Homework Report

Sobel Operators

Dennis Romero L.

1 Introduction

This document describes the concepts related to the implementation in C++ and TensorFlow of Sobel operators. The code in the GitHub repository is organized as follows:

- 1_single_kernel: Implementation that demonstrate a convolution process
- 2_openmp_sobel_kernels: Multi-threaded implementation of fixed-size kernels (for simplicity purposes)
- 3_openmp_custom_sobel_kernels_size: Multi-threaded implementation that supports different kernel sizes
- 4_custom_sobel_kernels_unit_test: Final results according to requirements
- tensorflow_py: TensorFlow implementation for faster computation
- images: Contains the reference image used

The mathematical description of the presented concepts was taken from the text cited in the bibliography of this document Pedrini and Shwartz (2008).

2 Brief Introduction to Image Filtering

Linear filters compute the resulting pixel value $f'(x, y)$ as a linear combination of the gray levels in a local neighborhood of the pixel $f(x, y)$ in the original image.

In the spatial domain, the filtering process is typically performed using matrices called *masks* or *kernels*, which are applied to the image. Each position in the mask is associated with a numerical value, called weight or coefficient. Applying the mask centered at coordinate (x, y), where x is the column position and y is the row position in the image, consists of replacing the pixel value at position (x, y) with a new value, which depends on the neighboring pixel values and the mask weights. The filter coefficients are multiplied by the gray levels of the corresponding pixels and then summed, replacing the gray level of the central pixel. Figure 2.1 shows a generic 3×3 pixel mask.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figure 2.1: 3×3 pixel mask with arbitrary coefficients.

Denoting the gray levels of the image under the mask by $z_i = f(x, y)$, $1 \leq i \leq 9$, the mask response is

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \tag{1.1}$$

where w_i represents the mask coefficients.

If the mask center is at position (x, y) in the image, the gray level of the pixel at position (x, y) will be replaced by R. The mask is then moved to the next pixel position in the image, and the process is repeated until all pixel positions have been covered.

Two concepts are related to spatial filtering: *correlation* and *convolution*. For correlation, let w be the correlation filter. Generally, a filter with an odd number of elements is selected so that, during its traversal of the image, the filter center is located over the pixel under consideration in the image. Thus, the correlation of image f with a filter w can be expressed as:

$$w \cdot f(x) = \sum_{i=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} w(i) f(x + i) \tag{2.2}$$

The two-dimensional correlation is similar. Considering that both the image and the filter now have two dimensions, the correlation is defined as:

$$w \cdot f(x, y) = \sum_{i=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} \sum_{j=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} w(i, j) f(x + i, y + i) \tag{2.3}$$

Convolution is a process similar to correlation, except that the filter w must be reflected (or equivalently rotated by 180 degrees) before being applied to the image. The one-dimensional convolution of an image f by a filter w can be expressed as:

$$w * f(x) = \sum_{i=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} w(i) f(x - i) \tag{2.4}$$

In two-dimensional convolution, the filter weights must be reflected both horizontally and vertically, i.e.,

$$w(x, y) * f(x, y) = \sum_{i=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} \sum_{j=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} w(i, j) f(x - i, y - j) \tag{2.5}$$

It should be noted that correlation and convolution are identical when the filter is symmetric. Algorithm 2.1 presents the convolution process for an image.

```cpp
// Single convolution for filtering purposes
void convolve(const Mat& input, Mat& output, const float kernel[3][3]) {
    output = Mat::zeros(input.size(), input.type());

    int kernelSize = 3;
    int border = kernelSize / 2;

    for (int y = border; y < input.rows - border; ++y) {
        for (int x = border; x < input.cols - border; ++x) {
            float sum = 0.0;

            // Perform convolution
            for (int ky = -border; ky <= border; ++ky) {
                for (int kx = -border; kx <= border; ++kx) {
                    int pixelVal = input.at<uchar>(y + ky, x + kx);
                    sum += pixelVal * kernel[ky + border][kx + border];
                }
            }
            output.at<uchar>(y, x) = static_cast<uchar>(sum);
        }
    }
}
```

Algorithm 2.1: Convolution process

3 Edge Detection

Basically, the idea behind most edge detection techniques is to compute a local differential operator. The first derivative is positive during transitions from dark to light regions, negative during transitions from light to dark regions, and zero in areas of constant gray levels. On the other hand, the second derivative is positive in the transition associated with the dark side of the edge, negative in the transition associated with the light side, and zero in areas of constant gray levels. Therefore, the magnitude of the first derivative can be used to detect an edge in an image, while the sign of the second derivative crosses zero, indicating a change in gray level and allowing edge localization in an image.

3.1 Gradient Operators

Edge detection is essentially the operation of identifying significant local changes in gray levels of an image. These changes can be described using the concept of derivatives.

Since an image depends on two spatial coordinates, the edges in the image can be expressed by partial derivatives. A commonly used operator in image differentiation is the *gradient*, which is a vector whose direction indicates where the gray levels vary the most. The direction of the gradient is always perpendicular to the tangent direction of the edge.

Thus:

$$\theta = \phi \pm \frac{\pi}{2} \tag{3.1}$$

The gradient vector $\nabla f(x, y)$ of an image at position (x, y) can be calculated by the partial derivatives:

$$\nabla f(x, y) = \frac{\partial f(x, y)}{\partial x}\mathbf{i} + \frac{\partial f(x, y)}{\partial y}\mathbf{j} \tag{3.2}$$

where **i** and **j** are unit vectors in the x and y directions, respectively. Thus, a rapid variation of $f(x, y)$ along the x direction and a slow variation along the y direction indicates the presence of a nearly vertical edge.

The identification of edge points based on the gradient concept is provided by Algorithm 3.1.

In matrix form, the gradient of the image can be expressed as:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.3}$$

The algorithm for determining edge points uses an input image f with dimensions $M \times N$ pixels and a threshold T.

The implementation in C++ can be presented as:

```
// Calculate the gradient magnitude
gradientMagnitude = Mat::zeros(image.size(), CV_32F);

int m = 0, n = 0;
float gm = 0.0, gn = 0.0;

for (n = 0; n < image.rows; ++n)
  {
    for (m = 0; m < image.cols; ++m)
{
  gm = gradientX.at<float>(n, m);
  gn = gradientY.at<float>(n, m);
  gradientMagnitude.at<float>(n, m) = sqrt(gm * gm + gn * gn);
      }
  }
```

Algorithm 3.1: C++ implementation

In edge detection, the magnitude of the gradient vector is an important measure, often simply referred to as the *gradient* and denoted by ∇f:

$$\nabla f = \text{mag}(\nabla f) = \sqrt{G_x^2 + G_y^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{3.4}$$

The magnitude of the gradient represents the highest rate of change of $f(x, y)$ per unit of distance in the direction of ∇f. Due to the computational cost of equation (3.2), the gradient magnitude can be approximated using absolute values:

$$\nabla f \approx |G_x| + |G_y| \tag{3.5}$$

or the maximum value between the gradients in the x and y directions:

$$\nabla f \approx \max(|G_x|, |G_y|) \tag{3.6}$$

The direction of the gradient vector is also an important measure. Let $\theta(x, y)$ be the angle of the direction of the vector ∇f at position (x, y). Then:

$$\theta(x, y) = \arctan\left(\frac{G_y}{G_x}\right) \tag{3.7}$$

where the angle θ is measured relative to the x axis.

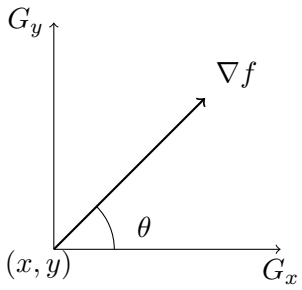

Figure 3.1: Gradient direction.

An intensity change can be detected by the difference between adjacent pixel values. *Vertical* edges can be detected by the *horizontal* difference between points, while *horizontal* edges can be detected by the *vertical* difference between adjacent points in the image.

References

Pedrini, H. and Shwartz, W. R. (2008). *Analise de imagens digitais, principios e aplicacoes*. Thomson.