合肥工業大學

（计算机与信息学院）

机器视觉实验报告

专　业　班　级　智能科学与技术 21-1 班

学 生 姓 名 学 号　郭宇童 2021214582

任　课　教　师　洪日昌

实　验　名　称　实验 1 图像滤波

2023-2024 学年第 1 学期

摘要： Sobel 算子是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导的概念，以在去除噪声的同时突出图像的边缘。本文解释了 Sobel 算子的数学原理，包括其如何通过卷积核对图像进行水平和垂直方向的微分来检测边缘。Sobel 算子包括两个卷积核，一个用于检测水平边缘，另一个用于检测垂直边缘。这些核在图像上滑动，计算相邻像素的强度变化。

接下来，文章讨论了 Sobel 算子在实际图像处理中的应用。在图像边缘检测中，Sobel 算子能有效识别物体的轮廓和其他重要特征。文章还提供了一些实现 Sobel 算子的编程技巧和算法，包括如何处理图像边界和选择适当的阈值以优化边缘检测效果。本文通过实例展示了 Sobel 算子处理前后的图像对比，说明了其在提高图像分析和处理质量方面的效果。尽管 Sobel 算子是一个相对简单的技术，但它在计算机视觉和数字图像处理领域中仍然是一个非常重要和有效的工具。。

关键词：Sobel 算子；索贝尔算子；图像滤波；计算机视觉；数字图像处理；

引言

在数字图像处理领域，边缘检测是一项关键任务，它涉及到从图像中识别出物体的轮廓和其他重要特征。边缘检测不仅对于图像分析至关重要，而且在计算机视觉、图像分割、模式识别等多个领域中都有广泛应用。在众多边缘检测技术中，Sobel 算子因其简单高效而广受欢迎。

Sobel 算子于 1968 年由 Irwin Sobel 和 Gary Feldman 提出，目的是在数字图像中识别边缘。该算子使用两个 3x3 的卷积核，分别对图像的水平和垂直方向进行微分，从而检测出图像边缘的位置。这种方法的优势在于其相对简单的计算过程和对噪声的鲁棒性，使其成为初学者和图像处理专家的首选工具。

本实验的目的是探索 Sobel 算子在图像滤波和边缘检测方面的应用。通过对不同类型的图像使用 Sobel 算子，我们将评估其在实际场景中的表现，包括对不同特征和噪声水平的响应。实验将涵盖 Sobel 算子的实现细节，包括核的应用、边界处理、以及最终图像的阈值设定。

通过这个实验，我们旨在深入了解 Sobel 算子的工作原理，以及在现代图像处理和计算机视觉应用中的实际效果。我们还将探讨 Sobel 算子与其他边缘检测算法的比较，以及在不同应用背景下的优缺点。

1 Sobel 算子简介

1.1 Sobel 算子的数学原理

Sobel 算子是一种用于边缘检测的计算算法，在图像处理中具有重要地位。它主要用于计算图像亮度函数的近似梯度，进而突出图像中的边缘部分。

1.1.1 梯度的概念

在数学中，梯度是一个向量，表示多变量函数在某一点的最大上升率及其方向。对于一

个给定的函数 $f(x,y,z,\ldots)$，其梯度表示为 ∇f，是一个向量场，指向最大变化率的方向。在三维空间中，梯度可以表示为：$f=(\partial x/\partial f,\partial y/\partial f,\partial z/\partial f)$。

在图像处理领域，梯度的概念被用于边缘检测。图像可以被视为二维函数，其中每个像素的强度或颜色值对应于函数的值。图像的梯度代表了像素强度的变化最快的方向和速率。较高的梯度值通常对应于图像中的边缘或颜色变化区域。

1.1.2 离散微分近似

在图像处理和其他计算机视觉任务中，离散微分近似是一个核心概念。由于图像是由离散像素点构成的，我们不能直接应用连续数学中的微分方法。因此，需要使用离散微分近似来估计图像中的梯度和其他导数。

在连续数学中，微分用于描述函数值随自变量的连续变化率。而在离散环境中，比如数字图像，我们只能访问有限的、离散的像素点。因此，需要将微分的概念**离散化**。离散微分近似通过计算相邻像素点间的差分来**近似导数**。例如，在一维情况下，函数 f 在点 x 的导数可以通过 $f(x+1)-f(x)$ 来近似。

对于二维图像，我们需要在两个方向（通常是水平和垂直）上计算梯度。图像梯度可以看作是图像亮度（或颜色强度）变化的向量表示。在实际应用中，我们通常使用**差分算子**，如 Sobel 算子，来计算图像的梯度。这些算子通过在特定方向上比较邻近像素值来估计梯度。

在一维情况下，离散微分可以用差分表达，例如 $\Delta f(x)=f(x+1)-f(x)$。在二维图像中，我们对每个像素计算两个方向（水平和垂直）上的差分。例如，水平方向的差分可以表示为 $\Delta f(x,y)=f(x+1,y)-f(x,y)$，垂直方向则为 $\Delta f(x,y)=f(x,y+1)-f(x,y)$。

1.1.3 卷积核

Sobel 算子使用两个 3x3 的矩阵（即卷积核）来计算水平和垂直方向上的梯度。这两个核分别为：

水平方向（Gx）：

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

垂直方向（Gy）：

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

不同的卷积核设计可以捕获不同类型的图像特征。Sobel 核是设计来突出边缘的，而其他核可能用于检测特定方向的纹理或模式。

1.1.4 梯度计算

对于图像中的每个像素点，Sobel 算子先分别应用这两个卷积核，通过卷积操作计算出该点在水平和垂直方向上的梯度 Gx 和 Gy。然后，根据这两个梯度值，可以计算出该点的梯度幅度和方向：

梯度幅度：$G=Gx^2+Gy^2$

梯度方向：$\theta=\arctan(Gx/Gy)$

1.1.5 边缘检测

在得到每个像素点的梯度幅度后，Sobel 算子将通过设置一个阈值来确定哪些点是边缘。梯度幅度高于该阈值的点被认为是边缘点

1.2 Sobel 算子的应用举例

Sobel 算子作为一种流行的边缘检测工具，在图像处理和计算机视觉领域有着广泛的应用。以下是一些具体的应用实例：

表 1 Sobel 算子的应用

应用	描述
图像边缘检测	Sobel 算子识别的边缘可以作为分割不同区域的基础
计算机视觉	物体识别与视觉效果分析
数字图像处理	特征提取与增强图像细节
医学影像分析	医学成像如 MRI 或 CT 扫描图像分析
实时视频处理	监控系统 Sobel 算子可以用于实时检测移动物体的边缘

2 图像滤波实验的编程实现

2.1 卷积的实现

首先定义卷积核的内容

```
# 对图像进行给定卷积核滤波
custom_kernel = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
```

图 定义卷积核

循环实现卷积操作：

- 外层循环：**for i in range(1, image.shape[0]-1):**

这个循环遍历图像的每一行，除了最上边和最下边的边界行。image.shape[0]是图像的高度（行数），循环从第二行开始到倒数第二行结束。

- 内层循环：**for j in range(1, image.shape[1]-1):**

这个循环遍历图像的每一列，除了最左边和最右边的边界列。image.shape[1]是图像的宽度（列数），循环从第二列开始到倒数第二列结束。

- 卷积操作：**filtered_result[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * custom_kernel)**

这行代码是卷积的核心。它首先截取以当前像素(i, j)为中心的 3x3 邻域（image[i-1:i+2, j-1:j+2]），这是因为卷积核通常是 3x3 大小。然后，将这个 3x3 区域与卷积核（custom_kernel）相乘。这里的乘法是逐元素相乘。使用 np.sum 计算上述逐元素乘积的总和，得到的结果是卷积操作的输出，赋值给 filtered_result[i, j]。这意味着每个像素点的新值是由其周围邻域像素值经过卷积核处理后得到的。

```
for i in range(1, image.shape[0]-1):
    for j in range(1, image.shape[1]-1):
        filtered_result[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * custom_kernel)
```

图 卷积操作实现

2.2 定义 Sobel 算子滤波函数

- 定义 Sobel 算子：

sobel_x 和 sobel_y 分别代表 Sobel 算子的水平和垂直核。水平核 sobel_x 用于检测垂直边缘，而垂直核 sobel_y 用于检测水平边缘。

- 应用 Sobel 算子：

使用 OpenCV 的 filter2D 函数，将 Sobel 核应用于输入的图像 image。filter2D 函数进行卷积操作，将核在整个图像上滑动，计算核与图像邻域的点积。

sobel_x_img 和 sobel_y_img 分别是应用水平和垂直 Sobel 核后得到的图像。

- 转换数据类型：

将 sobel_x_img 和 sobel_y_img 转换为浮点数类型 (np.float64)。这是为了确保在接下来的计算中不会有数据溢出或精度损失。

- 计算梯度幅度：

使用 cv2.magnitude 来计算 Sobel 算子在水平和垂直方向上的梯度的总和。这个函数计算每个像素的梯度幅度，使用了以下公式：$\sqrt{sobel\text{-}x\text{-}img\text{^}2+sobel\text{-}y\text{-}img\text{^}2}$。

得到的 sobel_combined 是综合了水平和垂直方向梯度信息的图像，它反映了图像中的边缘强度。

- 返回结果：

函数返回 sobel_combined，这是一个表示图像边缘强度的二维数组。较亮的区域表示边缘较强，较暗的区域表示没有或边缘较弱。

```python
def apply_sobel(image):
    # Sobel算子
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    # 对图像应用Sobel算子
    sobel_x_img = cv2.filter2D(image, -1, sobel_x)
    sobel_y_img = cv2.filter2D(image, -1, sobel_y)

    # 将Sobel滤波器的结果转换为浮点类型
    sobel_x_img = sobel_x_img.astype(np.float64)
    sobel_y_img = sobel_y_img.astype(np.float64)

    # 计算总的梯度近似值
    sobel_combined = cv2.magnitude(sobel_x_img, sobel_y_img)
```

图 Sobel 算子实现

2.3 颜色直方图的实现

- 初始化直方图数组：

这一步创建了一个 256x3 的二维数组，用于存储三个颜色通道（红、绿、蓝）的直方图。256 表示可能的像素值范围（0-255）。

- 计算每个颜色通道的直方图：

通过一个循环，对图像的每个颜色通道（通道 0 为红色，1 为绿色，2 为蓝色）进行处

理：

color_channel = image[:, :, i] 选取当前颜色通道的数据。

np.histogram 函数计算该颜色通道的直方图。bins=256 表示直方图分为 256 个柱状区间，range=[0, 256] 指定像素值的范围。

直方图结果存储在 color_histogram[:, i] 中，对应于当前处理的颜色通道。

- 可视化直方图：

使用 Matplotlib 的 plt.plot 函数来绘制每个颜色通道的直方图。

图像的标题、X 轴和 Y 轴标签分别设置为 'Color Histogram'、'Pixel Value' 和 'Frequency'。

- 展示图像：

plt.show() 调用显示出绘制的直方图。

```
# 可视化颜色直方图
color_histogram = np.zeros((256, 3), dtype=np.int)

for i in range(3):
    color_channel = image[:, :, i]
    hist, _ = np.histogram(color_channel, bins=256, range=[0, 256])
    color_histogram[:, i] = hist

plt.plot(color_histogram)
plt.title('Color Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()

# 保存纹理特征至npy格式
texture_feature = np.array([sobel_result])
texture_feature1=np.array([filtered_result])

load_npy(texture_feature1)
load_npy(texture_feature)


np.save('./assinment_1/texture_feature.npy', texture_feature)
```

图 颜色直方图实现

3 实验运行结果

原始的图像如下所示：

选取图像为我的 pc 桌面截图，可以看到图像比较复杂，但是有比较明显的边缘信息。

图 处理的源图片

经过普通卷积的图像如下，可见纹理粗糙边缘不明显，噪音较多。

图 卷积后的图像

经过 Sobel 算子处理的图像为：

图 sobel 滤波后的图像

可视化的颜色直方图为：

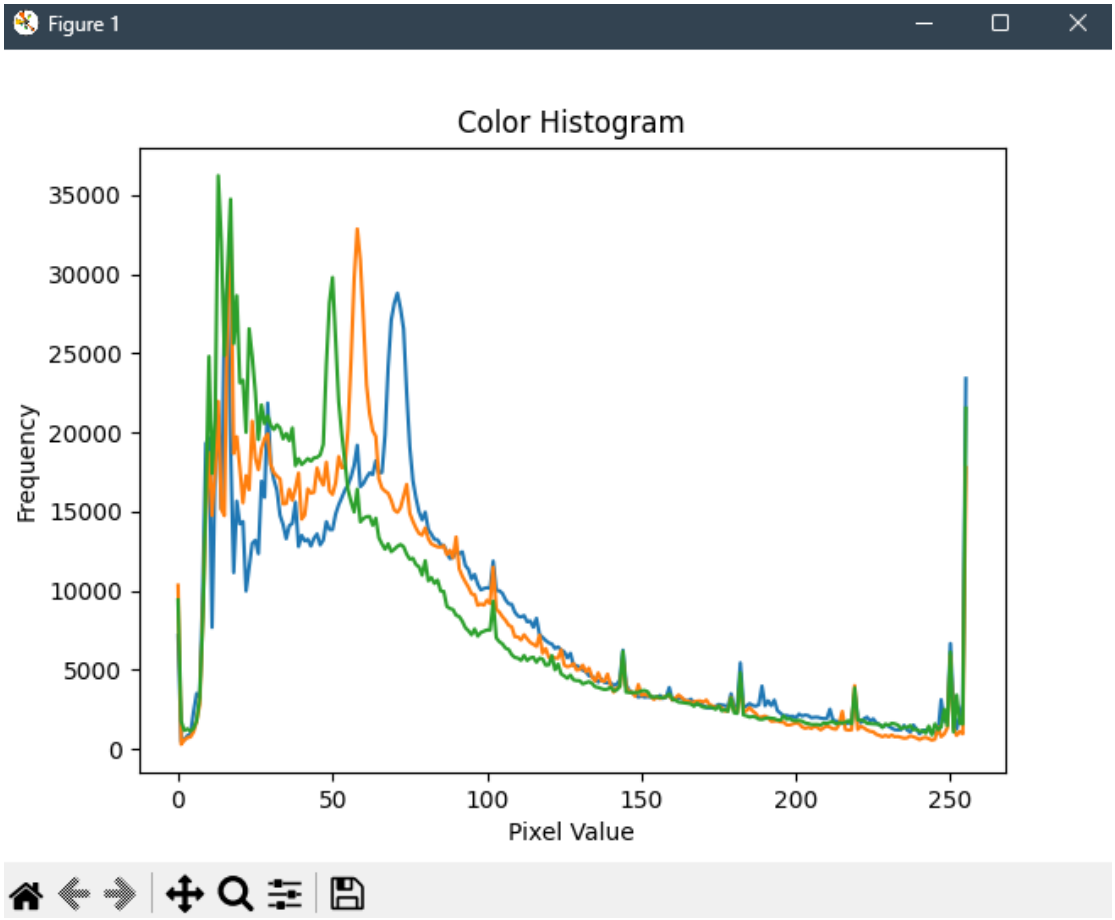

图 可视化的颜色直方图