# Convolution: Image Filters, CNNs and Examples in Python & Pytorch

Edward Roe · Follow

12 min read · Jun 7, 2023



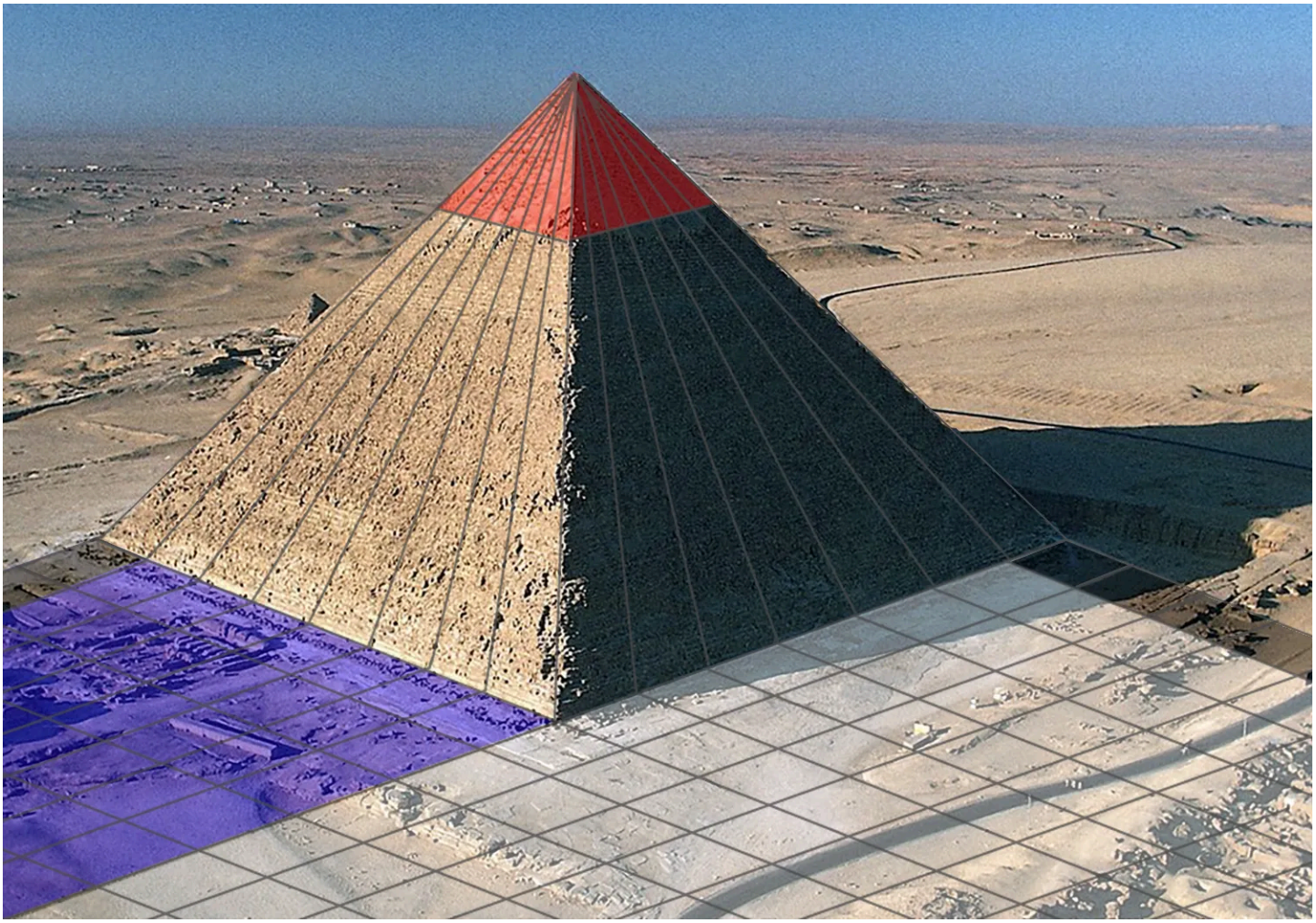


Image by author.

## Introduction

Two-dimensional (2D) convolution is well known in digital image processing for applying various filters such as blurring the image, enhancing sharpness, assisting in edge detection, etc. With the advent of Convolutional Neural Networks (CNN), convolutions gained a new interest. Convolutions are based on the idea of using a filter, also called a kernel, and iterating through an input image to produce an output image. This story will give a brief explanation of convolution using visual examples and code snippets in Python that show how to implement a simple convolution.

## Convolution

Convolution is a linear operator widely used in signal processing that, from two given functions, results in a third that measures the sum of the product of these functions along the domain implied by their superposition. In digital image processing in particular, convolution is a mathematical method for combining two images to produce a third image. Typically, one of the two combined images is not an image itself, but a matrix whose size and values determine the nature of the effect of the convolution process; this matrix is called a filter or kernel. The basic idea is to align the kernel over each pixel of the image and multiply and sum its values over the pixel and its local neighbours. For those more interested in mathematics, the formula for convolution in two dimensions is given by Equation 1 [1]:

$$I'(u,v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i,j) \qquad (1)$$

where $I$ and $H$ are two-dimensional functions, $(u, v)$ are the coordinates of the pixel in the image and $(i, j)$ are the coordinates of the kernel element. Or Equation 2 more closely related to the focus of this story [2]:

$$F * I(x,y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} F(i,j) I(x-i, y-j) \qquad (2)$$

where $F$ is the filter, or kernel, which having an odd number of elements, is represented by a matrix $(2N+1)$ x $(2N+1)$, $(x, y)$ are the coordinates of the pixel in the

image and (*i*, *j*) kernel element coordinates. The animation in Figure 1 illustrates this process on a 7x7 image. The image on the left is the original image and the one on the right is the result of the applied convolution. Note that the pixel resulting from the addition and multiplication operations is stored in the new image and the original image pixels remain unchanged. I have borrowed some of the following figures and animations from another story of mine on Binarisation of documents using the U-Net.

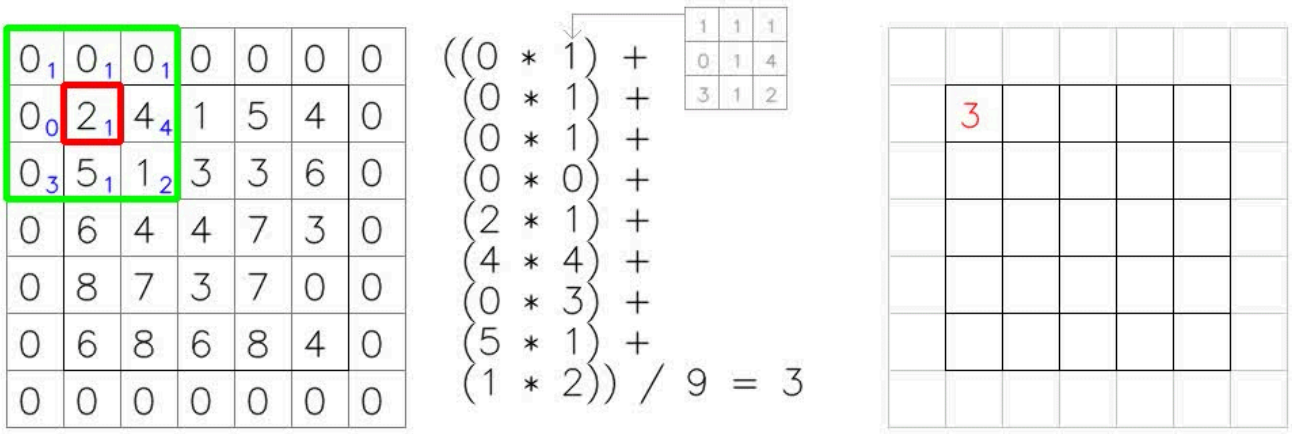



Figure 1. Animation illustrating how convolution works. In this case, no padding added and step 1. Note that the resulting image (right) is smaller than the original image (left), in this case 5×5 instead of 7×7. Animation by author.

The most common use of convolutions in digital image processing is the implementation of filters for edge detection, blurring and noise reduction. Although the effect of convolutions in intermediate layers in CNNs is well known, it can be made even clearer by showing their effect as digital filters.

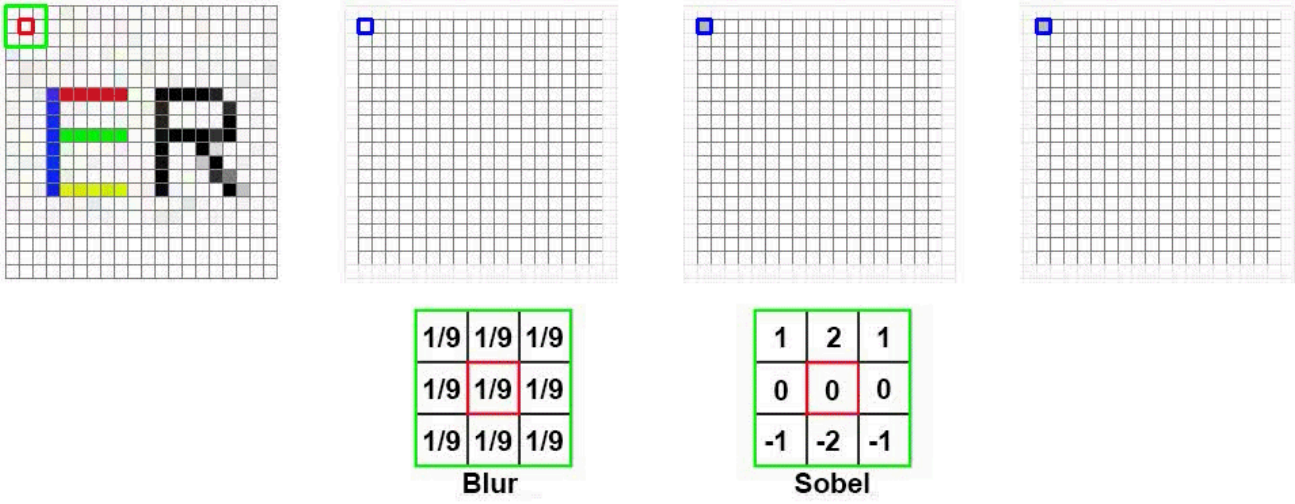



Figure 2. Example of the convolution process in image processing. The results in the centre are Sobel's blur and edge detection (the corresponding kernels are shown below the resulting image). Normally, images are first converted to greyscale before a filter is applied to avoid the undesirable result as shown in image on the right. In CNNs, the red, green and blue channels are used separately. Animation by author.

Figure 2 shows the convolutions applied to the left image with different kernels, first a blur and then Sobel edge detection. These two convolutions were applied to a greyscale version of the original image. In CNNs, convolution is performed separately for each RGB channel, which is not common in image processing as it leads to an undesired result, like the one on the right in Figure 2. The two matrices in the bottom row of Figure 2 represent the kernels used, and during convolution the target pixel in the image must match the centre of the kernel (red square). Convolutions are part of the implementation of various digital image processing filters such as blurring, edge detection (Sobel and Laplacian), etc. The kernel size directly affects the final result, as shown in the following example (Figure 3) where two kernels with the same element values but different sizes, one 3x3 and the other 5x5, are applied to the same image.

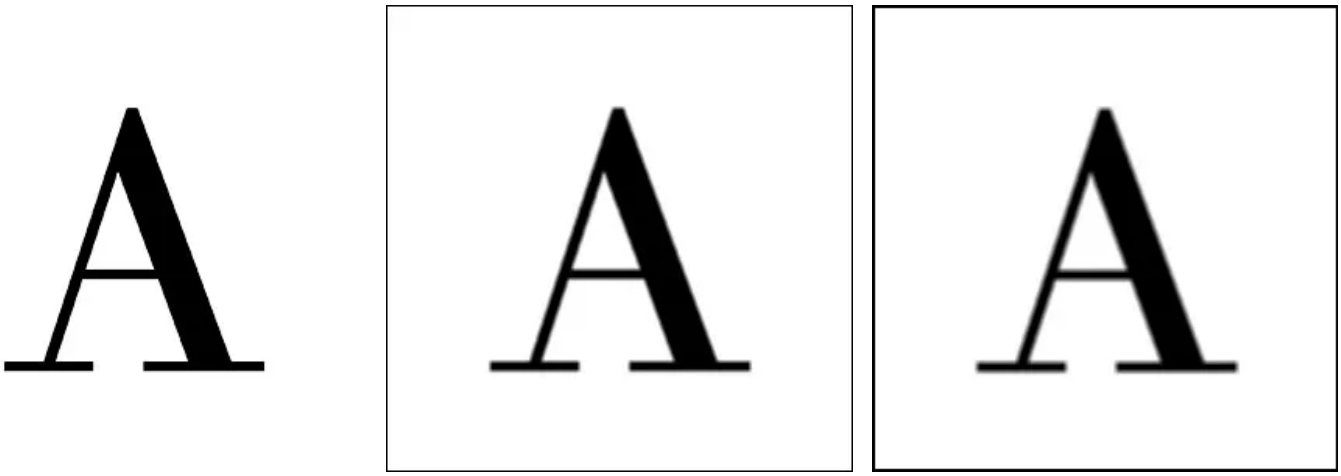


Figure 3. Example of applying a blur filter with the convolution process using the code in Listing 1. Original image (left), image after convolution with kernel blur_3×3 (centre) and image after convolution with kernel blur_5×5 (right). The black borders are not part of the resulting images, but are displayed to show that the resulting image is smaller than the original. Image by author.

The images in Figure 3 were created using the code in Listing 1 below. It's not the best way to implement convolutions but it's useful for illustrating the process from scratch.

```
################################## Listing 1 ##################################

blur_3x3 = np.array((
     [1, 1, 1],
     [1, 1, 1],
     [1, 1, 1]), dtype="int")
blur_5x5 = np.array((
```

```
        [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]), dtype="int")

def conv(img_in):
    img = cv2.imread(img_in)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) / 255.
    height = img.shape[0]
    width = img.shape[1]
    img_res = np.zeros([height, width, 3], dtype=np.uint8)
    img_res.fill(1)
    kernel = blur_4x4
    height_k = kernel.shape[0]
    width_k = kernel.shape[1]
    factor = height_k * width_k
    init_x = kernel.shape[1] // 2
    init_y = kernel.shape[0] // 2
    for x in range(init_x, width - init_x):
        for y in range(init_y, height - init_y):
            resulting_pixel = 0
            for xk in range(width_k):
                for yk in range(height_k):
                    resulting_pixel += kernel[yk, xk]*img[y+(yk-init_y),
                                                          x+(xk-init_x)]
            img_res[y, x] = int((resulting_pixel / factor) * 255)
    cv2.imshow('', img_res)
    cv2.waitKey(0)
```

The image in Figure 4 will be used as the input image in the various examples that follow in this story.

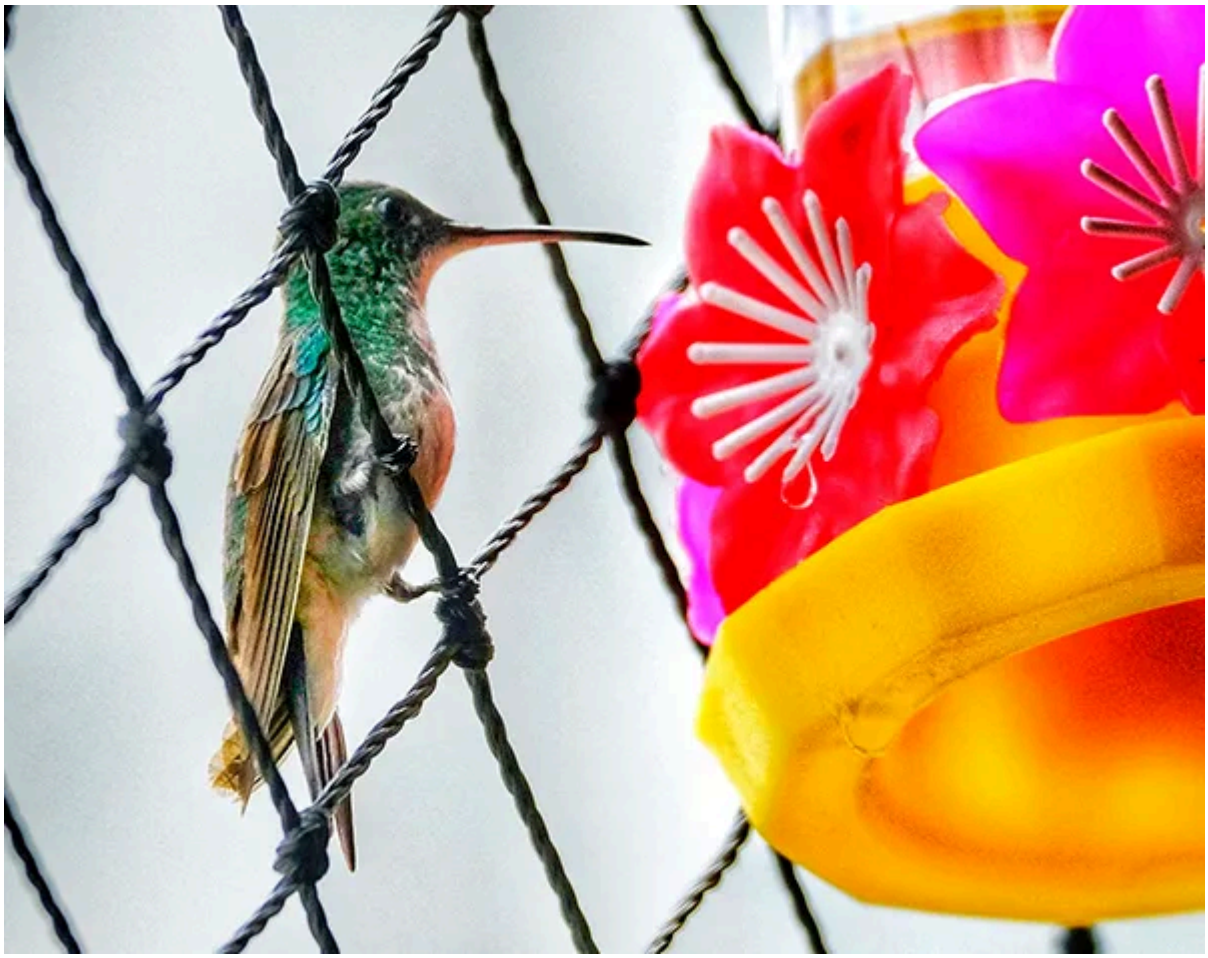


Figure 4. Input image used in following examples. Image by author.

Some more common kernels are shown in Figure 5. Depending on the intended use, the values and type (integer, float) may vary. For example, a blur filter might contain only 1/9 without requiring the final division by 9. The images next to each filter are representations commonly seen in visualisations of weights in CNNs. Figure 6 shows examples of the application of these filters.


Figure 5. Some examples of kernels used in digital imaging. Next to each kernel is a visual representation similar to that used in CNN weight visualisations. Image by author.

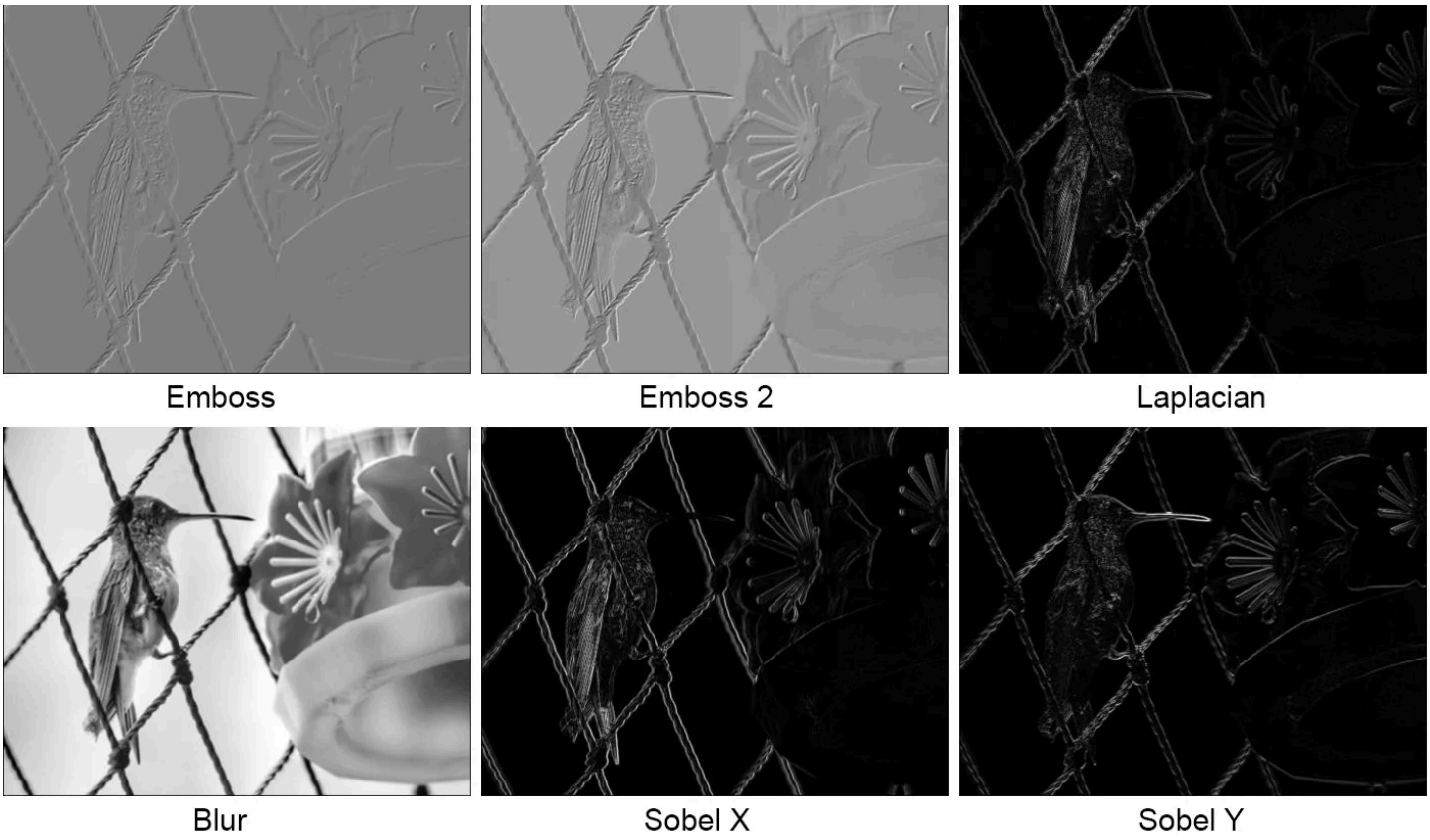



Figure 6. Examples of results from applying filters using kernels shown in Figure. Image by author..

Padding and stride are two important concepts in convolutional neural networks (CNNs) that affect the size of the output feature maps.

**Padding**: as can be seen from the examples in Figure 3, each resulting image is smaller than the original by an amount related to the kernel size; the longer the kernel, the farther the center is from the edge of the image. To produce an output the same size as the input, we pad the edges with extra pixels. That way, when sliding, the kernel can allow the original border pixels to be in its center, while extending the extra pixels beyond the border.

Figure 7 shows the original image used as a reference for the edge fill examples, four colored dots have been added in the corners to help demonstrate the difference between each method.

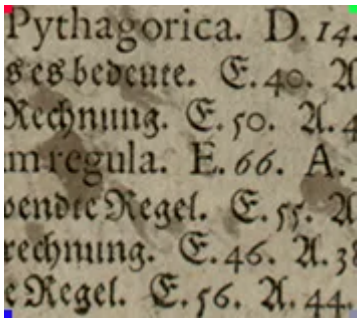


Figure 7. Original reference image for OpenCV padding applications. four colored dots (red, green, blue and gray) have been added at the corners to help demonstrate the difference in each method. Image by author.

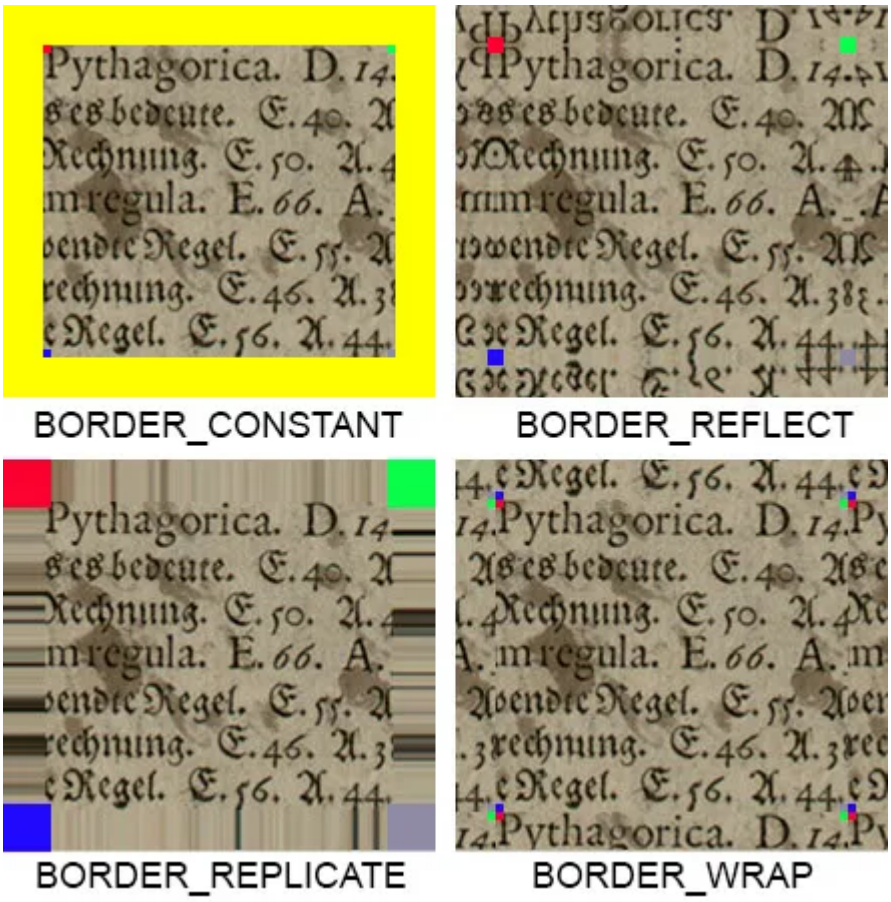



Figure 8. Four examples of population using OpenCV. (top left) with **BORDER_CONSTANT** (in yellow), (top right) with **BORDER_REFLECT**, (bottom left) with **BORDER_REPLICATE** and (bottom right) with **BORDER_WRAP**. Image by author.

Figure 8 shows four methods of padding using OpenCV's copyMakeBorder function. Which of these methods to choose depends on the problem at hand. Remembering that we apply padding so that the resulting image has the same dimensions as the input image, in addition to including the pixels at the edges of the image.

Figure 9 illustrates an animation similar to Figure 1, the difference here is that in this case the input image, the one on the right, has 5x5 dimensions and had a border with zeros added around it, generating a 7x7 image. The image on the right, the

result of the applied convolution, now has the same dimensions as the original image, 5x5.

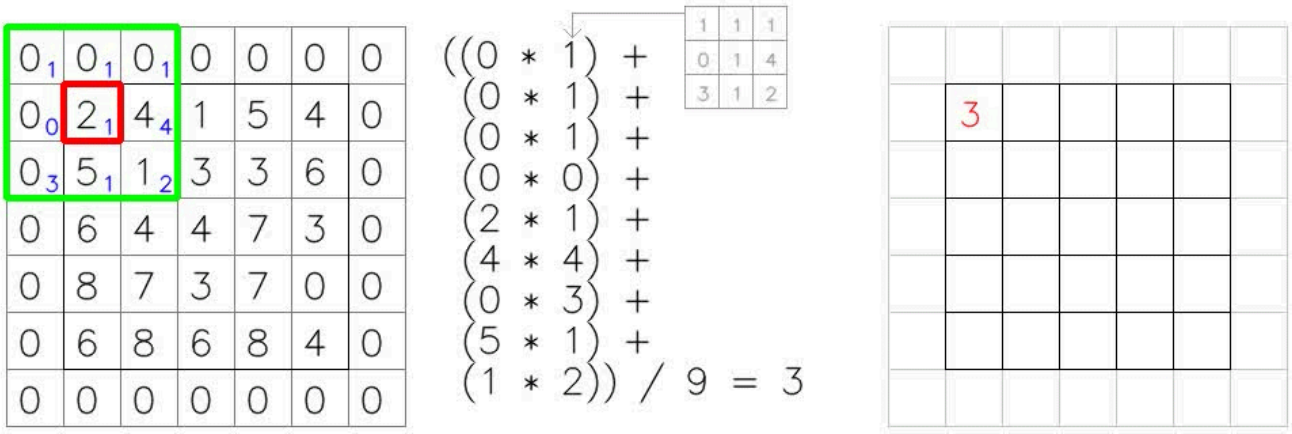



Figure 9. Same idea as Figure 1, but this time with size 1 padding added (the edge with zeros around the image). Note that the resulting image (right) has the same dimensions as the original image (5×5) without the border added (left). The green rectangle represents the kernel and the red the target pixel. Animation by author.

**Stride:** Stride is the number of pixels each kernel window offsets over the input array. A step of 1 means the kernel will go through all the pixels in the image without skipping any. A step of two means choosing offsets two pixels apart. The bigger the jump, the smaller the resulting image. In Figure 1 and Figure 9 stride is equal to 1. Figure 10 shows the difference in kernel movement with stride=2, resulting in a 3x3 image.

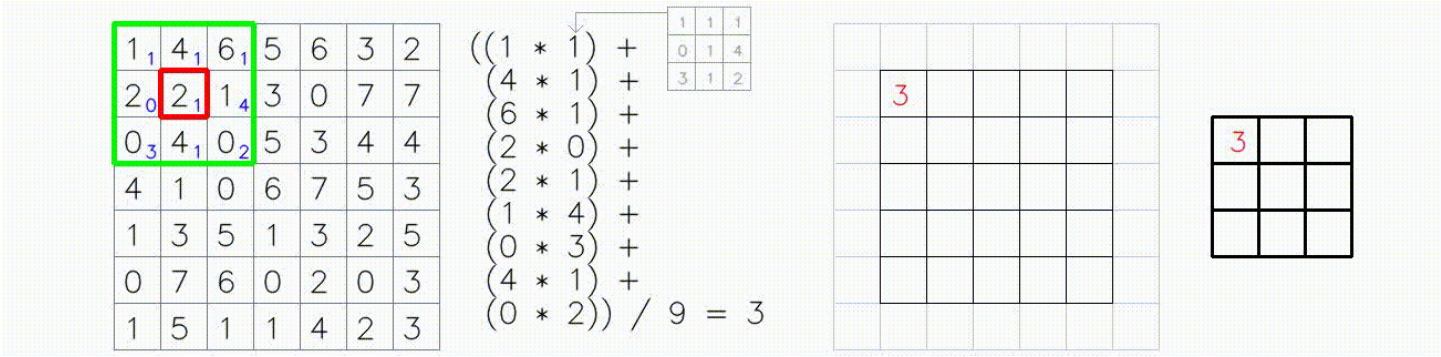



Figure 10. Same idea as Figure 1, but this time with padding of size 1 added (the border with zeros around the image) and stride=2. Note that the resulting image (right) has smaller dimensions (3×3)than the original image (7×7). The green rectangle represents the kernel and the red the target pixel. Animation by author.

The relationship between the input size, kernel size, padding, and stride can be expressed using the following formula:

$$output\_size = ((input\_size - kernel\_size + 2 * padding) / stride) + 1$$

In convolutional neural networks (CNNs), the convolution step is a fundamental operation that applies a set of filters (kernels) to an input image or feature map to extract important features resulting in a feture map. The number of resulting feature maps corresponds to the number of filters, as shown in Figure 11.

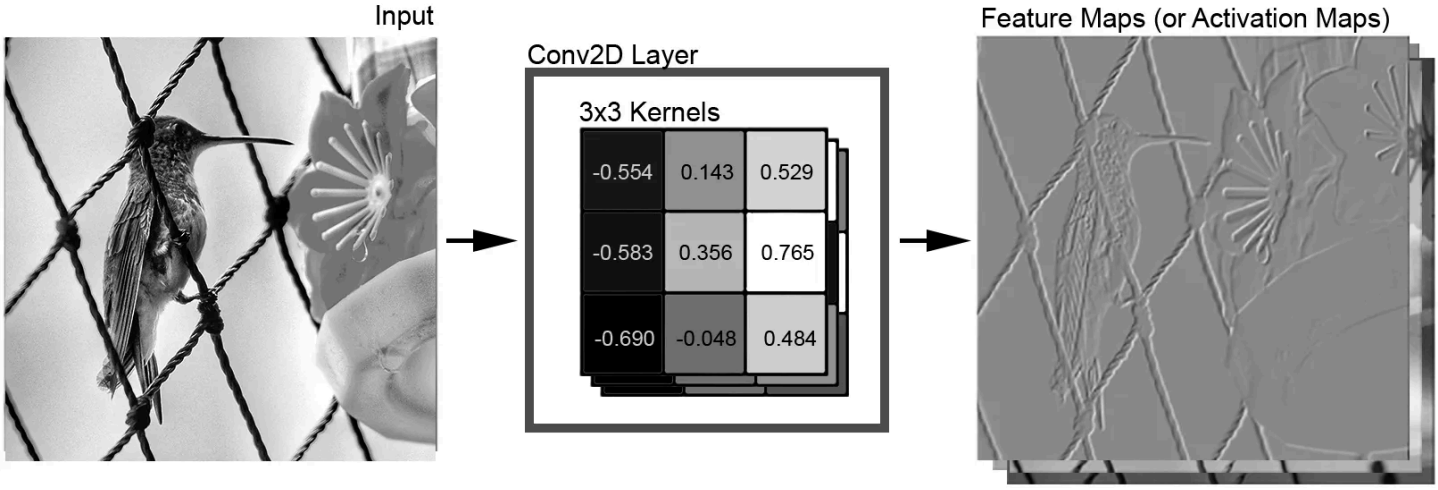



Figure 11. Participating elements of a convolutional layer. Input image (left), Conv2D layer with three kernels or filters (middle) and the three resulting feature maps (right). Image by author.

## 1×1 Convolution

One of the disadvantages of deep convolution networks is that the number of feature maps often increases with the depth of the network. This problem can lead to a significant increase in the number of parameters and computational complexity, especially when large filter sizes are used, such as 5x5 and 7x7. The solution is to use a 1×1 filter to reduce the depth or number of feature maps, and was introduced in the Network in Network paper by Lin et al. [3] and used extensively in Google's Inception architecture [4].

In 1x1 convolution, the input is convolved with filters of size 1x1, usually with zero padding and a stride of 1. It is a 1x1x*C* operation, where *C* is the number of channels or feature maps that do not include neighbours of the same channel. This 1x1 convolution can then be applied like a 2D convolution from left to right and top to bottom with a stride of 1, without the need for padding, resulting in a feature map with the same width and height as the input. The 1x1 convolution is also a linear combination of the channels, but you can add non-linearity by applying activation functions to the output.

The animation in Figure 12 shows an example of a 1x1x7 convolution applied to a 3x3x7 activation map, resulting in a 3x3x1 activation map.

See here for a detailed explanation of the 1x1 convolution.

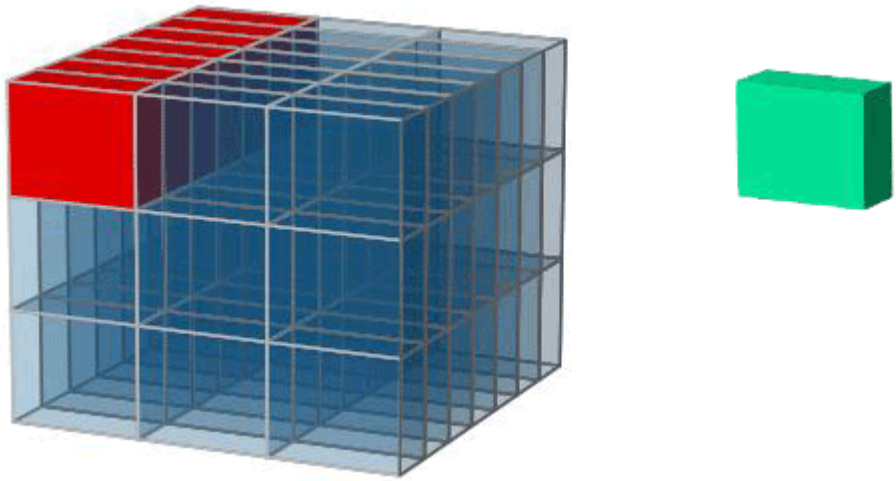


Figure 12. Example of a 1×1×7 convolution applied to a 3×3×7 activation map (left) resulting in a 3×3×1 activation map (right). Animation by author.

## 3D Convolution

In all the previous considerations and examples, convolution has been applied to images or matrices with two dimensions, but the same idea works for three-dimensional matrices. In the case of a three-dimensional matrix, as is often the case with convolutions in CNN-based models, the kernel will also be a 3D matrix. In the animation in Figure 8, the kernel is a cube of dimensions 3x3x3 that traverses another cube of dimensions 5x5x5. As with convolution in two dimensions, the result is one pixel at each iteration. Since stride=1 and no padding was applied in this example, the resulting cube has dimensions 3x3x3.

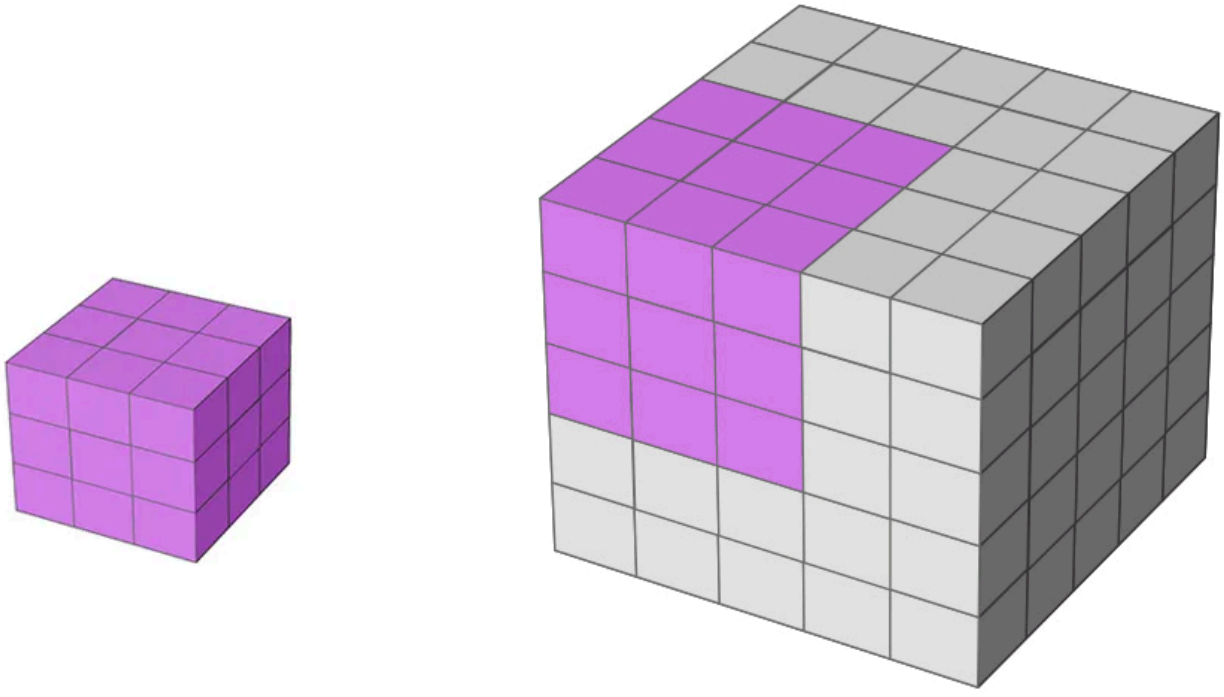


Figure 13. Example of the convolution of a 3D matrix. In this case the kernel is also 3D, the 3×3×3 cube (left), no padding and stride=1. The animation shows how the kernel traverses the three dimensional array (right). Animation

by author.

The same padding and stride concepts apply for three-dimensional convolutions.

## Visualisation of Filters in Pytorch

Neural networks are usually initialised with random values. In the case of CNNs, these initial values are the filter elements (kernels). As training progresses, these filters take on an "organised" aspect being possible to recognise a certain pattern in each of them. This difference between untrained and trained kernels can be seen in Figure 14.

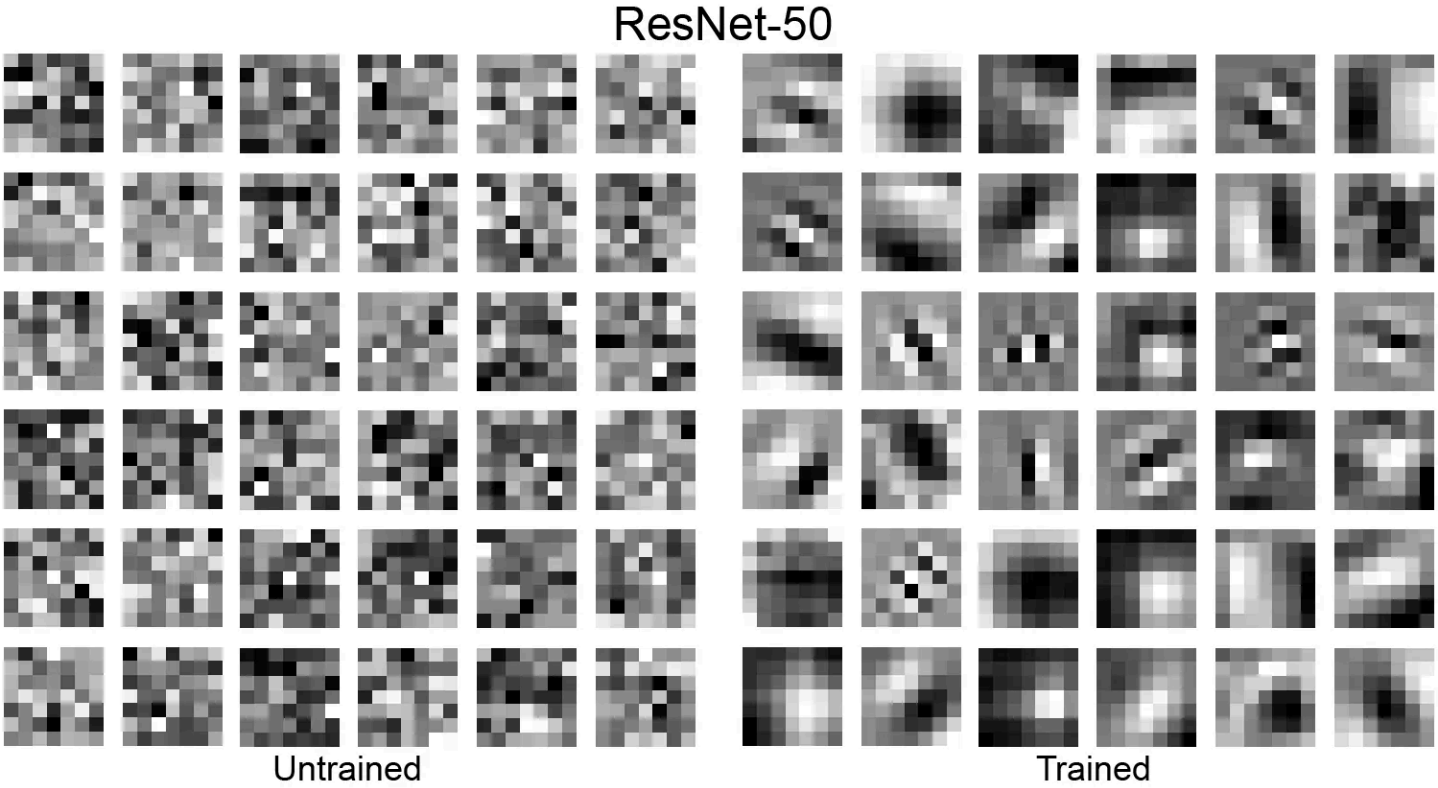



Figure 14. Some kernels of the first convolutional layers of the ResNet-50 model. Untrained kernels look very random, while trained kernels show distinct patterns. Image by author.

As the training evolves and the random aspect of the kernels is abandoned, the feature maps also change, as they are the result of convolution by these kernels. It is possible to observe (Figure 15) similarities in the feature maps resulting from the trained kernels with the results of applying the filters shown in Figure 6 (Sobel, emboss, Laplacian, etc.).

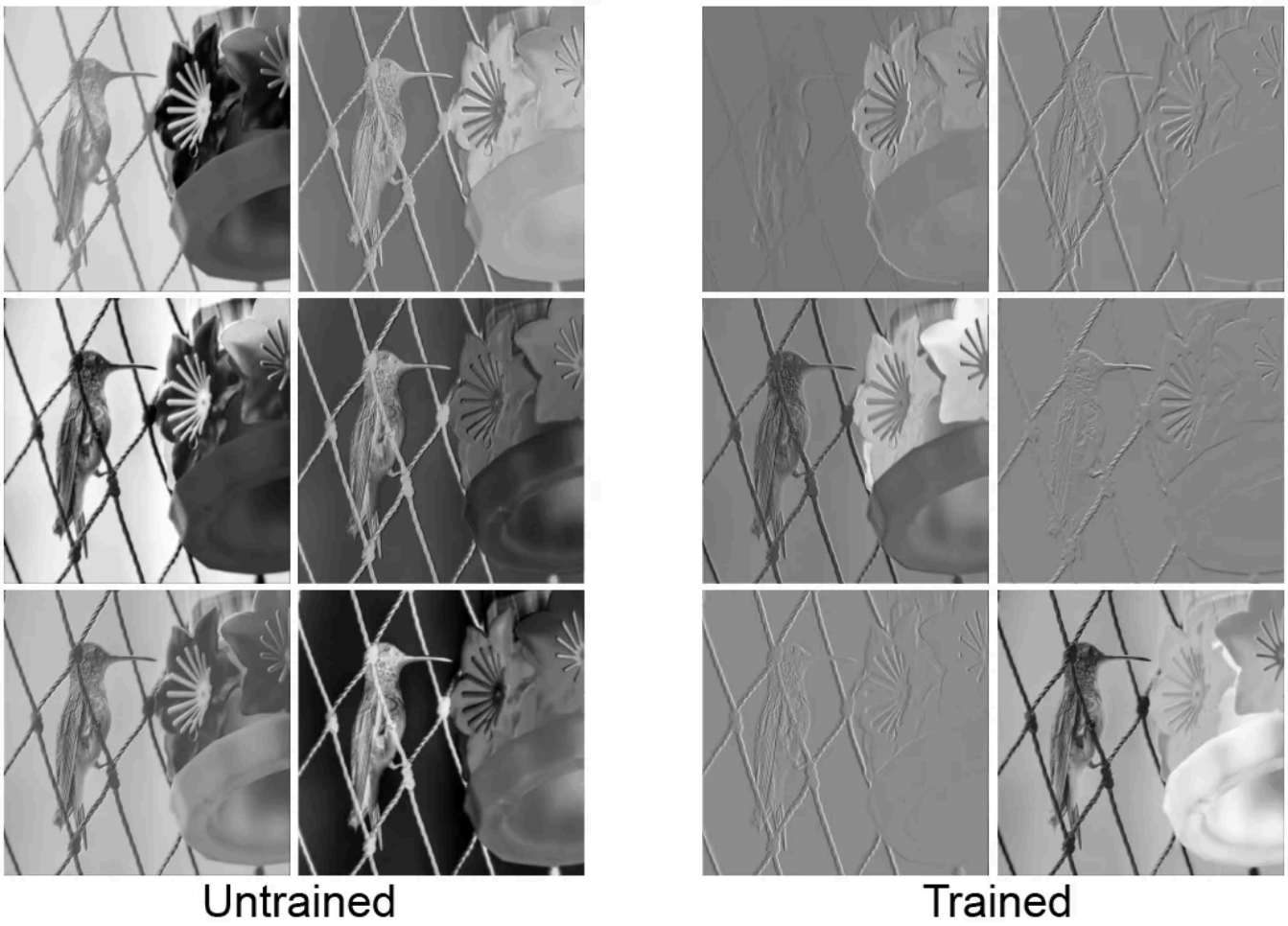



Figure 15. Some feature maps resulting from the first convolutional layers of the VGG-16 model. Note that some feature maps resulting from trained kernels (right) resemble the result of applying the embossing filter. Image by author.

The Figure 16 shows the behavior of the feature maps as the convolution layer gets deeper.

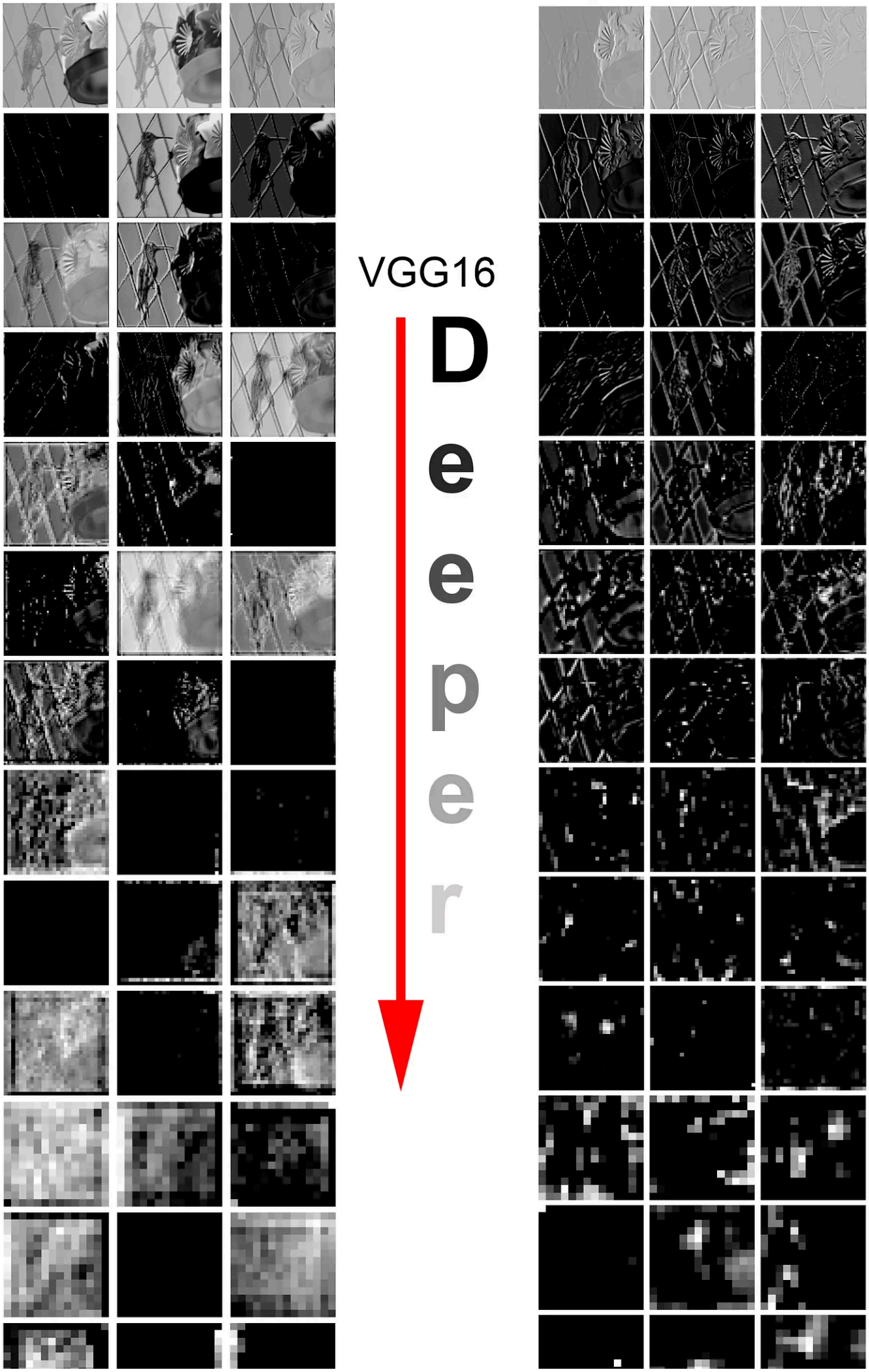

VGG16
Deeper


Figure 16. Example of resulting feature map from each 13 convolutional layers of the VGG-16 model. Images have been resized for better viewing. Image by author.

Figure 16 keeps the same size for all feature maps for better visualisation, but in reality there is a decrease in dimensions as shown in Figure 17.

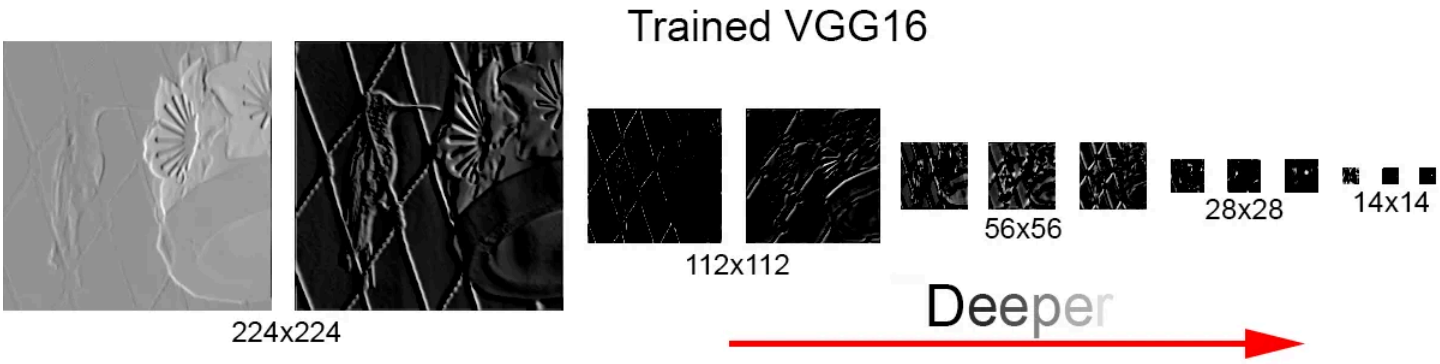



Figure 17. Evolution of feature maps size during forward pass in the VGG-16 model. Image by author.

## Transposed Convolution

The transposed convolutional layer, whose purpose is essentially to increase the dimensions (height and width) of its inputs, is also incorrectly called a deconvolutional layer. A deconvolution layer reverses the operation of a convolution layer and gets back the original input. The transposed convolution reverses the convolution not by values but by dimensions. The transposed convolutions can be considered as standard convolutions with a modified input feature map.
To change the input imageor feature map, sequences of zeros are inserted in the columns and rows between the existing values of the input image. The number of columns/rows to be inserted is indicated by stride-1 (stride and paddind also apply here). In the following example (Figure 18) stride=2 and padding=1.

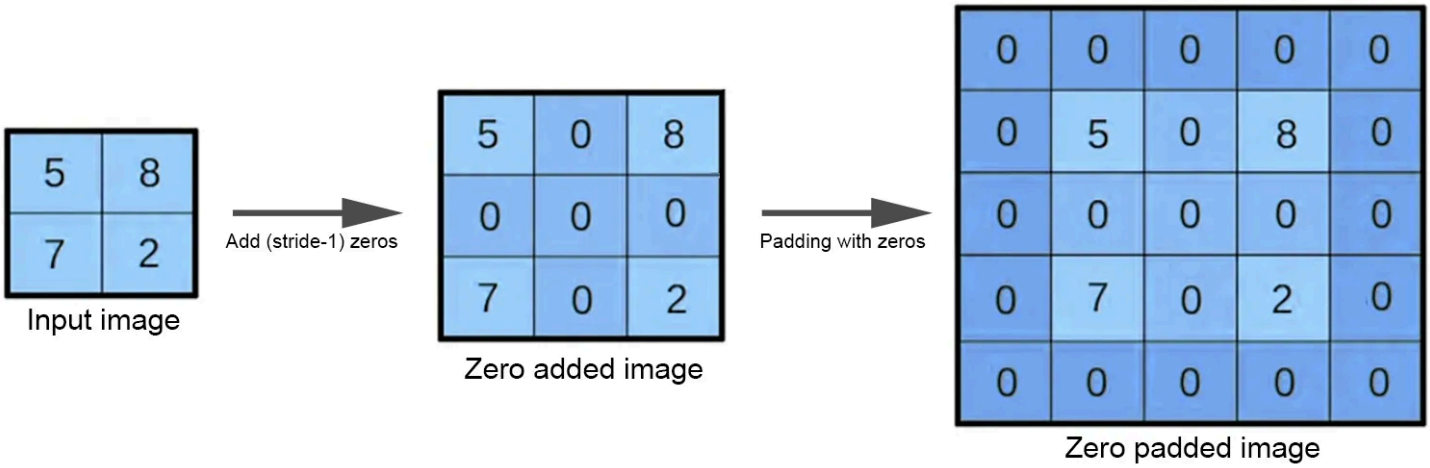



Figure 18. example of changing the input image/feature map for transposed convolution. Image by author.

The code in Listing 2 illustrates a very simple convolutional autoencoder (trained on STL10 dataset), just to illustrate the use of transposed convolutions. After training a thousand epochs, I obtained a very inefficient blur filter.

```
################################## Listing 2 ##################################

class ConvAutoEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, (3, 3), stride=(3, 3), padding=(1, 1)),
            nn.ReLU(True),
            nn.Conv2d(16, 32, (3, 3), stride=(3, 3), padding=(1, 1)),
            nn.ReLU(True),
            nn.Conv2d(32, 64, (3, 3), stride=(2, 2), padding=(1, 1)),
            nn.ReLU(True)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, (3, 3), stride=(2, 2), padding=(1, 1)),
            nn.ReLU(True),
            nn.ConvTranspose2d(32, 16, (3, 3), stride=(3, 3), padding=(1, 1)),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 1, (3, 3), stride=(3, 3), padding=(1, 1)),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
```

```
        return x
```

The evolution of a feature map through the layers can be seen in Figure 19.

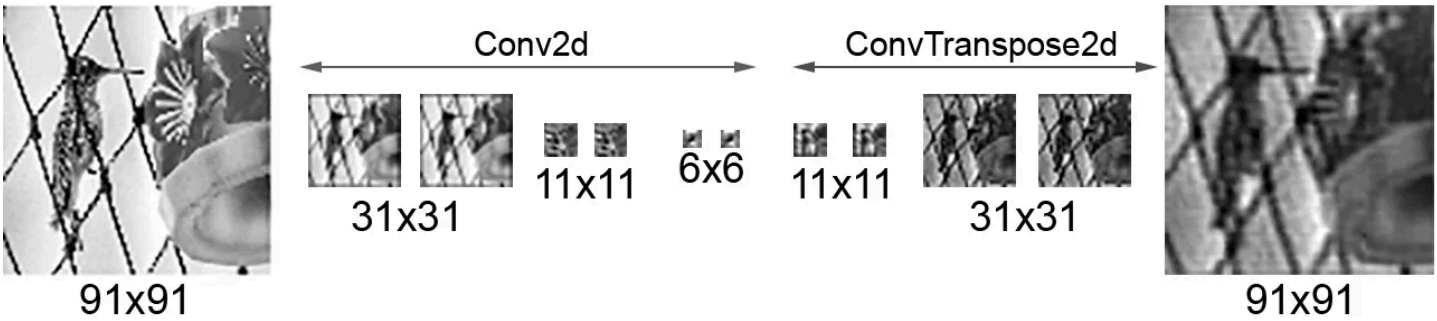



Figure 19. The evolution of a feature map through the six layers of the trained model. Image by author.

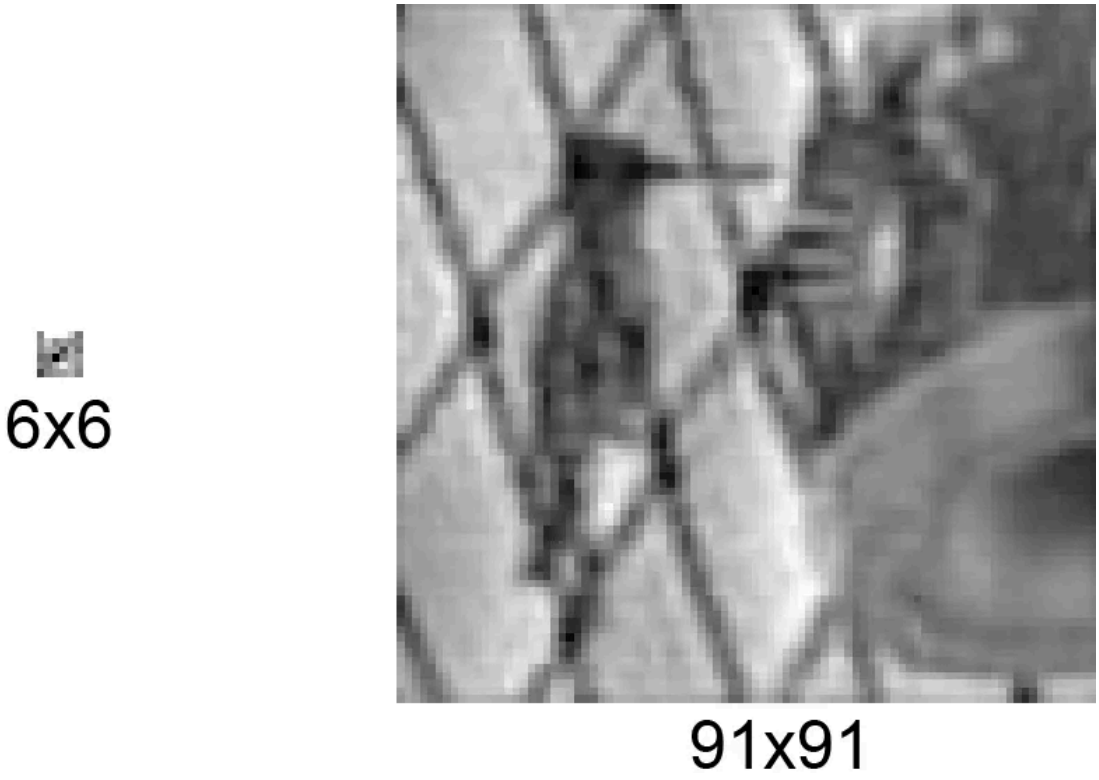



Figure 20. Comparison between two extremes of transposed convolution. The 6×6 image doesn't even remotely resemble the hummingbird figure. Image by author.

These two stories (here and here)provide further explanations on transposed convolutions.

## Convolution x Correlation

Convolution and correlation are mathematical operations that involve the combination of two functions or signals with similar in concept. There is a

fundamental difference between the two tha is the treatment of the kernel.

- In convolution, the filter/kernel is flipped (rotated 180 degrees) before the operation is performed. The flipped filter is then slid over the input signal, and at each position, an element-wise multiplication is performed between the filter and the corresponding portion of the input.
- In correlation, the filter/kernel is not flipped before the operation. It is directly slid over the input signal, and at each position, an element-wise multiplication is performed between the filter and the corresponding portion of the input.

In both cases the resulting products are summed up to obtain a single value. The process is repeated for each position, generating the output signal.

If the filter is symmetrical, the result of correlation and convolution would be the same. In this story, however, we can assume that we are applying a convolution in which the kernel has previously been rotated 180°.

## Final considerations

This story briefly discusses the following topics related to convolution:

- Concept of convolution
- Difference between greyscale and colour images
- Elements of convolution: kernel, stride, padding
- Two-dimensional convolution in greyscale and colour images
- Illustration of different kernels when using filters
- 1x1 convolution
- Example of three-dimensional convolution using an animation
- Transposed convolution with a toy example in Pytorch
- Difference between convolution and correlation

Codes with examples are available on Github.

## References

[1] Gonzalez, R. C., Woods, R. E., Digital Image Processing. 4th edition. Pearson Prentice Hall, 2017.

[2] Burger, W. e Burge, M. J., Digital Image Processing: An Algorithmic Introduction using Java. Springer, 1st edition, 2008.

[3] Lin, M., Chen, Q. and Yan, S., Sermanet, P., Szegedy, C., Network in network. CoRR, abs/1312.4400, 2013

[4] Szegedy, C., Liu, W., Jia, Y., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., Going Deeper with Convolutions, arXiv:1409.4842v1 [cs.CV], 2014.

Convolution | Filters | Cnn Model | Deep Learning | Image Processing

Follow

## Written by Edward Roe

31 Followers · 3 Following

System Engineer

## Responses (3)

What are your thoughts?

Respond

Dewald van Hoven
Nov 23, 2024

Fantastic explanation, thank you!

1 reply Reply

Enkhnyam Battulga
Jun 18, 2024

Can I ask how did you make your animations? Would you mind making it open source?

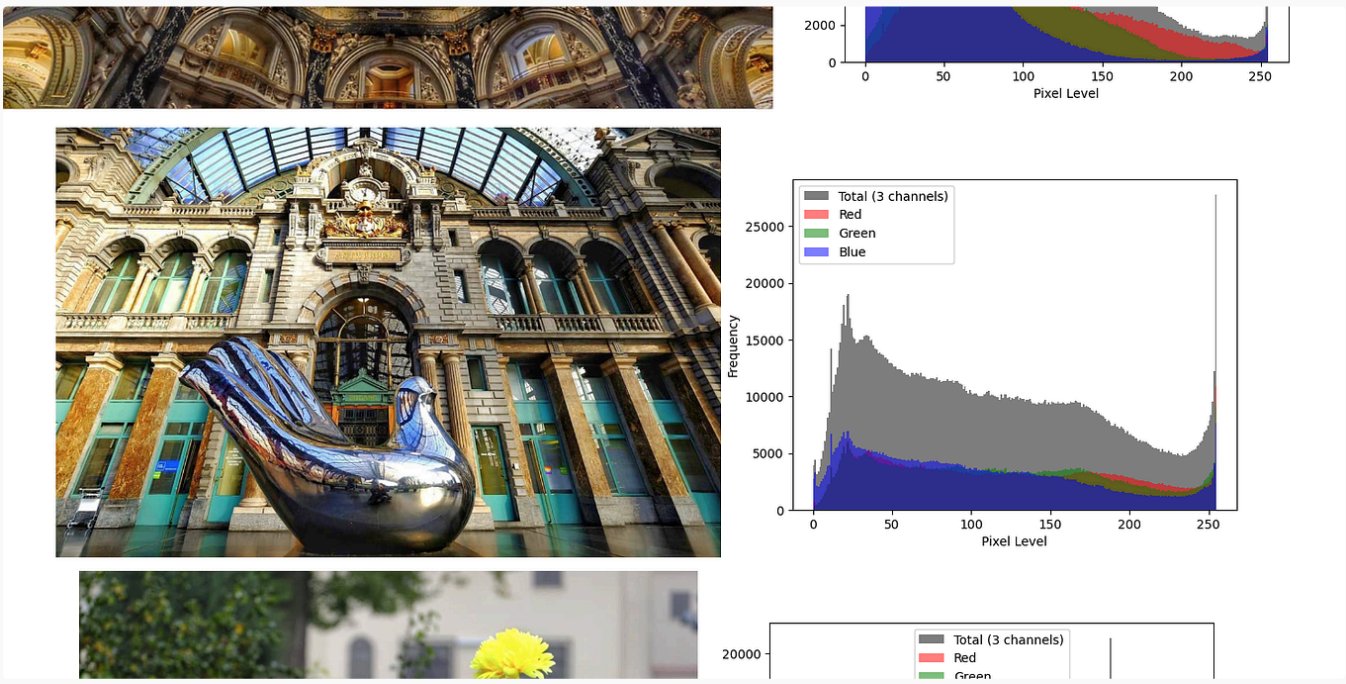
1 reply Reply

Candice Heaven
Apr 27, 2024

Thanks a lot for the on point explanations

Reply

## More from Edward Roe




Edward Roe

### Improving Images Using Equalisation and Histogram Matching with Python

Intoduction

May 25, 2023 19 1

Edward Roe

## ASL Recognition Using PointNet and MediaPipe

Sign language recognition plays a critical role in facilitating communication and inclusion of people with hearing disabilities. Recent...

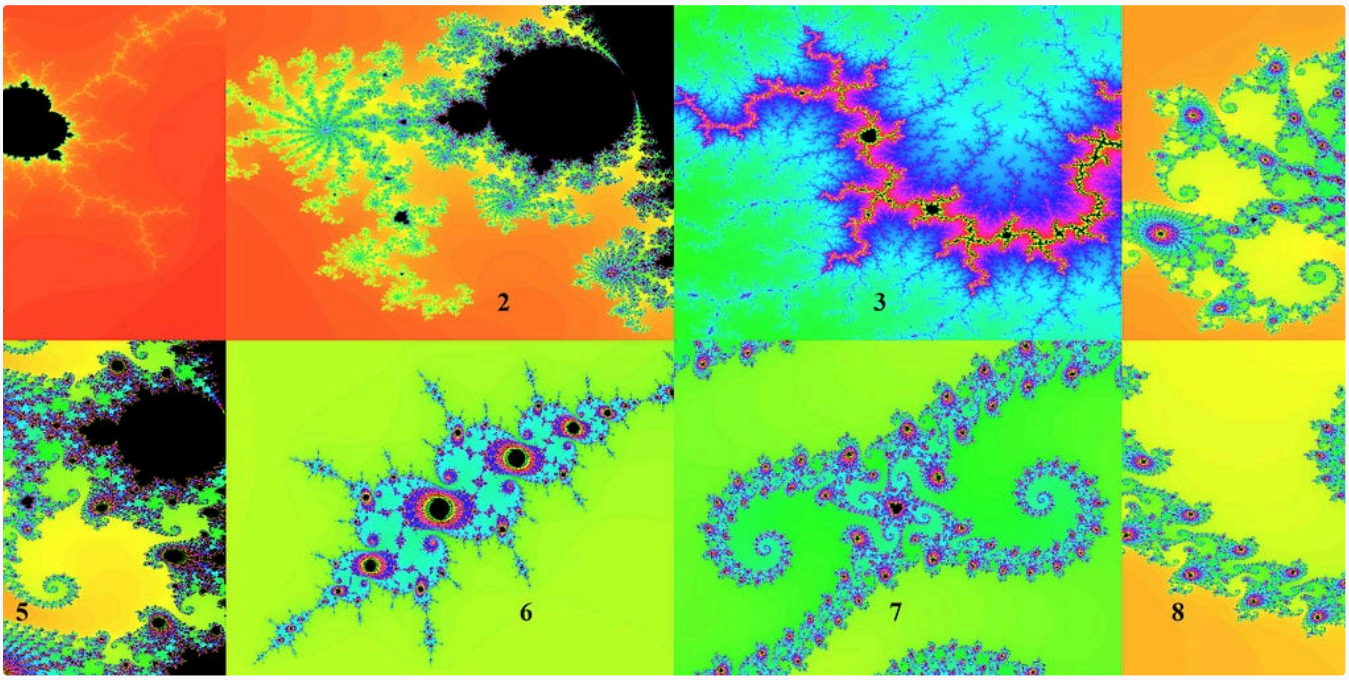
Jul 27, 2023 30

Edward Roe

## Colour Blindness simulator in Python: Detecting Visualisation Problems in Presentations and Reports

Intoduction

May 25, 2023 19




Edward Roe

## Animating Fractals with Python: Julia and Maldelbrot Sets

Computing Julia and Mandelbrot sets using a computer language are very easy, at least without any kind of optimizations as I will present...

Apr 7, 2022 4 1

See all from Edward Roe

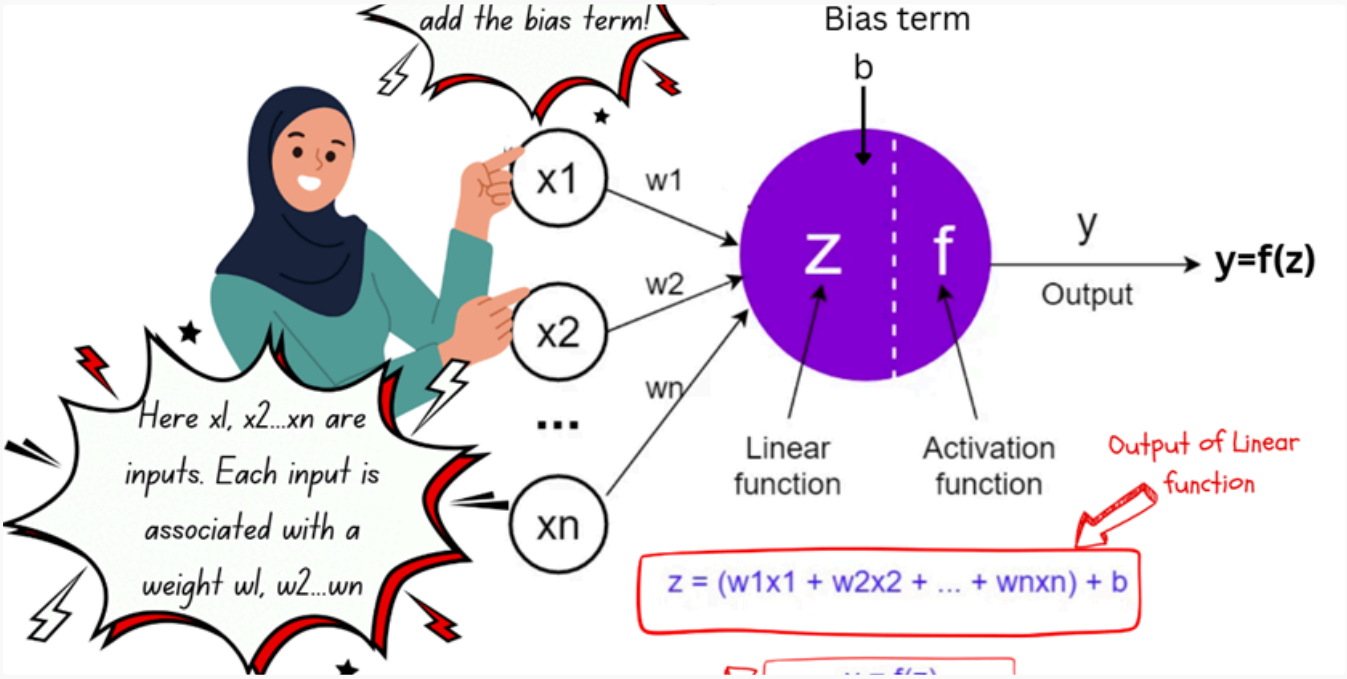
## Recommended from Medium


Sumitkrsharma

## Ideal High Pass Filter in Image preprocessing

Edges and fine detail in images are associated with high frequency components, if there is a use case which requires in depth details...

Aug 6, 2024




In GoPenAI by Mohana Roy Chowdhury

## Everything you need to know about CNNs Part 4: Dense Layer

So far in the series, we've talked about the Convolution layer, ReLU, and the Pooling layer. While these three are important components of...

Jan 15 102

## Lists

### Natural Language Processing

1894 stories · 1556 saves

### Practical Guides to Machine Learning

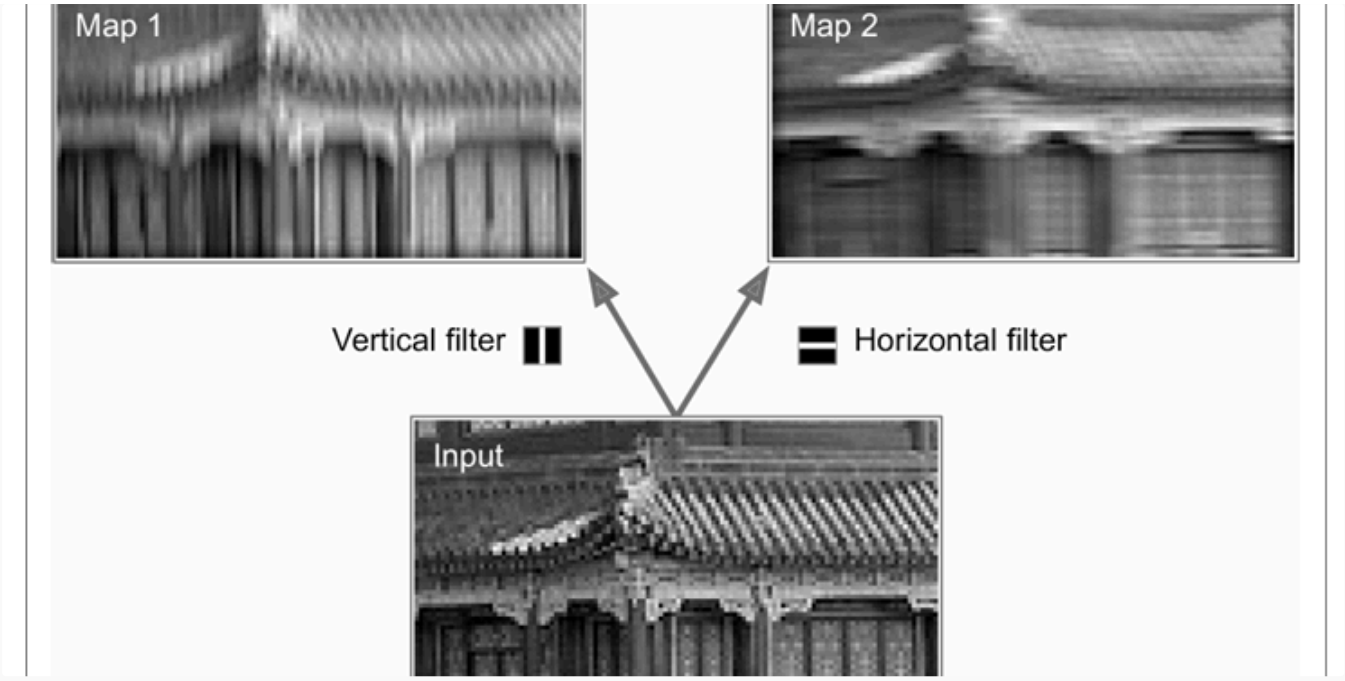
10 stories · 2173 saves

### data science and AI

40 stories · 322 saves


### Staff picks

804 stories · 1587 saves




Sanjay Dutta

## Exploring Filters and Feature Maps in Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have emerged as a powerful tool in the field of computer vision, excelling in tasks like image...

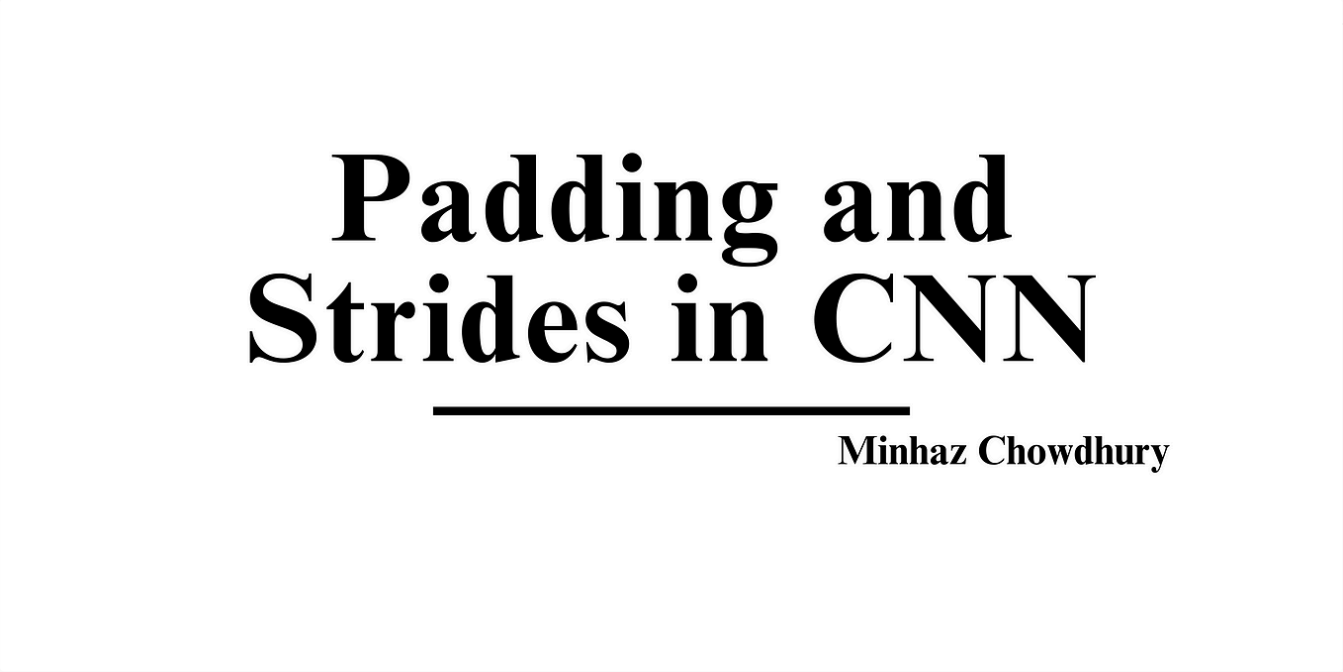
Oct 7, 2024 11

In Self Study Notes by Cevher Dogan

## What is CNN (Convolutional Neural Networks)?

A Convolutional Neural Network (CNN) is a type of deep learning algorithm specifically designed for processing structured data like images...

Nov 30, 2024 10

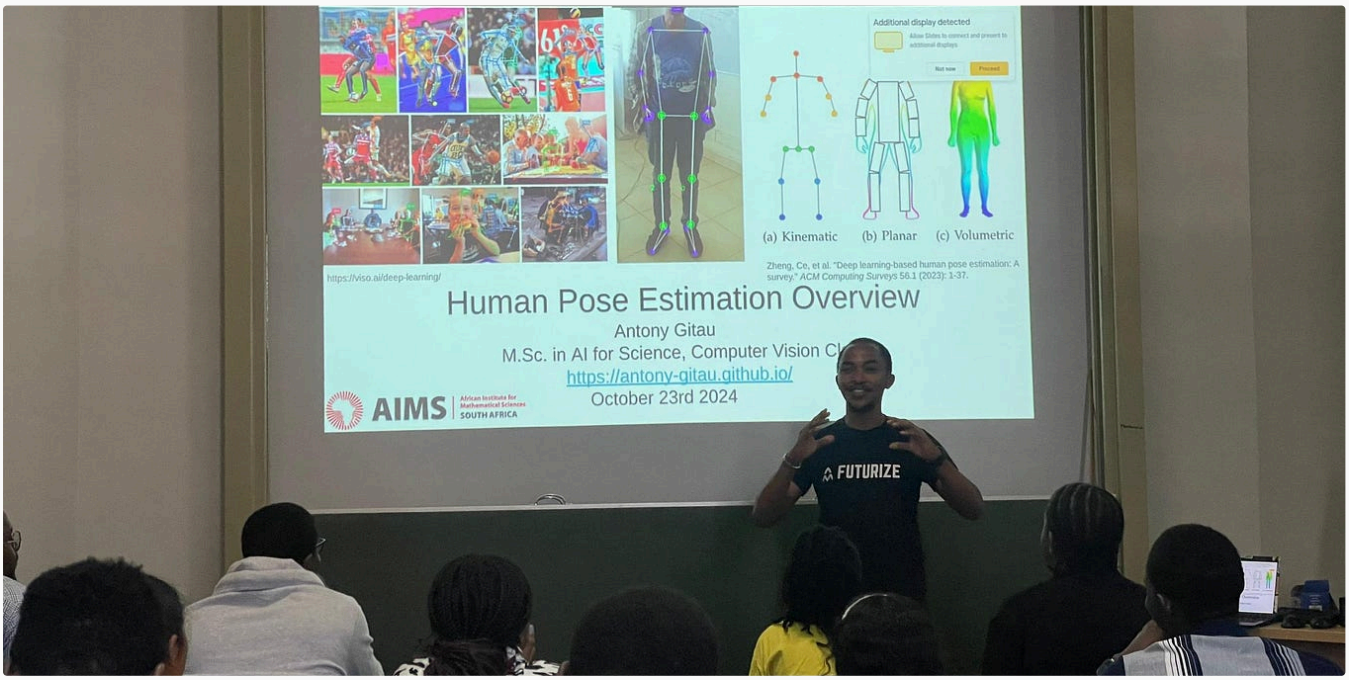



Minhaz Chowdhury

## Padding and Strides in CNN

In Convolutional Neural Networks (CNNs), padding and strides are important concepts that determine how the convolution operation is applied...

Aug 14, 2024 11



Antony M. Gitau

## Human Pose Estimation Overview

A presentation I made in Computer Vision Class at the African Institute for Mathematical Sciences (AIMS) South Africa

Oct 23, 2024 13