

# 基于小波变换的图像边缘检测

PB22050980 王开元

May 31 2025

## 1 背景介绍

### 1.1 选题介绍

边缘检测（Edge detection）是图像处理和计算机视觉中的基本问题。边缘检测可以大幅度地减少数据量，并且剔除可以认为不相关的信息，广泛应用于目标识别、特征提取、图像分割等领域。

小波变换以多分辨率分析（Multiresolution Analysis）为原理，将信号表示为不同分辨率下的近似和细节的叠加。这种分析方式非常适合处理具有局部特征或非平稳性质的信号，例如图像、音频、地震波等。正因如此，小波变换在检测图像边缘时比传统方法更为细致。

### 1.2 技术原理

#### 核心机制：二维离散小波变换（2D DWT）

二维离散小波变换将图像分解为四个部分：近似部分、水平细节、垂直细节和对角细节。而边缘信息主要存在于高频部分（即平细节、垂直细节和对角细节部分），通过分析高频部分的系数，我们可以有效地提取边缘。另一方面，多尺度检测可以允许我们分级提取粗边和细边。

我们使用 Python 语言来实现图像处理。用到的包如下所示：

---

```
1 import cv2 # OpenCV库, 用于图像处理
2 import numpy as np # 数值计算和数组处理
3 import matplotlib.pyplot as plt # 用于绘图和可视化
4 import pywt # Python Wavelet Transform 用于小波变换: 离散小波变换dwt
5 from skimage.filters import threshold_otsu # 用于Otsu二值化自动选择
6 from skimage.metrics import structural_similarity as ssim # 用于计算图像之间的结构相似度
7 from scipy.ndimage import zoom # 用于图像/数组的缩放
```

---

代码 1: 将图像转化为灰度图

## 2 算法流程

我们使用 Haar 小波基作为二维离散小波变换的基，并且根据输出结果手动调整二值化阈值，我们处理图片 *flower.jpg*，如下所示：（可选择的小波还有：Daubechies 小波、Symlets 小波、Coiflets 小波等。经过对比我们发现 Haar 小波和 Daubechies 1 小波的效果最好。以下变换默认都用 Haar 小波基）



图 1: flower.jpg (左) flower.jpg 的灰度图 (右)

## 2.1 预处理

导入图像并转化为转化为灰度图

```
1 plt.imshow(img, cmap='gray')
```

代码 2: 将图像转化为灰度图

## 2.2 小波分解

对图像进行小波分解，此处选择小波基为'haar' 小波

```
1 wavelet_Basis = 'haar'  
2 coeffs = pywt.wavedec2(img, wavelet = wavelet_Basis, level=2)  
3 cA2, (cH2, cV2, cD2), (cH1, cV1, cD1) = coeffs  
4  
5 # 准备画布，2层变换 + 原图共7幅图  
6 fig, axes = plt.subplots(3, 3, figsize=(15,8))  
7  
8 # 原图  
9 axes[0,0].imshow(img, cmap='gray')  
10 axes[0,0].set_title('原图')  
11 axes[0,0].axis('off')  
12  
13 # 近似系数  
14 axes[0,1].imshow(cA2, cmap='gray')  
15 axes[0,1].set_title('第2层\u2225近似系数\u2225cA2')  
16 axes[0,1].axis('off')  
17  
18 axes[0,2].axis('off')  
19  
20 #第二层  
21 axes[1,0].imshow(cH2, cmap='gray')  
22 axes[1,0].set_title('第2层\u2225水平细节\u2225cH2')  
23 axes[1,0].axis('off')
```

## 代码 3: 小波分解

得到的结果可视化如下。由于原图的分辨率较高（4928\*3264），可能不能很好地显示二次近似的效果，这里我们手动调整分辨率为 512\*341

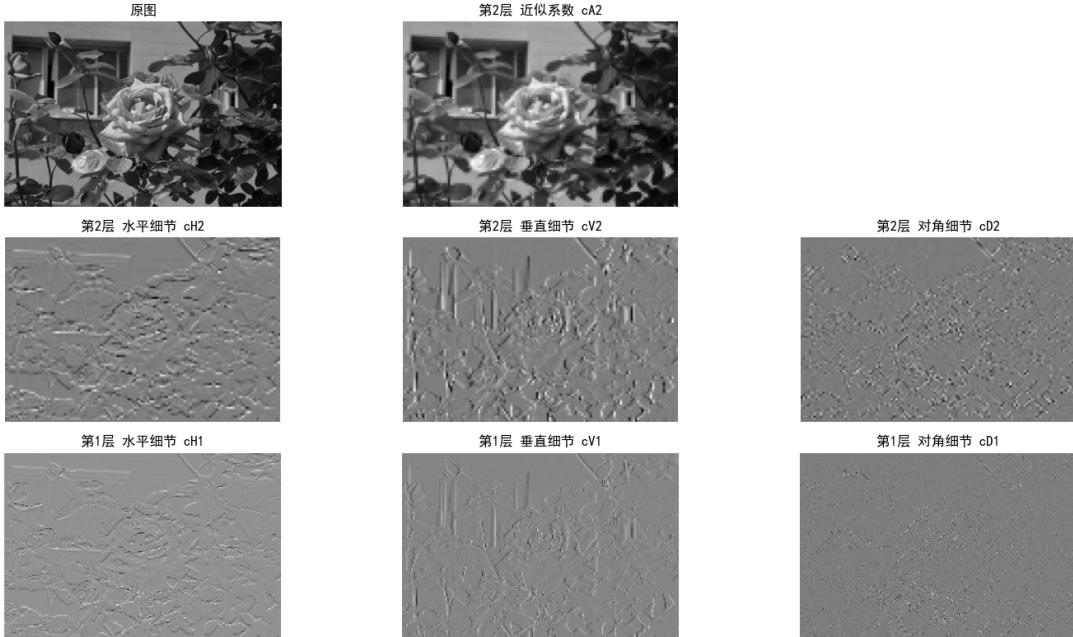


图 2: 两层二维小波分解的近似系数和细节系数

### 2.3 叠加边缘响应

我们知道，小波变换通过多尺度分解，将图像划分为近似部分和细节部分（对应近似系数和细节系数，它们都是灰度矩阵）。图像中的高频信息（也就是边缘）包含于细节部分：

1. cH: 水平细节，对垂直边缘敏感；
2. cV: 垂直细节，对水平边缘敏感；
3. cD: 对角细节，对斜边、角点敏感。

叠加边缘响应，就是将不同方向的细节系数强度相加（组合）起来，形成综合的**边缘强度图**。我们进行了两层小波变换，下面将每层的细节叠加在一起，并且将系数归一化：

```

1  #叠加边缘响应
2  edge_map = (np.abs(cH1_up) + np.abs(cV1_up) + np.abs(cD1_up) + np.abs(cH2_up) + np.
   abs(cV2_up) + np.abs(cD2_up))#这里的系数放大为和原图相同大小，加上后缀_up(这里的处
   理细节可忽略)
3
4  #归一化显示
5  edge_map_norm = cv2.normalize(edge_map, None, 0, 255, cv2.NORM_MINMAX).astype(np.
   uint8)

```

## 代码 4: 叠加边缘响应

我们的边缘响应叠加经过归一化和反色之后，得到了如下的边缘图：



图 3: 边缘响应叠加后的边缘图

可以看到我们已经提取到了令人满意的图片边缘。至此，和以小波分析为原理的图像处理已经基本结束。下面我们选择合适的灰度阈值对边缘图进行二值化，使其更加清晰。

### 3 阈值选择

在图像处理中，阈值分割是把灰度图像转化为二值图像的常用方法。确定阈值  $T$  之后，将灰度值大于  $T$  的像素赋值为 255，灰度值小于  $T$  的赋值为 0。选择合适的阈值，才能保留有效的边界信息，得到清楚的边界图。当然我们可以按照输出结果手动设置阈值，下面介绍一些计算阈值的方法。

#### 3.1 Otsu 阈值

(1) 原理：Otsu 方法寻找的阈值按照灰度，将像素划分为背景和前景两个部分，然后寻找阈值  $k$  使得类间方差  $\sigma_b^2$  取值最大，这里：

$$\sigma_b^2 = \omega_0\omega_1(\mu_0 - \mu_1)^2 \quad (1)$$

其中  $\omega_i, \mu_i$  分别为  $i$  类灰度的概率和平均灰度。(2) 特性：阈值的选择取决于全局的灰度分布情况，计算方便；同时，因为其全局性，可能无法很好地捕捉局部信息，例如局部灰度梯度所产生的边界可能被 Otsu 阈值忽略，也就是说光照不均时效果差。

#### 3.2 局部阈值

每个像素点的阈值由其邻域的平均值决定，处理更为复杂

这里对于归一化后的灰度图自动选取阈值，输出的结果为  $thresh = 28$ 。

---

<sup>1</sup> (`from skimage.filters import threshold_otsu`)

```

2
3 otsu_thresh = threshold_otsu(edge_map_norm) #自动计算Otsu阈值

```

---

或者手动选择阈值  $thresh = 17, 25, 30, 35$  对应的二值化图如下：

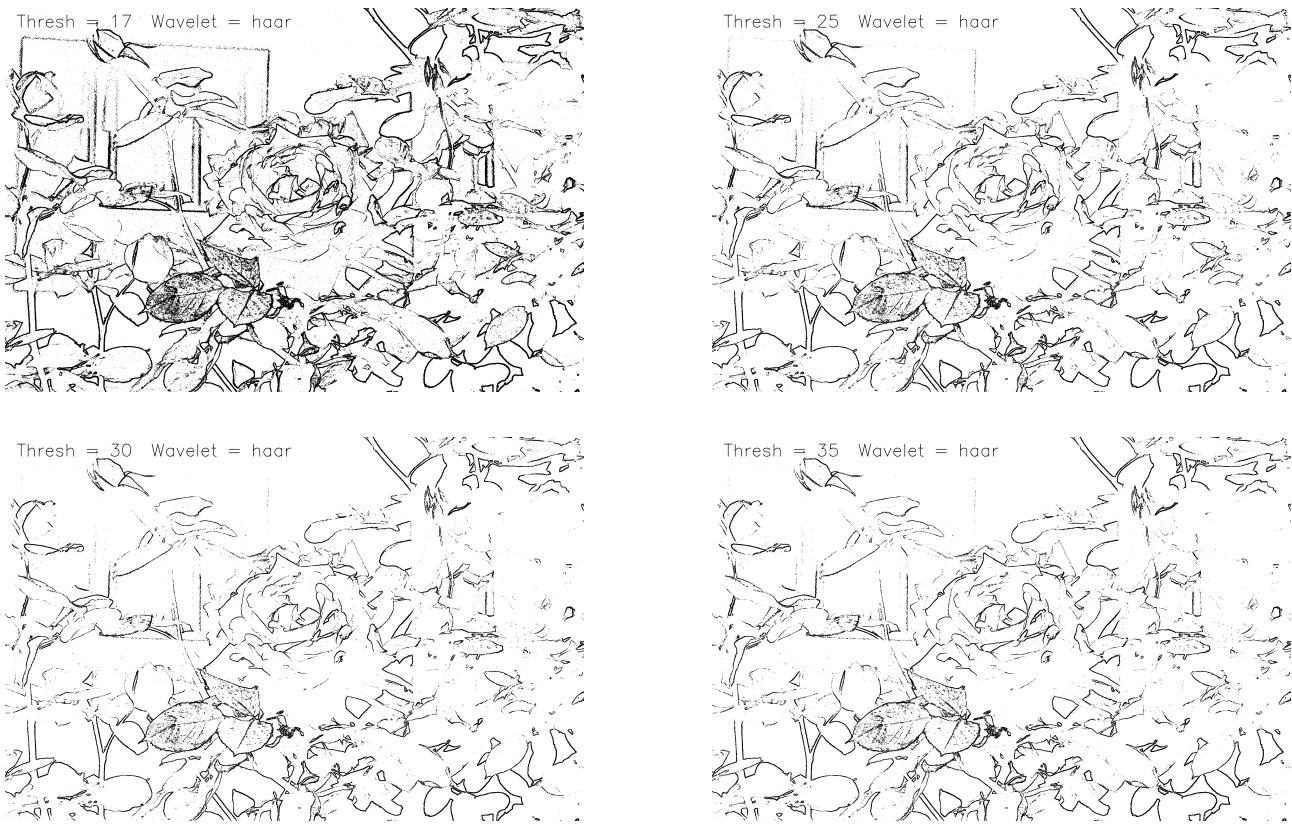


图 4: 不同阈值下二值图的对比

## 4 对照实验

下面我们选择 Canny 边缘检测算法（来自 OpenCV 库）作为对照。Canny 算法的原理如下：

1. 高斯模糊平滑图像，减少噪声影响；
2. 计算梯度，也就是边缘强度和方向，利用的是 Sobel 算子；
3. 非极大值抑制，只保留局部最大值，细化边缘，把粗边缘变成细线（1 像素宽）；
4. 双阈值处理：当边缘强度  $>$  高阈值时，认为其为强边缘并保留；当边缘强度  $<$  低阈值时，认为是非边缘并丢弃；当边缘强度介于两个阈值之间时，认为是弱边缘，若与强边缘相连则保留，否则丢弃；

我们发现当选择两个阈值分别为  $t1 = 80, t2 = 100$  的时候提取边缘效果较好，代码如下：

```

1 t1 = 50 #手动设置上下阈值
2 t2 = 100
3 canny_edges = 255- cv2.Canny(img, t1, t2)
4 plt.imshow(canny_edges, cmap='gray') #用灰度图显示边缘图
5 plt.axis('off') #关闭坐标轴
6 plt.title("Canny Edges")
7 plt.show() #显示图像

```

```
8  
9     filename = f"flower_canny_edge_{t1}_{t2}.png"  
10    cv2.imwrite(filename, canny_edges)
```

---

代码 5: Canny 边缘检测

和二值化阈值为 25 的 Haar 小波算法的比较如下:

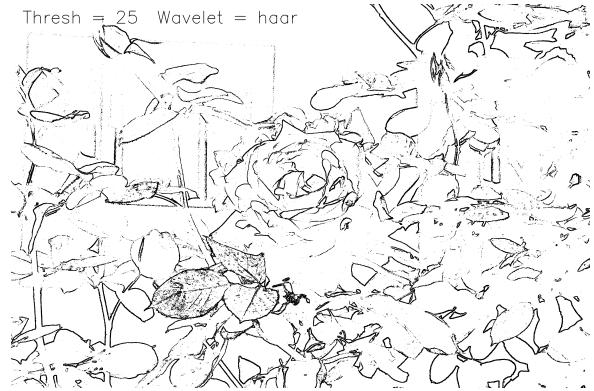


图 5: Canny 边缘检测 ( $t_1=80, t_2=100$ ) (左) Haar 小波算法 ( $t=25$ ) (右)

可以看到, 因为 Canny 算法进行了细化边缘的操作, 所以得到的边缘线宽度固定为 1 像素, 颜色较浅 (尽管本身就是二值化的灰度图), 因此不会反映出原图边缘的宽度。另一方面, 因为边缘宽度小, 所以更可能显得不连续或者不光滑。但总体上 Canny 算法也很好地刻画出原图像的边界, 应该根据不同的需求选择合适的算法。

## 5 Python 代码

< 见附件 *wavelet\_Edge.ipynb* >

## 6 更多的边缘提取例子

< 见附件 *examples* >