数字图像处理实验报告

班级：智能专研 25

姓名：王耀彬

学号：2025354100103

目录

实验项目一：腐蚀与膨胀

前言

（1）边缘：图像中亮度发生剧烈变化的地方，例如“建筑物的轮廓”、“物体与背景的分界线”。边缘对应于图像灰度值的突变。

（2）膨胀与腐蚀：为更好实现“膨胀与腐蚀”效果，先将实际拍出的图片进行边缘检测。在边缘检测之后，我们通常得到一张“黑白图”（白色是边缘，黑色是背景）。这张二值图常会有“小毛刺”“小黑洞”或“边缘断裂”。膨胀与腐蚀就是用来解决这些问题的。

1 实现图像的边缘检测

1.1 实验原理

1.1.1 图像预处理

1. 灰度化：将彩色图像转换为灰度图像，简化计算。

2. 对比度增强（CLAHE）：有些图像对比度低，边缘不明显。

3. 高斯模糊（降噪）：避免图像中的噪声被误认为边缘。利用卷积操作，每个像素的新值 = 邻域像素的加权平均。

高斯模糊 ≈ 低通滤波。基于二维高斯函数：

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

1.1.2 导数算子

1. 一阶 Sobel 算子：计算图像在 X 方向和 Y 方向的灰度变化率。

2. 二阶 Laplacian 算子：检测灰度变化率的变化

1.1.3 最优边缘检测

1. Canny 算法：最经典、最优秀的边缘检测算法。算法步骤：

（1）高斯滤波降噪：去除噪声

（2）计算梯度幅值与方向：使用 Sobel 算子

（3）非极大值抑制(NMS)：边缘变细，只保留梯度方向上的最大值。

（4）双阈值检测：区分强边缘、弱边缘、非边缘。

（5）边缘连接：将弱边缘连接到强边缘

1.2 实验代码

```
import cv2
import numpy as np
import os

def robust_norm_abs(mat64):
```

```
    """稳健归一化
    用 1%-99%分位抑制极端值，避免“全黑/全白”
    """
    abs_m = np.abs(mat64).astype(np.float64)
    p1, p99 = np.percentile(abs_m, [1, 99])
    rng = max(p99 - p1, 1e-6)
    out = np.clip((abs_m - p1) * (255.0 / rng), 0, 255).astype(np.uint8)
    return out

def quantile_canny_thresholds(gray_u8, low_q=10, high_q=90):
    """自适应阈值
    """
    p_low, p_high = np.percentile(gray_u8, [low_q, high_q])
    lower = int(np.clip(p_low, 0, 255))
    upper = int(np.clip(p_high, 0, 255))
    if lower >= upper:
        lower = max(0, upper - 30)
    return lower, upper

img = cv2.imread('./Data/NCUT_Building.jpg', cv2.IMREAD_GRAYSCALE)

# 局部对比度增强，减少“发灰/无纹理”
use_clahe = True
if use_clahe:
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    img_proc = clahe.apply(img)
else:
    img_proc = img

# 高斯降噪
blur = cv2.GaussianBlur(img_proc, (5, 5), 1.0)

# Sobel 算子
sx64 = cv2.Sobel(blur, cv2.CV_64F, 1, 0, ksize=3)
sy64 = cv2.Sobel(blur, cv2.CV_64F, 0, 1, ksize=3)
sobel_x_viz = robust_norm_abs(sx64)
sobel_y_viz = robust_norm_abs(sy64)
mag = cv2.magnitude(sx64, sy64)
mag_viz = robust_norm_abs(mag)

# Laplacian 算子
lap64 = cv2.Laplacian(blur, cv2.CV_64F, ksize=3)
lap_viz = robust_norm_abs(lap64)

# Canny：分位数阈值更稳健；若“全黑”减小 low_q，若“全白”增大 low_q/high_q
lower, upper = quantile_canny_thresholds(blur, low_q=10, high_q=90)
edges = cv2.Canny(blur, lower, upper, L2gradient=True)

# 叠加显示
overlay = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
overlay[edges != 0] = (0, 0, 255)

output_dir = './Result/Edge_Detection'
os.makedirs(output_dir, exist_ok=True)

cv2.imwrite(os.path.join(output_dir, '1_Original.jpg'), img)
cv2.imwrite(os.path.join(output_dir, '2_Sobel_X.jpg'), sobel_x_viz)
cv2.imwrite(os.path.join(output_dir, '3_Sobel_Y.jpg'), sobel_y_viz)
cv2.imwrite(os.path.join(output_dir, '4_Sobel_Magnitude.jpg'), mag_viz)
cv2.imwrite(os.path.join(output_dir, '5_Laplacian.jpg'), lap_viz)
cv2.imwrite(os.path.join(output_dir, '6_Canny.jpg'), edges)
cv2.imwrite(os.path.join(output_dir, '7_Overlay.jpg'), overlay)

print(f'所有边缘检测结果已保存到: {output_dir}')
print(f'- 1_Original.jpg: 原始图像')
print(f'- 2_Sobel_X.jpg: Sobel X 方向梯度')
print(f'- 3_Sobel_Y.jpg: Sobel Y 方向梯度')
```

```
print(f'- 4_Sobel_Magnitude.jpg: Sobel 梯度幅值')
print(f'- 5_Laplacian.jpg: Laplacian 二阶导')
print(f'- 6_Canny.jpg: Canny 边缘检测')
print(f'- 7_Overlay.jpg: Canny 边缘叠加到原图')
```

1.3 实验结果与分析

待分割图片来源：瀚学楼拍摄北方工业大学景观。

图 1-1 NCUT 景观图

分割结果：

(a) (b)

(c) (d)

图 1-2 Sobel、Laplacian 及 Canny 分割结果

(a)灰度图 (b)Sobel 处理图 (c)Laplacian 处理图 (d)分割结果图

效果对比：

算法	特点	适用场景
灰度图	-	基准参考
Sobel	综合所有方向	全方位边缘
Laplacian	二阶导，各向同性	精细边缘定位

2 实现图像的腐蚀与膨胀

2.1 实验原理

2.1.1 二值图像与结构元素

1. 二值图像：每个像素只有两种值（0 => 黑，255 => 白）
2. 前景、背景：白色视为前景，黑色为背景
3. 结构元素(SE)：小矩阵（类似卷积核），决定“检查邻域的大小”

2.1.2 腐蚀与膨胀(Erosion and Dilation)

1. 腐蚀(Erosion)：只有当以结构元素为模板覆盖该点邻域时，模板对应的所有位置都是白（前景），该点才保留白；否则变黑。即：让白色区域“收缩”。突出作用：去掉突出的小白点、减少毛刺、让细线变细甚至消失。对细线其副作用，容易直接去除。

2. 膨胀(Dilation)：只要结构元素模板盖住的邻域里“有白”，该点就会变白。即：让白色区域“扩张”。突出作用：填平小的裂缝、连接断裂处、让细线变粗。对噪点处理不好，令噪点放大。

3. 开运算：由腐蚀 => 膨胀。先腐蚀去小白点，再膨胀把主体“长回去”。去除小的白色噪声、平滑物体边界；对“细细的白线”会保留得较差（因为先腐蚀）。边缘去噪常用。

4. 闭运算：膨胀 => 腐蚀。先膨胀填裂缝，再腐蚀把主体“收回”。填补小黑洞、连接近邻的白色区域；对断裂边缘更友好。边缘连接、孔洞填补。

5. 形态学梯度：膨胀 + 腐蚀。膨胀结果 − 腐蚀结果。

2.2 实验代码

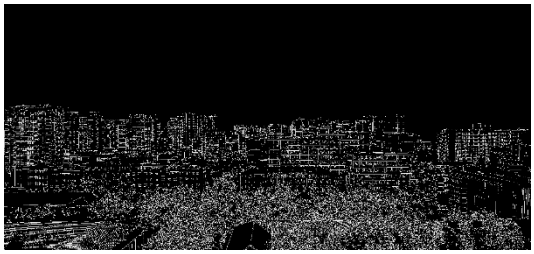
```
import cv2
import numpy as np
import os
from typing import Optional

BASE_DIR = os.path.dirname(__file__)
INPUT_PATH = os.path.join(BASE_DIR, 'Result', 'Edge_Detection', '6_Canny.jpg')
OUTPUT_DIR = os.path.join(BASE_DIR, 'Result', 'Erosion_AND_Dilation')
MIN_RATIO = 1e-4  # 判定“几乎全黑”的阈值（非零像素比例）

def imread_gray(path: str):
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise FileNotFoundError(f'未找到输入图像: {path}\n 请先生成 6_Canny.jpg 或检查路径。')
    return img

def binarize_otsu(gray_u8: np.ndarray) -> np.ndarray:
```

```
    _t, bin_img = cv2.threshold(gray_u8, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
    return bin_img

def nz_ratio(img_u8: np.ndarray) -> float:
    h, w = img_u8.shape[:2]
    return float(np.count_nonzero(img_u8)) / float(h * w + 1e-9)

def save_img(name: str, img: np.ndarray):
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    cv2.imwrite(os.path.join(OUTPUT_DIR, name), img)

# 若输入边缘图过于稀疏，尝试一次性“从原图恢复”
def try_recover_from_original(base_dir: str) -> Optional[np.ndarray]:
    candidates = [
        os.path.join(base_dir, 'Result', 'Edge_Detection', '1_Original.jpg'),
        os.path.join(base_dir, 'Data', 'NCUT_Building.jpg'),
    ]
    for p in candidates:
        if os.path.exists(p):
            orig = cv2.imread(p, cv2.IMREAD_GRAYSCALE)
            if orig is not None:
                blur = cv2.GaussianBlur(orig, (5, 5), 1.0)
                # Sobel 幅值 + Otsu 二值化（一次性简单恢复）
                sx = cv2.Sobel(blur, cv2.CV_64F, 1, 0, ksize=3)
                sy = cv2.Sobel(blur, cv2.CV_64F, 0, 1, ksize=3)
                mag = cv2.magnitude(sx, sy)
                mag_u8 = cv2.normalize(mag, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
                return binarize_otsu(mag_u8)
    return None

# 执行一次操作；若“过空”，先轻度膨胀再重做，避免纯黑
def ensure_nonempty(src: np.ndarray, kernel: np.ndarray, op, min_ratio: float =
MIN_RATIO) -> np.ndarray:
    out = op(src, kernel)
    if nz_ratio(out) >= min_ratio:
        return out
    thick = cv2.dilate(src, kernel, iterations=1)
    return op(thick, kernel)

def op_erode(src, k):
    return cv2.erode(src, k, iterations=1)
def op_dilate(src, k):
    return cv2.dilate(src, k, iterations=1)
def op_open(src, k):
    return cv2.morphologyEx(src, cv2.MORPH_OPEN, k)
def op_close(src, k):
    return cv2.morphologyEx(src, cv2.MORPH_CLOSE, k)
def op_grad(src, k):
    return cv2.morphologyEx(src, cv2.MORPH_GRADIENT, k)

# ========== 主流程 ==========
def main():
    # 读入边缘图并标准化为二值
    gray = imread_gray(INPUT_PATH)
    bin_img = binarize_otsu(gray)

    # 输入过稀疏时，尝试一次从原图恢复
    if nz_ratio(bin_img) < MIN_RATIO:
        rec = try_recover_from_original(BASE_DIR)
        if rec is not None and nz_ratio(rec) > nz_ratio(bin_img):
            bin_img = rec
    k3_rect = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    k5_rect = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
    k3_ell  = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
```

```
    # 形态学五项（用 ensure_nonempty 简化回退）
    erosion_3x3  = ensure_nonempty(bin_img, k3_rect, op_erode)
    erosion_5x5  = ensure_nonempty(bin_img, k5_rect, op_erode)
    dilation_3x3 = op_dilate(bin_img, k3_rect)
    dilation_5x5 = op_dilate(bin_img, k5_rect)
    opening_3x3  = ensure_nonempty(bin_img, k3_rect, op_open)
    closing_3x3  = op_close(bin_img, k3_rect)
    gradient_3x3 = op_grad(bin_img, k3_ell)

    save_img('1_Input.png', gray)
    save_img('2_Input_Binary.png', bin_img)
    save_img('3_Erosion_3x3.png', erosion_3x3)
    save_img('4_Erosion_5x5.png', erosion_5x5)
    save_img('5_Dilation_3x3.png', dilation_3x3)
    save_img('6_Dilation_5x5.png', dilation_5x5)
    save_img('7_Opening_3x3.png', opening_3x3)
    save_img('8_Closing_3x3.png', closing_3x3)
    save_img('9_Gradient_3x3.png', gradient_3x3)

    stats = {
        'input_bin': nz_ratio(bin_img),
        'erosion_3x3': nz_ratio(erosion_3x3),
        'erosion_5x5': nz_ratio(erosion_5x5),
        'dilation_3x3': nz_ratio(dilation_3x3),
        'dilation_5x5': nz_ratio(dilation_5x5),
        'opening_3x3': nz_ratio(opening_3x3),
        'closing_3x3': nz_ratio(closing_3x3),
        'gradient_3x3': nz_ratio(gradient_3x3),
    }
    print(f'形态学结果已保存到: {OUTPUT_DIR}')
    for k, v in stats.items():
        print(f'- {k:>14}: 非零像素比例 = {v:.4%}')

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

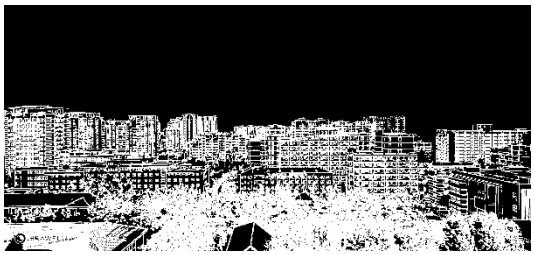
2.3 实验结果与分析

运行结果：



(a)

(b)



(c)

(d)

(e)

(f)

图 2-1 腐蚀与膨胀效果图

(a)输入二值图 (b)腐蚀图 (c)膨胀图 (d)开处理图 (e)闭处理图 (f)形态学梯度图

结果分析：

算法	特点	适用场景
二值	白色占比通常较低	前景/背景分离
腐蚀	细线易消失	去除孤立噪点
膨胀	边缘变粗，断裂连接	突出目标区域
开运算	去小白点、平滑轮廓	消除椒盐噪声
闭运算	填小黑洞、连接近邻边缘	增强整体性
形态学梯度	细致的轮廓线	提取目标边缘