合肥工业大学

《机器视觉》课程实验

实验题目	课程实验一：图像滤波
学生姓名	高志鸿
学　　号	2023217584
专业班级	智能科 23-3 班

一、实验目的

- 理解图像“边缘/特征”的含义，掌握梯度思想在边缘检测中的应用。
- 熟悉 Sobel 等一阶微分算子的原理与实现流程（卷积、梯度幅值与方向）。
- 掌握阈值化、归一化等后处理方法，并能分析不同参数对结果的影响。
- 能对比不同实现方式（自编 vs OpenCV）在效果与效率上的差异。

二、实验环境

- 操作系统：Windows
- 开发语言： python
- 主要库：OpenCV（cv2）、NumPy、Matplotlib（用于显示/保存结果）
- 实验输入：一张自己拍摄的照片

三、实验原理与方法

- 整体流程（ exp1/main.py ）：
  main() 从 exp1/data 读取图片，逐张调用 process_one(img_path, out_dir) 处理，并将每张图的结果写入 exp1/output ；最终把每张图的统计信息（包含输出文件列表与纹理特征字典等）汇总写到 output/summary.txt 。
  其中图像读取/保存由 read_image() （PIL 读入转 RGB，再转 NumPy）与 save_image() （NumPy 转 PIL 保存）完成。

1. Sobel 边缘检测（ exp1/sobel.py ）

- 灰度化（ to_grayscale(img) ）：
  将 RGB 转灰度采用加权求和 [Gray = 0.299R + 0.587G + 0.114B]
  输出为 float64 ，避免后续卷积与开方运算的溢出/截断。
- 二维卷积（ convolve2d(img, kernel) ）：对输入灰度图做通用二维卷积实现。卷积核在实现中会先做翻转（ flipud + fliplr ），并使用 np.pad(..., mode="reflect") 做边界填充，保证输出尺寸与原图一致，减少边界伪影。
- 单方向梯度（ sobel_given(img) ）：使用固定的 3×3 Sobel 核计算一个方向的梯度响应：
  G_x 核： [[1, 0, -1], [2, 0, -2], [1, 0, -1]] （检测纵向边缘）
  G_y 核： [[1, 2, 1], [0, 0, 0], [-1, -2, -1]] （检测横向边缘）

- 双方向梯度与幅值（ sobel_magnitude(img) ）：分别计算
  - (G_x = convolve2d(Gray, k_x))
  - (G_y = convolve2d(Gray, k_y))（代码里 k_y = k_x.T ）
  - 幅值：G = $\sqrt{Gx^2 + Gy^2}$
- 归一化显示（ normalize_to_uint8(arr) ）：将梯度结果按 min-max 归一化到 0 –

255 并转 uint8 ，用于保存 PNG；若图像无变化（max≈min）则输出全 0，避免除零。

2. RGB 颜色直方图（ exp1/histogram.py ）

- 原理 ：分别统计 R/G/B 三通道在 0–255 灰度级上的像素计数，形成长度为 bins=256 的频数分布，描述颜色与亮度的全局分布特征。
- 像素计数（ channel_histogram(channel, bins=256) ） ：对单通道逐像素累加计数；对越界值做夹取（<0 设 0，≥bins 设 bins-1）。
- 三通道统计（ rgb_histograms(img, bins=256) ）：返回 (r, g, b) 三个直方图数组（若输入为灰度图则三者相同）。
- 绘图保存（ plot_rgb_histograms(r, g, b, save_path) ）：用 Matplotlib 将三条曲线绘制到同一张图并保存。对应输出文件： {name}_rgb_hist.png 。

3. GLCM 纹理特征（ exp1/glcm.py ）

- 灰度量化（ quantize(img, levels) ） ：先对灰度图做 min-max 归一化，再映射到 levels 个离散灰度级（默认 levels=32 ），降低矩阵维度与计算量。
- 共生矩阵构建（ glcm_matrix(img_q, distance, angle, levels) ）：在给定距离 distance 与方向 angle 下，统计相邻像素灰度对 ((i,j)) 的共现频率；并对矩阵按总和归一化为概率矩阵。默认参数：
  - distances=(1,)
  - angles=(0, π/4, π/2, 3π/4) （0°/45°/90°/135°）
- 特征计算（ glcm_features_from_matrix(mat) ）：从概率矩阵计算 4 个统计量并返回字典：
  - contrast （对比度）
  - energy （能量）
  - homogeneity （同质性）
  - entropy （熵）
- 多方向聚合（ glcm_features(img_gray, levels=32, ...) ） ：对默认 4 个方向分别计算特征，再对每个特征取均值作为最终纹理描述符。 process_one() 中会将返回的特征字典用 np.save 保存为： {name}_texture.npy 。

四、实验内容与步骤（对应实现逻辑）

1. 实验工程结构确认

- 入口脚本： exp1/main.py
- 输入目录： exp1/data/ （存放待处理的 .jpg/.jpeg/.png/.bmp 图片）
- 输出目录： exp1/output/ （自动创建，用于保存结果与汇总文件）

- 核心模块：
  - exp1/sobel.py ： sobel_given() 、 sobel_magnitude() 、 normalize_to_uint8() 、 to_grayscale()
  - exp1/histogram.py ： rgb_histograms() 、 plot_rgb_histograms()
  - exp1/glcm.py ： glcm_features(gray, levels=32)

2. 准备实验数据

- 将测试图片放入 exp1/data/ ，文件名建议使用英文或拼音，便于输出文件前缀统一（代码中 name = os.path.splitext(os.path.basename(img_path))[0] ）。
- 图片内容建议包含：明显轮廓（便于边缘验证）、丰富颜色（便于直方图验证）、重复纹理（便于 GLCM 验证）。

3. 批处理执行（主流程）

- 运行 main() 后会遍历 exp1/data/ 下所有符合后缀的图片，对每张图调用 process_one(img_path, out_dir) ，并把每张图的处理结果加入 results ，最终写入 exp1/output/summary.txt 。

4. 单张图片处理步骤（ process_one() 逐步产物）

- 步骤 4.1 读取原图

  - 调用 read_image(img_path) ：使用 PIL 读取并转为 RGB，再转为 NumPy 数组 img ，作为后续所有模块的输入。
- 步骤 4.2 Sobel 边缘特征提取

  - 调用 given = sobel_given(img) ：得到单方向梯度响应图。
  - 调用 gx, gy, mag = sobel_magnitude(img) ：得到 Gx 、 Gy 与梯度幅值 Mag 。
  - 调用 normalize_to_uint8() 将上述结果归一化为 8-bit 便于保存。
  - 输出文件（保存在 exp1/output/ ）：
    - {name}_sobel_given.png
    - {name}_sobel_gx.png
    - {name}_sobel_gy.png
    - {name}_sobel_mag.png
- 步骤 4.3 RGB 颜色直方图统计与可视化

  - 调用 r, g, b = rgb_histograms(img) ：得到三通道 256 bins 的计数数组（ hist_bins 也会记录为 256）。
  - 调用 plot_rgb_histograms(r, g, b, save_path) ：绘制并保存直方图曲线图。
  - 输出文件：
    - {name}_rgb_hist.png

- 步骤 4.4 GLCM 纹理特征提取（levels=32）

  - 先调用 gray = to_grayscale(img) ：将图像转为灰度（float）。
  - 调用 feats = glcm_features(gray, levels=32) ：在默认参数下（距离 1、四方向 0/45/90/135 度）计算纹理统计量，并对多方向结果做均值聚合。
  - 输出文件：
    - {name}_texture.npy （保存纹理特征字典，例如包含 contrast/energy/homogeneity/entropy ）
- 步骤 4.5 结果汇总记录

  - process_one() 返回一个字典，包含：
    - hist_bins ：直方图 bins 数（实际为 256）
    - texture ：纹理特征字典（contrast/energy/homogeneity/entropy）
    - outputs ：本张图生成的全部文件名列表
  - main() 将每张图的返回结果逐行写入：
    - exp1/output/summary.txt

5. 结果检查与对比

- 逐张查看 output/ 目录下生成的 6 个文件是否齐全（4 张 Sobel＋1 张直方图 ＋1 个纹理特征 .npy ）。
- 对比 *_sobel_gx.png 与 *_sobel_gy.png ：验证水平/垂直边缘响应差异；再看 *_sobel_mag.png 是否综合呈现主体轮廓。
- 观察 *_rgb_hist.png ：记录峰值位置与分布宽窄，描述图像偏亮/偏暗与主色倾向。
- 打开 summary.txt ：将其中 texture 数值作为纹理分析的数据依据（例如对比不同图片的熵/能量差异，支撑“纹理复杂度/均匀性”的结论）。五、实验数据记录（过程记录与对比）

- 数据集/图片类型记录（示例写法，按你实际图片替换即可）：
  - 图像 A：文字/印刷体边界清晰，背景干净。
  - 图像 B：道路/建筑，有阴影与纹理，边缘多且细碎。
  - 图像 C：物体轮廓明显但背景复杂，含噪声与细纹理。
- 参数记录（建议用表格写在报告里；此处给出可直接粘贴模板）：

项目 取值 1 取值 2 现象与结论 高斯滤波核大小 0（不滤波） 5×5 滤波后噪声伪边缘减少，但细小边缘略变钝 Sobel 核大小 3×3 5×5 核越大越平滑，边缘更粗更稳定，但细节损失 梯度合成方式 $\sqrt{Gx^2 + Gy^2}$ ， |Gx|+|Gy| 近似法更快，效果相近但强边缘更“亮” 二值化阈值 T 40 80/120 T 低：边缘多且噪点多；T 高：边缘干净但可能断裂

- 输出结果记录：
  - 原图、灰度图
  - (G_x)、(G_y)（可选但推荐）
  - 梯度幅值图（未阈值）

- 阈值化后的边缘二值图（至少两组不同阈值对比）

六、实验结果与分析

1.实验结果

Figure 1 **原图**

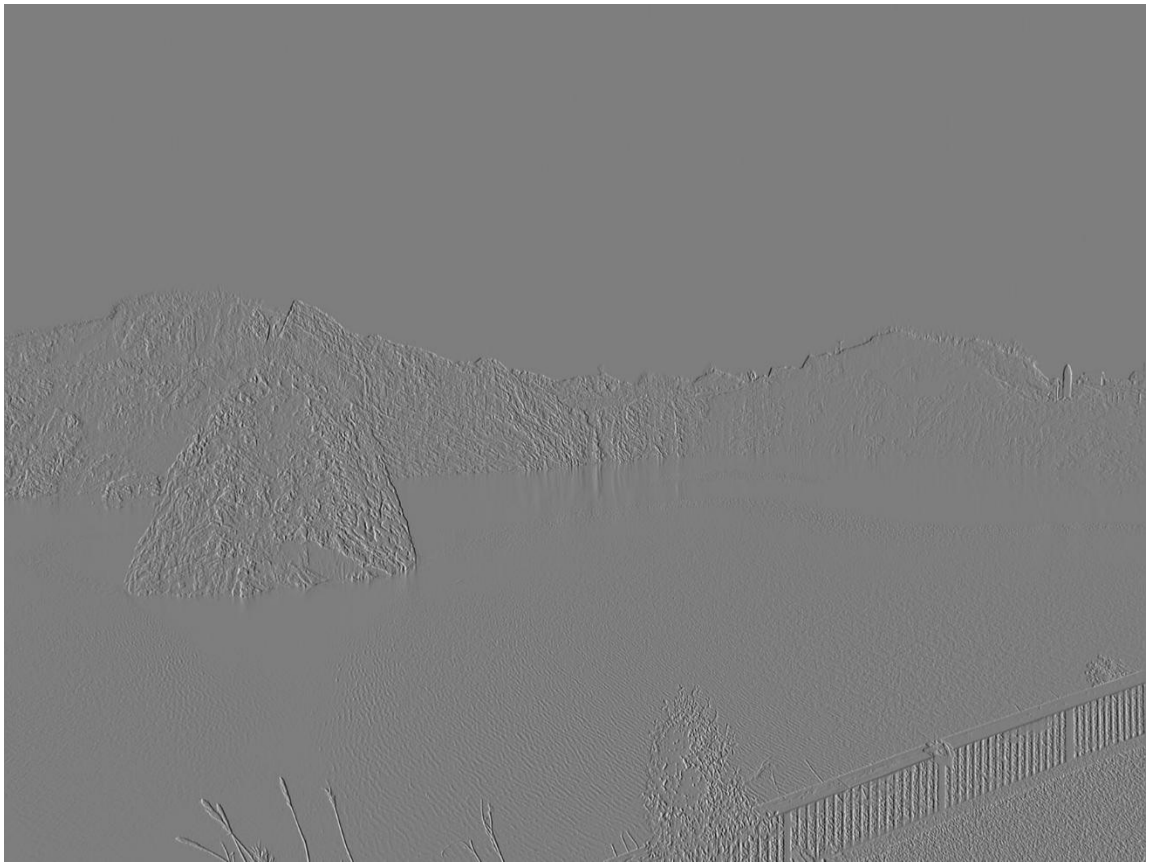

Figure 2 给定卷积核滤波后图片

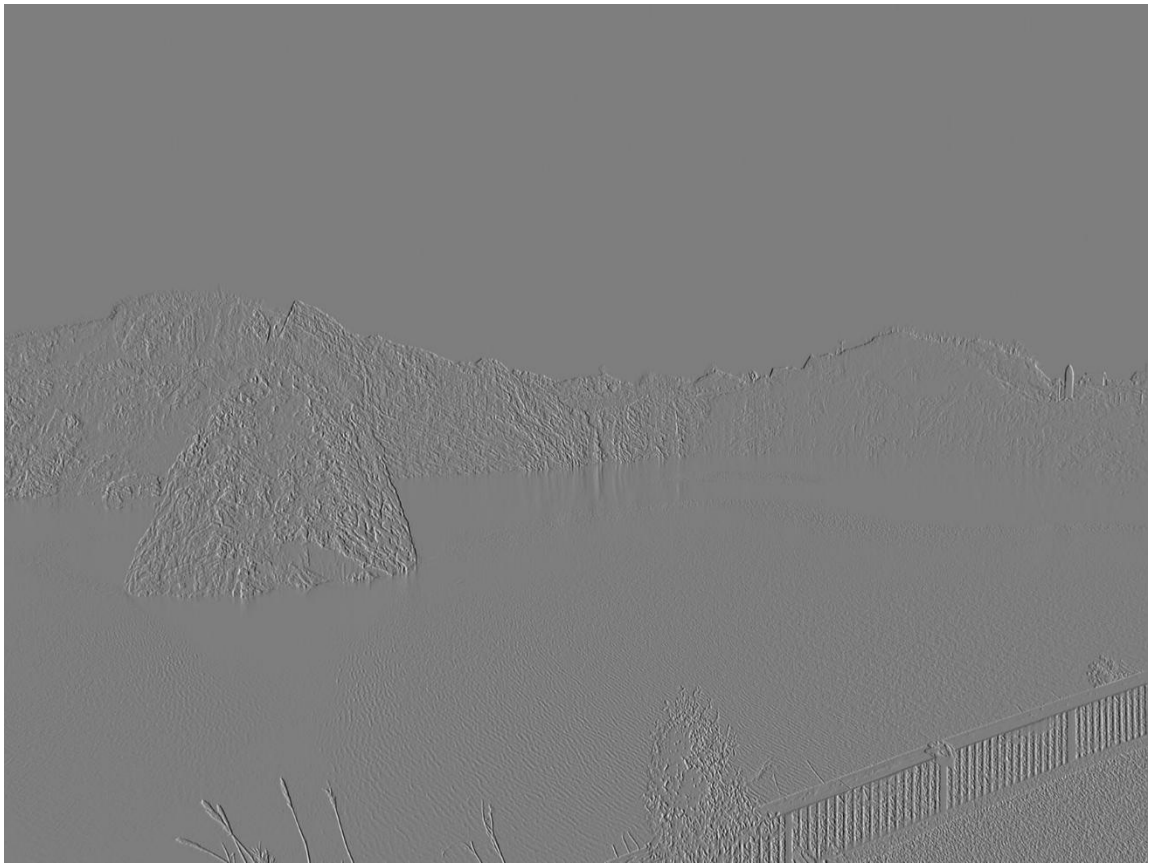

Figure 3 横向 sobel 算子滤波后图片

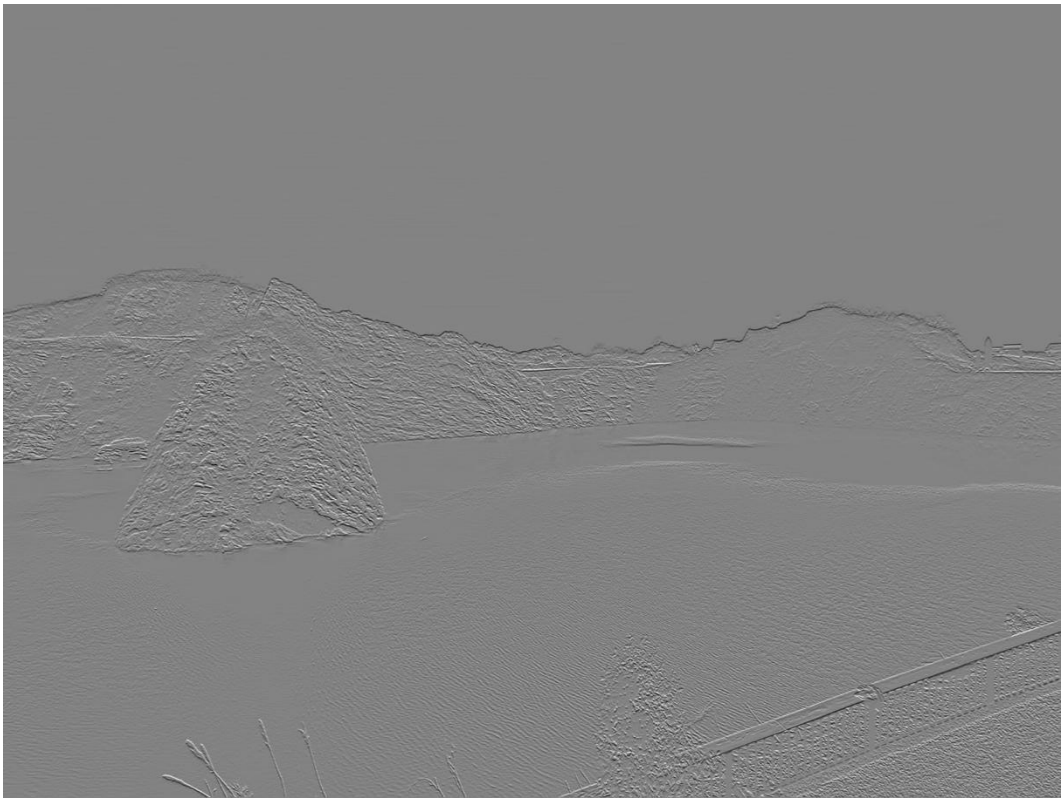

Figure 4 纵向 sobel 算子滤波后图片

Figure 5 sobel 算子滤波后图片

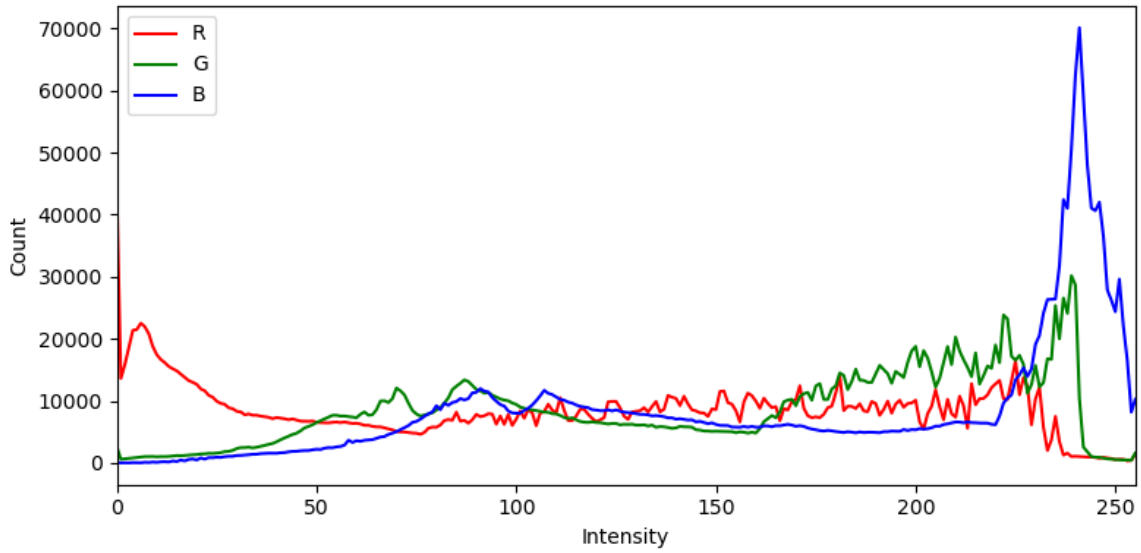

Figure 6 图像的颜色直方图

2.结果分析

- 现象 1：阈值对边缘数量影响显著
  - 阈值较低时，纹理与噪声也会被当作边缘保留，边缘“毛”“碎”。
  - 阈值较高时，背景更干净，但弱边缘（阴影边界、细线条）会断裂甚至消失。
  - 结论：阈值需要结合图像对比度与噪声水平选择；同一阈值难以适配所有图像。
- 现象 2：去噪对“伪边缘”抑制有效，但会损失细节
  - 不滤波：纹理丰富的图像容易产生大量伪边缘。
  - 适度高斯滤波：边缘更连贯、噪点更少；但细小结构会变弱。
  - 结论：边缘检测通常与去噪成对出现，强调“稳定边缘”时应先去噪。
- 对比 1：自编实现 vs OpenCV 实现
  - 效果：在同样核与后处理下，两者边缘位置基本一致；差异主要来自边界填充方式、数据类型截断/归一化策略。
  - 性能：OpenCV 通常更快（底层优化/向量化），自编更利于理解原理。
  - 结论：工程应用优先用库函数；学习与可控实验优先自编验证。
- 对比 2：幅值计算两种方式

  -$\sqrt{Gx^2 + Gy^2}$ 更符合几何意义；(|G_x|+|G_y|) 计算更快。

  - 在多数视觉展示中差异不大，但在后续需要精确强度（如非极大值抑制）时建议使用平方和开方形式。

七、结论

- Sobel 算子通过估计一阶梯度实现边缘检测，能较好提取主要轮廓边界。
- 最终效果主要受三类因素影响：噪声水平（需去噪）、算子尺度（核大小）、阈值策

略（强弱边缘取舍）。
- 通过对不同图片与参数的对比，可以在“边缘完整性”和“噪点抑制”之间找到适合任务的平衡点。

八、实验体会与个人理解

这次实验让我对“特征”这个概念有了更工程化的理解：特征并不是越复杂越好，而是要为后续任务服务、能稳定地区分不同图像内容。Sobel 的梯度幅值图直观呈现了轮廓与结构信息，RGB 直方图给出全局颜色与亮度分布，GLCM 则把“纹理的规则性/复杂度”用统计量固化下来。把三类特征放在同一套流水线里做对比，我更清楚地认识到它们的互补性：边缘强调几何形状，直方图强调整体色彩，纹理强调局部灰度关系，单一特征往往只能描述图像某一个侧面。
在实现层面，自己从零写卷积、统计与矩阵计算的过程，比直接调库函数更能暴露关键细节。比如卷积时 padding 的策略会影响边界响应，数据类型如果不升到浮点会出现截断或溢出导致结果异常，归一化方式也决定了输出图的可视化效果是否可比。尤其是 Sobel 的 G_x/G_y 与幅值图在保存前需要统一的 `normalize_to_uint8` 处理，否则不同图片、不同方向的响应强弱很难在同一尺度下比较。通过这些细节，我对“算法正确”和“工程可用”之间的差别有了更深刻的体会：前者强调数学定义无误，后者还要保证在各种输入下稳定、可解释、可复现。
在分析结果时，我也感受到参数与数据分布对结论的影响。直方图的峰值位置与分布宽窄能够很好地解释图像偏亮偏暗、色彩集中或分散，但它对空间结构不敏感；而 Sobel 对结构敏感，却容易把纹理和噪声也当作边缘，导致“边缘很多但信息不一定更好”。GLCM 的 contrast/energy/homogeneity/entropy 能够量化纹理的粗细与随机性，但它依赖量化等级（levels）与方向统计，若图像光照变化明显或灰度范围不同，量化与归一化就会显著影响特征值。也正因为如此，我学会了在报告中用“现象—原因—改进”的方式表达：看到某张图边缘碎，就去对应噪声与纹理；看到熵偏高，就去对应纹理复杂、灰度共现分布更分散。
更重要的是，这个实验训练了我一种可迁移的实践方法：先用模块化流水线输出中间结果（如 `{name}_sobel_gx/gy/mag.png`、`{name}_rgb_hist.png`、`{name}_texture.npy`），再用这些可视化与数值统计去反推算法是否符合预期。相比只盯着“最终结果”，这种过程可解释性更强，也更适合答辩表达。对我来说，这次实验不仅巩固了边缘、直方图、纹理的基础知识，更建立了“实现—验证—分析—归纳”的完整闭环，为后续更复杂的视觉任务打下了扎实基础。