深 圳 大 学 实 验 报 告

课程名称：数字图像处理

实验项目名称：**Exp1 Basic Operations and Algebraic Operators for Digital Images**

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：06

实验时间：2024 年 3 月 5 日、12 日、19 日、26 日

实验报告提交时间：2024 年 3 月 27 日

教务部制

实验目的（Aim of Experiment）：

(1) Establish a Python environment and install some digital image processing libraries such as scikit-image, OpenCV, PIL, and matplotlib.

(2) Learn how to load, display, and save images.

(3) Be familiar with some basic image processing operations such as adding noise, image type conversion, image file format conversion, etc.

(4) Learn how to perform algebraic operations on digital images.

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

(1) **Load, Save and Display Images.**

(a) Load a PNG image with Scikit-Image. (Tips: io.imread).

(b) Convert it to PIL image format. (Tips: Image.fromarray).

(c) Display this image with Matplotlib.

(d) Convert PIL image format to OpenCV image format, save as a JPEG image with a quality factor of 90. (Tips: cv2.imwrite).

(2) **Display Three Individual Color Components of RGB Image.**

(a) Load an image with OpenCV.

(b) Display the R, G, and B color component of image, respectively. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(c) Answer the question: What are the differences in R, G, and B color components?

(3) **Convert Color Image to Grayscale.**

(a) Display the original Lena image and the grayscale images obtained by three grayscaling methods in the same figure. Add the corresponding title.

a1) Maximum of the three components;

a2) The average of the three components;

a3) gray=0.30r + 0.59g + 0.11*b.

(b) Answer the question: What are their differences?

(4) **Image Cropping.**

(a) Load a RGB image.

(b) Select the 128x128 central region of the image.

(c) Display the full image and its central part.

(d) Save the central part as an image file in the same format as the full image.

(5) **Adding Noise to Image.**

(a) Load an RGB image in with Scikit-Image.

(b) Add Gaussian additive noise, salt noise, pepper noise, salt and pepper noise, or speckle noise to it. The parameters can be chosen by yourself. (Tips: You may use random_noise in the util module of Scikit-Image).

(c) Display these six images in the same figure and add the corresponding title.

(6) **Image Denoising by Averaging.**

(a) Load an RGB image.

(b) Add Gaussian noise with a mean value of 0 and a variance of 0.1 to it.

(c) Display and compare the images before and after adding noise.

(d) Use the for loop to add 3, 30, and 300 images with random Gaussian noise and find their average value, respectively.

(7) **Image Algebraic Operations.**

(a) Download two pictures by yourself, and load these two pictures with OpenCV.

(b) Perform algebraic operations of addition and subtraction (Tips: Pay attention to the size and type of the image during the calculation. If they are different, the larger image should be cropped or scaled).

(c) Display the images before and after processing.

(8) **Text adding.**

(a) Load the Lena image with OpenCV.

(b) Employ a red rectangle to mark the 64x64 rectangle in the middle of image. (Tips: You may use cv2.rectangle)

(c) Adding some black text on it. (Tips: You may usecv2.putText). An example of the generated result is shown in the figure below.

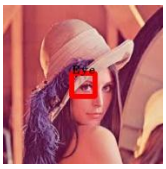


Figure 1. The result of adding content to Lean image

(9) **Adding image mask.**

(a) Load the Lena image with OpenCV.

(b) Adding a circular mask on the Lena image. (Tips: You may usenumpy.ogrid). An example of the generated result is shown in the figure below.

Figure 2. The result of adding mask to Lean image

(10)**Capture Images by Laptop Camera with OpenCV. (Bonus Practice).**

(a) Read the video stream from the laptop camera.

(b) Implement Time-lapse photography.(It can be used to record the whole process of the evaporation of water droplets in the cup. Tips: You may usetime.sleep).

(c) Generate corresponding MP4 video or GIF image.

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

(1) **Load, Save and Display Images.**

(a) Load a PNG image with Scikit-Image. (Tips: io.imread).

(b) Convert it to PIL image format. (Tips: Image.fromarray).

(c) Display this image with Matplotlib.

(d) Convert PIL image format to OpenCV image format, save as a JPEG image with a quality factor of 90. (Tips: cv2.imwrite).

Code：

```python
# (a) Load the PNG image with Scikit-Image
image_path = 'images/bunny.png'
image_skimage = io.imread(image_path)

# (b) Convert it to PIL image format
image_pil = Image.fromarray(image_skimage)

# (c) Display the image with Matplotlib
plt.imshow(image_pil)
plt.axis('off')
plt.show()

# (d) Convert PIL image format to OpenCV image format
image_cv2 = cv2.cvtColor(image_skimage, cv2.COLOR_RGBA2BGR)

# Save it as a JPEG image with quality factor of 90
output_jpeg_path = 'results/bunny.jpeg'
cv2.imwrite(output_jpeg_path, image_cv2, [cv2.IMWRITE_JPEG_QUALITY, 90])

# (e) Load the JPEG image saved above
image_jpeg = cv2.imread(output_jpeg_path)

# Save it as a BMP format image
output_bmp_path = 'results/bunny.bmp'
cv2.imwrite(output_bmp_path, image_jpeg)

# Save it as a TIFF format image
output_tiff_path = 'results/bunny.tiff'
cv2.imwrite(output_tiff_path, image_jpeg)
```

Result：

bunny.bmp　bunny.jpeg　bunny.tiff

(2) **Display Three Individual Color Components of RGB Image.**

(a) Load an image with OpenCV.

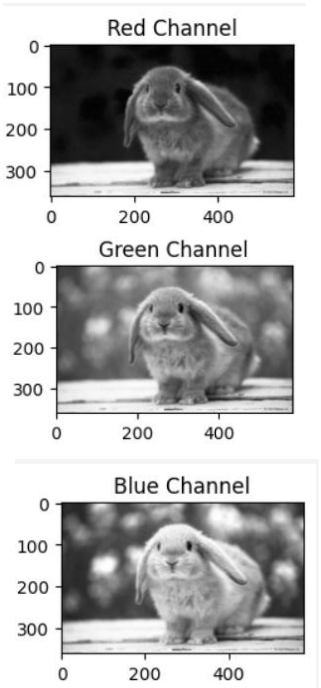
(b) Display the R, G, and B color component of image, respectively. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(c) Answer the question: What are the differences in R, G, and B color components?

Code：

```python
# Load an image with OpenCV
bunny = cv2.imread(output_jpeg_path)

# Split the image into its RGB components
bunny_r, bunny_g, bunny_b = cv2.split(bunny)

# Display the RGB image
plt.figure(figsize=(10, 5))
plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(bunny, cv2.COLOR_BGR2RGB))
plt.title('RGB Image')

# Display the R component
plt.subplot(1, 4, 2)
plt.imshow(bunny_r, cmap='gray')
plt.title('Red Channel')

# Display the G component
plt.subplot(1, 4, 3)
plt.imshow(bunny_g, cmap='gray')
plt.title('Green Channel')

# Display the B component
plt.subplot(1, 4, 4)
plt.imshow(bunny_b, cmap='gray')
plt.title('Blue Channel')

# Adjust layout to prevent overlapping of titles
plt.tight_layout()

# Show the plot
plt.show()
```

Result：



The main differences between the RGB image and its individual R/G/B components lie in the information each component carries:

1. RGB Image: This is the composite image created by combining the red (R), green (G), and blue (B) color channels. It represents the full-color image as perceived by human vision, where each pixel's color is a combination of varying intensities of red, green, and blue.

2. Red (R) Component: This component represents the intensity of red in the image. In the red component image, areas with high red intensity appear bright, while areas with low or no red intensity appear dark. The red component highlights elements in the image that are predominantly red in color.

3. Green (G) Component: Similar to the red component, the green component represents the intensity of green in the image. In the green component image, areas with high green intensity appear bright, while areas with low or no green intensity appear dark. The green component highlights elements in the image that are predominantly green in color.

4. Blue (B) Component: The blue component represents the intensity of blue in the image. In the blue component image, areas with high blue intensity appear bright, while areas with low or no blue intensity appear dark. The blue component highlights elements in the image that are predominantly blue in color.

In summary, the RGB image represents the full-color information of the image, while the individual R/G/B components isolate and highlight specific color information corresponding to red, green, and blue channels respectively.

(3) **Convert Color Image to Grayscale.**

(a) Display the original Lena image and the grayscale images obtained by three grayscaling methods in the same figure. Add the corresponding title.

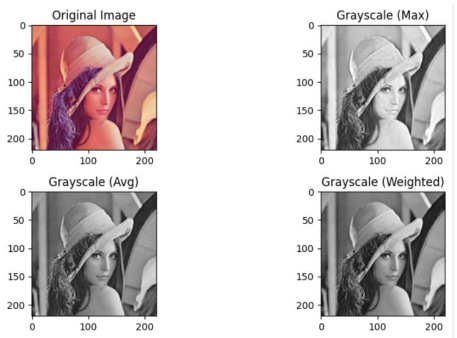
a1) Maximum of the three components;

a2) The average of the three components;

a3) gray=0.30r + 0.59g + 0.11*b.

(b) Answer the question: What are their differences?

```
# Load the Lena image
lena = cv2.imread('images/lena.jpg')

# Convert the Lena image to grayscale using different methods
# Method 1: Maximum of the three components
gray_max = np.maximum(np.maximum(lena[..., 0], lena[..., 1]), lena[..., 2])

# Method 2: Average of the three components
gray_avg = np.mean(lena, axis=2)

# Method 3: Weighted sum method
gray_weighted = 0.30 * lena[..., 2] + 0.59 * lena[..., 1] + 0.11 * lena[..., 0]
```



The differences between these methods are in how they calculate the grayscale values:

Maximum of the three components: It takes the maximum value among the three RGB components for each pixel. This method tends to preserve the brighter parts of the image.

Average of the three components: It calculates the average value of the three RGB components for each pixel. This method provides a balanced representation of the image.

Weighted sum method: It uses specific weights for each RGB component based on the luminosity perception of human vision. This method emphasizes the green component while also considering the contribution of red and blue components.

(4) **Image Cropping.**

(a) Load a RGB image.

(b) Select the 128x128 central region of the image.

(c) Display the full image and its central part.

(d) Save the central part as an image file in the same format as the full image.

```python
# Load the RGB image
image_path = 'images/lena.jpg'
full_image = cv2.imread(image_path)

# Get the dimensions of the full image
height, width = full_image.shape[:2]

# Calculate the coordinates for the central region
center_x = width // 2
center_y = height // 2
half_size = 64  # Half of the size (128x128)

# Define the region of interest (ROI) for cropping
x1 = center_x - half_size
y1 = center_y - half_size
x2 = center_x + half_size
y2 = center_y + half_size

# Crop the central region from the full image
cropped_image = full_image[y1:y2, x1:x2]

# Display the full image and its central part
cv2.imshow('Full Image', full_image)
cv2.imshow('Central Part', cropped_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Save the central part as an image file
output_path = 'results/cropped_lena.jpg'
cv2.imwrite(output_path, cropped_image)
```

(5) **Adding Noise to Image.**

(a) Load an RGB image in with Scikit-Image.

(b) Add Gaussian additive noise, salt noise, pepper noise, salt and pepper noise, or speckle noise to it. The parameters can be chosen by yourself. (Tips: You may use random_noise in the util module of Scikit-Image).

(c) Display these six images in the same figure and add the corresponding title.

```python
# Add different types of noise to the image
gaussian_noise = util.random_noise(original_image, mode='gaussian', mean=0, var=0.01)
salt_noise = util.random_noise(original_image, mode='salt', amount=0.02)
pepper_noise = util.random_noise(original_image, mode='pepper', amount=0.02)
salt_pepper_noise = util.random_noise(original_image, mode='s&p', amount=0.02)
speckle_noise = util.random_noise(original_image, mode='speckle', mean=0, var=0.01)
```

(6) **Image Denoising by Averaging.**

(a) Load an RGB image.

(b) Add Gaussian noise with a mean value of 0 and a variance of 0.1 to it.

(c) Display and compare the images before and after adding noise.

(d) Use the for loop to add 3, 30, and 300 images with random Gaussian noise and find their average value, respectively.

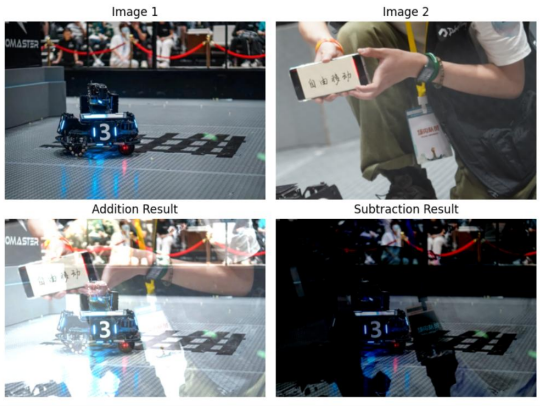
```python
# (d) Use a for loop to add 3, 30, and 300 images with random Gaussian noise and find their average value
num_images = [3, 30, 300]
averaged_images = []

for num in num_images:
    images = [original_image_float32 + np.random.normal(loc=0, scale=0.1, size=original_image.shape) for _ in range(num)]
    averaged_image = np.mean(images, axis=0)
    averaged_images.append(averaged_image)

# (e) Display the averaged images
plt.figure(figsize=(12, 4))

for i, averaged_image in enumerate(averaged_images):
    plt.subplot(1, len(num_images), i+1)
    plt.imshow(cv2.cvtColor((averaged_image * 255).astype(np.uint8), cv2.COLOR_BGR2RGB))
    plt.title(f'Averaged Image ({num_images[i]} Images)')
    plt.axis('off')

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

(7) **Image Algebraic Operations.**

(a) Download two pictures by yourself, and load these two pictures with OpenCV.

(b) Perform algebraic operations of addition and subtraction (Tips: Pay attention to the size and type of the image during the calculation. If they are different, the larger image should be cropped or scaled).

(c) Display the images before and after processing.



(8) **Text adding.**

(a) Load the Lena image with OpenCV.

(b) Employ a red rectangle to mark the 64x64 rectangle in the middle of image. (Tips: You may use cv2.rectangle)

(c) Adding some black text on it. (Tips: You may usecv2.putText). An example of the generated result is shown in the figure below.

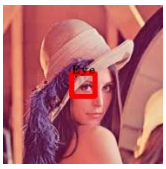
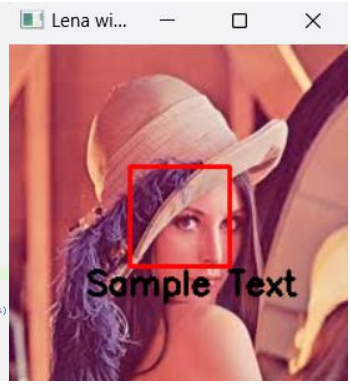


Figure 1. The result of adding content to Lean image

```python
# (a) Load the Lena image with OpenCV
lena_path = 'images/lena.jpg'
lena_image = cv2.imread(lena_path)

# (b) Add a red rectangle to mark the 64x64 rectangle in the middle of the image
# Get the dimensions of the image
height, width, _ = lena_image.shape

# Calculate the coordinates for the rectangle
top_left = (width // 2 - 32, height // 2 - 32)
bottom_right = (width // 2 + 32, height // 2 + 32)

# Draw the rectangle
rectangle_color = (0, 0, 255)  # Red color in BGR format
thickness = 2  # Thickness of the rectangle border
cv2.rectangle(lena_image, top_left, bottom_right, rectangle_color, thickness)

# (c) Add some black text on the rectangle
text = 'Sample Text'
text_position = (width // 2 - 60, height // 2 + 50)  # Position of the text
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.7
text_color = (0, 0, 0)  # Black color in BGR format
text_thickness = 2  # Thickness of the text

cv2.putText(lena_image, text, text_position, font, font_scale, text_color, text_thickness)

# Display the image with the rectangle and text
cv2.imshow('Lena with Content', lena_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



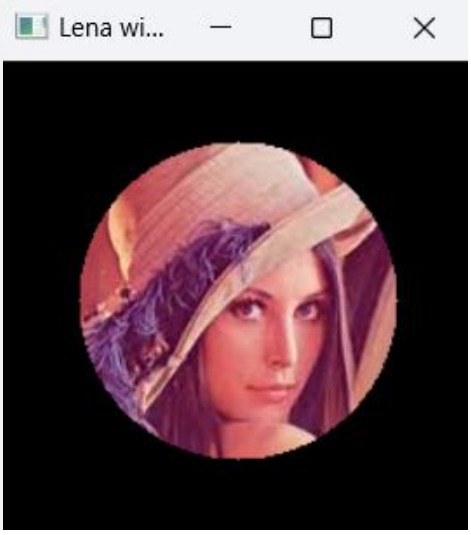
(9) **Adding image mask.**

(a) Load the Lena image with OpenCV.

(b) Adding a circular mask on the Lena image. (Tips: You may usenumpy.ogrid). An example of the generated result is shown in the figure below.

Figure 2. The result of adding mask to Lean image

```python
# (a) Load the Lena image with OpenCV
lena_path = 'images/lena.jpg'
lena_image = cv2.imread(lena_path)

# (b) Add a circular mask on the Lena image
# Get the dimensions of the image
height, width, _ = lena_image.shape

# Create a black mask with the same size as the image
mask = np.zeros((height, width), dtype=np.uint8)

# Define the center and radius of the circle
center = (width // 2, height // 2)
radius = min(width, height) // 3

# Generate the coordinates grid for the entire image
x, y = np.ogrid[:height, :width]

# Create the circular mask
mask[((x - center[1])**2 + (y - center[0])**2) <= radius**2] = 255

# Apply the mask to the Lena image
masked_image = cv2.bitwise_and(lena_image, lena_image, mask=mask)

# Display the masked image
cv2.imshow('Lena with Mask', masked_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



(10)**Capture Images by Laptop Camera with OpenCV. (Bonus Practice).**

(a) Read the video stream from the laptop camera.

(b) Implement Time-lapse photography.(It can be used to record the whole process of the evaporation of water droplets in the cup. Tips: You may usetime.sleep).

(c) Generate corresponding MP4 video or GIF image.

```python
# (a) Read the video stream from the laptop camera
cap = cv2.VideoCapture(0)  # 0 for the default camera, change the index if needed

# Check if the camera is opened successfully
if not cap.isOpened():
    print("Error: Unable to open the camera.")
    exit()

# Create a VideoWriter object to save the frames as a video
frame_width = int(cap.get(3))  # Width of the frames
frame_height = int(cap.get(4))  # Height of the frames
out = cv2.VideoWriter('results/timelapse_video.mp4', cv2.VideoWriter_fourcc(*'mp4v'), 10, (frame_width, frame_height))

# (b) Implement Time-lapse photography
# Capture frames for 10 seconds with a delay of 1 second between each frame
start_time = time.time()
duration = 10  # Duration of capturing (in seconds)
fps = 1  # Number of frames per second

while time.time() - start_time < duration:
    ret, frame = cap.read()  # Read a frame from the camera

    if not ret:
        print("Error: Unable to read frame from the camera.")
        break

    out.write(frame)  # Write the frame to the video file
    cv2.imshow('Frame', frame)  # Display the frame
    cv2.waitKey(1000 // fps)  # Wait for 1 second between each frame

# Release the VideoCapture and VideoWriter objects
cap.release()
out.release()
```

实验分析与结论（Analysis and Conclusion）：

Load, Save and Display Images:
Conclusion: By using Scikit Image to load PNG images, converting them to PIL image format, and displaying images using Matplotlib, image processing can be easily performed.
Gain: Learned how to use different libraries to load, save, and display images, laying a foundation for subsequent image processing operations.

Display Three Individual Color Components of RGB Image:
Conclusion: The red (R), green (G), and blue (B) components of RGB images display the intensity distribution of corresponding colors in the image, which helps to analyze the color characteristics of the image.
Gain: Understood the meaning of various color channels in RGB images and how to use Matplotlib to display them in the same image.

Convert Color Image to Grayscale:
Conclusion: Different grayscale methods (maximum, average, weighted sum) will produce different grayscale images, reflecting different brightness and contrast of the images.
Gain: Learned how to convert color images into grayscale images and understood the advantages and disadvantages of different methods.

Image Cropping:
Conclusion: Image cropping can be achieved by adjusting the index of the image array and selecting the regions of interest in the image for further processing.
Gain: Master the basic methods of image cropping and understand how to extract specific parts of an image for subsequent processing.

Adding Noise to Image:
Conclusion: Different types of noise (Gaussian noise, salt noise, salt and pepper noise, speckle noise, etc.) can have varying degrees of impact on the visual effect of images.
Gain: Learned how to add different types and degrees of noise to images, and understood the impact of noise on image quality.

Image Denoising by Average:
Conclusion: By averaging images with Gaussian noise, it is possible to effectively reduce noise and restore image clarity.
Gain: Master the basic principles and methods of using average processing for image denoising.
Through these experiments, I have gained a deeper understanding of the basic operations and

techniques commonly used in image processing, including loading, saving, and displaying images, separating color channels, graying, image cropping, and adding and removing noise. These knowledge and skills are of great significance for learning and practice in fields such as image processing, computer vision, and deep learning.

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 3 月 30 日

备注：