IMAGE CONVOLUTION AND EDGE DETECTION

- Convolution
- Kernels for sharpening
- Kernels for embossing
- Kernels for edge detection
- Canny edge detector

Author: prof. Yevhenii Borodavka

Convolution

In image processing, a kernel, convolution matrix, or mask is a small matrix used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between the kernel and an image. Or more simply, when each pixel in the output image is a function of the nearby pixels (including itself) in the input image, the kernel is that function.

The general expression of a convolution is

$$g(x,y) = \omega \cdot f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega(i,j) \cdot f(x-i, y-j)\,,$$

where **g(x,y)** is the filtered image;

f(x,y) is the original image;

$\boldsymbol{\omega}$ is the filter kernel.

Every element of the filter kernel is considered by **–a<=i<=a** and **–b<=j<=b**.

Kernels for sharpening

The algorithm for sharpening an image is then:

1. Blur an image using whatever blur you wish (e.g., Box, Tent, Gaussian)
2. Subtract the blurred result from the original image to get high-frequency details.
3. Add the high-frequency details to the original image.

Algebraically, this can be expressed as

image + (image – blurred) or 2 * image – blurred.

Blurring is most commonly done by convolving an image with a low-frequency kernel that sums to 1. If we are assuming that path to blurring, we can actually build a sharpening kernel that encodes the equation we just derived. For “image”, we’ll just use the identity matrix for convolution which is all zeros except a 1 in the center. That gives us this:

2 * identity – blur

Kernels for sharpening

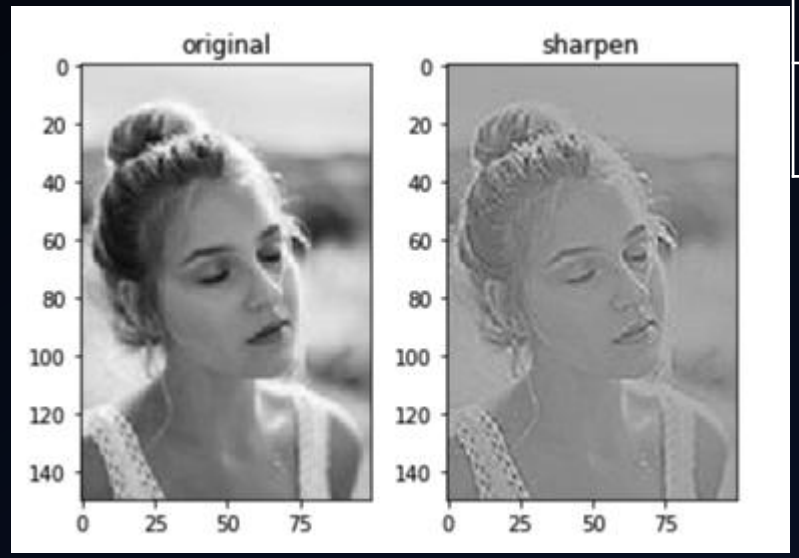
The sharpening kernels are

0	-1	0
-1	5	-1
0	-1	0

-1	-1	-1
-1	9	-1
-1	-1	-1

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	25	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	49	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1



Kernels for embossing

Image embossing is a computer graphics technique in which each pixel of an image is replaced either by a highlight or a shadow, depending on light/dark boundaries on the original image. Low-contrast areas are replaced by a gray background. The filtered image will represent the rate of color change at each location of the original image. Applying an embossing filter to an image often results in an image resembling a paper or metal embossing of the original image, hence the name.

The emboss filter, also called a directional difference filter will enhance edges in the direction of the selected convolution mask(s). When the emboss filter is applied, the filter matrix is in convolution calculation with the same square area on the original image. So it involves a large amount of calculation when either the image size or the emboss filter mask dimension is large. The emboss filter repeats the calculation as encoded in the filter matrix for every pixel in the image; the procedure itself compares the neighboring pixels on the image, leaving a mark where a sharp change in pixel value is detected. In this way, the marks form a line following an object's contour. The process yields an embossed image with edges highlighted.

KERNELS FOR EMBOSSING

Four primary and two rotated emboss filter masks are:

0	+1	0
0	0	0
0	-1	0

+1	0	0
0	0	0
0	0	-1

0	0	0
+1	0	-1
0	0	0

0	0	+1
0	0	0
-1	0	0

-1	0	0
0	0	0
0	0	+1

0	0	-1
0	0	0
+1	0	0

To control the depth of edges, we can enlarge the emboss filter masks, such as:

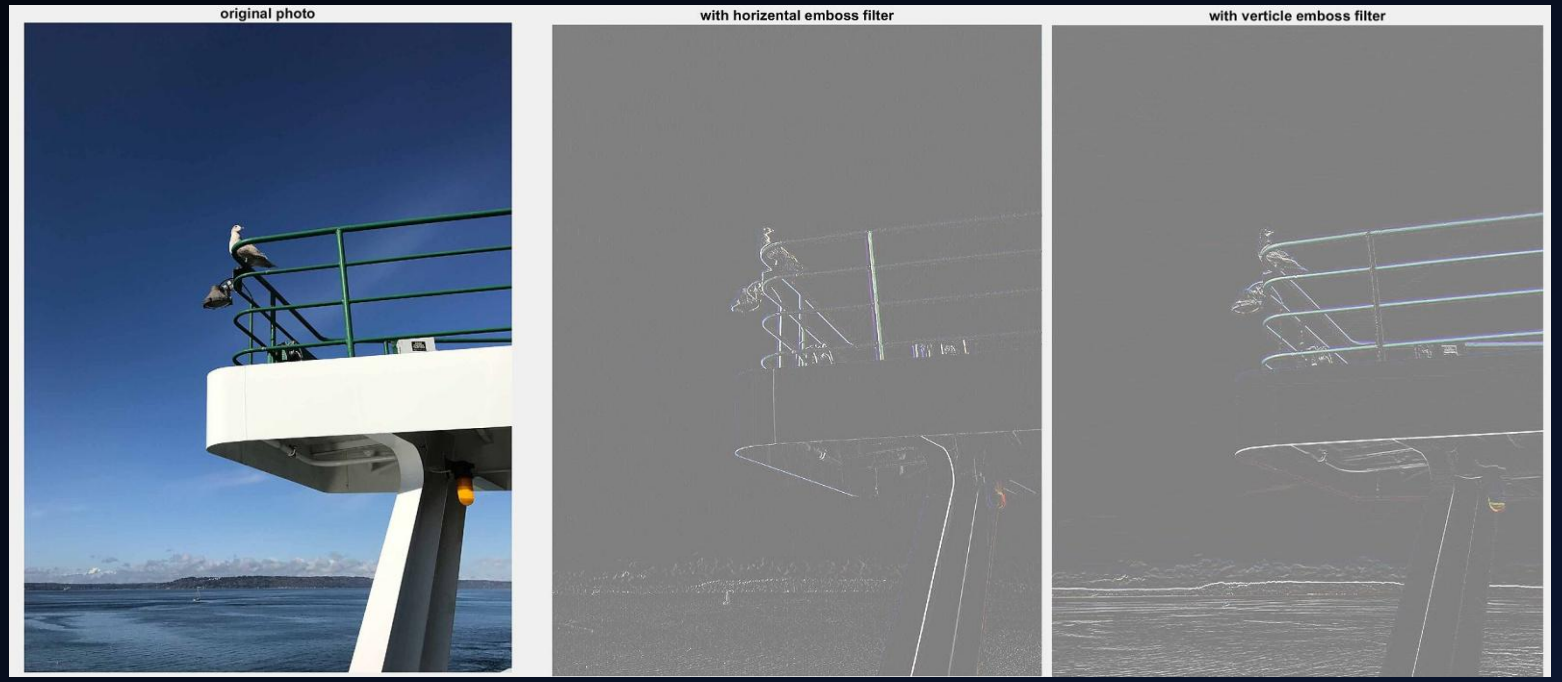
+1	0	0	0	0
0	+1	0	0	0
0	0	0	0	0
0	0	0	-1	0
0	0	0	0	-1

-2	-1	0
-1	1	1
0	1	2

The emboss kernel and its convolution result

Kernels for embossing

Two different emboss filters are applied to the original photo. Image (a) is the result of a 5×5 filter with the +1 and -1 in the horizontal direction, which emphasizes vertical lines. Image (b) is the result of a 5×5 filter with the +1 and -1 in the vertical direction; it emphasizes horizontal lines. Since the entries of a given emboss filter matrix sum to zero, the output image has an almost completely black background, with only the edges visible. Adding a 128 value of brightness (half the 0-255 range) to each pixel creates the final, displayed images with grey-toned backgrounds:



Kernels for edge detection

Edges represent the object boundaries. So edge detection is a very important preprocessing step for any object detection or recognition process. Simple edge detection kernels are based on the approximation of gradient images.

We will consider the Prewitt operator, the Sobel operator, and the Laplacian filter.

Technically, the Prewitt and the Sobel are discrete differentiation operators, computing an approximation of the gradient of the image intensity function.

An image is defined as a two-dimensional function f(x,y) the amplitude of **f** at any pair of coordinates (x,y) is called the intensity or gray level of the image at that point. Thus for performing edge detection, the image must be converted to gray levels considered as image intensity (**I**).

In simple terms, both operators calculate the **gradient** of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction.

Kernels for edge detection

The Prewitt and Sobel operators use two kernels — the first one for the horizontal direction and the second one for the vertical derivative approximations.

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

The Prewitt kernels

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

The Sobel kernels

The gradient magnitude of the image relates both directions G_x and G_y. It represents the strength of the edge. While the gradient angle represents the direction of the edge or direction of intensity variation. Gradient magnitude (**G**) and the gradient direction (Θ) are given by:

$$G = \sqrt{G_x^2 + G_y^2}; \quad \Theta = tan^{-1}(\frac{G_y}{G_x})$$

Kernels for edge detection

The gradient magnitude is computed as image intensity (**I**) convoluted with kernel.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * I$$

The Prewitt kernels

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I$$

The Sobel kernels

Kernels for edge detection

The grayscale image

Gradient with Prewitt

Gradient with Sobel

Kernels for edge detection

In digital image processing, we use a Laplacian filter to compute the second-order derivative of an image to detect edges. We need a Laplacian filter so that we can extract the features of the image in a better way. The better we can extract the features of the image, the better we will make the model to train.

The question arises of why we need to compute the second-order derivative when the first-order derivative is doing the work. We detect the horizontal and vertical edges in first-order derivatives and combine them. On the other hand, the second-order derivative allows us to detect all the edges of an image at once. Have a look at the Laplacian filters given below.

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Kernels for edge detection

As we have discussed earlier, in the Laplacian filter, we are interested in finding the second-order derivatives of the image vertically and horizontally. So the equation of this scenario is given below.

$$\nabla I(x,y) = \frac{\partial^2 I}{\partial^2 x^2} + \frac{\partial^2 I}{\partial^2 y^2}$$

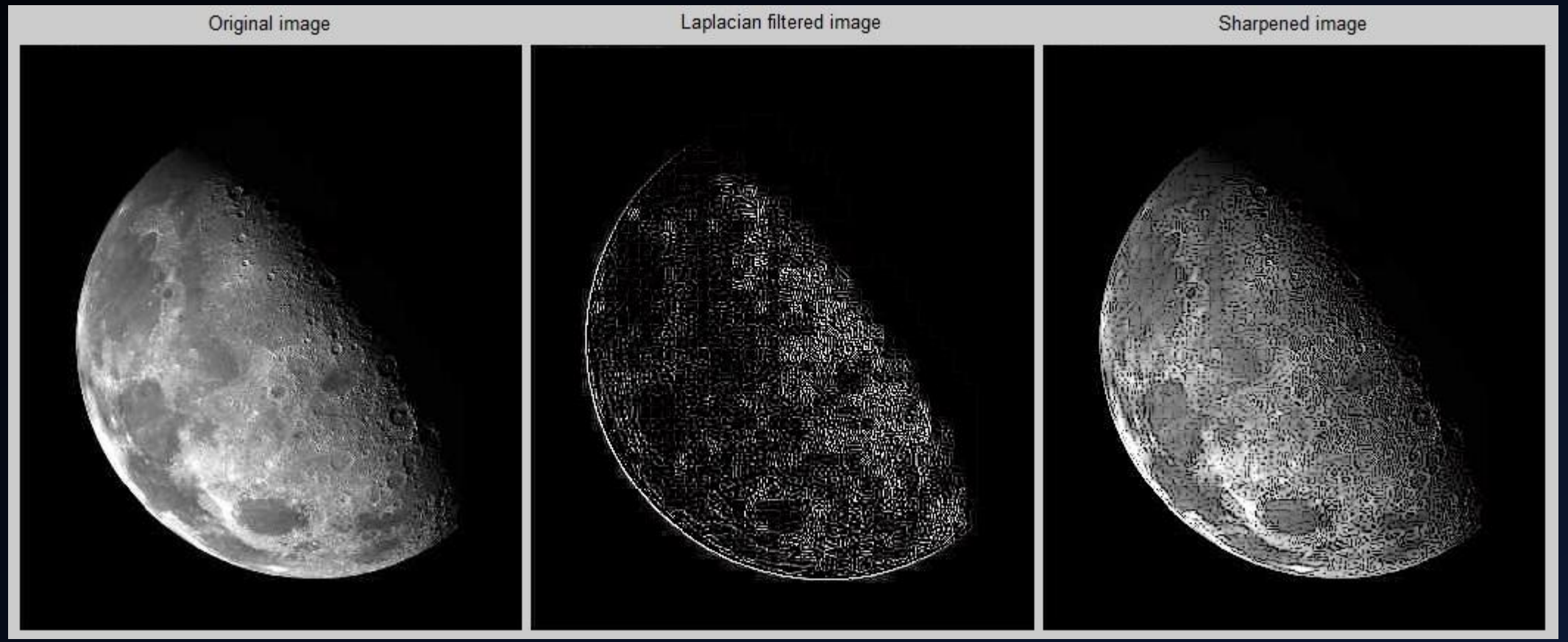
In this equation,

- *x* is the column index of the pixel.
- *y* is the row index of the pixel.
- *I*(x,y) represents the intensity of the image at a pixel (x,y).
- ∇ represents the gradient.

This equation represents how we take second-order derivation of an image.

Kernels for edge detection

The example of the Laplacian filter.



Canny edge detector

It was developed by John F. Canny in 1986. The Canny operator is a multi-stage algorithm that detects a wide range of edges.

The Canny edge detection algorithm is composed of 5 steps:

1. Smoothing for noise removal.
2. Finding Gradients.
3. None-maximum suppression.
4. Double Thresholding.
5. Edge Tracking by hysteresis.

The algorithm is based on grayscale pictures. Therefore, the pre-requisite is to convert the image to grayscale before following the above-mentioned steps.

Canny edge detector

Stage 1. Smoothing for noise removal.

The first stage in the Canny edge detection algorithm is smoothing to remove noise that may cause false edges.

The equation for a Gaussian filter kernel of size (2*k*+1)×(2*k*+1) is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right); 1 \le i,j \le (2k+1)$$

The kernel used in this step is a 5x5 Gaussian kernel with $\sigma = 1.4$ and that it:

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Canny edge detector

After applying the Gaussian blur, we get the following result:

Original image (left) — Blurred image with a Gaussian filter (sigma=1.4 and kernel size of 5x5)

Canny edge detector

Stage 2. Finding Gradients. We use the Sobel operator for finding gradients.

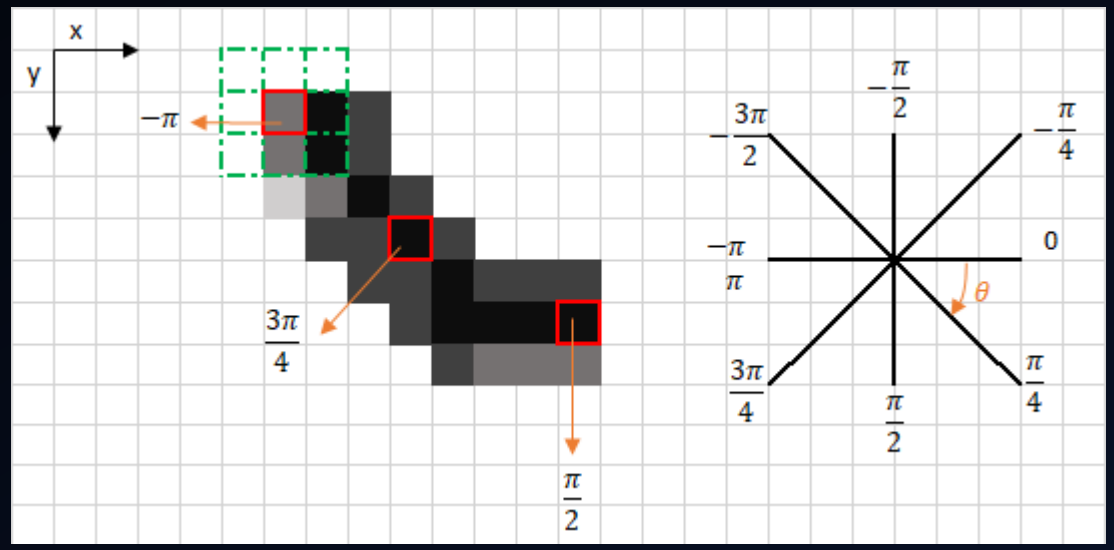
Blurred image (left) — Gradient intensity (right)

Canny edge detector

Stage 3. None-maximum suppression.

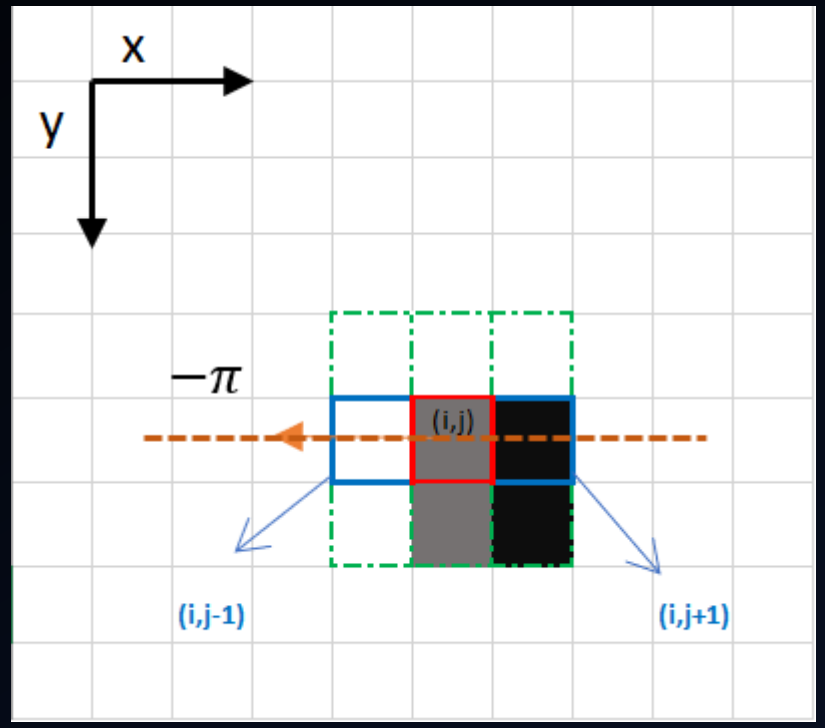
Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression to thin out the edges.

The principle is simple: the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions. Let's take an easy example:



The upper left corner red box present in the image represents an intensity pixel of the gradient Intensity matrix being processed. The corresponding edge direction is represented by the orange arrow with an angle of -pi radians (+/- 180 degrees).

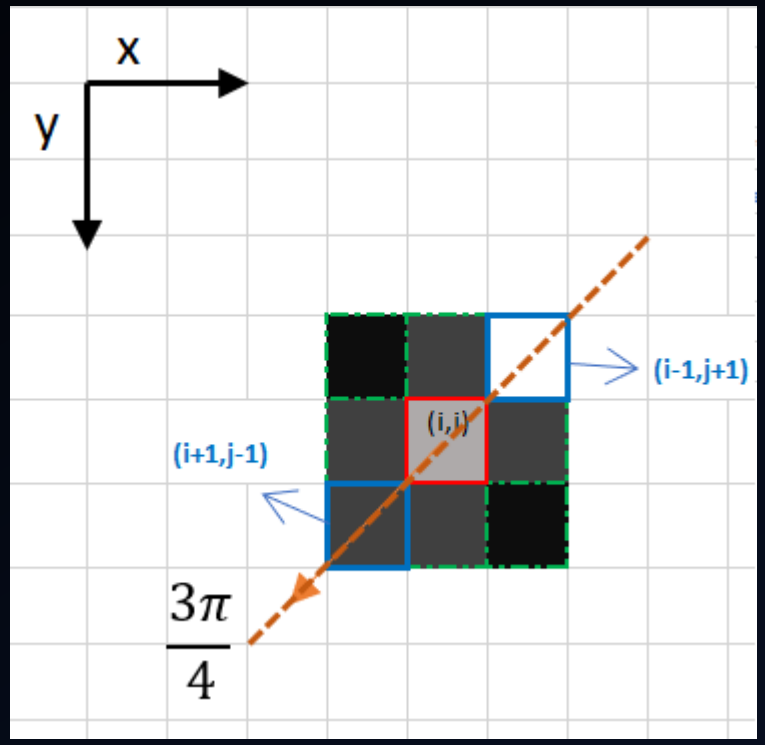
Canny edge detector



The edge direction is the orange dotted line (horizontal from left to right). The purpose of the algorithm is to check if the pixels in the same direction are more or less intense than the ones being processed. In the example, the pixel (i,j) is being processed, and the pixels in the same direction are highlighted in blue (i,j-1) and (i,j+1). If one of those two pixels is more intense than the one being processed, then only the more intense one is kept. Pixel (i,j-1) seems to be more intense because it is white (value of 255). Hence, the intensity value of the current pixel (i,j) is set to 0. If there are no pixels in the edge direction having more intense values, then the value of the current pixel is kept.

Canny edge detector

Let’s now focus on another example. In this case, the direction is the orange-dotted diagonal line. Therefore, the most intense pixel in this direction is the pixel (i-1, j+1).

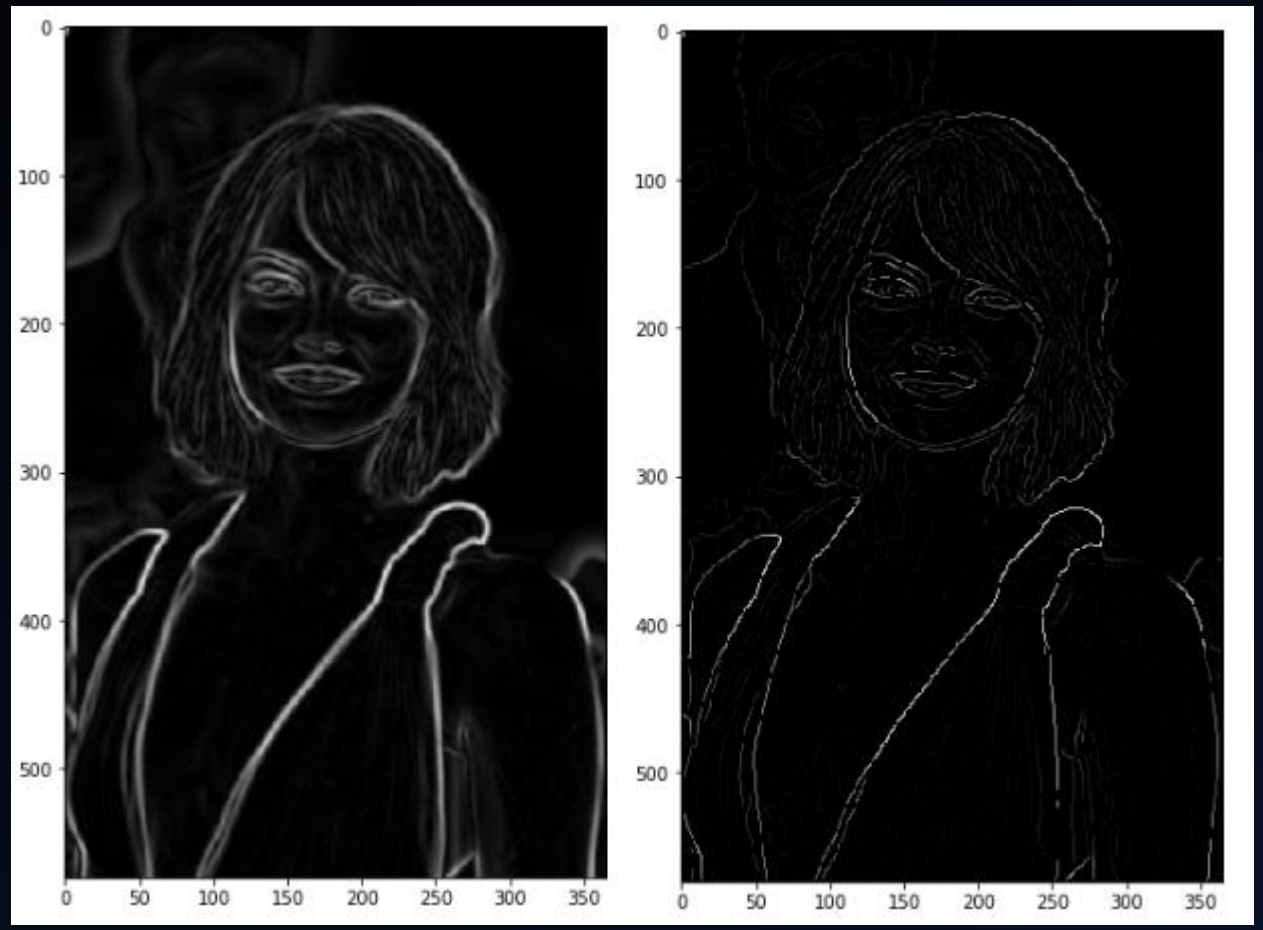


Let’s sum this up. Each pixel has 2 main criteria (edge direction in radians, and pixel intensity (between 0–255)). Based on these inputs the non-max-suppression steps are:

- Create a matrix initialized to 0 of the same size as the original gradient intensity matrix;
- Identify the edge direction based on the angle value from the angle matrix;
- Check if the pixel in the same direction has a higher intensity than the pixel that is currently processed;
- Return the image processed with the non-max suppression algorithm.

Canny edge detector

The result is the same image with thinner edges.



Result of the non-max suppression.

We can however still notice some variation regarding the edges' intensity: some pixels seem to be brighter than others, and we will try to cover this shortcoming with the two final steps.

Canny edge detector

Stage 4. Double threshold.

The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant:

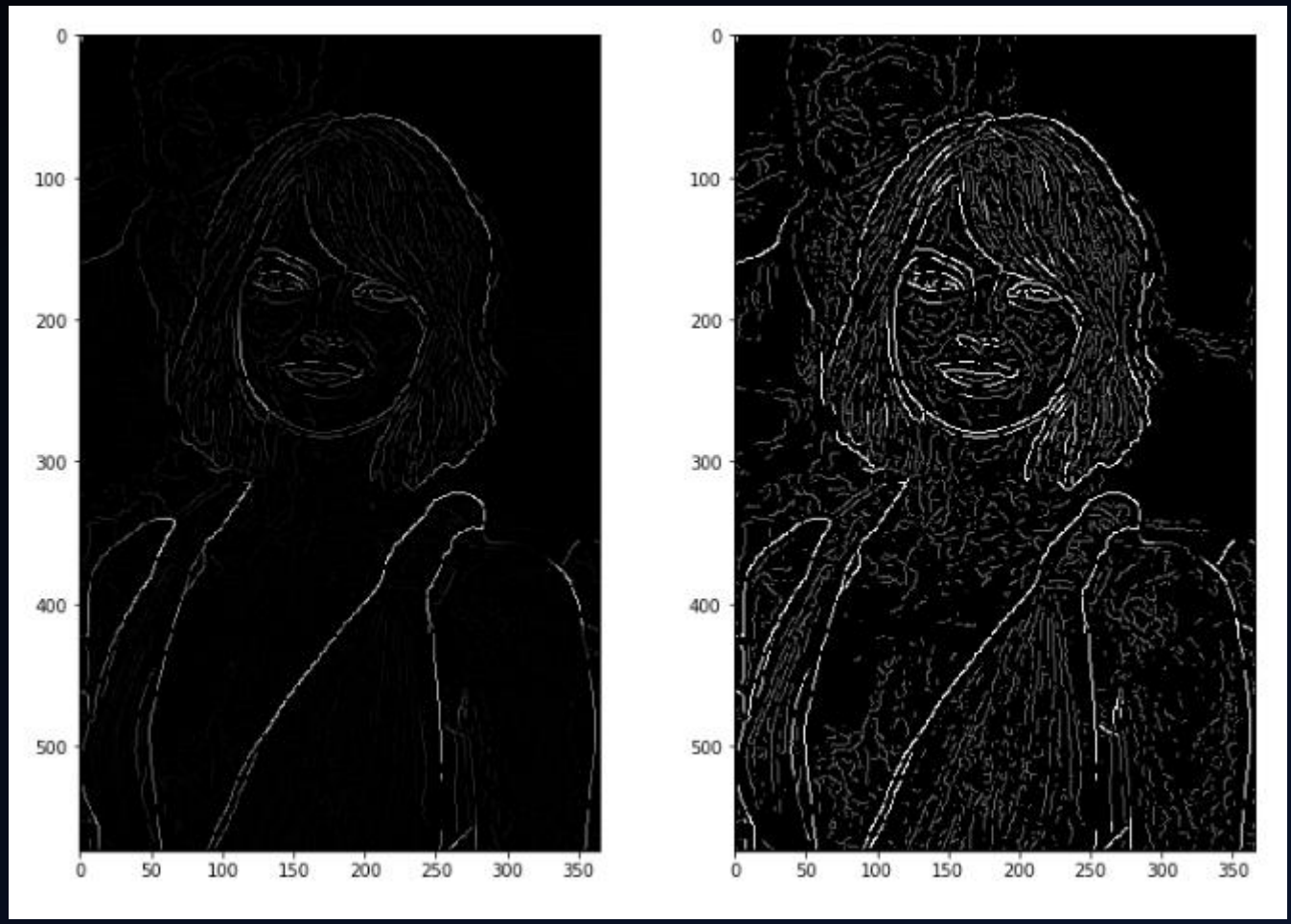
- Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.
- Weak pixels are pixels that have an intensity value that is not enough to be considered the strong ones, but yet not small enough to be considered as non-relevant for edge detection.
- Other pixels are considered non-relevant for the edge.

Now you can see what the double threshold holds for:

- The high threshold is used to identify the strong pixels (intensity higher than the high threshold)
- The low threshold is used to identify the non-relevant pixels (intensity lower than the low threshold)
- All pixels having intensity between both thresholds are flagged as weak and the Hysteresis mechanism (next step) will help us identify the ones that could be considered strong and the ones that are considered non-relevant.

Canny edge detector

The result of this step is an image with only 2 pixel intensity (strong and weak):

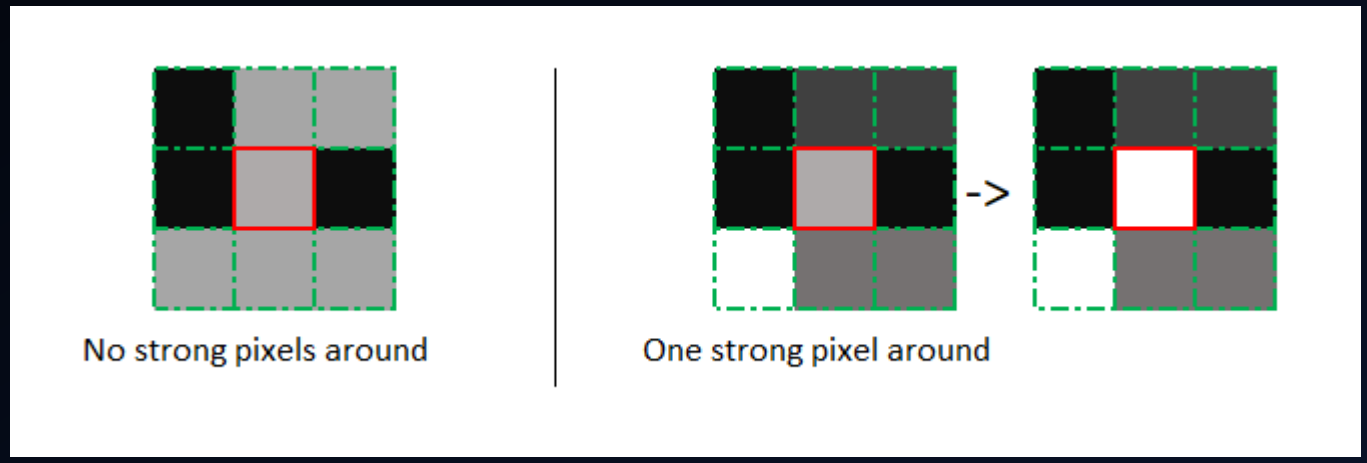


Non-Max Suppression image — Threshold result: weak pixels are gray and strong are white

Canny edge detector

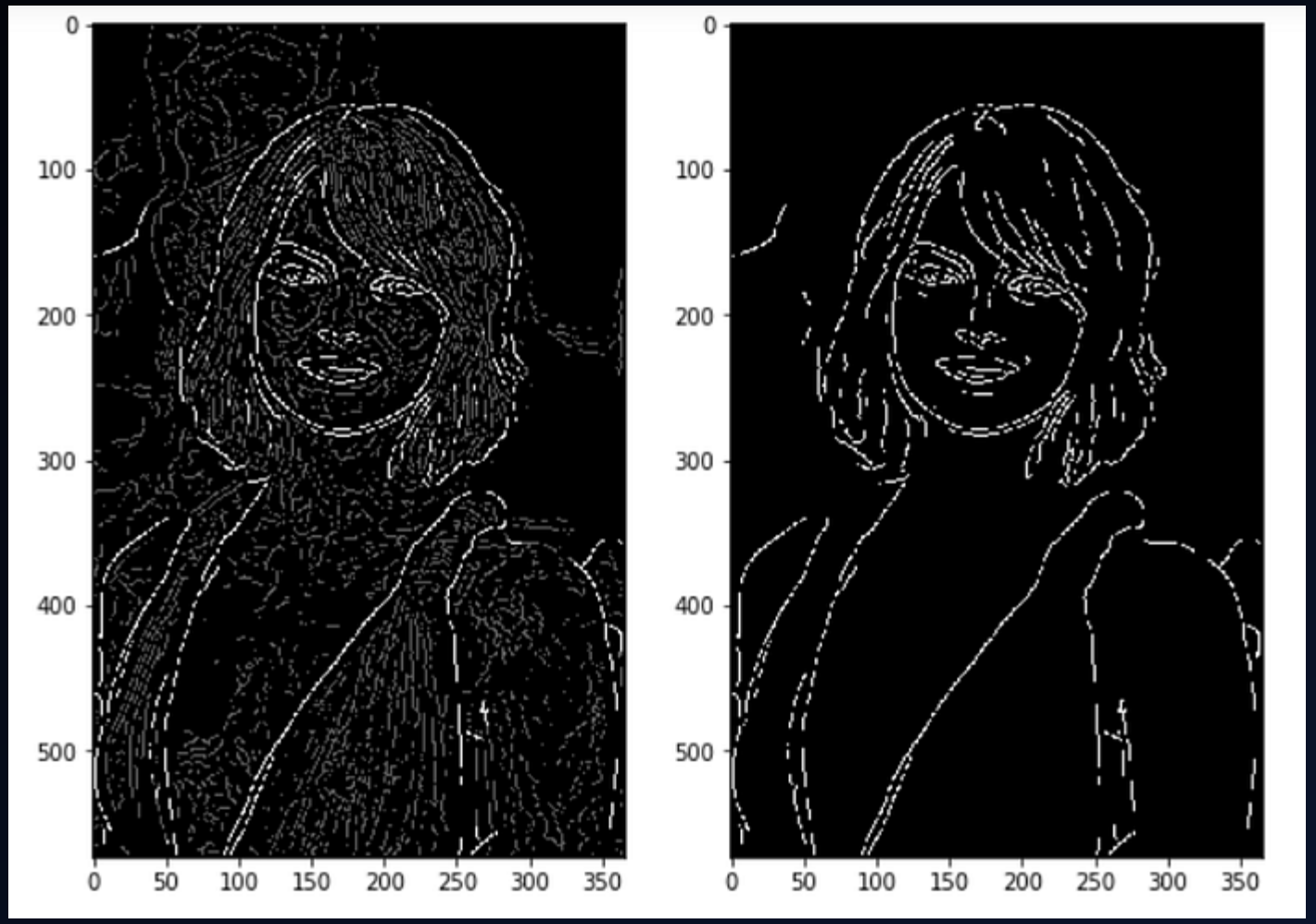
Stage 5. Edge Tracking by Hysteresis.

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one, as described below:



Canny edge detector

The result of the Hysteresis process:



Thank you!