

**计算机视觉实验报告**

**实验题目**  图像滤波

**学生姓名**  杜述超

**学 号**  2021214745

**专业班级**  21计科-1班

**指导教师**  吴晶晶

**完成日期**  2023.12.20

**合肥工业大学 计算机与信息学院**

# 实验目的

使用Sobel算子、给定卷积核滤波自己拍摄的图像，并提取图像的颜色直方图和纹理特征

具体要求：

➢ 任务输入：自己拍摄的图像。

➢ 任务输出：经过Sobel算子滤波的图像，经过给定卷积核滤波的图像，可视化图像的颜色直方图，保存纹理特征至npy格式。

➢ 滤波、直方图计算、纹理特征提取过程不可以调用函数包。

➢ 代码语言不限，纹理特征提取方法不限，要求提交整个算法源代码，实验结果（算法输入图片、输出图片 、直方图，纹理结果），算法分析等内容

# 实验原理

1. **Sobel算子**：Sobel算子是一种用于边缘检测的算法，它通过计算图像亮度的空间梯度来突出显示图像中的边缘。Sobel算子包括两个卷积核（或过滤器），分别用于水平和垂直方向的边缘检测。这些卷积核在图像上应用时，会分别计算图像在水平和垂直方向上的梯度强度，从而揭示出图像的边缘结构。
2. **卷积核滤波**：卷积核（或滤波器）是一种在图像处理中用于图像增强、平滑、锐化等目的的工具。它通过与图像中的每个像素及其周围像素的加权和运算来改变该像素的值。当使用特定的卷积核对图像进行滤波时，可以提取图像的某些特征或实现特定的效果，如边缘增强、模糊等。
3. **颜色直方图**：颜色直方图是图像颜色分布的图形表示，它统计了图像中每种颜色的像素数。在数字图像处理中，颜色直方图可用于图像分析、图像检索、图像分割等。通过提取图像的颜色直方图，可以得到有关图像颜色分布的信息，这对于理解和分析图像内容非常有帮助。
4. **纹理特征提取**：纹理是指图像中重复出现的局部图案和结构，是图像区分不同区域的重要特征之一。在图像处理中，纹理特征的提取是为了获取图像中纹理的信息，这些信息可以用于图像分类、识别等。提取纹理特征通常涉及计算图像中某个区域的统计度量，如灰度共生矩阵、局部二值模式等。

本实验的核心在于综合运用图像处理的多种基本技术（边缘检测、滤波、颜色和纹理特征提取）来分析和处理图像。通过实际操作，可以加深对这些技术原理及其应用的理解。

# 实验内容

1. 滤波

由于Sobel算子和给定算子滤波算法思路一模一样，故下面仅给出Sobel算子的核心算法思路。

Sobel算子核心算法思路：

Sobel边缘检测是一种常用的图像处理技术，用于在数字图像中突出显示边缘。该算法通过计算图像中每个像素点的灰度变化程度来识别边缘。具体步骤如下：

1. **输入图像**：该函数接收一个**cv::Mat**类型的图像（**img**）作为输入，该图像为灰度图像。
2. **初始化结果矩阵**：函数创建了一个与输入图像大小相同的新矩阵**result**，作为处理结果的存储。
3. **遍历图像像素**：函数通过两层嵌套循环遍历输入图像中的每个像素（除了边缘像素，因为边缘像素没有足够的邻域像素进行计算）。循环中的**i**和**j**分别表示像素的行和列。
4. **计算索引**：对于每个像素点，计算其及其8个相邻像素在图像数据数组中的索引位置。
5. **Sobel运算**：算法通过两个方向的Sobel卷积核来计算该像素点的梯度：
   * **gx**为水平方向的梯度，计算方法是将中心像素左侧3个像素的灰度值与右侧3个像素的灰度值进行比较，以找出水平方向上的灰度变化。
   * **gy**为垂直方向的梯度，计算方法是将中心像素上方3个像素的灰度值与下方3个像素的灰度值进行比较，以找出垂直方向上的灰度变化。
6. **计算边缘强度**：使用公式**sum = abs(gx) + abs(gy)**计算每个像素点的边缘强度。这个值是水平和垂直方向梯度的绝对值之和。
7. **阈值处理**：如果计算得到的边缘强度超过255（灰度图像的最大值），则将其限制在255。
8. **生成结果图像**：将计算得到的边缘强度作为新图像中对应像素的灰度值。
9. **返回结果**：函数返回处理后的图像，即边缘被突出显示的图像。

核心代码如下：

// Sobel边缘检测函数

cv::Mat Sobel(const cv::Mat& img)

{

int rows = img.rows; // 图像的行数

int cols = img.cols; // 图像的列数

cv::Mat result = img.clone(); // 创建结果图像，初始化为输入图像的副本

uchar\* data = img.data; // 指向图像数据的指针

// 遍历图像中的每个像素，除了边界像素

for (int i = 1; i < rows - 1; i++)

{

for (int j = 1; j < cols - 1; j++)

{

// 计算当前像素及其邻域像素在一维数组中的索引

int index = i \* cols + j;

int index1 = (i - 1) \* cols + j - 1;

int index2 = (i - 1) \* cols + j;

int index3 = (i - 1) \* cols + j + 1;

int index4 = i \* cols + j - 1;

int index5 = i \* cols + j + 1;

int index6 = (i + 1) \* cols + j - 1;

int index7 = (i + 1) \* cols + j;

int index8 = (i + 1) \* cols + j + 1;

// 计算Sobel水平和垂直方向上的梯度

int gx = data[index1] + 2 \* data[index2] + data[index3] - data[index6] - 2 \* data[index7] - data[index8];

int gy = data[index1] + 2 \* data[index4] + data[index6] - data[index3] - 2 \* data[index5] - data[index8];

// 计算边缘强度

int sum = abs(gx) + abs(gy);

// 限制边缘强度不超过255

if (sum > 255)

sum = 255;

// 更新结果图像中对应像素的灰度值

result.data[index] = sum;

}

}

return result; // 返回处理后的边缘检测图像

}

运行结果：

Sobel：



图表 1 Sobel算子滤波结果

指定算子：



图表 2 指定算子滤波结果

由上图可以看见指定滤波在y轴方向上明显会弱化不少，Sobel算子的结果会在两个方向都有相应的效果，所以符合实验结果。

1. 提取颜色直方图

核心算法思路：

**颜色直方图计算** (**calcHist**函数)：

* 输入一个RGB颜色空间的图像。
* 初始化一个无序映射（哈希表），分别为红（R）、绿（G）、蓝（B）三个颜色通道创建一个长度为256的向量，用于存储每个颜色值的频率。
* 遍历图像的每个像素，对每个颜色通道的颜色值进行计数，增加对应颜色值在直方图中的频率。
* 返回包含三个颜色通道直方图的哈希表。

核心代码：

// 计算图像的颜色直方图

std::unordered\_map<char, std::vector<int>> calcHist(const cv::Mat& img)

{

int rows = img.rows; // 图像的行数

int cols = img.cols; // 图像的列数

// 初始化颜色直方图数据结构

std::unordered\_map<char, std::vector<int>> hist;

hist['r'] = std::vector<int>(256, 0); // 红色通道

hist['g'] = std::vector<int>(256, 0); // 绿色通道

hist['b'] = std::vector<int>(256, 0); // 蓝色通道

// 遍历图像的每个像素

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

{

cv::Vec3b cur = img.at<cv::Vec3b>(i, j); // 获取当前像素的RGB值

hist['b'][cur[0]]++; // 对蓝色值进行计数

hist['g'][cur[1]]++; // 对绿色值进行计数

hist['r'][cur[2]]++; // 对红色值进行计数

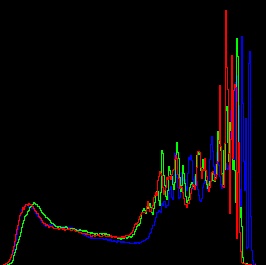
}

}

return hist; // 返回颜色直方图

}

运行结果：



图表 3 颜色直方图结果

我拍摄的照片是一副校园的雪景，其中白色为主色，纯白色在RGB中的值为[R,G,B]=[255,255,255]，由颜色统计直方图可以看出，RGB三原色的值最为集中的部分就在接近255的部分，所以该结果符合预取。由于我是使用的C++实现的代码，所以这里就没有保存至.npy格式，就保存了.jpg格式。

1. 提取纹理特征

核心算法思路：

1. **局部二值模式 (Local Binary Pattern, LBP) 特征提取**：
   * LBP是一种用于纹理分析的非参数化特征描述符。
   * 算法的核心是比较中心像素与其周围邻域像素的灰度值。
2. **LBP 特征计算流程**：
   * 对图像中的每个像素（除去边缘像素，因为边缘像素没有完整的邻域），考虑其周围8个邻域像素。
   * 将中心像素的灰度值与周围每个邻域像素的灰度值进行比较。
   * 如果邻域像素的灰度值大于中心像素的灰度值，则该邻域像素的位置被标记为1，否则为0。
   * 从中心像素的一个邻域像素开始（例如左上角），按顺时针或逆时针方向遍历所有邻域像素，形成一个8位的二进制数。
   * 将这个二进制数转换为十进制，作为该中心像素位置的新像素值。
3. **结果**：
   * 生成的新图像（result）将反映原始图像的纹理特征，其中每个像素的灰度值是基于其局部邻域的二进制模式计算得出的。

核心代码：

cv::Mat LBP(const cv::Mat& img)

{

int rows = img.rows; // 图像的行数

int cols = img.cols; // 图像的列数

cv::Mat result = img.clone(); // 创建结果图像，初始化为输入图像的副本

// 遍历图像的每个像素，除了边界像素

for (int i = 1; i < rows - 1; i++)

{

for (int j = 1; j < cols - 1; j++)

{

uchar cur = img.at<uchar>(i, j); // 获取当前像素的灰度值

uchar code = 0; // 初始化LBP代码

// 比较中心像素与周围邻域像素的灰度值，并生成LBP代码

code |= (img.at<uchar>(i - 1, j - 1) > cur) << 7;

code |= (img.at<uchar>(i - 1, j) > cur) << 6;

code |= (img.at<uchar>(i - 1, j + 1) > cur) << 5;

code |= (img.at<uchar>(i, j + 1) > cur) << 4;

code |= (img.at<uchar>(i + 1, j + 1) > cur) << 3;

code |= (img.at<uchar>(i + 1, j) > cur) << 2;

code |= (img.at<uchar>(i + 1, j - 1) > cur) << 1;

code |= (img.at<uchar>(i, j - 1) > cur);

// 将计算出的LBP值赋予结果图像的相应位置

result.at<uchar>(i, j) = code;

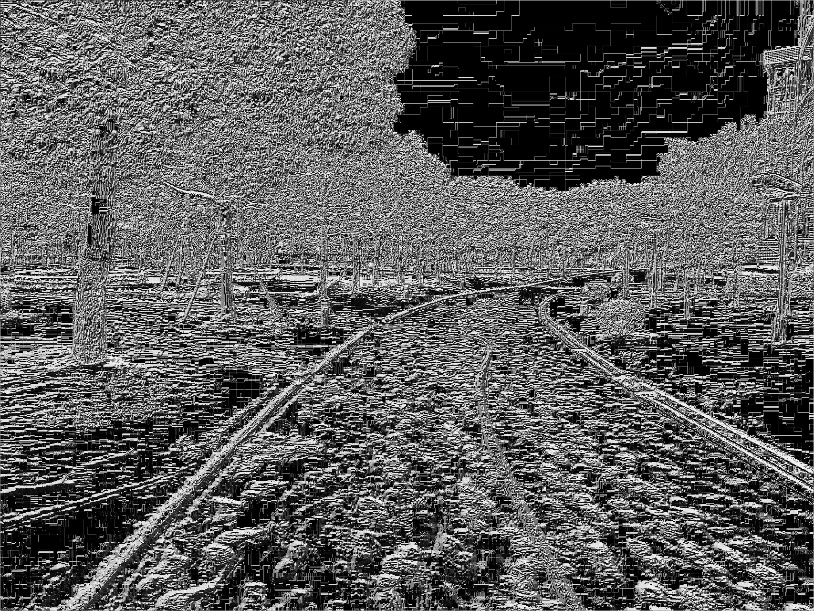
}

}

return result; // 返回LBP特征图像

}

运行结果：



图表 4 LBP特征提取运行结果

可以看见，每一个纹理相同的位置的特征都是类似的，比如树叶部分，提取出的纹理都是一致的，比如天空，纹理页基本是一样的，所以运行结果正确。

# 总结和感想体会

**实验概述**：

在这个实验中，我采用了Sobel算子和自定义卷积核对自拍的图像进行滤波处理，随后提取了图像的颜色直方图和纹理特征。整个过程中，我没有使用任何现成的函数包，而是手动实现了所有的处理步骤。这个过程不仅增强了我对图像处理基本概念的理解，还提升了我的编程技能。

**实验过程**：

1. **滤波处理**：
   * 使用Sobel算子处理图像，这一步骤帮助我深入理解了边缘检测的原理。通过计算图像的梯度，我能够清晰地看到图像中的边缘和细节部分。
   * 应用自定义卷积核进行滤波处理。这个步骤让我了解了滤波器对图像的影响，如何通过改变卷积核的权重来获得不同的滤波效果。
2. **颜色直方图提取**：
   * 手动计算图像的颜色直方图是一个挑战性的任务。这一过程要求我对图像的颜色分布有深入的理解，同时也需要精确的编程实现。
3. **纹理特征提取**：
   * 纹理特征提取让我探索了图像纹理的多种表达方式。尽管这部分相对复杂，但它极大地提高了我解决问题的能力和对图像特征理解的深度。

**实验结果与分析**：

* 通过Sobel算子和自定义卷积核滤波后的图像清晰地展示了滤波处理的效果。Sobel算子突出了图像的边缘，而自定义卷积核则根据其设计产生了y方向的滤波效果。
* 手动计算的颜色直方图成功地展示了图像中各种颜色的分布情况，这为理解图像的颜色构成提供了有价值的视角。
* 纹理特征的提取虽然复杂，但为进一步的图像分析提供了重要的基础。

**总体体会**：

这次实验是一次全面的学习经历，不仅提高了我的编程技能，还增强了我对图像处理技术的理解。手动实现滤波处理、直方图计算和纹理特征提取等过程，虽然具有一定的挑战性，但却极大地提升了我的分析和解决问题的能力。通过这次实验，我不仅学习了图像处理的理论知识，还获得了实际操作的经验，这对我的未来学习和研究将是非常宝贵的资产。