Int. j. adv. multidisc. res. stud. 2024; 4(3):1039-1044

Received: 18-04-2024
Accepted: 28-05-2024

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

The Comparison of 2D Convolution and Max Pooling Process in Real Time

Panca Mudjirahardjo
Department of Electrical Engineering, Faculty of Engineering, Universitas Brawijaya, Malang, Indonesia

Corresponding Author: **Panca Mudjirahardjo**

Abstract
Convolution and max pooling process are the necessary processes to get an object's feature in convolutional neural network (CNN). There are many filters or kernels to be convolved with the input image, to get another form of the object's edge. The max pooling is one way to reduce the spatial dimension. In this paper, we study a comparison of 2D convolution and max pooling process in real time. The comparison are the process to get the output image. The first one, we perform convolution first, then the max pooling process. The second one is performing the max pooling first, then convolution process. Both process is required to get the object's feature to be fed to classifier. The experiment is performed using programming language C++ and openCV library.

Keywords: Real Time, 2D Convolution, Max Pooling, Feature Extraction, CNN

1. Introduction

Nowadays CNN is a powerful system in object classification. Convolutional neural network (or CNN) is a special type of multilayer neural network or deep learning architecture inspired by the visual system of living beings. The CNN is very much suitable for different fields of computer vision and natural language processing [1]. Fig 1 is shown the conceptual model of CNN. Inspired by the work of Hubel and Wiesel, in 1980, Kunihiko Fukushima proposed Neocognitron [1, 2], which is a self-organizing Neural Network, containing multiple layers, capable of recognizing visual patterns hierarchically through learning and this architecture became the first theoretical model of CNN as in the Fig 2.

A CNN consists of pre-processing, feature extraction, and classification stages. Pre-processing stages are grayscale conversion and contrast improvement. Contrast improvement is performed usually by contrast limited adaptive histogram equalization (CLAHE) process. The aim of CLAHE process is to improve image's contrast, especially in dark condition.

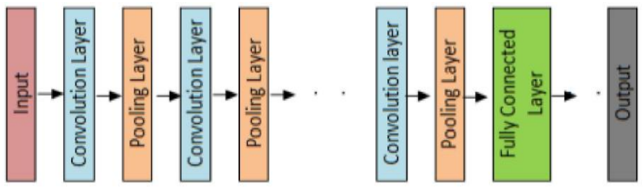

Fig 1: Conceptual model of CNN [1]

The next stage is feature extraction. The first step in feature extraction is convolution process. Convolutional layer is the most important component of any CNN architecture. It contains a set of convolutional kernels (also called filters), which gets convolved with the input image (N-dimensional metrics) to generate an output feature map.

The second step of feature extraction if pooling process. The pooling layers are used to sub-sample the feature maps (produced after convolution operations), i.e. it takes the larger size feature maps and shrinks them to lower sized feature maps. While shrinking the feature maps it always preserve the most dominant features (or information) in each pool steps. The pooling operation is performed by specifying the pooled region size and the stride of the operation, similar to convolution operation [1].

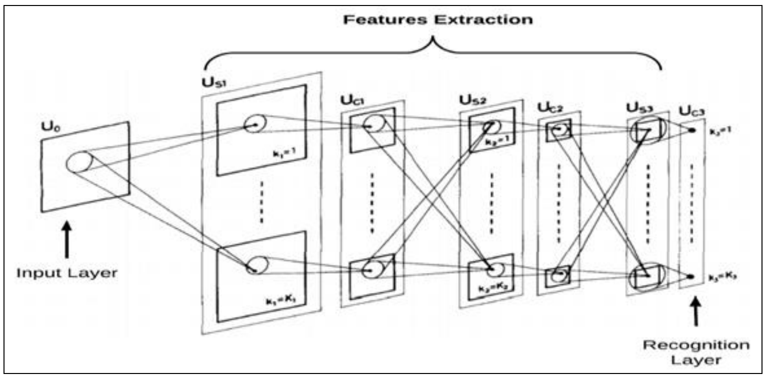

Fig 2: Schematic diagram illustrating the interconnections between layers in the neocognitron [2]

2. The Proposed Method

The method of this study is shown in Fig 3. We have two study scenarios. The first one, the input image will be converted into CLAHE image first then be filtered with a convolution process. The filtered image, then be processed max pooling to reduce the spatial dimension. The second scenario is the input image will be processed max pooling first, then be filtered by a convolution process.

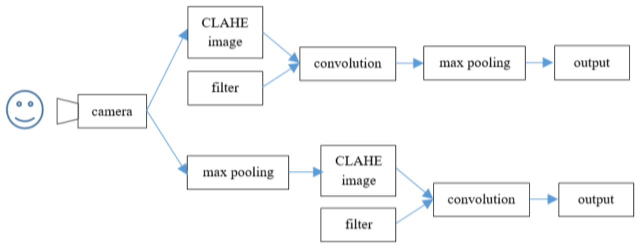

Fig 3: The method of this study

2.1 Contrast Limited Adaptive Histogram Equalization (CLAHE)

Adaptive histogram equalization (AHE) is a computer image processing technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image [1, 3, 4].

However, AHE has a tendency to over amplify noise in relatively homogeneous regions of an image. A variant of adaptive histogram equalization called contrast limited adaptive histogram equalization (CLAHE) prevents this by limiting the amplification. The one implementation of CLAHE is used for improve the visibility level of foggy image or video [4].

In CLAHE, the contrast amplification in the vicinity of a given pixel value is given by the slope of the transformation function. This is proportional to the slope of the neighborhood cumulative distribution function (CDF) and therefore to the value of the histogram at that pixel value. CLAHE limits the amplification by clipping the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and therefore of the transformation function. The value at which the histogram is clipped, the so-called clip limit, depends on the normalization of the histogram and thereby on the size of the neighborhood region. Common values limit the resulting amplification to between 3 and 4.

It is advantageous not to discard the part of the histogram that exceeds the clip limit but to redistribute it equally among all histogram bins [5-8].

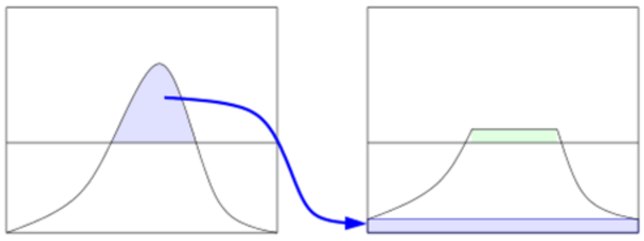

Fig 4: The histogram distribution in CLAHE [5]

The redistribution will push some bins over the clip limit again (region shaded green in the Fig 4), resulting in an effective clip limit that is larger than the prescribed limit and the exact value of which depends on the image. If this is undesirable, the redistribution procedure can be repeated recursively until the excess is negligible.

The CLAHE algorithm has three major parts: Tile generation, histogram equalization, and bilinear interpolation. The input image is first divided into sections. Each section is called a tile. Histogram equalization is then performed on each tile using a pre-defined clip limit. Histogram equalization consists of five steps: Histogram computation, excess calculation, excess distribution, excess redistribution, and scaling and mapping using a cumulative distribution function (CDF). The histogram is computed as a set of bins for each tile. Histogram bin values higher than the clip limit are accumulated and distributed into other bins. CDF is then calculated for the histogram values. CDF values of each tile are scaled and mapped using the input image pixel values. The resulting tiles are stitched together using bilinear interpolation, to generate an output image with improved contrast.

To increase image contrast, use the CLAHE algorithm as below. Grayscale and color photos can both be processed using this approach.

CLAHE algorithm steps are as follows [9]:

Step 1: Input image
Step 2: Segment input images into tiles
Step 3: Compute histogram for each tiles
Step 4: Apply TFM to compute clip limit
Step 5: Limit the contrast based on computed clip limit
Step 6: Check for enhanced image
Step 7: Enhanced image

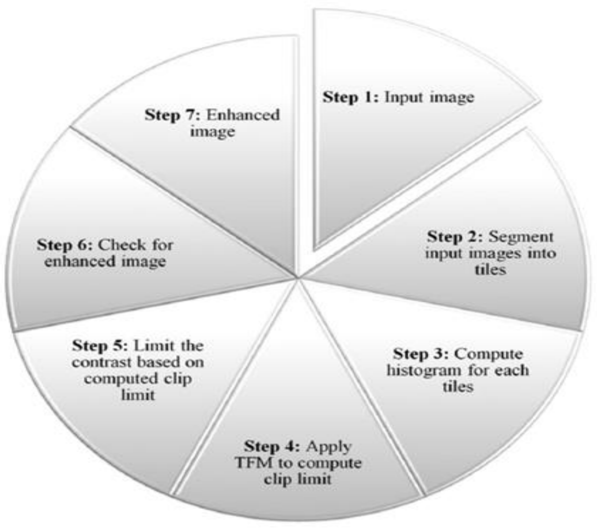
Step 1: Input image
Step 2: Segment input images into tiles
Step 3: Compute histogram for each tiles
Step 4: Apply TFM to compute clip limit
Step 5: Limit the contrast based on computed clip limit
Step 6: Check for enhanced image
Step 7: Enhanced image

Fig 5: Steps followed in CLAHE algorithm [9]

Fig 5 illustrates the steps to be followed in the CLAHE algorithm. Prior to creating a histogram for each context region, a given input image is first separated into context regions. So that various portions of the image may be easily linked, a mapping function is used to produce an image mapping. The image noise is subsequently reduced using an interpolation approach. This enables us to lessen the noise in particular regions of the image. Although the method denoises the image, it does not do so fully.

2.2 Filter Kernel

A filter, or kernel, in a CNN is a small matrix of weights that slides over the input data (such as an image), performs element-wise multiplication with the part of the input it is currently on, and then sums up all the results into a single output pixel. This process is known as convolution.

Filters are at the heart of what makes CNNs work. They are the primary component that helps the model extract useful features from the input data. There are many filters, such as Prewitt filters, Sobel filters, Laplacian filter, Robinson compass filters, Krisch compass filters, etc. Some of filters are shown in Fig 6 [10].

Prewitt Filters

$$\underset{\text{horizontal}}{\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{\text{vertical}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}}$$

Sobel Filters

$$\underset{\text{horizontal}}{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{\text{vertical}}{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}$$

Laplacian Filters

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Fig 6: Some of filter kernel [10]

2.3 2D Convolution

2D convolutions, a convolution generalized to matrices, are useful in computer vision for a variety of reasons, including edge detection and convolutional neural networks. Their exact usage will not be discussed here, and instead we will discuss an efficient way to calculate a 2D convolution with the FFT we have already developed. We have a "data" matrix, representing an image, and we have a kernel matrix, which is the matrix we imagine sliding over the image. This is also known as a filter [11, 12, 13].

For 2D convolutions, the result is slightly ambiguous depending on how one defines it. We will use scipy's definition, where to calculate the value of the convolution at a particular point, we imagine the bottom right corner of the kernel placed over that point.

We define the 2D convolution between an image x of size M×N and a kernel h of size H×W as follows (similar to the 1D case, we assume both matrices are padded with 0's):

$$(x * h)[i,j] = \sum_{k=0}^{i} \sum_{l=0}^{j} x[k][l]h[i-k][j-l] \tag{1}$$

This operation is also symmetric, so what we call the image and the kernel is essentially arbitrary (by convention, the kernel is the smaller matrix). The resulting matrix is going to be of size $(M + H - 1)\times(N + W - 1)$ from the same logic as the 1D case. Thus, the time it takes to compute the convolution is O($MNHW$). We can, however, take advantage of a trick if the kernel has a certain property.

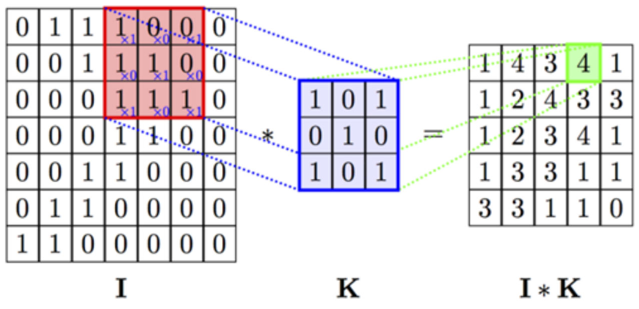

Fig 7: A convolution process [11]

2.4 Pooling Layer

The pooling layers are used to sub-sample the feature maps (produced after convolution operations), i.e. it takes the larger size feature maps and shrinks them to lower sized feature maps. While shrinking the feature maps it always preserve the most dominant features (or information) in each pool steps. The pooling operation is performed by specifying the pooled region size and the stride of the operation, similar to convolution operation [1].

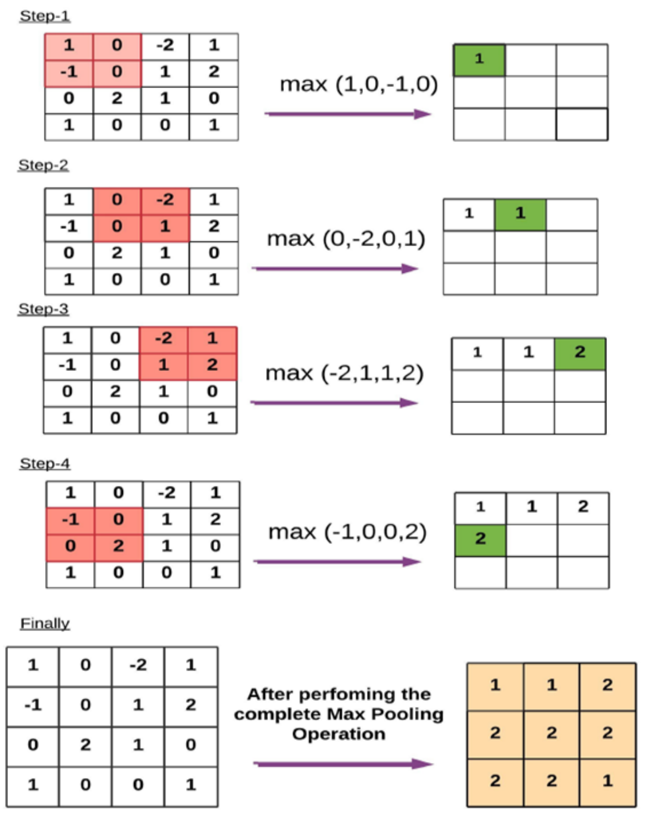

Fig 8: Illustrating the max pooling process [1]

There are different types of pooling techniques are used in different pooling layers such as max pooling, min pooling, average pooling, gated pooling, tree pooling, etc. Max Pooling is the most popular and mostly used pooling technique.

The main drawback of pooling layer is that it sometimes decreases the overall performance of CNN. The reason behind this is that pooling layer helps CNN to find whether a specific feature is present in the given input image or not without caring about the correct position of that feature [1].

Typically, the size of the pooling window is 3×3, and the stride with which the window is moved is also 2 pixels, as shown in Fig 7. This setup reduces the size of the input by half, both in height and width, effectively reducing the total number of pixels by 75%.

Max pooling offers several benefits in the context of CNNs [8]:

Feature Invariance: Max pooling helps the model to become invariant to the location and orientation of features. This means that the network can recognize an object in an image no matter where it is located.

Dimensionality Reduction: By down sampling the input, max pooling significantly reduces the number of parameters and computations in the network, thus speeding up the learning process and reducing the risk of overfitting.

Noise Suppression: Max pooling helps to suppress noise in the input data. By taking the maximum value within the window, it emphasizes the presence of strong features and diminishes the weaker ones.

3. The Experimental Result

In this section, we explain our experimental result. We use an input image captured by a camera. The image size is 640×480 pixels. This experiment is performed using programming language C++ and openCV library.

The programming code to convert the RGB input image into grayscale is:

```
cvtColor(imgOriginal,imgGrey,COLOR_BGR2GRA
Y);
```

The programming code to convert the grayscale image into CLAHE image with clip limit 4, are:

```
Ptr<CLAHE> clahe = createCLAHE();
clahe->setClipLimit(4);
clahe->apply(imgGrey,imgClahe);
```

To create the filter kernel 3×3 is as follows:

```
kernelPFH = (Mat_<int>(3,3) << -1, 0, 1, -1, 0, 1, -1,
0, 1); //Prewitt filter horizontal
```

Then the convolution process is performed in filter2D(src,dst,ddepth,kernel,anchor,delta,BORDER_DEFAULT) as:

```
filter2D(imgClahe,output, -1, kernel, Point(-1, -1), 0,
4);
```

Where the arguments denote:

- *src*: Source image.
- *dst*: Destination image.
- *ddepth*: The depth of *dst*. A negative value (such as -1) indicates that the depth is same as the source.
- *kernel*: The kernel to be scanned through the image.
- *anchor*: The position of the anchor relative to its kernel. The location *Point(-1,-1)* indicates the center by default.
- *delta*: A value to be added to each pixel during the correlation. By default it is 0.
- *BORDER_DEFAULT*: We let this value by default.

The result images are depicted in Figure 9, 10 and 11. Fig 9 is the result of convolution process in light and no-light condition. Fig 10 are the original, CLAHE and convolved image with different filter and filter size. Fig 11 is the convolved image before max pooling process (Fig. 11.c) and after max pooling process (Fig. 11.e). In Fig 11, it is shown the output image similar for both processes.

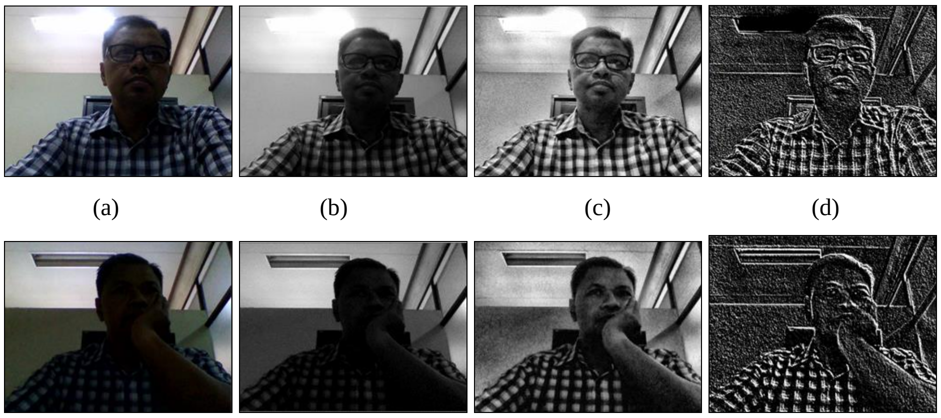

Fig 9: Processing real time video, (top) at light condition, (bottom) no light condition, (a) original frame (b) grayscale frame (c) CLAHE frame (d) convolved frame

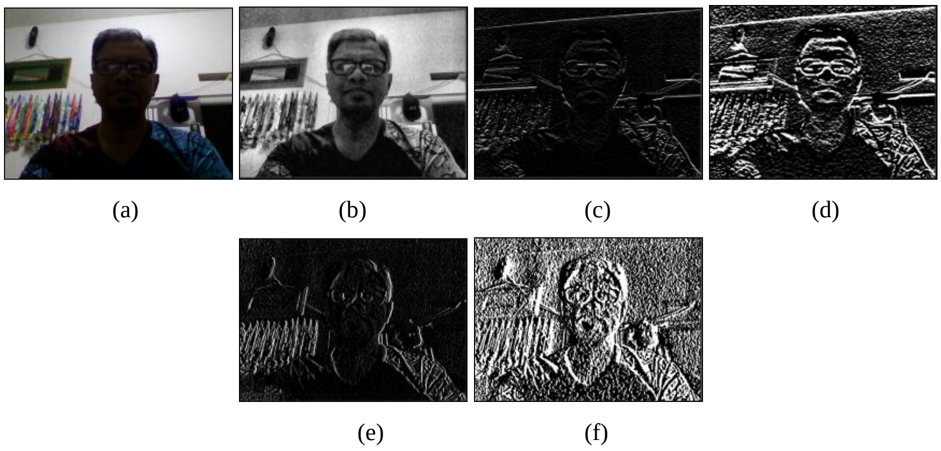

Fig 10: The original, CLAHE and convolved image (a) original image (b) CLAHE image (c) convolved image by Prewitt filter 3x3 (d) by Prewitt filter 5x5 (e) by Sobel filter 3x3 (f) by Sobel filter 5x5

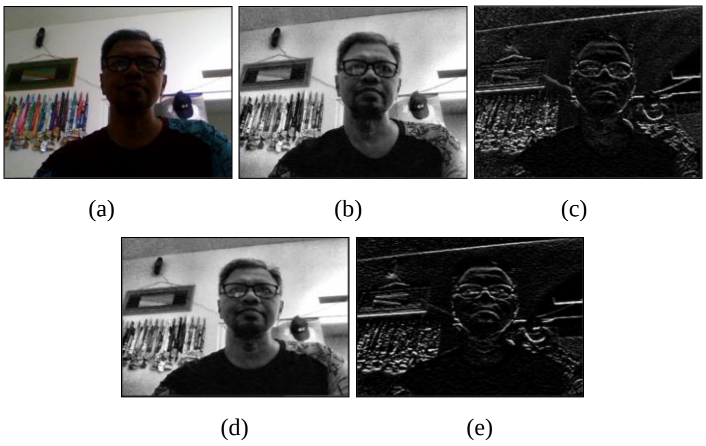

Fig 11: The original, CLAHE and convolved image, (top) convolution then max pooling process, (bottom) max pooling then convolution process, (a) original image, 640×480 pixels (b) CLAHE image, 640×480 pixels (c) convolved image, 320×240 pixels (d) CLAHE image, 320×240 pixels (e) convolved image, 320×240 pixels

4. Conclusion

In this paper, we demonstrate and observe the output image as the result of convolution and max pooling process. It is shown the output image similar for two processes. The first process is convolved image before max pooling process and the second process is convolved image after max pooling process.

5. References

1. Ghosh A, Sufian A, Sultana F, Chakrabarti A, De Debashis. Fundamental Concepts of Convolutional Neural Network, 2020. Doi: 10.1007/978-3-030-32644-9_36
2. Fukushima K Neocognitron. A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics. 1980; 36(4):193-202.
3. Pizer SM, Johnston RE, Ericksen JP, Yankaskas BC, Muller KE. Contrast-limited adaptive histogram equalization: Speed and effectiveness. [1990] Proceedings of the First Conference on Visualization in Biomedical Computing, Atlanta, GA, USA, 1990, 337-

345.
4. Yadav G, Maheshwari S, Agarwal A. Contrast limited adaptive histogram equalization-based enhancement for real time video system. 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, 2392-2397.
5. Pizer SM, Amburn EP, Austin JD, *et al*. Adaptive Histogram Equalization and Its Variations. Computer Vision, Graphics, and Image Processing. 1987; 39:355-368.
6. Zuiderveld K. Contrast Limited Adaptive Histogram Equalization. In: P. Heckbert: Graphics Gems IV, Academic Press, 1994. ISBN 0-12-336155-9
7. Sund T, Moystad A. Sliding window adaptive histogram equalization of intra-oral radiographs: Effect on diagnostic quality. Dentomaxillofac Radiol. 2006; 35(3):133-138.
8. Vidhya GR, Ramesh H, Effectiveness of contrast limited adaptive histogram equalization technique on multispectral satellite imagery, Proc. Int. Conf. Video Image Process, 2017, 234-239.
9. Venkatesh S, John De Britto C, Subhashini P, Somasundaram K. Image Enhancement and Implementation of CLAHE Algorithm and Bilinear Interpolation. Cybernetics and systems: An International Journal, 2022.
10. Filters in convolutional neural networks, 2022. https://blog.paperspace.com/filters-in-convolutional-neural-networks/.
11. Stephen Huan. Fast Fourier Transform and 2D Convolutions, 2020.
12. Vincent Mazet. Convolution. Basics of Image Processing — (Université de Strasbourg), 2020-2024. https://vincmazet.github.io/bip/filtering/convolution.html
13. Pupeikis Rimantas. Revised 2D convolution, 2022. Doi: 10.13140/RG.2.2.11346.68809