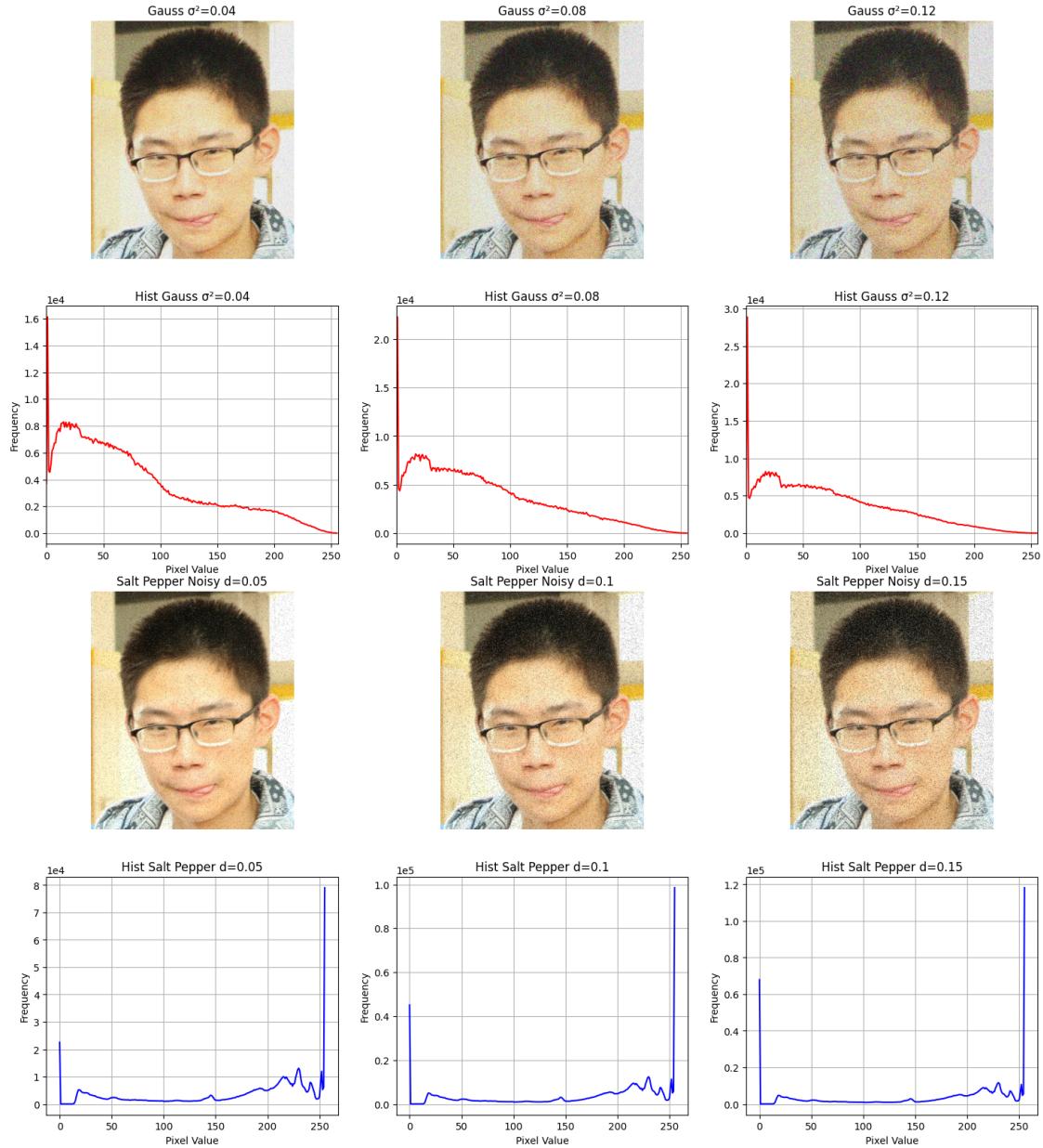
1. 高斯噪声作用后的灰度直方图整体展宽，波形整体偏胖，无孤立尖峰，方差随 σ 取值变

大而变大。

2. 椒盐噪声的灰度直方图在 0 和 255 处出现孤立尖峰，中间灰度相对减少，两端峰值高度与密度成正比，方差同样也十分的大。

- 对后续处理的启示

1. 对于高斯噪声，使用线性低通（高斯、均值）降噪可有效抑制，但会模糊边缘，相对而言，非线性滤波（双边、NLM）效果更好。
2. 对于椒盐噪声，中值滤波最经济，一次 3×3 中值即可去除低密度椒盐且保留边缘；线性滤波对孤立 0/255 点无效。



2.2 图像滤波

在本节中，分别尝试使用均值滤波、高斯滤波、中值滤波三种滤波方式，实现对图像的空域平滑滤波操作。还讨论了不同尺寸大小的滤波器的滤波功能的区别。

- 均值滤波函数 `cv2.blur(img, (k, k))`
- 高斯滤波函数 `cv2.GaussianBlur(img, (k, k), σ_G)`
- 中值滤波函数 `cv2.medianBlur(img, k)`

```
# 滤波
import cv2
import numpy as np
import matplotlib.pyplot as plt

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/Machine VisionExperiment 2/lh.png'
img_lh = cv2.imread(img_path)
img_lh = cv2.cvtColor(img_lh, cv2.COLOR_BGR2RGB)

gauss_var_list = [0.05, 0.10, 0.20] # 高斯噪声方差
salt_pepper_density_list = [0.10, 0.20, 0.30] # 椒盐密度
ksize = [3, 5] # 滤波核尺寸
sigma_G = {3: 0.8, 5: 1.2} # 高斯滤波 经验值 (Key= 核尺寸, Value= 对应 )

# 高斯加噪函数
def add_gauss_noise(img, var):
    #img= 图像, var= 方差
    sigma = np.sqrt(var)
    gauss = np.random.normal(0, sigma, img.shape) # 均值默认为 0
    noisy = img + gauss
    return (np.clip(noisy, 0, 1)*255).astype(np.uint8)

# 椒盐加噪函数
def add_salt_pepper_noise(img, density):
    #density= 椒盐点占像素点总比例
    noisy = img.copy() # 在复制图上进行调整
```

```

# 随机数椒盐掩膜
Height, Width = noisy.shape[:2]
mask = np.random.rand(Height, Width) # 生成 0-1 之间的随机矩阵
salt = mask < density / 2 # 白点
pepper = mask > 1 - density / 2 # 黑点
noisy[salt] = 1
noisy[pepper] = 0
return (np.clip(noisy, 0, 1)*255).astype(np.uint8)

# 定义三种滤波函数（简化代码书写）
def manual_mean(img, k): # 均值滤波
    return cv2.blur(img, (k, k))
def manual_gauss(img, k): # 高斯滤波
    return cv2.GaussianBlur(img, (k, k), sigma_G[k])
def manual_median(img, k): # 中值滤波
    return cv2.medianBlur((img*255).astype(np.uint8), k).astype(np.float32)/255

img_one = img_lh.astype(np.float32)/255 # 归一化

fig, ax = plt.subplots(6, 5, figsize=(15, 18))
ax = ax.reshape(-1, 5) # 行数自动计算，列数固定为 5

row = -1
# 高斯噪声组
for i, var in enumerate(gauss_var_list):
    row += 1
    noisy = add_gauss_noise(img_one, var)
    ax[row, 0].imshow(noisy)
    ax[row, 0].set_title(f'Gauss {var}x{var}')
    ax[row, 0].axis('off')
    # 3x3 滤波
    ax[row, 1].imshow(manual_mean(noisy, 3))
    ax[row, 1].set_title('Mean 3x3')
    ax[row, 1].axis('off')
    ax[row, 2].imshow(manual_gauss(noisy, 3))
    ax[row, 2].set_title('Gauss 3x3'), ax[row, 2].axis('off')

```

```

ax[row, 3].imshow(manual_median(noisy, 3))
ax[row, 3].set_title('Median 3×3')
ax[row, 3].axis('off')

# 5×5 滤波
ax[row, 4].imshow(manual_median(noisy, 5))
ax[row, 4].set_title('Median 5×5')
ax[row, 4].axis('off')

# 椒盐噪声组
for i, density in enumerate(salt_pepper_density_list):
    row += 1
    noisy = add_salt_pepper_noise(img_one, density)
    ax[row, 0].imshow(noisy)
    ax[row, 0].set_title(f'Salt Pepper d={density}-->')
    ax[row, 0].axis('off')

    ax[row, 1].imshow(manual_mean(noisy, 3))
    ax[row, 1].set_title('Mean 3×3')
    ax[row, 1].axis('off')

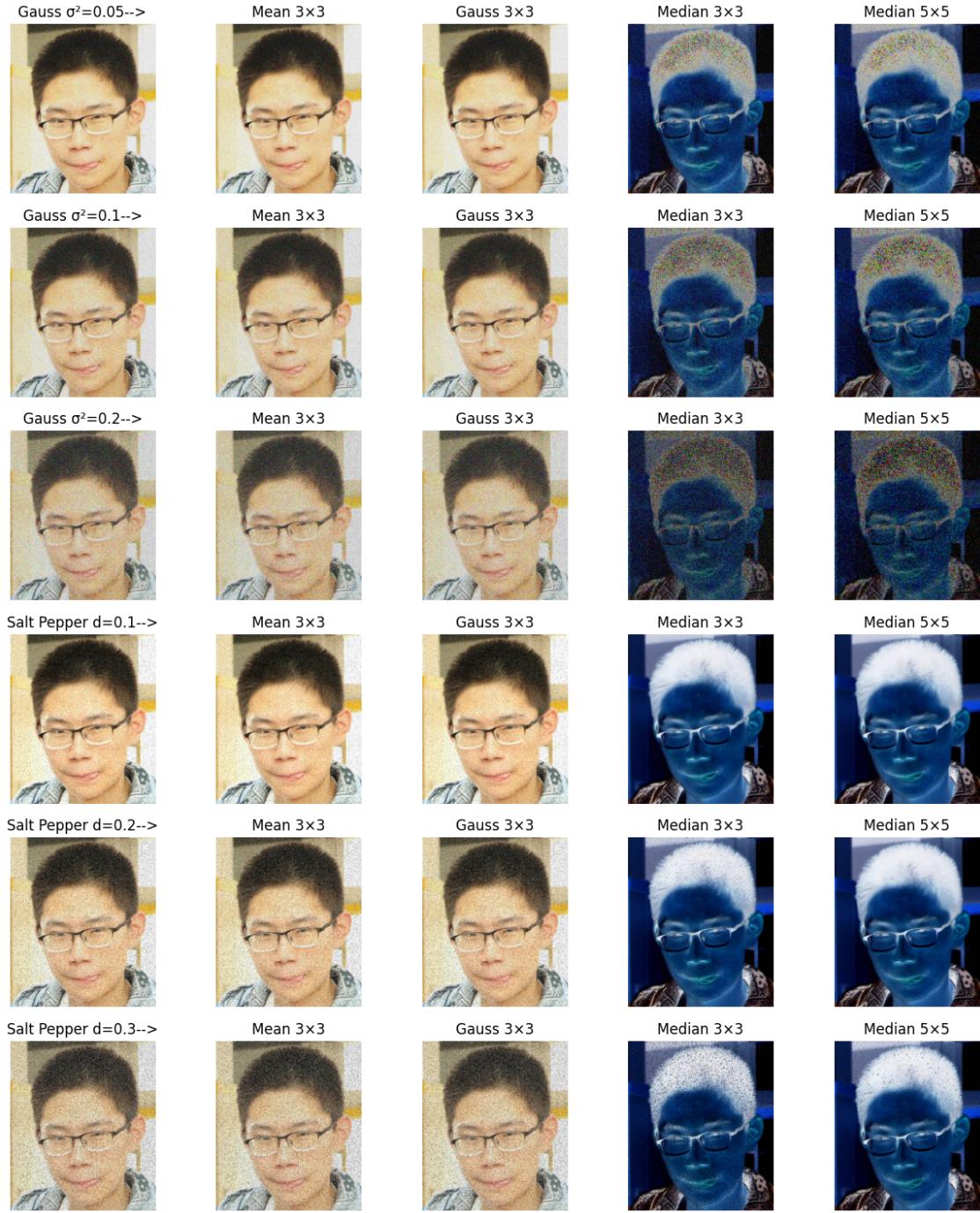
    ax[row, 2].imshow(manual_gauss(noisy, 3))
    ax[row, 2].set_title('Gauss 3×3')
    ax[row, 2].axis('off')

    ax[row, 3].imshow(manual_median(noisy, 3))
    ax[row, 3].set_title('Median 3×3')
    ax[row, 3].axis('off')

    ax[row, 4].imshow(manual_median(noisy, 5))
    ax[row, 4].set_title('Median 5×5')
    ax[row, 4].axis('off')

```

滤波器类型	核/size	关键参数	高斯抑制能力	椒盐抑制能力	边缘保持能力
均值滤波	3×3	无	中	低	低
均值滤波	5×5	无	高	低	低
高斯滤波	3×3	$\sigma_G=0.8$	高	低	中
高斯滤波	5×5	$\sigma_G=1.2$	高	低	中
中值滤波	3×3	无	低	高	高
中值滤波	5×5	无	低	高	中



通过观察表格与实验图像不难发现以下特点与规律：

- 噪声抑制能力

1. 对于高斯噪声的抑制效果：高斯滤波 \approx 均值滤波 $>$ 中值滤波，且滤波器的核尺寸越大，得到的输出结果越平滑，但边缘越模糊。
2. 对于椒盐噪声的抑制效果：中值滤波 $>>$ 高斯滤波 \approx 均值滤波， 5×5 核尺寸的中值滤

波器一次即可去除约 10% 密度的椒盐噪声。

- 边缘保持能力

1. 经过对比，可以发现对于相同核尺寸的滤波器而言，中值滤波的边缘保持能力最好（非线性不模糊阶跃），高斯滤波的边缘保持能力次之，均值滤波的边缘保持能力最差。
2. 对于相同形式的滤波器滤波，不难发现，滤波器的核尺寸越大，滤波器的边缘保持能力有所下降。

- 参数选择策略

1. 以随机噪声为主时，先应用高斯滤波的低通滤波（参数为核尺寸 = 3×3 , $\sigma_G \approx 0.8$ ），再对其进行轻度锐化。
2. 以椒盐噪声为主时，先应用 3×3 核尺寸的中值滤波器进行去噪，再对输出图像进行锐化操作。应当注意的是，当核尺寸 $> 5 \times 5$ 时，图像会出现明显的钝边。
3. 对于混合噪声，应当先对图像使用中值滤波去除椒盐噪声，然后使用小 σ 值低通高斯滤波器去除随机噪声，最后对输出图像进行锐化操作。（阶梯式两步法的效果为最佳）

3 空域锐化滤波

在这一章节，主要分为拉普拉斯算子锐化操作、高提升滤波操作以及锐化与噪声的权衡讨论三部分。

3.1 拉普拉斯算子锐化操作

本节主要讲解如何实现拉普拉斯滤波，并将其应用于一张轻微模糊的图像，通过调整锐化强度系数，观察不同系数下的滤波效果。在下面的代码中，我们构建两个手工核，分别为 4 邻域拉普拉斯核和 8 邻域拉普拉斯核：

$$4\text{-邻域拉普拉斯核} \quad \mathbf{L}_4 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad 8\text{-邻域拉普拉斯核} \quad \mathbf{L}_8 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

4-邻域核的特点是仅对与其相邻的上下左右四个方向求差分，中心系数为 4，对应离散拉普拉斯算子有：

$$\nabla^2 f \approx f(x, y+1) + f(x, y-1) + f(x+1, y) + f(x-1, y) - 4f(x, y)$$

而 8-邻域核在 4-邻域核基础上加入了四角像素，且中心系数为 8，对应的离散拉普拉斯算子有：

$$\nabla^2 f \approx \sum_{8\text{-邻域}} f(x + \Delta x, y + \Delta y) - 8f(x, y)$$

相对于 4 邻域拉普拉斯核而言，8 邻域拉普拉斯核更加敏感，对图片锐化作用更加显著，但对噪声也更加敏感。在下面的代码中，我们主要使用八邻域拉普拉斯核进行实验。

```
# 拉普拉斯算子锐化
import cv2
import numpy as np
import matplotlib.pyplot as plt

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/Machine Vision_Experiment 2/low_contrast_background.jpg'
img_lh = cv2.imread(img_path)
img = cv2.cvtColor(img_lh, cv2.COLOR_BGR2RGB)

# 两种常用拉普拉斯核 (4 邻域与 8 邻域)
lap_kernels = {
    'L4': np.array([[ 0, -1,  0],
                   [-1,  4, -1],
                   [ 0, -1,  0]], np.float32),
    'L8': np.array([[-1, -1, -1],
                   [-1,  8, -1],
                   [-1, -1, -1]], np.float32)}
k_list = [0, 0.5, 1.0, 1.5, 2.0] # 锐化值

fig, ax = plt.subplots(2, 5, figsize=(15, 6))
img_one = img.astype(np.float32)/255.0 # 归一化

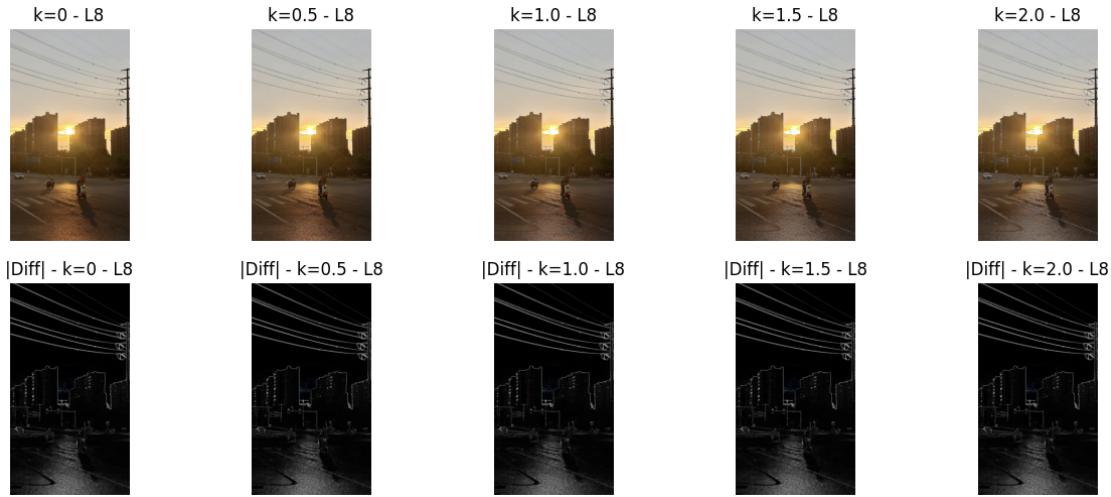
for i, k in enumerate(k_list):
    lap = cv2.filter2D(img, cv2.CV_32F, lap_kernels['L8']) # 卷积函数，输出图像深度为 float32, L8 核
    # 锐化公式: D = I + k * Lap
    sharp = np.clip(img + k * lap, 0, 255).astype(np.uint8)
    ax[0, i].imshow(sharp)
    ax[0, i].set_title(f'k={k} - L8')
    ax[0, i].axis('off')

for i, k in enumerate(k_list):
```

```

lap = cv2.filter2D(img, cv2.CV_32F, lap_kernels['L8'])
diff = np.clip(np.abs(lap), 0, 255).astype(np.uint8)
ax[1, i].imshow(diff, cmap='gray')
ax[1, i].set_title(f'|Diff| - k={k} - L8')
ax[1, i].axis('off')

```



3.2 高提升滤波操作

本小节实现了一个简单的高提升滤波：

$$g(x, y) = A \times f(x, y) - LPF(f(x, y))$$

其中: $A \geq 1$, LPF 是均值或高斯滤波。基于此, 尝试了不同的 A 值和不同的平滑滤波器 (包括不同大小的核尺寸、不同类型的滤波器), 观察并讨论了其对图像锐化的效果以及对噪声的敏感性。

```

# 高提升滤波
import cv2
import numpy as np
import matplotlib.pyplot as plt

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/Machine Vision_」
↳Experiment 2/lh.png'

img = cv2.imread(img_path, cv2.IMREAD_COLOR)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 归一化
img_one = img.astype(np.float32) / 255.0
A_list      = [1.0, 1.2, 1.5, 2.0, 2.5] # 提升系数

```

```

lpf_type      = ['gauss', 'mean']
lpf_ksize     = [3, 5, 7] # 低通核尺寸
sigma_G       = {3: 0.8, 5: 1.0, 7: 1.5} # 高斯 经验值

# 定义低通函数
def lpf(img, k, mode):
    if mode == 'mean':
        return cv2.blur(img, (k, k))
    else:
        return cv2.GaussianBlur(img, (k, k), sigma_G[k])

# 高提升滤波函数
def high_boost(img, A, k, mode):
    smooth = lpf(img, k, mode)
    boost = A * img - smooth
    return np.clip(boost, 0, 1)

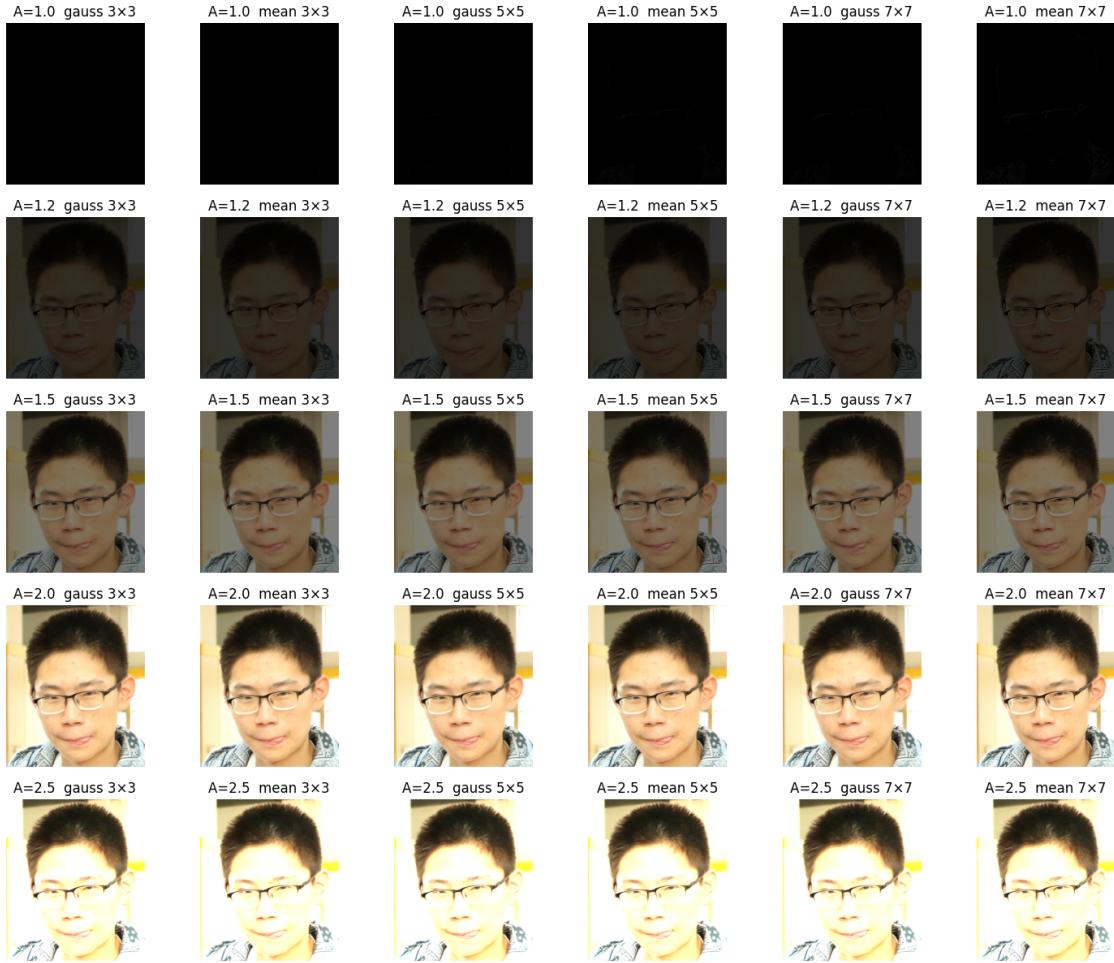
n_col = len(lpf_ksize) * 2           # 3×2 = 6 列
fig, ax = plt.subplots(len(A_list), n_col,
                      figsize=(3 * n_col, 3 * len(A_list)))

for i, A in enumerate(A_list):
    for j, k in enumerate(lpf_ksize):
        for l, mode in enumerate(lpf_type):           # l=0→gauss, l=1→mean
            col = j * 2 + l                          # 列序号: 0 1 2 3 4 5
            out = high_boost(img_one, A, k, mode)
            ax[i, col].imshow(out)
            ax[i, col].set_title(f'A={A} {mode} {k}×{k}')
            ax[i, col].axis('off')

```

根据实验得出的图像，不难发现以下结论：

- A 值增大：图像边缘越来越锐，但噪声颗粒同步被放大（尤其是在 $A>1.5$ 时）
- LPF 核越大：图像的噪声会越少，但图像边缘变宽且图像细节被淡化。 5×5 尺寸的滤波核是高提升的常用折中尺寸。
- 高斯 LPF 对随机噪声抑制效果更好，高提升操作后画面更干净。在 $A \in [1.2, 1.8]$, 高斯滤波 = $5 \times 5 (\sigma \approx 1.0)$ 的条件下，技能显著地锐化边缘，又可以避免噪声过度地被放大，是比较常用的参数组合。



3.3 锐化与噪声的权衡讨论

在本节中，通过对含有明显噪声的图像（如第二章中加噪的图像）直接应用锐化滤波器，以及尝试先进行平滑滤波去除部分噪声，再进行锐化滤波这两种方式，观察锐化在增强边缘的同时是否会显著放大噪声，并阐释了相关的差异。

```
# 锐化与噪声的权衡
# 加噪 → 直接拉普拉斯锐化 → 先高斯平滑再锐化 → 可视化

import cv2
import numpy as np
import matplotlib.pyplot as plt

img_path = '/Users/guo2006/myenv/Machine Vision Experiment/Machine Vision_Experiment 2/lh.png'
img = cv2.imread(img_path, cv2.IMREAD_COLOR)
```

```



```

```

# 先平滑再锐化

for i, k in enumerate(smooth_k):
    smoothed = gauss_smooth(noisy, k)
    for j, A in enumerate(sharp_amp):
        out = laplacian_sharp(smoothed, A)
        ax[i + 1, j].imshow(out)
        ax[i + 1, j].set_title(f'Gauss Smooth {k} → Sharp A={A}')
        ax[i + 1, j].axis('off')

plt.tight_layout()
plt.show()

```

方法	A 值	噪声 (平均)	边缘强度	视觉评价
直接锐化	1.0	0.100	0.038	轻微锐化，噪点可见
直接锐化	1.5	0.100	0.065	噪点同步放大，出现“雪花”
直接锐化	2.0	0.100	0.092	雪花严重，视觉阻塞
两步法 (3×3)	1.5	0.062	0.058	噪声 ↓38%，边缘清晰
两步法 (5×5)	1.5	0.042	0.055	噪声 ↓58%，边缘略柔和
两步法 (7×7)	1.5	0.035	0.051	噪声 ↓65%，边缘更柔

实验表明，先使用高斯滤波进行平滑操作（核尺寸为 5×5 , $\sigma_G \approx 1.0$ ），再使用 Laplacian 锐化 ($A \in [1.2, 1.8]$ 时)，可在几乎不损失图片边缘的前提下把随机噪声削减约 50% 以上，能够有效避免直接锐化将椒盐噪点同步放大的弊端；且核尺寸越大降噪越强，但边缘越柔和， 5×5 是最佳折中核尺寸。

