REPORT

PRACTICAL PROJECT 1

Students.
Piera i Jiménez, David 1703730
Salguero Martinez, Héctor 1704247
Sanllehi Vico, David 1709952

Degree. Artificial Intelligence, 2n year

Professors. Jorge Bernal
Guillermo Torres

Class. Fundamentals of computer vision

INDEX:

CONVOLUTION

1. How would you describe the convolution process to another student? Use your own words.

The convolution process in image processing can be thought of as a way to apply a filter to an image to emphasize certain features, such as edges, textures, or other patterns.
The process applies a small matrix, called a kernel, over an image to highlight specific features. The kernel slides over the image, and at each position, it multiplies and sums the underlying pixels to create a new output pixel. This process transforms the image, emphasizing patterns based on the kernel's values.
We use convolution in image processing because it efficiently captures local patterns, which are essential for understanding image content. It's adaptable, allowing us to emphasize specific features by adjusting the kernel. Compared to other methods, convolution better preserves spatial relationships and is computationally efficient.

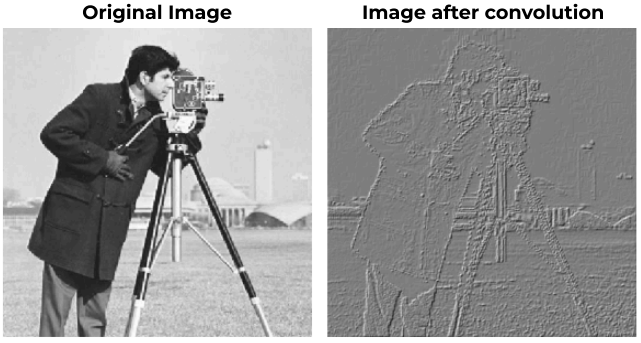

Figure 1

In Figure 1, we can see the Original Image shows a grayscale photograph of a cameraman. In the Image after Convolution, we see a processed version where specific features, such as edges and transitions in brightness, are highlighted. The details of the cameraman and the tripod are more pronounced, especially around the contours, creating a textured effect.
This change is due to the convolution operation, which uses a kernel to emphasize differences in pixel intensity across the image. As explained earlier, convolution highlights edges and patterns by capturing localized changes in brightness, which allows us to detect and analyze key features. Padding in this operation helps retain the image size, ensuring that even the edges of the original image are represented in the convolved output.

2. What is the role of padding in the convolution operation, and what are the different types of padding?

In convolution operations, padding involves adding extra pixels around the edges of an image before applying the kernel. Padding plays a crucial role by allowing the kernel to be centered over every pixel in the original image, even those at the borders, which would otherwise be left out. Without padding, the output image shrinks with each convolution, which can cause a loss of information or other more computationally costly methods. Some padding methods are the following:

figure 2

1	2	3	4	5	6
7	8	9	10	11	12
1	2	3	4	5	6
7	8	9	10	11	12
1	2	3	4	5	6
7	8	9	10	11	12

No padding

Valid Padding (No Padding): No extra pixels are added to the image borders. This causes the output to be smaller than the input image because the kernel doesn't fully cover the borders. (figure 2)

figure 3

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	1	2	3	4	5	6	0
0	7	8	9	10	11	12	0
0	0	0	0	0	0	0	0

Zero padding with a one-pixel thick boundary

Zero Padding (Same Padding): Extra pixels with a value of zero are added around the borders, ensuring the output image has the same dimensions as the input. This padding captures border features and preserves image size. (figure 3)

figure 4

11	10	9	10	11	12	11	10
7	6	5	6	7	8	7	6
3	2	1	2	3	4	3	2
7	6	5	6	7	8	7	6
11	10	9	10	11	12	11	10
15	14	13	14	15	16	15	14
11	10	9	10	11	12	11	10
7	6	5	6	7	8	7	6

Mirror padding with a two-pixel thick boundary

Reflect Padding: The edge pixels are mirrored across the padding, creating a reflection effect around the image. This helps maintain continuity at the borders, which can improve feature detection near edges. (figure 4)

Additional padding methods include:

Replicate Padding: Extends the edge pixels outward by replicating the values.
Constant Padding: Adds padding with a specified constant value (other than zero).

3. How does the stride parameter influence the convolution operation?

The stride parameter affects directly the size of the output matrix, result of the convolution, as it follows this formula:

i = size of the image
k = kernel size
p = padding
s - stride

output size = (i - k + 2*p) / s

In the case that the value of stride is equal to 0 the formula changes as if not the value would be infinite.

output_size = i - k + 2*p

The stride also skips intermediations in the convolution and that leads to a reduction of the computation time. For example, with stride 1 the computation time is 2.7 seconds and for stride 5 is 0.5 seconds.

In the figure 5, we can appreciate the changes on the images depending on the stride. The first one is the original image which we will take as a reference for the upcoming images in order to compare them.

As a consequence of the convolution and the value of the stride, we can appreciate that it only remains the main figure of the picture (the cameraman and the camera) and the background is undifferentiable. But then when the value of the stride is 5 the convolution filter skips over more pixels and fewer details of the image are retained. As a consequence we end up with the finer details of the cameraman disappearing and only some geometric shapes remain as we can see in Figure 5

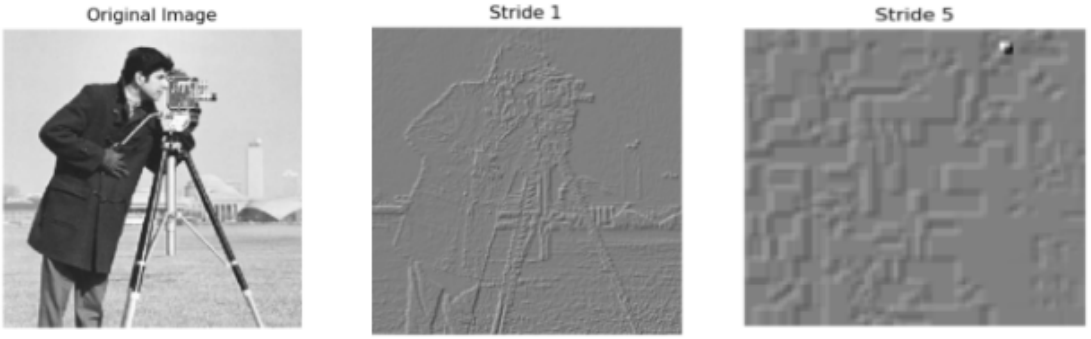

Figure 5

4. How do small kernels (e.g., 3x3) and large kernels (e.g., 5x5 or 7x7) affect the outcome of the convolution operation?

The size of a kernel in a convolution operation significantly affects the operation's outcome in terms of preserving details and in the amount of blurring or edge detection applied.

On one hand, small kernel sizes, such as a 3x3 kernel, imply less blurring in pixels, but they are more precise in detecting edges, as the smaller the kernel, the more sensitive it is to small changes in intensity. This approach makes small sizes more effective in highlighting fined details in the image. Furthermore, smaller sizes are more efficient in computation, as they are faster to apply and therefore more useful for real-time applications.

On the other hand, large kernels, for instance, 5x5 or 7x7, imply covering more surrounding pixels in computation, producing a stronger blurring or smoothing effect. However, larger kernels tend to obscure finer details, so they might lose small edges.

Finally, larger edges are computationally harder to manage, so they may slow down processing.

Let’s have a view on the result of a convoluted image with a 3x3 kernel vs a convoluted one having applied a 5x5 kernel:

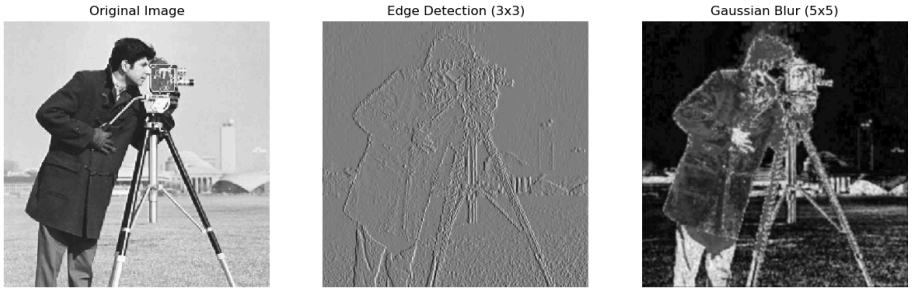

Figure 6

In the image that we can observe above (Figure 6), there is presented the original image, next to the output of applying a 3x3 kernel to it whereas the output of applying it a 5x5 kernel. We can appreciate that the convoluted image with a 3x3 kernel distinguishes in a better way the edges and contours, whereas the one with a 5x5 kernel gets blurred, although it quite preserves some original colors.

5. Why is OpenCV's convolution result so different from your implementation and SciPy's convolution?

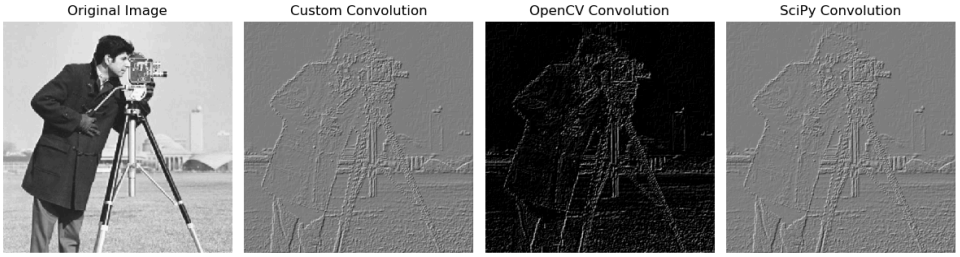

Figure 7

In Figure 7, and as the statement also says, we can easily observe that our customized convolution and the SciPy convoluted image are really similar, almost identical, while the OpenCV convoluted image differs a lot from them.

This distinction may be due to the fact that OpenCV convolution constraints the output values of each pixel to the range [0, 255], so if a value is outside this range, it approximates it to the nearest bound. On the other hand, our customized and SciPy convolution allows float type, as well as negative, output results.

But the main reason is that OpenCV convolution uses a different type of padding by default, in this case, Border replication, strategy that we have already explained in question 2. In contrast, our customized convolution and the SciPy one both use constant padding.

Therefore, the product of the image pixels plus the padding with the kernel gives really different values for the OpenCV compared with the other two approaches, which leads to a different image result.

LINEAR FILTERING

6. Briefly describe the goal of each linear filter. Use own words.

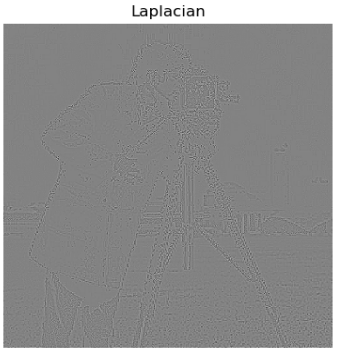

Figure 8

1. Laplacian Filter: The Laplacian filter is used for detection of edges in an image. It highlights areas in which intensity changes rapidly producing a picture of all the edges in an image. It is a second derivative function designed to measure changes in intensity without being overly sensitive to noise. The function produces a peak at the start of the change in intensity and then at the end of the change.

2. Sobel Filter: The Sobel filter is used to find edges in a specific direction, typically horizontal or vertical. It combines gradient calculations (first-order derivatives) in the x and y directions. The result is an image that highlights areas where there are significant changes in brightness.

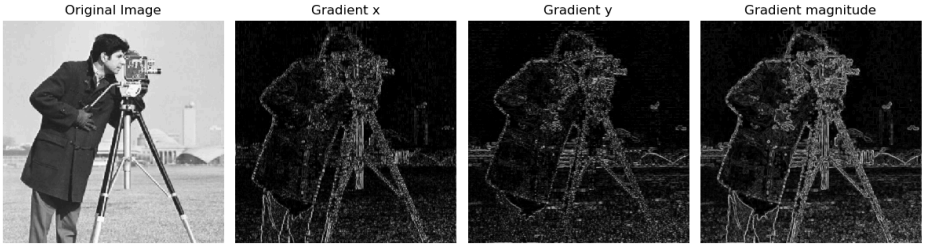

Figure 9

Figure 10

3. Average Box Filter: The average box filter smooths an image by averaging the pixel values in a defined neighborhood. However, this filter can also blur edges, making fine details less sharp. The goal is to create a softer look by averaging variations. This filtering approach is useful for noise reduction.

Figure 11

4. Gaussian Filter: The Gaussian filter is also used for blurring, but by applying weights based on a Gaussian distribution. In other words, pixels closer to the center of the filter receive more weight, making the effect smoother and less harsh on edges. This filter is commonly used to reduce noise while preserving more of the image structure.

7. The Sobel filter detects edges in specific directions (horizontal or vertical). What are the potential advantages and disadvantages of using a directional filter like Sobel versus a non-directional filter like Laplacian in edge detection tasks?

Using a directional filter like the Sobel filter versus a non-directional filter like the Laplacian for edge detection has specific advantages and disadvantages based on the task's requirements and the properties of each filter.

The Sobel filter, (a directional edge detector), offers specific advantages in applications where understanding edge orientation is important, as it can highlight horizontal or vertical edges separately. This directional sensitivity allows for selective edge detection, which can reduce noise from edges in non-target directions, making Sobel particularly effective in structured scenes where edges align predominantly along certain axes. However, its directional nature is also a limitation: to capture edges comprehensively, multiple Sobel filters (e.g., both horizontal and vertical) must be applied, which can increase computational requirements. In contrast, the Laplacian filter detects edges in all directions simultaneously, providing a better edge map with only one pass. This makes it suitable when omnidirectional edge detection is needed or when edge orientation is unknown. However, as a second-order derivative, the Laplacian is more sensitive to noise, which can result in unwanted artifacts and spurious edges in noisy images. Additionally, the Laplacian does not provide directional information, which may be a drawback in tasks where edge orientation is critical. Consequently, the choice between Sobel and Laplacian depends on the need for directional specificity versus comprehensive edge detection, as well as the level of noise resilience required in the application.

In figure 12 we can see an example of a solver filter vs a Laplacian filter, here we can observe that in that case, a directional filter like the Sobel filter (horizontal) seems to do a better job. This is because the Laplacian filter tends to make the image noisy.

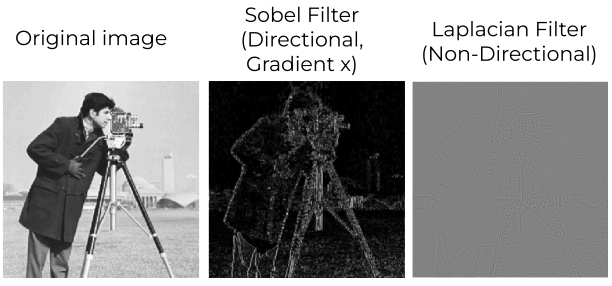

Figure 12

8. After adding salt-and-pepper noise to the image, which filter (Laplacian, Sobel, Mean or Gaussian) is the most effective at reducing noise? Justify your answer with observations from the filtered outputs.

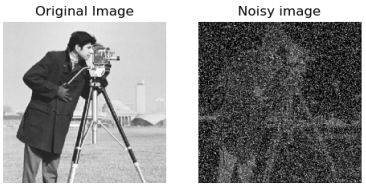

Figure 13

After adding salt-and-pepper noise to an image(figure 13) , the mean filter (image 14) is typically the most effective among the Laplacian, Sobel, Mean, and Gaussian filters for reducing this type of noise. Salt-and-pepper noise consists of random bright and dark pixels scattered throughout the image, and the mean filter, which replaces each pixel value with the average of its neighborhood, helps smooth these outliers. This averaging effect reduces the prominence of isolated black-and-white pixels without severely blurring edges.

Figure 14

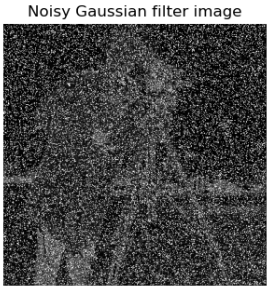

Figure 15

In comparison, the Gaussian filter (figure 15) is less effective for salt-and-pepper noise because it is designed for general smoothing and may not specifically address extreme outliers. The Sobel and Laplacian filters, focused on edge detection, often exacerbate salt-and-pepper noise by highlighting the sharp intensity contrasts it introduces, making them unsuitable for noise reduction. Consequently, the mean filter stands out for its balance of noise reduction and edge preservation in cases of salt-and-pepper noise.

Finally, in image 16, we can see how the Sobel filter and Laplacian filter do not do a good job of reducing the noise of the image because both filters are designed for edge detection rather than noise reduction.

Figure 16

TEMPLATE MATCHING (OPTIONAL)

9. What is the goal of template matching in computer vision?
Briefly describe the purpose and typical applications of template matching.

The main goal of the template matching technique in computer vision is to identify patterns in larger images. It looks like a simple task but it is a really great technique when it refers to image recognition and object detection. It is useful to define a similarity between two or more different images and it involves many applications that go from security-based to biomedical-based projects.

Some of the most famous examples are face recognition, license plate recognition, eye detection, and automatic PCB boarding test.

10. Explain the role of Normalized Cross-Correlation (NCC) in template matching. Why is normalization important in comparing image regions and templates?

Normalized Cross-Correlation (NCC) plays a key role in template matching by measuring the similarity between a template and different regions of a larger image. NCC provides a value between -1 and 1, where 1 indicates a perfect match and -1 indicates an inverse correlation. By sliding the template across the image and calculating NCC values, we can identify areas that closely match the template.

Normalization is crucial in NCC because it compensates for variations in brightness and contrast between the template and the image. Without normalization, differences in lighting or contrast could lead to unreliable results, as these variations could affect the matching accuracy.

11. In your own words, explain the meaning of an NCC value of 1, 0, and -1.
What does each of these values indicate about the similarity between the image region and the template? Relate your answers to your output results.

As we have commented before NCC gave us some values between 1, 0, and -1 and they indicate properties referred to the similarity.

An NCC value of 1 indicates a perfect correlation between the template and the image region, meaning they are identical in both content and alignment. This is the highest possible similarity, showing that the region matches the template exactly. Related to sour output there are values above the 0.8 threshold, they would signify regions of high similarity between the template and those parts of the main image. These areas would likely have bounding boxes drawn around them in the final image to highlight the match. (see on figure 17)

figure 17

An NCC value of 0 means there is no correlation between the template and the image region. This implies that there is no meaningful similarity between them, so they likely have no features in common. In our output, regions with NCC values close to 0 would be disregarded during filtering since they do not represent matches. These values would not pass the threshold, so they wouldn't appear with bounding boxes in the result image.

An NCC value of -1 indicates a perfect negative correlation, meaning the image region is essentially the inverse of the template. This would be the case if the region's features are completely opposite to those in the template. In our output, if any values were close to -1, they would suggest that those image regions are the inverse of the template (e.g., a dark area in the template aligning with a light area in the image). Like values near 0, these would also not pass the matching threshold and wouldn't have bounding boxes.

These values help interpret the output results by showing how closely each region in the larger image resembles the template, allowing us to locate similar areas effectively.

BIBLIOGRAPHY

Naukri. (n.d.). *Padding in convolutional neural network*. Naukri. Retrieved October 29, 2024, https://www.naukri.com/code360/library/padding-in-convolutional-neural-network

GeeksforGeeks. (n.d.). *CNN: Introduction to padding*. GeeksforGeeks. Retrieved October 29, 2024, https://www.geeksforgeeks.org/cnn-introduction-to-padding/

Educative. (n.d.). *Types of padding in images*. Educative. Retrieved October 29, 2024, https://www.educative.io/answers/types-of-padding-in-images

LearnOpenCV. (n.d.). *Image filtering using convolution in OpenCV*. LearnOpenCV. Retrieved October 29, 2024, from https://learnopencv.com/image-filtering-using-convolution-in-opencv/

WebSupergoo. (n.d.). *Laplacian effect*. WebSupergoo. Retrieved October 29, 2024, from https://www.websupergoo.com/helpie/default.htm?page=source%2F2-effects%2Flaplacian.htm

Author(s). (2016). *Template matching advances and applications in image analysis*. arXiv. Retrieved October 29, 2024, from https://arxiv.org/pdf/1610.07231