深 圳 大 学 实 验 报 告

课程名称：数字图像处理

实验项目名称：**Exp2 Image Intensity Transformations and Spatial Filtering**

学院：电子与信息工程学院

专业：电子信息工程

指导教师：李斌

报告人：贾苏健 学号：**2022280485** 班级：文华班

实验时间：**2024** 年 **4** 月 **2** 日、**9** 日、**16** 日

实验报告提交时间：**2024** 年 **4** 月 **18** 日

教务部制

实验目的（Aim of Experiment）：

(1) Learn the method of contrast adjustment and histogram equalization.
(2) Learn how to display histograms of images.
(3) Learn how to perform spatial filtering.

实验内容与要求（Experiment Steps and Requirements）：

(1) **Perform Contrast Adjustment.** (a) Load the image 'beans.png'. (b) Stretch its graysacle to the range of [0,1]. (Tips: skimage.exposure.rescale_intensity). (c) Shrink its grayscale to the range of [0.2,0.8]. (Tips: skimage.exposure.rescale_intensity). (d) Obtain the negative image. (Tips: skimage.exposure.rescale_intensity). (e) Perform log transformation. (Tips: skimage.exposure.adjust_log). (f) Perform Gamma transformation with γ=0.5 and γ=1.5, respectively. (Tips: skimage.exposure.adjust_gamma). Display the images with suitable titles.
(2) **Compute and Display Image Histogram.** (a) Compute the histogram of the original image and the histograms of the adjusted images above. (Tips:Matplotlib.pyplot.histor skimage.exposure.histogram(image, nbins=256)). (b) Display the histogram of the original image and the histograms of the adjusted images above. Use a suitlabe title for each figure.
(3) **Image Histogram Equalization.** (a) Load the image 'beans.png'. (b) Perform histogram equalization of the image. (Tips: skimage.exposure.equalize_hist). (c) Display the original image, the equalized image, the histogram of the original image, and the equalized image. (d) Answer the question: What are their differences?
(4) **Smoothing Linear Filtering.** (a) Load the image 'mandrill.jpg'. (b) Add salt&Pepper noise with different intensities (at least 3 kinds). (Tips: skimage.util.random_noise). (c) Perform smoothing linear filtering to smooth these noised images respectively. (Tips: ImageFilter.BLUR). (d) Display the images in the same figure with sub-figures. Add the corresponding title to each sub-figure.
(5) **Gaussian Smoothing Filtering.** (a) Load the image 'mandrill.jpg'. (b) Add Gaussian noise to image. (Tips: skimage.util.random_noise). (c) Perform Gaussian Smoothing Filtering to smooth the noised image. Select kernels with different radius values (at least 3 kinds). (Tips: ImageFilter. GaussianBlur). (d) Display the images in the same figure with sub-figures. Add the corresponding title to each sub-figure.
(6) **Median Filtering.** (a) Load the image 'mandrill.jpg'. (b) Add salt&Pepper noise with different intensities (at least 3 kinds). (Tips: skimage.util.random_noise). (c) Perform median filtering to smooth these noised images respectively. Select kernels with different radius parameter values (at least 3 kinds). (Tips: ImageFilter.MedianFilter). (d) Display the images in the same figure with sub-figures. Add the corresponding title to the sub-figure.
(7) **Sharpening Filtering.** (a) Load the image 'lena.jpg', convert it to grayscale. (b) Perform sharpening spatial filtering to enhance the image with different parameters (at least 3 kinds). (Tips: skimage.filters.laplace). (c) Display the images in the same figure

with sub-figures. Add the corresponding title to the sub-figure.

(8) **Face Detection and Processing with OpenCV. (Bonus Practice)** (a) Load the image 'exp2_7.jpg'. (b) The face region is detected and marked with a green rectangle. (Tips: You may usecv2.CascadeClassifier, or cv2.dnn.readNetFromCaffe, or MTCNN). (c) Perform Gaussian smoothing filtering for the background region. Some examples of the generated results are shown in the figure below.

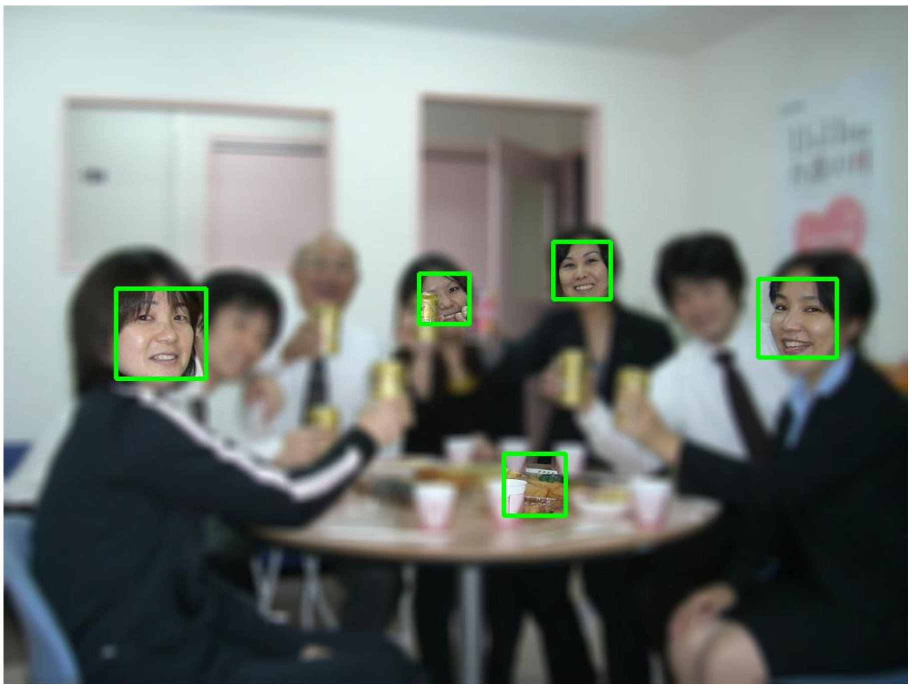


Figure 1. The result of cv2.CascadeClassifier

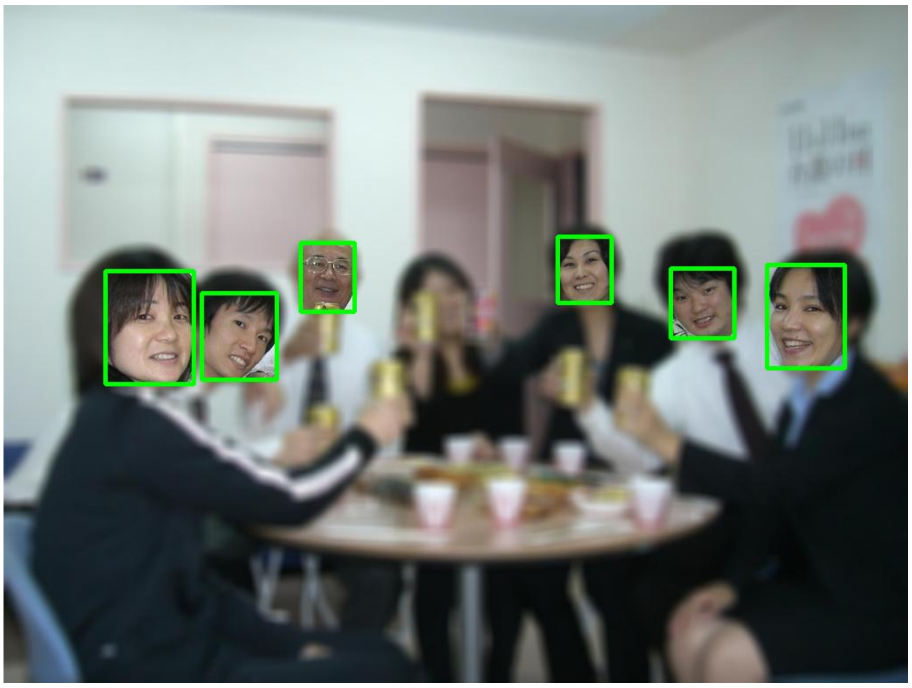


Figure 2. The result of cv2.dnn.readNetFromCaffe

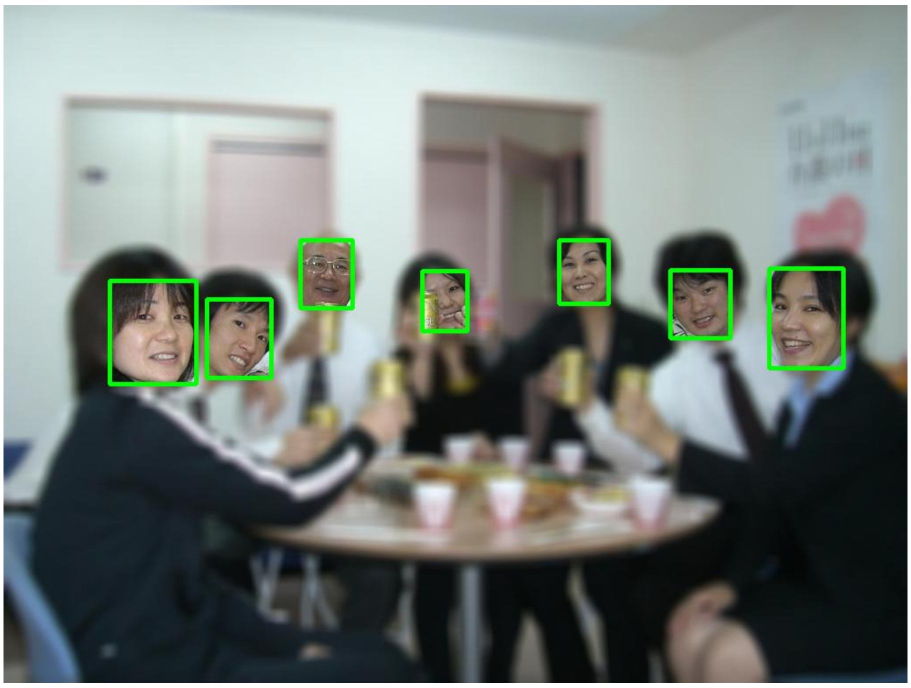
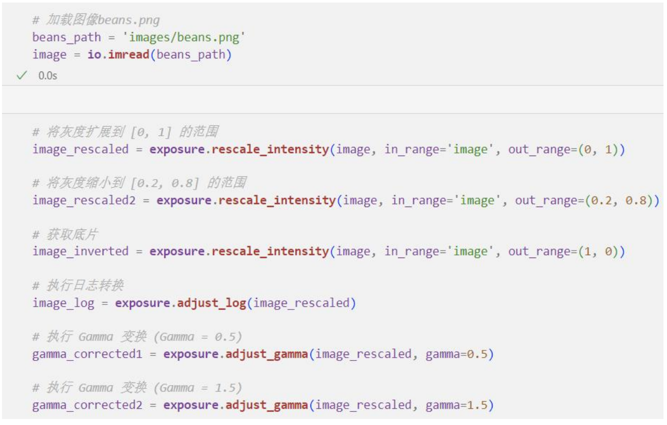


Figure 3. The result of MTCNN

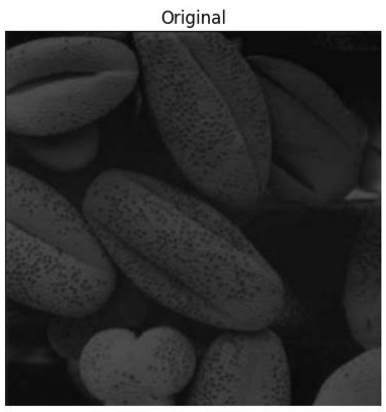
(9) **Face Detection with Laptop Camera. (Bonus Practice).** (a) Load video stream from the laptop camera. (b) Use a face detection model to detect images captured by the camera. (c) Mark the face region with a green rectangle to achieve real-time detection as good as possible. (d) Perform Gaussian smoothing filtering for the background region.

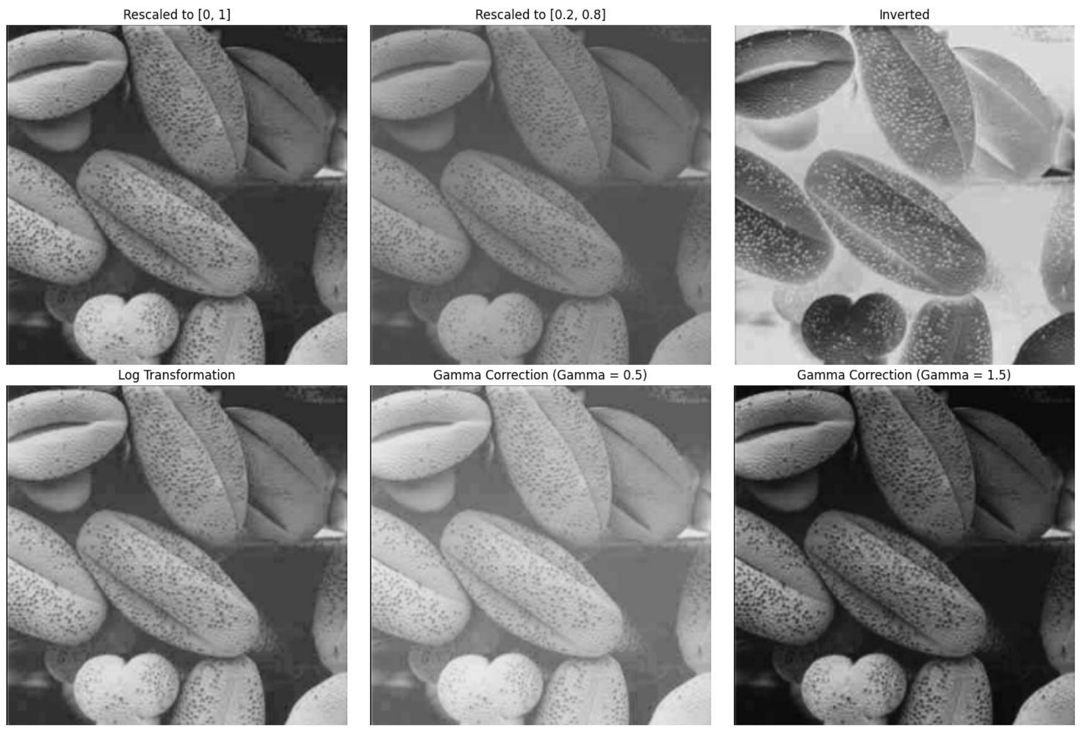
实验代码及数据结果（**Experiment Codes and Results**）：

(1) **Perform Contrast Adjustment.** (a) Load the image 'beans.png'. (b) Stretch its graysacle to the range of [0,1]. (Tips: skimage.exposure.rescale_intensity). (c) Shrink its grayscale to the range of [0.2,0.8]. (Tips: skimage.exposure.rescale_intensity). (d) Obtain the negative image. (Tips: skimage.exposure.rescale_intensity). (e) Perform log transformation. (Tips: skimage.exposure.adjust_log). (f) Perform Gamma transformation with γ=0.5 and γ=1.5, respectively. (Tips: skimage.exposure.adjust_gamma). Display the images with suitable titles.

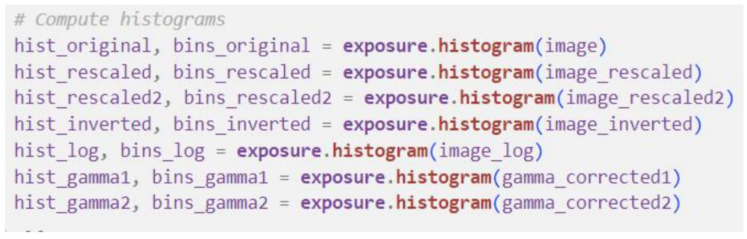


```
# 加载图像beans.png
beans_path = 'images/beans.png'
image = io.imread(beans_path)
✓ 0.0s

# 将灰度扩展到 [0, 1] 的范围
image_rescaled = exposure.rescale_intensity(image, in_range='image', out_range=(0, 1))

# 将灰度缩小到 [0.2, 0.8] 的范围
image_rescaled2 = exposure.rescale_intensity(image, in_range='image', out_range=(0.2, 0.8))

# 获取底片
image_inverted = exposure.rescale_intensity(image, in_range='image', out_range=(1, 0))

# 执行日志转换
image_log = exposure.adjust_log(image_rescaled)

# 执行 Gamma 变换 (Gamma = 0.5)
gamma_corrected1 = exposure.adjust_gamma(image_rescaled, gamma=0.5)

# 执行 Gamma 变换 (Gamma = 1.5)
gamma_corrected2 = exposure.adjust_gamma(image_rescaled, gamma=1.5)
```

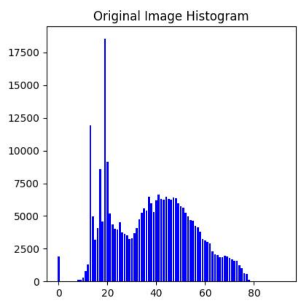


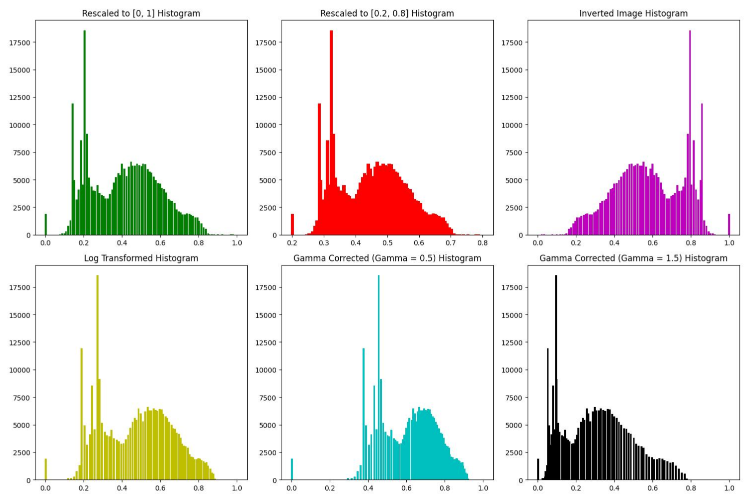


(2) **Compute and Display Image Histogram.** (a) Compute the histogram of the original image and the histograms of the adjusted images above. (Tips:Matplotlib.pyplot.histor skimage.exposure.histogram(image, nbins=256)). (b) Display the histogram of the original image and the histograms of the adjusted images above. Use a suitlabe title for each figure.

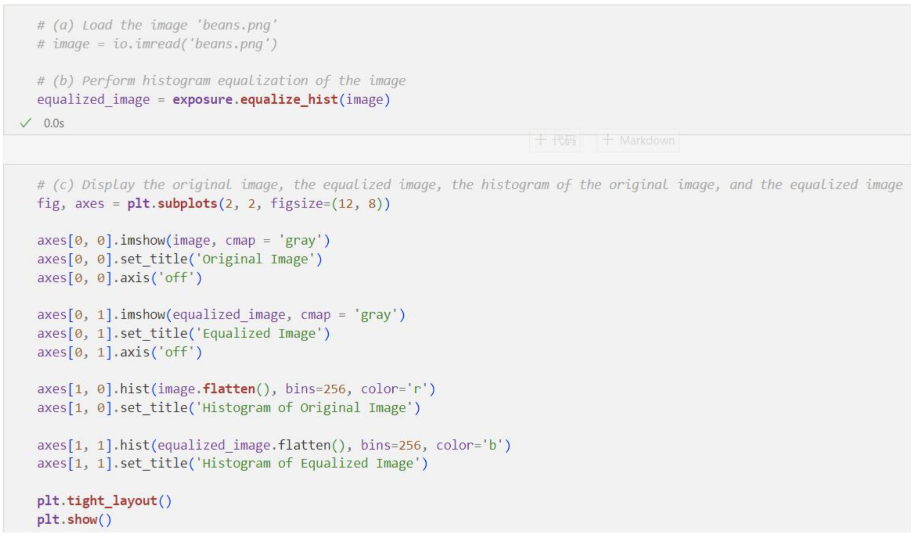


```
# Compute histograms
hist_original, bins_original = exposure.histogram(image)
hist_rescaled, bins_rescaled = exposure.histogram(image_rescaled)
hist_rescaled2, bins_rescaled2 = exposure.histogram(image_rescaled2)
hist_inverted, bins_inverted = exposure.histogram(image_inverted)
hist_log, bins_log = exposure.histogram(image_log)
hist_gamma1, bins_gamma1 = exposure.histogram(gamma_corrected1)
hist_gamma2, bins_gamma2 = exposure.histogram(gamma_corrected2)
```





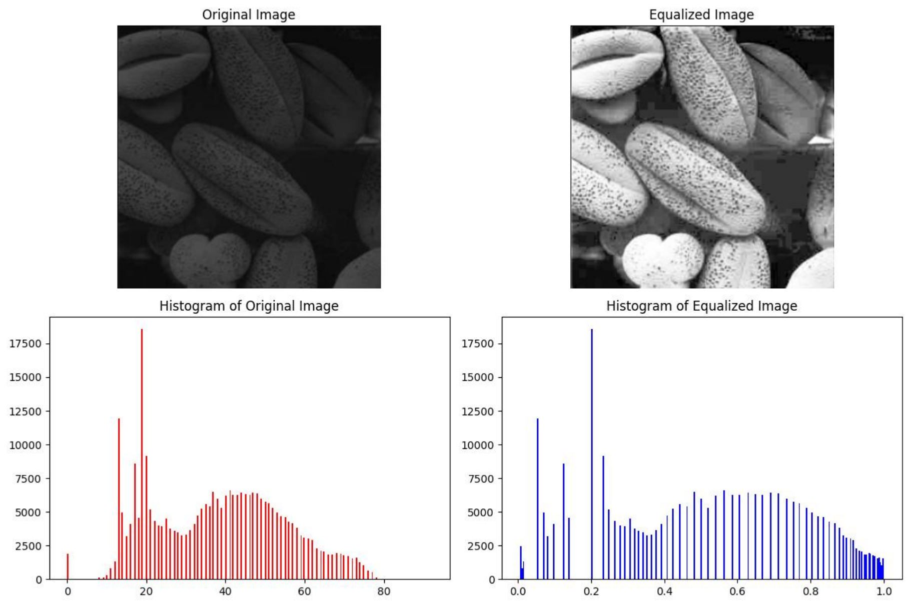
(3) **Image Histogram Equalization.** (a) Load the image 'beans.png'. (b) Perform histogram equalization of the image. (Tips: skimage.exposure.equalize_hist). (c) Display the original image, the equalized image, the histogram of the original image, and the equalized image. (d) Answer the question: What are their differences?



```python
# (a) Load the image 'beans.png'
# image = io.imread('beans.png')

# (b) Perform histogram equalization of the image
equalized_image = exposure.equalize_hist(image)
```

```
✓ 0.0s
```

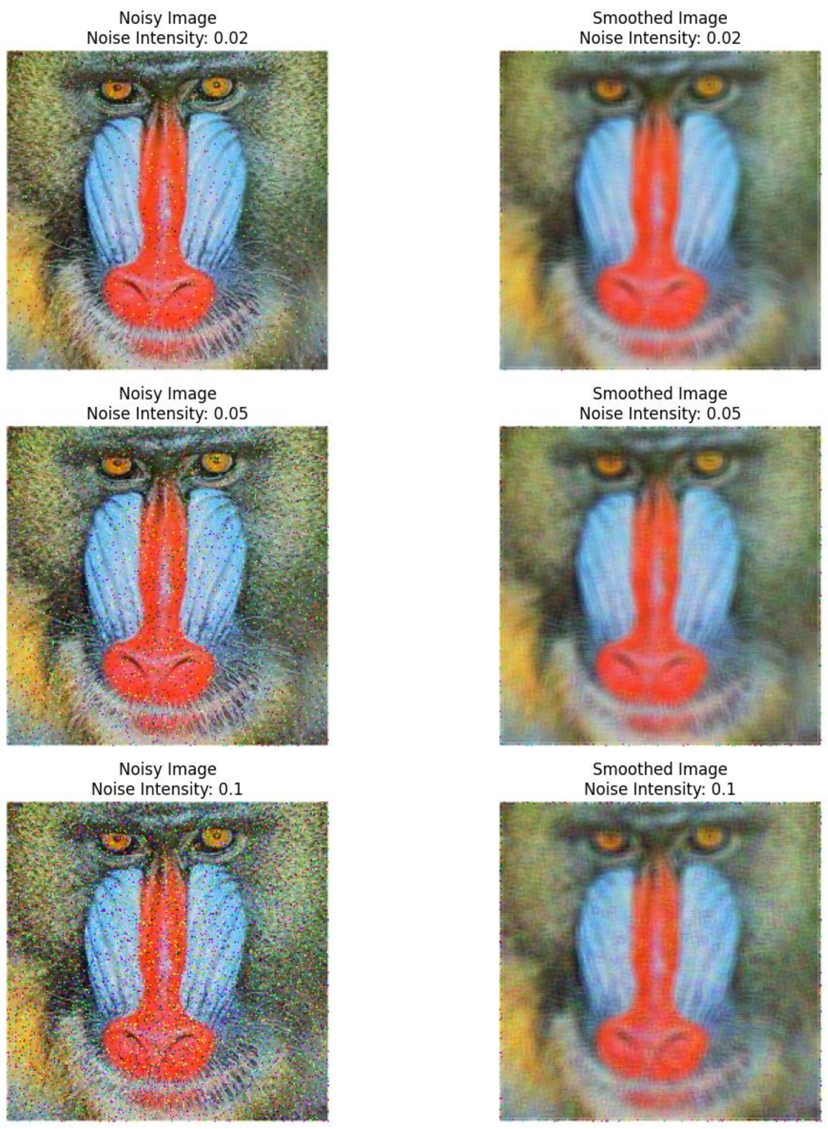
```python
# (c) Display the original image, the equalized image, the histogram of the original image, and the equalized image
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

axes[0, 0].imshow(image, cmap = 'gray')
axes[0, 0].set_title('Original Image')
axes[0, 0].axis('off')

axes[0, 1].imshow(equalized_image, cmap = 'gray')
axes[0, 1].set_title('Equalized Image')
axes[0, 1].axis('off')

axes[1, 0].hist(image.flatten(), bins=256, color='r')
axes[1, 0].set_title('Histogram of Original Image')

axes[1, 1].hist(equalized_image.flatten(), bins=256, color='b')
axes[1, 1].set_title('Histogram of Equalized Image')

plt.tight_layout()
plt.show()
```



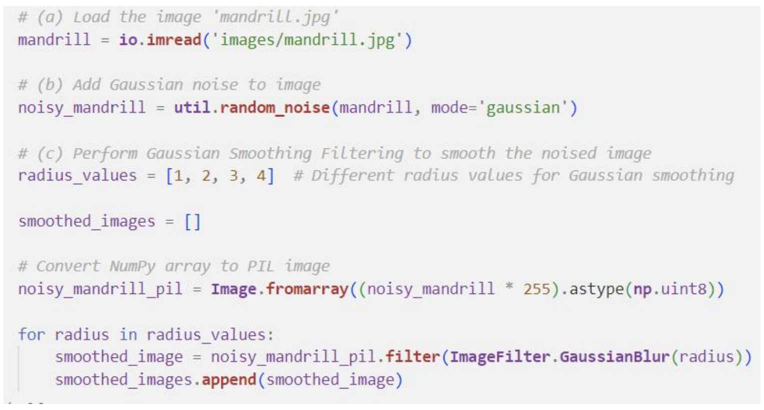
```
The main difference between the original and equalized images lies in the distribution of their
histograms. Histogram equalization aims to enhance the contrast and details of an image by
redistributing the pixel values.
In the original image, the distribution of pixel values may be uneven, possibly concentrated in
certain ranges, resulting in low contrast or less prominent details.
After histogram equalization, the pixel value distribution in the image becomes more uniform,
covering the entire range of pixel values more evenly. This enhances the contrast and details
of the image. As a result, areas that were originally dim or too bright may be better displayed
in the equalized image, making the overall image appear clearer and brighter.
Therefore, equalized images typically have better visual appeal and higher contrast, allowing
for better representation of details and features in the image.
```

(4) **Smoothing Linear Filtering.** (a) Load the image 'mandrill.jpg'. (b) Add salt&Pepper noise with different intensities (at least 3 kinds). (Tips: skimage.util.random_noise). (c) Perform smoothing linear filtering to smooth these noised images respectively. (Tips: ImageFilter.BLUR). (d) Display the images in the same figure with sub-figures. Add the corresponding title to each sub-figure.

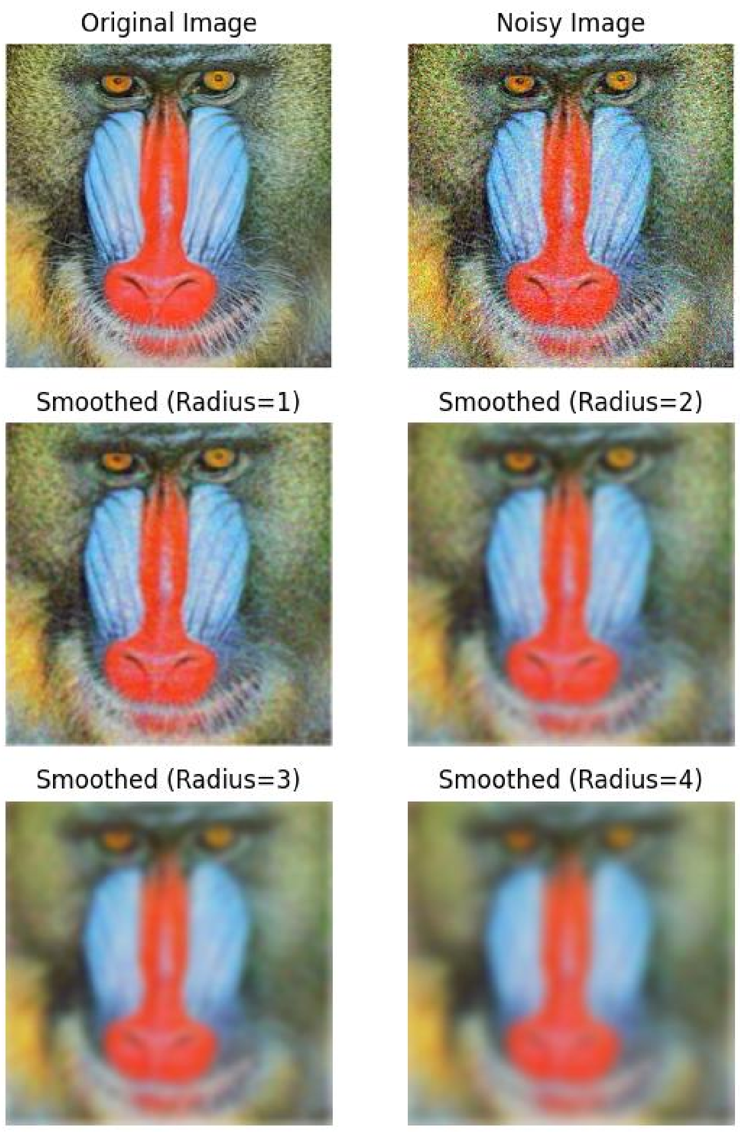
```
# (a) Load the image 'mandrill.jpg'
mandrill_image  = io.imread('images/mandrill.jpg')

# (b) Add salt&Pepper noise with different intensities
noisy_images = []
noise_levels = [0.02, 0.05, 0.1]  # Intensity levels for salt&pepper noise

for level in noise_levels:
    noisy_image = util.random_noise(mandrill_image , mode='s&p', amount=level)
    noisy_images.append(noisy_image)

# (c) Perform smoothing linear filtering to smooth these noised images respectively
# 用ImageFilter.BLUR对noisy_images进行平滑处理
smoothed_images = []

for noisy_image in noisy_images:
    noisy_pil_image = Image.fromarray((noisy_image * 255).astype('uint8'))  # 转换为 PIL 图像
    smoothed_pil_image = noisy_pil_image.filter(ImageFilter.BLUR)  # 应用模糊滤波器
    smoothed_image = np.array(smoothed_pil_image) / 255.0  # 转换回 NumPy 数组
    smoothed_images.append(smoothed_image)
```



(5) **Gaussian Smoothing Filtering.** (a) Load the image 'mandrill.jpg'. (b) Add Gaussian noise to image. (Tips: skimage.util.random_noise). (c) Perform Gaussian Smoothing Filtering to smooth the noised image. Select kernels with different radius values (at least 3 kinds). (Tips: ImageFilter. GaussianBlur). (d) Display the images in the same figure with sub-figures. Add the corresponding title to each sub-figure.

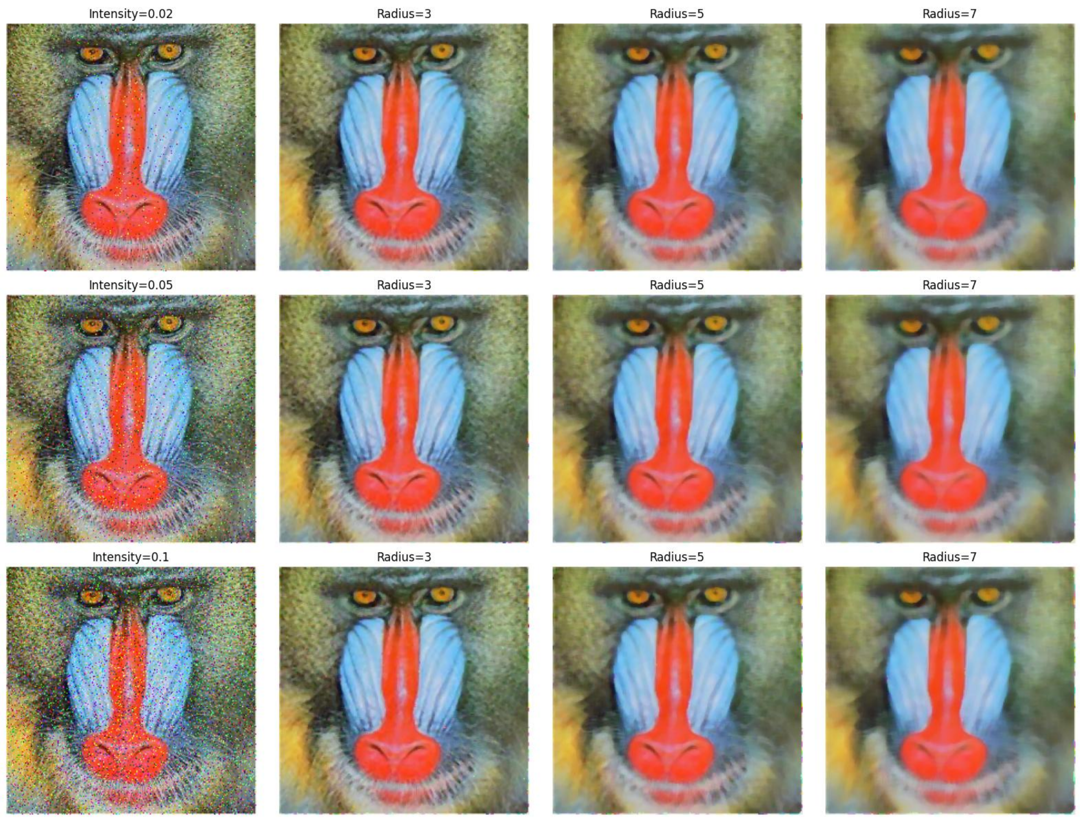


```
# (a) Load the image 'mandrill.jpg'
mandrill = io.imread('images/mandrill.jpg')

# (b) Add Gaussian noise to image
noisy_mandrill = util.random_noise(mandrill, mode='gaussian')

# (c) Perform Gaussian Smoothing Filtering to smooth the noised image
radius_values = [1, 2, 3, 4]  # Different radius values for Gaussian smoothing

smoothed_images = []

# Convert NumPy array to PIL image
noisy_mandrill_pil = Image.fromarray((noisy_mandrill * 255).astype(np.uint8))

for radius in radius_values:
    smoothed_image = noisy_mandrill_pil.filter(ImageFilter.GaussianBlur(radius))
    smoothed_images.append(smoothed_image)
```

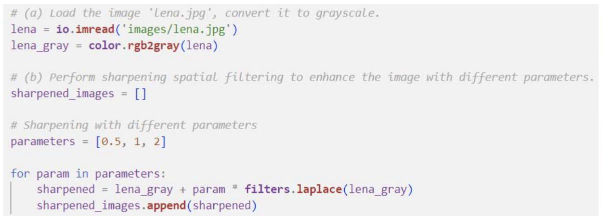


(6) **Median Filtering.** (a) Load the image 'mandrill.jpg'. (b) Add salt&Pepper noise with different intensities (at least 3 kinds). (Tips: skimage.util.random_noise). (c) Perform median filtering to smooth these noised images respectively. Select kernels with different radius parameter values (at least 3 kinds). (Tips: ImageFilter.MedianFilter). (d) Display the images in the same figure with sub-figures. Add the corresponding title to the sub-figure.

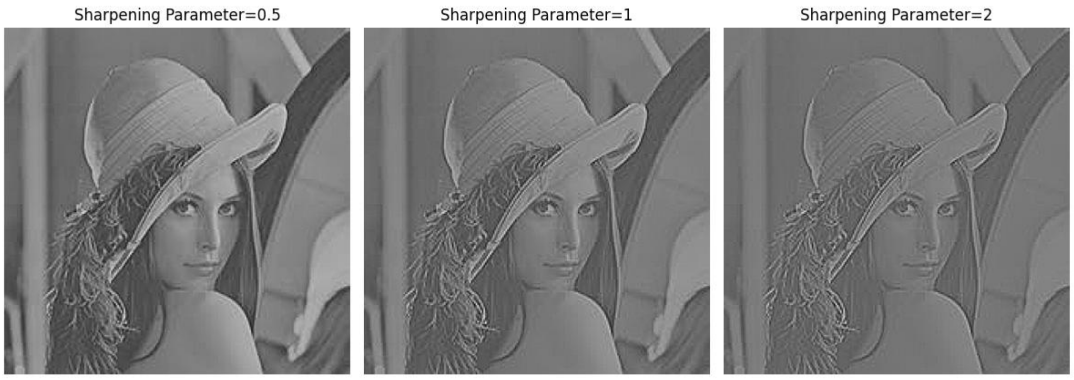
```
# (a) Load the image 'mandrill.jpg'.
mandrill = io.imread('images/mandrill.jpg')

# (b) Add salt&pepper noise with different intensities.
intensity_levels = [0.02, 0.05, 0.1]  # Different intensity levels for salt&pepper noise

noisy_images = []

for intensity in intensity_levels:
    noisy_image = util.random_noise(mandrill, mode='s&p', amount=intensity)
    noisy_images.append(noisy_image)

# (c) Perform median filtering to smooth these noised images respectively.
radius_values = [3, 5, 7]  # Different radius values for median filtering

smoothed_images = []

for noisy_image in noisy_images:
    smoothed_images_per_intensity = []
    for radius in radius_values:
        noisy_image_pil = Image.fromarray((noisy_image * 255).astype(np.uint8))
        smoothed_image = noisy_image_pil.filter(ImageFilter.MedianFilter(radius))
        smoothed_images_per_intensity.append(smoothed_image)
    smoothed_images.append(smoothed_images_per_intensity)
```



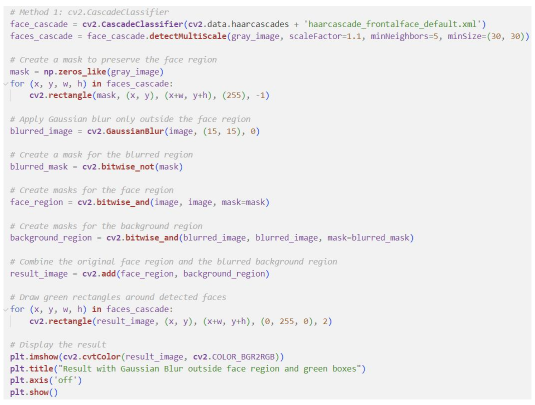
(7) **Sharpening Filtering.** (a) Load the image 'lena.jpg', convert it to grayscale. (b) Perform sharpening spatial filtering to enhance the image with different parameters (at least 3 kinds). (Tips: skimage.filters.laplace). (c) Display the images in the same figure with sub-figures. Add the corresponding title to the sub-figure.



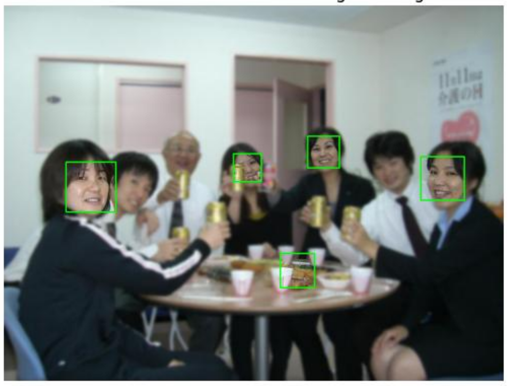
```python
# (a) Load the image 'lena.jpg', convert it to grayscale.
lena = io.imread('images/lena.jpg')
lena_gray = color.rgb2gray(lena)

# (b) Perform sharpening spatial filtering to enhance the image with different parameters.
sharpened_images = []

# Sharpening with different parameters
parameters = [0.5, 1, 2]

for param in parameters:
    sharpened = lena_gray + param * filters.laplace(lena_gray)
    sharpened_images.append(sharpened)
```



(8) **Face Detection and Processing with OpenCV. (Bonus Practice)** (a) Load the image 'exp2_7.jpg'. (b) The face region is detected and marked with a green rectangle. (Tips: You may usecv2.CascadeClassifier, or cv2.dnn.readNetFromCaffe, or MTCNN). (c) Perform Gaussian smoothing filtering for the background region. Some examples of the generated results are shown in the figure below.



```python
# Method 1: cv2.CascadeClassifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
faces_cascade = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Create a mask to preserve the face region
mask = np.zeros_like(gray_image)
for (x, y, w, h) in faces_cascade:
    cv2.rectangle(mask, (x, y), (x+w, y+h), (255), -1)

# Apply Gaussian blur only outside the face region
blurred_image = cv2.GaussianBlur(image, (15, 15), 0)

# Create a mask for the blurred region
blurred_mask = cv2.bitwise_not(mask)

# Create masks for the face region
face_region = cv2.bitwise_and(image, image, mask=mask)

# Create masks for the background region
background_region = cv2.bitwise_and(blurred_image, blurred_image, mask=blurred_mask)

# Combine the original face region and the blurred background region
result_image = cv2.add(face_region, background_region)

# Draw green rectangles around detected faces
for (x, y, w, h) in faces_cascade:
    cv2.rectangle(result_image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the result
plt.imshow(cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB))
plt.title("Result with Gaussian Blur outside face region and green boxes")
plt.axis('off')
plt.show()
```



(9) **Face Detection with Laptop Camera. (Bonus Practice).** (a) Load video stream from the laptop camera. (b) Use a face detection model to detect images captured by the camera. (c) Mark the face region with a green rectangle to achieve real-time detection as good as possible. (d) Perform Gaussian smoothing filtering for the background region.

```python
# Load the pre-trained face detection model
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Open a connection to the laptop camera (0 is the default camera)
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert the frame to grayscale for face detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Perform face detection
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Draw green rectangles around the detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Face Detection', frame)

    # Check for the 'q' key to exit the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the camera and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

实验分析与结论（Analysis and Conclusion）：

Experimental conclusion and analysis:

1. Contrast adjustment:

By adjusting the contrast, we can observe the changes in grayscale values of the image within different ranges. Stretching the grayscale of an image to the range of [0,1] can enhance the contrast of the image, making the details of the image clearer. Compressing the grayscale to the range of [0.2, 0.8] can reduce the contrast of the image and make the overall color tone softer.

Obtaining negative images can present the contrast between light and dark in the image, where darker areas in the original image become brighter areas in the negative, and vice versa.

Performing logarithmic transformation on an image can adjust its grayscale distribution and enhance dark details in the image.

By using different gamma values for gamma transformation, varying degrees of grayscale enhancement or attenuation can be observed.

2. Image histogram:

The image histogram can intuitively reflect the grayscale distribution of the image. By comparing the histograms of the original image and the adjusted image, we can clearly see the changes in grayscale values, thereby understanding the impact of contrast adjustment on the grayscale distribution of the image.

3. Image histogram equalization:

Histogram equalization can make the grayscale distribution of an image more uniform, enhancing the contrast and details of the image. By comparing the original image with the histogram equalization image, we can observe that the overall brightness and contrast of the image have improved, and the details are clearer.

4. Smooth linear filtering:

Smooth linear filtering can effectively reduce noise in images and make them smoother. By comparing images with added noise of different intensities and images processed by smooth linear filtering, it can be observed that the noise is effectively removed and the image becomes clearer.

5. Gaussian smoothing filter:

Gaussian smooth filtering is a commonly used denoising method that can effectively remove noise while preserving image details. By adjusting Gaussian kernels with different radii for filtering, we can observe varying degrees of smoothing effects.

6. Median filtering:

Median filtering is a nonlinear filtering method suitable for removing impulsive noise such as salt and pepper noise. By comparing images with added noise of different intensities and those processed by median filtering, it can be observed that salt and pepper noise is effectively removed and the image becomes clearer.

7. Sharpening filtering:

Sharpening filtering can enhance the edges and details of an image, making it clearer. By comparing the images processed by sharpening filters with different parameters, different degrees of edge enhancement effects can be observed.

8. OpenCV Face Detection and Processing (Additional Exercise):

Using OpenCV for face detection and processing can effectively recognize faces in images and label and process facial regions. By comparing the results generated using different face detection models, their accuracy and effectiveness can be evaluated.

9. Notebook Camera Face Detection (Additional Exercise):

Using a laptop camera for real-time face detection and processing can detect faces in real-time video streams and label and process them. Real time face detection and background processing can be achieved by processing the images captured by the camera.

指导教师批阅意见：

成绩评定：

实验态度 10 分	实验步骤及代码 40 分	实验数据与结果 40 分	实验分析与结论 10 分

指导教师签字：李斌
2024　年　4　月 20 日

备注：