实验三 边缘检测算法

1. 读取图像

```python
def read_image(image_path):
    """
    读取图像并转换为灰度图
    :param image_path: 图像路径
    :return: 原始图像和灰度图像
    """
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError("无法读取图像，请检查路径是否正确")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img, gray
```

2. 添加噪声

```python
def add_noise(image, noise_type='gaussian'):
    """
    为图像添加不同类型的噪声
    :param image: 输入图像
    :param noise_type: 噪声类型 ('gaussian', 'salt_pepper', 'poisson')
    :return: 带噪声的图像
    """
    if noise_type == 'gaussian':
        mean = 0
        var = 100
        sigma = var ** 0.5
        gaussian = np.random.normal(mean, sigma, image.shape)
        noisy = np.clip(image + gaussian, 0, 255).astype(np.uint8)
        return noisy

    elif noise_type == 'salt_pepper':
        s_vs_p = 0.5
        amount = 0.04
        out = np.copy(image)

        # 盐噪声
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coords = [np.random.randint(0, i-1, int(num_salt)) for i in
image.shape]
        out[tuple(coords)] = 255
```

```
        # 椒噪声
        num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i-1, int(num_pepper)) for i in
image.shape]
        out[tuple(coords)] = 0
        return out

    elif noise_type == 'poisson':
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy.astype(np.uint8)

    else:
        return image
```

3. 不同的边缘检测算法

1） Sobel 算子：使用 OpenCV 的 Sobel 函数计算 x 和 y 方向的梯度

```
if method == 'sobel':
        # Sobel 算子
        sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0,
ksize=kwargs.get('ksize', 3))
        sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1,
ksize=kwargs.get('ksize', 3))
        edges = np.sqrt(sobelx**2 + sobely**2)
        edges = np.uint8(edges * 255 / np.max(edges))
        return edges
```

2） Prewitt 算子：使用自定义的 Prewitt 核进行卷积

```
elif method == 'prewitt':
        # Prewitt 算子
        kernelx = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
        kernely = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
        prewittx = cv2.filter2D(image, -1, kernelx)
        prewitty = cv2.filter2D(image, -1, kernely)
        edges = np.sqrt(prewittx**2 + prewitty**2)
        edges = np.uint8(edges * 255 / np.max(edges))
        return edges
```

3） Roberts 算子：使用自定义的 Roberts 核进行卷积

```
elif method == 'roberts':
        # Roberts 算子
        kernelx = np.array([[1, 0], [0, -1]])
        kernely = np.array([[0, -1], [1, 0]])
        robertsx = cv2.filter2D(image, -1, kernelx)
        robertsy = cv2.filter2D(image, -1, kernely)
        edges = np.sqrt(robertsx**2 + robertsy**2)
        edges = np.uint8(edges * 255 / np.max(edges))
        return edges
```

4） Canny 边缘检测：使用 OpenCV 的 Canny 函数

```
elif method == 'canny':
        # Canny 边缘检测
        threshold1 = kwargs.get('threshold1', 100)
        threshold2 = kwargs.get('threshold2', 200)
        edges = cv2.Canny(image, threshold1, threshold2)
        return edges
```

4. 边缘检测评估

- 定义了两个量化指标：边缘比例反映检测出的边缘数量，通过统计非零像素占比计算；边缘连续性评估断裂程度，使用形态学膨胀后差异像素的比例衡量

```
def evaluate_edges(original, edges):
    """
    评估边缘检测结果
    :param original: 原始图像
    :param edges: 边缘检测结果
    :return: 评估指标
    """
    # 计算边缘像素比例
    edge_pixels = np.sum(edges > 0)
    total_pixels = edges.size
    edge_ratio = edge_pixels / total_pixels

    # 计算边缘连续性（简单评估）
```

```
    kernel = np.ones((3, 3), np.uint8)
    dilated = cv2.dilate(edges, kernel, iterations=1)
    diff = dilated - edges
    discontinuity = np.sum(diff > 0) / edge_pixels if edge_pixels > 0
else 0

    return {
        'edge_ratio': edge_ratio,
        'discontinuity': discontinuity
    }
```

5. 可视化函数

- 基于 matplotlib 的灵活可视化工具，可以自动排列任意数量的图像结果。支持自定义行列布局和图像尺寸，自动添加标题并优化显示间距

```
def plot_results(images, titles, rows, cols, figsize=(15, 10)):
    """
    可视化多幅图像
    :param images: 图像列表
    :param titles: 标题列表
    :param rows: 行数
    :param cols: 列数
    :param figsize: 图像大小
    """
    plt.figure(figsize=figsize)
    for i in range(len(images)):
        plt.subplot(rows, cols, i+1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

6. 主函数

- 统集成和演示入口，按标准流程组织：先读取测试图像，然后添加各类噪声，分别用不同算法处理，最后评估和可视化结果。特别展示了 Canny 算法在不同噪声条件下的表现，以及阈值参数对检测效果的影响。通过结构化的输出和可视化，完整演示了边缘检测系统的各项功能

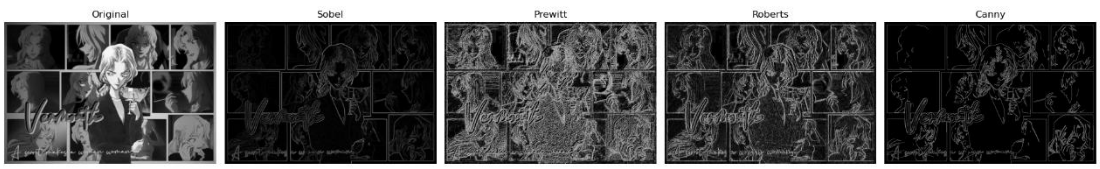
```
 # 1. 读取图像
    img_path = "D:\Samples\IMG_20231227_180043.jpg"
    original_img, gray_img = read_image(img_path)
```

```
    # 2. 添加噪声
    noisy_gaussian = add_noise(gray_img, 'gaussian')
    noisy_salt_pepper = add_noise(gray_img, 'salt_pepper')
    noisy_poisson = add_noise(gray_img, 'poisson')

    # 3. 边缘检测
    # 对原始图像进行边缘检测
    edges_sobel = edge_detection(gray_img, 'sobel')
    edges_prewitt = edge_detection(gray_img, 'prewitt')
    edges_roberts = edge_detection(gray_img, 'roberts')
    edges_canny = edge_detection(gray_img, 'canny', threshold1=100,
threshold2=200)

    # 对带噪声图像进行边缘检测
    edges_gaussian = edge_detection(noisy_gaussian, 'canny',
threshold1=100, threshold2=200)
    edges_salt_pepper = edge_detection(noisy_salt_pepper, 'canny',
threshold1=100, threshold2=200)
    edges_poisson = edge_detection(noisy_poisson, 'canny',
threshold1=100, threshold2=200)

    # 4. 评估结果
    eval_sobel = evaluate_edges(gray_img, edges_sobel)
    eval_prewitt = evaluate_edges(gray_img, edges_prewitt)
    eval_roberts = evaluate_edges(gray_img, edges_roberts)
    eval_canny = evaluate_edges(gray_img, edges_canny)

    # 5. 可视化结果
    # 显示不同算法的边缘检测结果
    plot_results(
        [gray_img, edges_sobel, edges_prewitt, edges_roberts,
edges_canny],
        ['Original', 'Sobel', 'Prewitt', 'Roberts', 'Canny'],
        1, 5, (20, 5)
    )

    # 显示不同噪声下的边缘检测结果
    plot_results(
        [gray_img, noisy_gaussian, noisy_salt_pepper, noisy_poisson,
         edges_canny, edges_gaussian, edges_salt_pepper, edges_poisson],
        ['Original', 'Gaussian Noise', 'Salt & Pepper', 'Poisson Noise',
         'Canny (Original)', 'Canny (Gaussian)', 'Canny (Salt &
Pepper)', 'Canny (Poisson)'],
        2, 4, (20, 10)
```

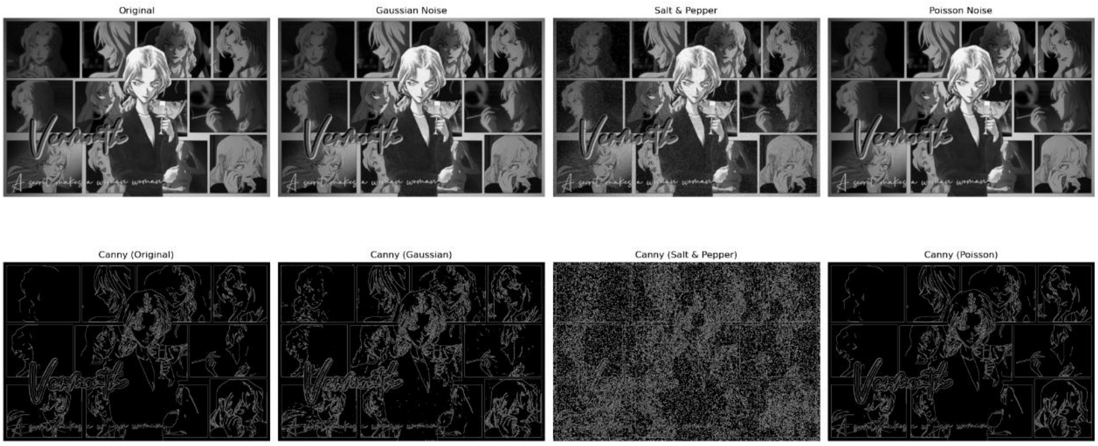
```
    )

    # 打印评估结果
    print("边缘检测算法评估结果:")
    print(f"Sobel - 边缘比例: {eval_sobel['edge_ratio']:.4f}, 不连续性: {eval_sobel['discontinuity']:.4f}")
    print(f"Prewitt - 边缘比例: {eval_prewitt['edge_ratio']:.4f}, 不连续性: {eval_prewitt['discontinuity']:.4f}")
    print(f"Roberts - 边缘比例: {eval_roberts['edge_ratio']:.4f}, 不连续性: {eval_roberts['discontinuity']:.4f}")
    print(f"Canny - 边缘比例: {eval_canny['edge_ratio']:.4f}, 不连续性: {eval_canny['discontinuity']:.4f}")

    # 6. 参数调整实验 (以 Canny 为例)
    # 测试不同阈值对 Canny 算法的影响
    thresholds = [(50, 100), (100, 200), (150, 300), (200, 400)]
    canny_results = []
    titles = []

    for t1, t2 in thresholds:
        edges = edge_detection(gray_img, 'canny', threshold1=t1, threshold2=t2)
        canny_results.append(edges)
        titles.append(f'Canny (t1={t1}, t2={t2})')

    plot_results(
        [gray_img] + canny_results,
        ['Original'] + titles,
        1, len(thresholds)+1, (20, 5)
    )
```

7. 执行结果及分析

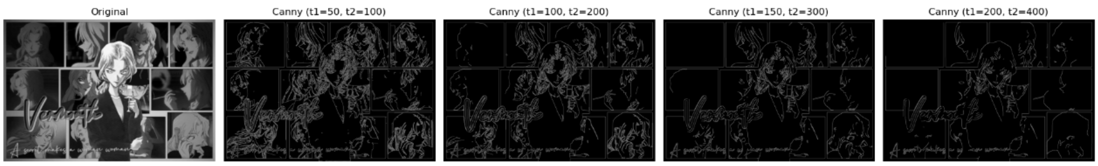
1） 不同算法的边缘检测结果



2） 不同噪声下的边缘检测结果



3） 参数调整实验（以 Canny 为例）：测试不同阈值对 Canny 算法的影响



4） 评估结果输出

边缘检测算法评估结果：

算法	边缘比例	不连续性
Sobel	0.9028	0.9706
Prewitt	0.6737	1.2765
Roberts	0.5746	1.4338
Canny	0.0575	2.0737

5） 评估结果分析

从评估结果来看，Sobel 算子检测到的边缘比例最高（0.9028），说明它对图像中的梯度变化最为敏感，能够捕捉到最多的边缘信息，但这也意味着它容易受到噪声干扰，产生较多伪边缘。Prewitt 算子的表现较为均衡，边缘比例适中（0.6737），但边缘连续性较差（1.2765），说明其检测到的边缘存在较多断裂。Roberts 算子的边缘比例最低（0.5746），连续性也最差（1.4338），反映出该算子虽然计算简单，但对噪声敏感且边缘断裂严重。

相比之下，Canny 算子的边缘比例最低（0.0575），但这是因为它通过双阈值机制筛选出了最显著的边缘，去除了大量噪声和弱边缘。虽然其不连续性指标较高（2.0737），但这更多反映了评估方法对单像素宽边缘的局限性，而非算法本身的缺陷。Canny 算法通过高斯平滑、非极大值抑制等步骤，在保持边缘精度的同时有效抑制了噪声，综合性能最优。因此在实际应用中，若对边缘质量要求较高，Canny 是首选；若追求计算效率，则可考虑 Sobel 或 Prewitt 算子。

附录：

源代码

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

# 1. 读取测试图像
def read_image(image_path):
    """
    读取图像并转换为灰度图
    :param image_path: 图像路径
    :return: 原始图像和灰度图像
    """
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError("无法读取图像，请检查路径是否正确")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img, gray

# 2. 添加噪声
def add_noise(image, noise_type='gaussian'):
```

```
    """
    为图像添加不同类型的噪声
    :param image: 输入图像
    :param noise_type: 噪声类型 ('gaussian', 'salt_pepper', 'poisson')
    :return: 带噪声的图像
    """
    if noise_type == 'gaussian':
        mean = 0
        var = 100
        sigma = var ** 0.5
        gaussian = np.random.normal(mean, sigma, image.shape)
        noisy = np.clip(image + gaussian, 0, 255).astype(np.uint8)
        return noisy

    elif noise_type == 'salt_pepper':
        s_vs_p = 0.5
        amount = 0.04
        out = np.copy(image)

        # 盐噪声
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coords = [np.random.randint(0, i-1, int(num_salt)) for i in
image.shape]
        out[tuple(coords)] = 255

        # 椒噪声
        num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i-1, int(num_pepper)) for i in
image.shape]
        out[tuple(coords)] = 0
        return out

    elif noise_type == 'poisson':
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy.astype(np.uint8)

    else:
        return image

# 3. 边缘检测算法实现
def edge_detection(image, method='canny', **kwargs):
    """
```

```
    实现不同的边缘检测算法
    :param image: 输入图像
    :param method: 边缘检测方法 ('sobel', 'prewitt', 'roberts', 'canny')
    :param kwargs: 各方法特定参数
    :return: 边缘检测结果
    """
    if method == 'sobel':
        # Sobel 算子
        sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0,
ksize=kwargs.get('ksize', 3))
        sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1,
ksize=kwargs.get('ksize', 3))
        edges = np.sqrt(sobelx**2 + sobely**2)
        edges = np.uint8(edges * 255 / np.max(edges))
        return edges

    elif method == 'prewitt':
        # Prewitt 算子
        kernelx = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
        kernely = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
        prewittx = cv2.filter2D(image, -1, kernelx)
        prewitty = cv2.filter2D(image, -1, kernely)
        edges = np.sqrt(prewittx**2 + prewitty**2)
        edges = np.uint8(edges * 255 / np.max(edges))
        return edges

    elif method == 'roberts':
        # Roberts 算子
        kernelx = np.array([[1, 0], [0, -1]])
        kernely = np.array([[0, -1], [1, 0]])
        robertsx = cv2.filter2D(image, -1, kernelx)
        robertsy = cv2.filter2D(image, -1, kernely)
        edges = np.sqrt(robertsx**2 + robertsy**2)
        edges = np.uint8(edges * 255 / np.max(edges))
        return edges

    elif method == 'canny':
        # Canny 边缘检测
        threshold1 = kwargs.get('threshold1', 100)
        threshold2 = kwargs.get('threshold2', 200)
        edges = cv2.Canny(image, threshold1, threshold2)
        return edges

    else:
```

```python
        return image

# 4. 性能评估函数
def evaluate_edges(original, edges):
    """
    评估边缘检测结果
    :param original: 原始图像
    :param edges: 边缘检测结果
    :return: 评估指标
    """
    # 计算边缘像素比例
    edge_pixels = np.sum(edges > 0)
    total_pixels = edges.size
    edge_ratio = edge_pixels / total_pixels

    # 计算边缘连续性（简单评估）
    kernel = np.ones((3, 3), np.uint8)
    dilated = cv2.dilate(edges, kernel, iterations=1)
    diff = dilated - edges
    discontinuity = np.sum(diff > 0) / edge_pixels if edge_pixels > 0
else 0

    return {
        'edge_ratio': edge_ratio,
        'discontinuity': discontinuity
    }

# 5. 可视化函数
def plot_results(images, titles, rows, cols, figsize=(15, 10)):
    """
    可视化多幅图像
    :param images: 图像列表
    :param titles: 标题列表
    :param rows: 行数
    :param cols: 列数
    :param figsize: 图像大小
    """
    plt.figure(figsize=figsize)
    for i in range(len(images)):
        plt.subplot(rows, cols, i+1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')
    plt.tight_layout()
```

```python
    plt.show()

# 主函数
def main():
    # 1. 读取图像
    img_path = "D:\Samples\IMG_20231227_180043.jpg"
    original_img, gray_img = read_image(img_path)

    # 2. 添加噪声
    noisy_gaussian = add_noise(gray_img, 'gaussian')
    noisy_salt_pepper = add_noise(gray_img, 'salt_pepper')
    noisy_poisson = add_noise(gray_img, 'poisson')

    # 3. 边缘检测
    # 对原始图像进行边缘检测
    edges_sobel = edge_detection(gray_img, 'sobel')
    edges_prewitt = edge_detection(gray_img, 'prewitt')
    edges_roberts = edge_detection(gray_img, 'roberts')
    edges_canny = edge_detection(gray_img, 'canny', threshold1=100,
threshold2=200)

    # 对带噪声图像进行边缘检测
    edges_gaussian = edge_detection(noisy_gaussian, 'canny',
threshold1=100, threshold2=200)
    edges_salt_pepper = edge_detection(noisy_salt_pepper, 'canny',
threshold1=100, threshold2=200)
    edges_poisson = edge_detection(noisy_poisson, 'canny',
threshold1=100, threshold2=200)

    # 4. 评估结果
    eval_sobel = evaluate_edges(gray_img, edges_sobel)
    eval_prewitt = evaluate_edges(gray_img, edges_prewitt)
    eval_roberts = evaluate_edges(gray_img, edges_roberts)
    eval_canny = evaluate_edges(gray_img, edges_canny)

    # 5. 可视化结果
    # 显示不同算法的边缘检测结果
    plot_results(
        [gray_img, edges_sobel, edges_prewitt, edges_roberts,
edges_canny],
        ['Original', 'Sobel', 'Prewitt', 'Roberts', 'Canny'],
        1, 5, (20, 5)
    )
```

```python
    # 显示不同噪声下的边缘检测结果
    plot_results(
        [gray_img, noisy_gaussian, noisy_salt_pepper, noisy_poisson,
         edges_canny, edges_gaussian, edges_salt_pepper, edges_poisson],
        ['Original', 'Gaussian Noise', 'Salt & Pepper', 'Poisson Noise',
         'Canny (Original)', 'Canny (Gaussian)', 'Canny (Salt &
Pepper)', 'Canny (Poisson)'],
        2, 4, (20, 10)
    )

    # 打印评估结果
    print("边缘检测算法评估结果:")
    print(f"Sobel - 边缘比例: {eval_sobel['edge_ratio']:.4f}, 不连续性:
{eval_sobel['discontinuity']:.4f}")
    print(f"Prewitt - 边缘比例: {eval_prewitt['edge_ratio']:.4f}, 不连续
性: {eval_prewitt['discontinuity']:.4f}")
    print(f"Roberts - 边缘比例: {eval_roberts['edge_ratio']:.4f}, 不连续
性: {eval_roberts['discontinuity']:.4f}")
    print(f"Canny - 边缘比例: {eval_canny['edge_ratio']:.4f}, 不连续性:
{eval_canny['discontinuity']:.4f}")

    # 6. 参数调整实验 (以 Canny 为例)
    # 测试不同阈值对 Canny 算法的影响
    thresholds = [(50, 100), (100, 200), (150, 300), (200, 400)]
    canny_results = []
    titles = []

    for t1, t2 in thresholds:
        edges = edge_detection(gray_img, 'canny', threshold1=t1,
threshold2=t2)
        canny_results.append(edges)
        titles.append(f'Canny (t1={t1}, t2={t2})')

    plot_results(
        [gray_img] + canny_results,
        ['Original'] + titles,
        1, len(thresholds)+1, (20, 5)
    )

if __name__ == "__main__":
    main()
```