

MODULE 7 –Spatial Filter: Low Pass Filter, High Pass Filter, Point Detection, Line Detection, Edge Detection

A. PURPOSE

- Students are able to understand the concept of Spatial Filters
- Students are able to distinguish types of filter Spatial
- Students are able to create a simple image spatial filter using 2D convolution .

B. TOOLS AND MATERIALS

1. PC/LAPTOP
2. Github
3. Google Colaborator

C. THEORY REVIEW

C.1. Filter Process

In general , the process of filtering an image is a mechanism that can change optical , electronic or digital signals according to certain criteria. Filtering is a way to extract certain part of a set of data, by eliminating the parts data is not desirable. The aim of the performed process filter on an image is making the image become more visible, or seem more obvious to be analyze. In digital image processing, filters are used to:

1. Hold high frequency on imagea, smoothening images(smoothing)
2. Hold low frequency on images, clarifies or detects edges in the image.

Filter in image processing can be done in the frequency domain and spatial domain .

- a. **Frequency Domain.** Filtering the image in the frequency scopes will change the image into frequency domain, multiplying by a function of the frequency filter, transform back the results to spatially space scope. The function of the filter works to reduce (attenuates) the frequency specific and amplify the frequency of the other , or can also be interpreted to pass (receive) component frequency specific and eliminate (reject) the other frequency. For instance lowpass filter function means to pass components of low frequencies. Low Pass Filter generates value 1 for a frequency less than the threshold and 0 to the other so that the obtained image blurred (soft/smooth).
- b. **Spatial Domain.** Spatial domain filtering is the process of manipulating a collection of pixels from an image to produce a new image. Filtering domain spatial is one of the methods that are used in many fields for a variety of applications, especially for the improvement of the quality of the image and fix image. Filtering the image of the space scope of spatial are doing convolution of image input $f(i,j)$ with the function of the filter $h(i,j)$, where the function of the filter that used to be simulated in the form of kernel discrete certain.

$$g(i,j) = h(i,j) \odot f(i,j)$$

C.2. Convolution Process

Convolution is the sum of the multiplication of each point in the kernel by each point in the input function. Kernel operate as a shift in the function of the input image input $f(x)$. Total multiplication of each point on the function that is the result of convolution that is expressed by $g(x)$. The kernel uses the concept of neighboring pixels, where the kernel matrix is made with the assumption that the value of a pixel can be influenced by their neighbor pixels. Neighbor pixel is the number of pixels which is adjacent to the direct (adjacent) with a pixel center. Illustration of neighboring pixels can be seen in Figure 1.

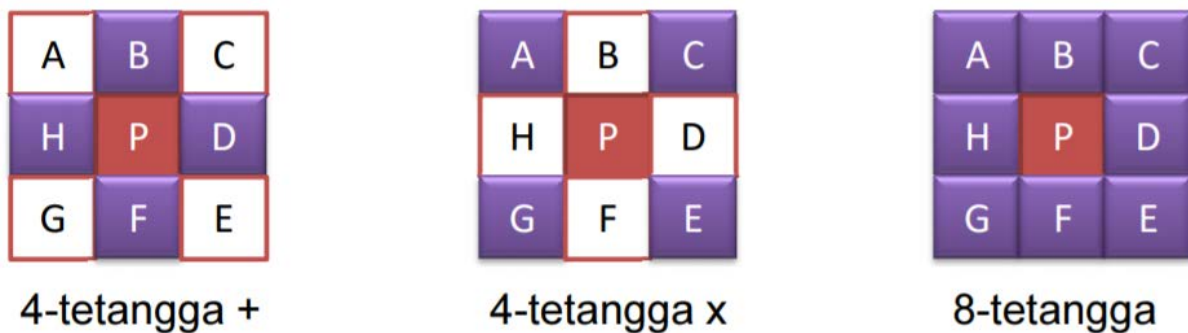


Figure 1. Illustration of neighboring pixels of pixel P

In digital image processing, convolution is carried out in two dimensions in an image, as shown by the equation:

$$g(x, y) = f(x, y) \odot h(x, y) = \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} f(a, b) g(x - a, y - b)$$

The illustration of the convolution process from the above equation is shown in Figure 2.

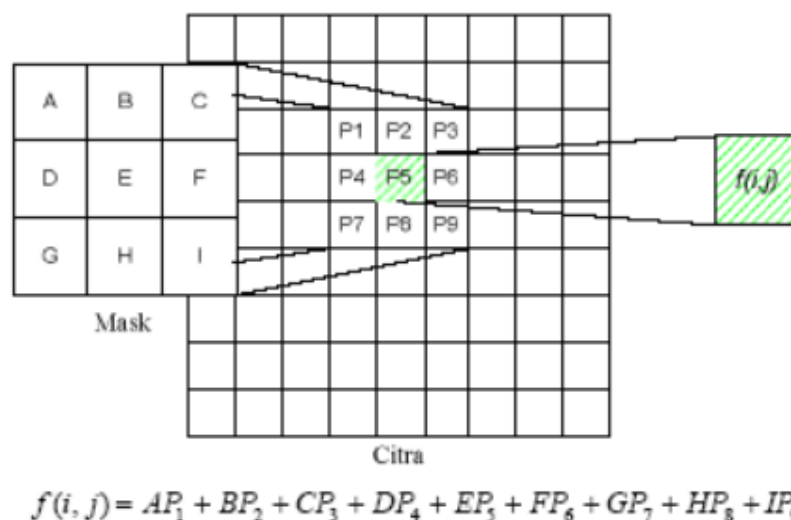


Figure 2. Illustration of the convolutional process

For example, there is an input image matrix and a kernel matrix as follows :

| | | | |
|---|---|---|---|
| 5 | 5 | 6 | 6 |
| 5 | 4 | 4 | 7 |
| 0 | 0 | 2 | 2 |
| 0 | 1 | 1 | 3 |

(a) Citra input

| | | |
|---------------|---------------|---------------|
| $\frac{1}{4}$ | $\frac{1}{4}$ | $\frac{1}{4}$ |
| $\frac{1}{4}$ | $\frac{1}{4}$ | $\frac{1}{4}$ |
| $\frac{1}{4}$ | $\frac{1}{4}$ | $\frac{1}{4}$ |

(b) Kernel

So the results of the calculation of convolution area are :

$$g(2,2) = 5 \times \frac{1}{4} + 5 \times \frac{1}{4} + 6 \times \frac{1}{4} + 5 \times \frac{1}{4} + 4 \times \frac{1}{4} + 4 \times \frac{1}{4} + 0 \times \frac{1}{4} + 0 \times \frac{1}{4} + 2 \times \frac{1}{4}$$

$$= 1,25 + 1,25 + 1,5 + 1,25 + 1 + 1 + 0 + 0 + 0,5 = 7,75$$

Convolution is carried out until all the pixels of the input image are subjected to the convolution calculation . Here is an example of the stages of the process of convolution in a matrix image of size 5x5 with a kernel size of 3x3.

$$f(x,y) = \begin{bmatrix} 4 & 4 & 3 & 5 & 4 \\ 6 & 6 & 5 & 5 & 2 \\ 5 & 6 & 6 & 6 & 2 \\ 6 & 7 & 5 & 5 & 3 \\ 3 & 5 & 2 & 4 & 4 \end{bmatrix} \quad g(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & * & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

note : mark * indicates the position (0.0) of the kernel

The convolution operation $f(x,y) \odot h(x,y)$ are as follows:

1. Place the kernel in the upper left corner , then calculate the pixel value at the position (0,0) of the kernel.

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 3 | 5 | 4 |
| 6 | 6 | 5 | 5 | 2 |
| 5 | 6 | 6 | 6 | 2 |
| 6 | 7 | 5 | 5 | 3 |
| 3 | 5 | 2 | 4 | 4 |

→

| | | | | |
|--|---|--|--|--|
| | | | | |
| | 3 | | | |
| | | | | |
| | | | | |
| | | | | |

Hasil konvolusi = 3. Nilai ini dihitung dengan cara berikut:

$$(0 \times 4) + (-1 \times 4) + (0 \times 3) + (-1 \times 6) + (4 \times 6) + (-1 \times 5) + (0 \times 5) + (-1 \times 6) + (0 \times 6) = 3$$

2. Move the kernel one pixel to the right , then calculate the pixel value at the position (0,0) of the kernel

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 3 | 5 | 4 |
| 6 | 6 | 5 | 5 | 2 |
| 5 | 6 | 6 | 6 | 2 |
| 6 | 7 | 5 | 5 | 3 |
| 3 | 5 | 2 | 4 | 4 |

→

| | | | | |
|--|---|---|--|--|
| | | | | |
| | 3 | 0 | | |
| | | | | |
| | | | | |
| | | | | |

Hasil konvolusi = 0. Nilai ini dihitung dengan dengan cara berikut:

$$(0 \times 4) + (-1 \times 3) + (0 \times 5) + (-1 \times 6) + (4 \times 5) + (-1 \times 5) + (0 \times 6) + (-1 \times 6) + (0 \times 6) = 0$$

3. Move the kernel one pixel to the right , then calculate the pixel value at the position (0,0) of the kernel

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 3 | 5 | 4 |
| 6 | 6 | 5 | 5 | 2 |
| 5 | 6 | 6 | 6 | 2 |
| 6 | 7 | 5 | 5 | 3 |
| 3 | 5 | 2 | 4 | 4 |

→

| | | | | |
|--|---|---|---|--|
| | | | | |
| | 3 | 0 | 2 | |
| | | | | |
| | | | | |
| | | | | |

Hasil konvolusi = 2. Nilai ini dihitung dengan cara berikut:

$$(0 \times 3) + (-1 \times 5) + (0 \times 4) + (-1 \times 5) + (4 \times 5) + (-1 \times 2) + (0 \times 6) + (-1 \times 6) + (0 \times 2) = 2$$

4. Furthermore shift kernel one pixel to the bottom, then began to perform the convolution of the left image . Every convolution process, slide kernel one pixel to the right and calculate the value of the pixel at position (0,0) of the kernel .

(i)

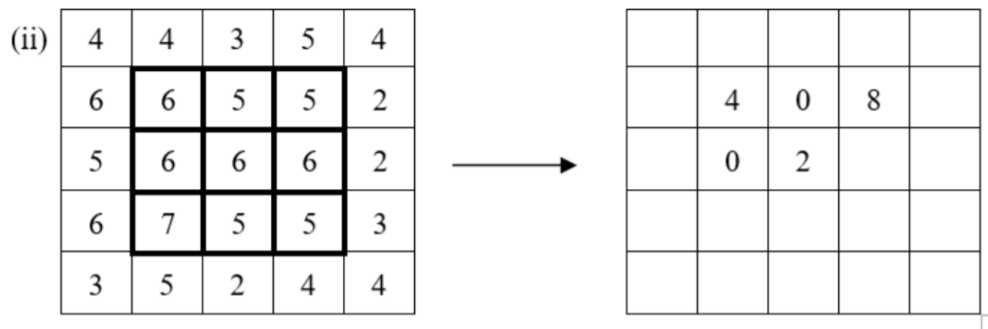
| | | | | |
|---|---|---|---|---|
| 4 | 4 | 3 | 5 | 4 |
| 6 | 6 | 5 | 5 | 2 |
| 5 | 6 | 6 | 6 | 2 |
| 6 | 7 | 5 | 5 | 3 |
| 3 | 5 | 2 | 4 | 4 |

→

| | | | | |
|--|---|---|---|--|
| | | | | |
| | 3 | 0 | 2 | |
| | 0 | | | |
| | | | | |
| | | | | |

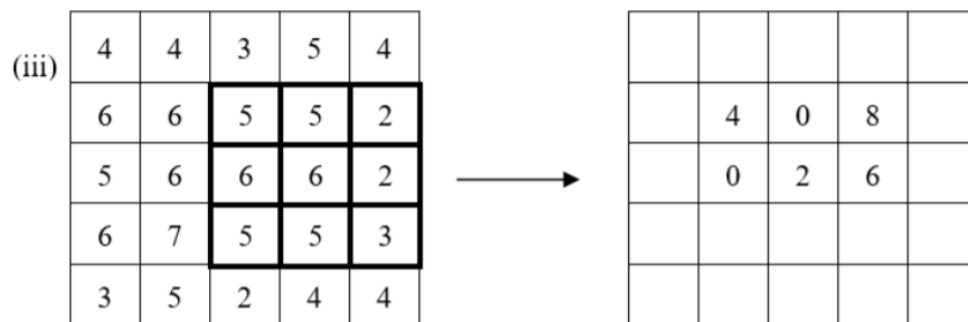
Hasil konvolusi = 0. Nilai ini dihitung dengan cara berikut:

$$(0 \times 6) + (-1 \times 6) + (0 \times 5) + (-1 \times 5) + (4 \times 6) + (-1 \times 6) + (0 \times 6) + (-1 \times 7) + (0 \times 5) = 0$$



Hasil konvolusi = 2. Nilai ini dihitung dengan cara berikut:

$$(0 \times 6) + (-1 \times 5) + (0 \times 5) + (-1 \times 6) + (4 \times 6) + (-1 \times 6) + (0 \times 7) + (-1 \times 5) + (0 \times 5) = 2$$



Hasil konvolusi = 6. Nilai ini dihitung dengan cara berikut:

$$(0 \times 5) + (-1 \times 5) + (0 \times 2) + (-1 \times 6) + (4 \times 6) + (-1 \times 2) + (0 \times 5) + (-1 \times 5) + (0 \times 3) = 6$$

5. By the same way as in the above, then the pixels in row three convolve to produce :

| | | | | |
|--|---|---|---|--|
| | | | | |
| | 4 | 0 | 8 | |
| | 0 | 2 | 6 | |
| | 6 | 0 | 2 | |
| | | | | |

Problems arise when the pixel convolve is in the edge of the image (border), because some of the convolution coefficients is not able to be positioned on these image pixels (the "hang" effect) as shown in the Figure 3. "hang" problems like this always happens in the left, right, top, and bottom of image pixels edges.

| | | | | | |
|---|---|---|---|---|---|
| 4 | 4 | 3 | 5 | 4 | ? |
| 6 | 6 | 5 | 5 | 2 | ? |
| 5 | 6 | 6 | 6 | 2 | ? |
| 6 | 7 | 5 | 5 | 3 | |
| 3 | 5 | 2 | 4 | 4 | |

Figure 3. An example of the hang effect on a convolution process

The solution to a problem early this are :

1. Edge pixels are ignored, do not convolve. This solution is widely used in the usages of image processing functions. Using this alternative, all the edge of the pixels value remains the same as the image of origin , such as the example in Figure 4.

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 3 | 5 | 4 |
| 6 | 4 | 0 | 8 | 2 |
| 5 | 0 | 2 | 6 | 2 |
| 6 | 6 | 0 | 2 | 3 |
| 3 | 5 | 2 | 4 | 4 |

Gambar 4. Edge pixels (un-shaded) are not convolve

2. Duplicate element of the image. For example, elements of the first column copied into column $M + 1$, also on the contrary , the convolution can be performed on the edge as illustrated in Figure 5.

| | | | | | |
|---|---|---|---|---|---|
| 5 | 5 | 5 | | | |
| 5 | 5 | 5 | 5 | 6 | 6 |
| 5 | 5 | 4 | 4 | 7 | |
| | 0 | 0 | 2 | 2 | |
| | 0 | 1 | 1 | 3 | |

Figure 5. Illustration of the pixel value duplication process

3. Elements that are marked with a "?" In Figure 3 , it is assumed that the value is 0 or some other constant , so that the edge pixel convolution can be carried out , as illustrated in Figure 6.

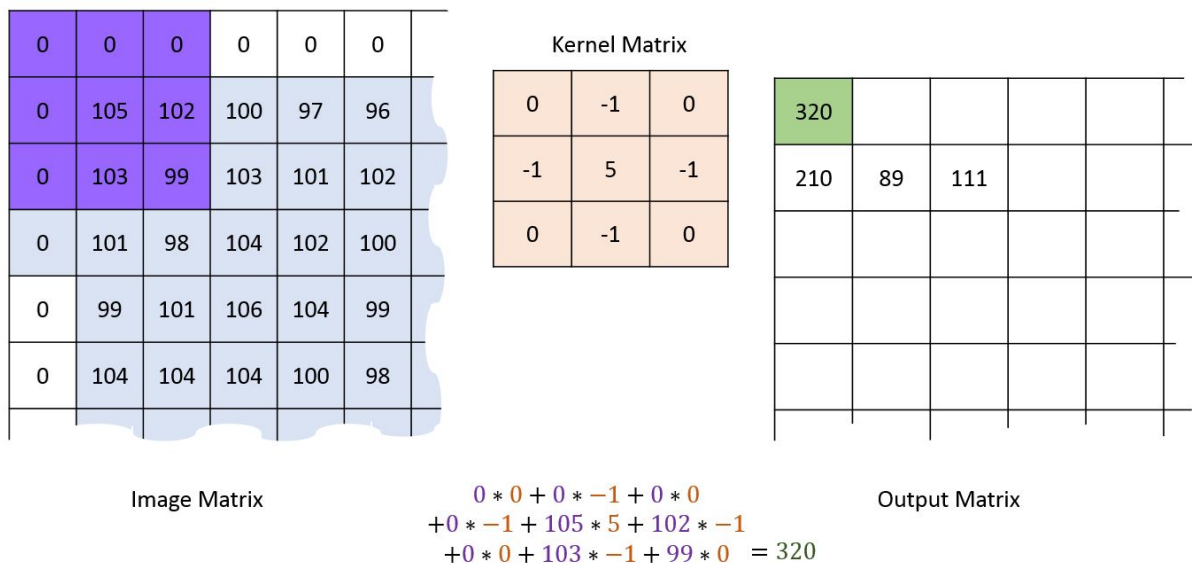


Figure 6. Illustration of giving the value 0 to the input image

Solutions with three approaches in the above assumes part of the edge of the image width is very small (only one pixel) relative compared denagn size of the image , so the pixels on the edge does not show the effect from our naked eyes.

The following is a convolutional process algorithm

```

For x = 0 to image_width-1
  For y = 0 image_height-1
    z(x,y) = 0
    For k1 = 0 to filter_width-1
      For k2 = 0 to filter_height-1
        z(x,y) = z(x,y) + H(k1,k2)*Img_pixel(x+k1, y+k2)
      next k2
    next k1
  next y
next x
    
```

The pseudocode for the convolution process is as follows :

```
void konvolusi(citra Image, citra ImageResult, imatriks Mask, int N, int M)
/* Mengkonvolusi citra Image yang berukuran N x M dengan mask 3 x 3. Hasil
konvolusi disimpan di dalam matriks ImageResult.
*/
{ int i, j;

  for (i=1; i<=N-3; i++)
    for(j=1; j<=M-3; j++)
      ImageResult[i][j]=
        Image[i-1][j-1]*Mask[0][0] +
        Image[i-1][j+1]*Mask[0][1] +
        Image[i-1][j]*Mask[0][2] +
        Image[i][j-1]*Mask[1][0] +
        Image[i][j]*Mask[1][1] +
        Image[i][j+1]*Mask[1][2] +
        Image[i+1][j-1]*Mask[2][0] +
        Image[i+1][j]*Mask[2][1] +
        Image[i+1][j+1]*Mask[2][2];
}
```

The convoluted pixels are elements (i, j). Eight pieces of pixels adjacent to the pixel (i, j) is as follows :

| | | |
|------------|----------|------------|
| $i-1, j-1$ | $i-1, j$ | $i-1, j+1$ |
| $i, j-1$ | i, j | $i, j+1$ |
| $i+1, j-1$ | $i+1, j$ | $i+1, j+1$ |

C.3. Spatial Filter Type

The results of the filter spatially determined by the element matrix of the kernel, and can produce effects that vary depending on the image Input. Some of the types of spatial filters are :

- Sharpening (sharpening)
- Blurring (blurring)
- Emboss
- Edge detection (detection edge)

C.3.1. Sharpening

Kernel for sharpening / sharpening the image using the principle that the intensity of the pixels centers should be strengthened in the direction that is opposite to its neighbors . Example of a kernel for image enhancement :

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

courtesy

<http://docs.gimp.org/en/plugin-in-convmatrix.html>

| | | |
|---|----|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

courtesy

<http://www.imagesincontext.com/IICFeatures/convolution-filter.htm>

While examples of sharpening results can be seen in Figure 7.

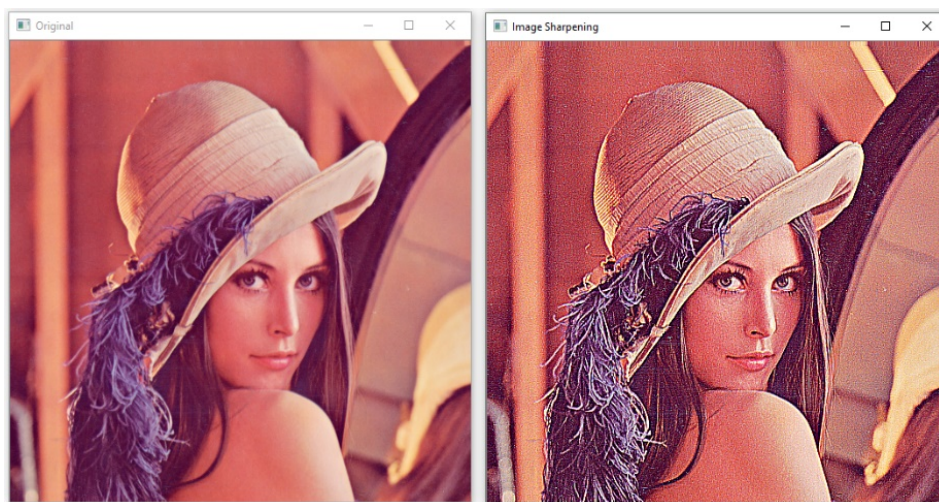


Figure 7. Example of a Sharpening Filter

C.3.2. Blurring

The kernel for blurring / blurring uses the principle that the central pixel value should be made closer to neighboring pixels (reducing discrepancies). An example of a kernel for blurring images, one can use Gaussian Blurring:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{or} \quad \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

While examples of blurring results can be seen in Figure 9 .

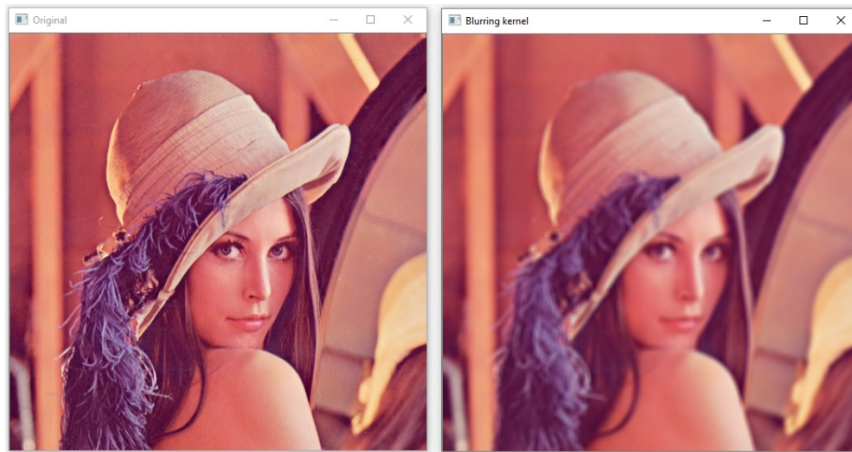


Figure 9 . Example of a Gaussian Blurring Filter

C.3.3. Emboss

The kernel for embossing is based on the principle of reinforcing edges in one particular direction , without losing any other color. The direction of strengthening is indicated by the change from negative to positive elements . Example kernel to emboss:

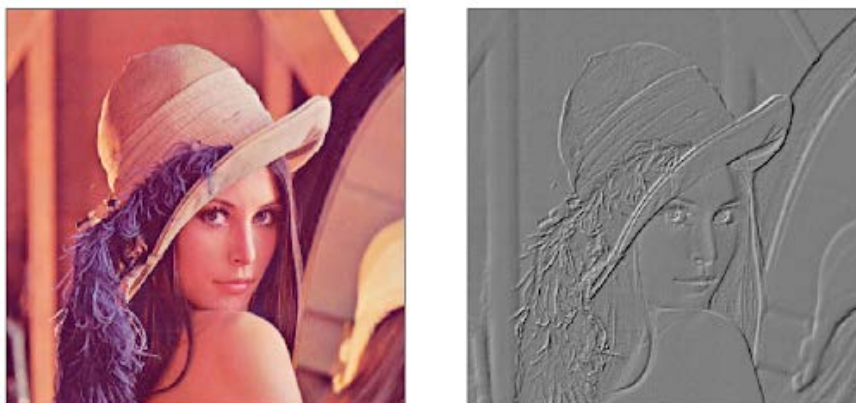
| | | |
|----|----|---|
| -2 | -1 | 0 |
| -1 | 1 | 1 |
| 0 | 1 | 2 |

courtesy
<http://docs.gimp.org/en/plugin-convmatrix.html>

| | | |
|---|---|----|
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | -1 |

courtesy
<http://www.imagesincontext.com/IICFeatures/convolution-filter.htm>

While an example of the embossed result can be seen in Figure 10.



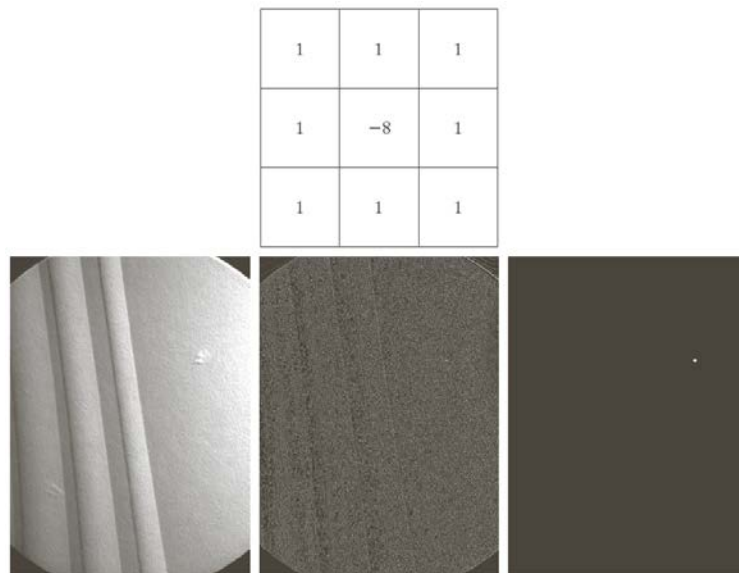
Here 's a few examples of other kernel filter base:

- *Average Filter 3x3* = $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

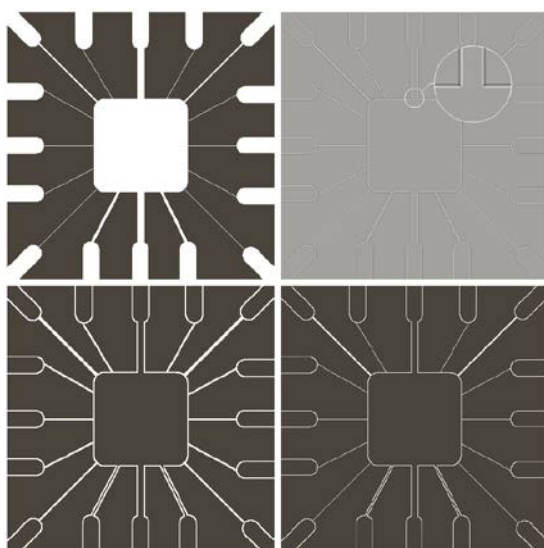
- *Low Pass Filter* = $\frac{1}{12} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

- *High Pass Filter* = $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 3 \\ -3 & 0 & 1 \end{bmatrix}$

- **Point Detection**



- **Line Detection**



| | | | | | |
|----|----|----|----|----|----|
| -1 | -1 | -1 | 2 | -1 | -1 |
| 2 | 2 | 2 | -1 | 2 | -1 |
| -1 | -1 | -1 | -1 | -1 | 2 |

Horizontal

+45°

| | | | | | |
|----|---|----|----|----|----|
| -1 | 2 | -1 | -1 | -1 | 2 |
| -1 | 2 | -1 | -1 | 2 | -1 |
| -1 | 2 | -1 | 2 | -1 | -1 |

Vertical

-45°

- **Edge Detection**

| | | |
|-------|-------|-------|
| z_1 | z_2 | z_3 |
| z_4 | z_5 | z_6 |
| z_7 | z_8 | z_9 |

| | | | |
|----|---|---|----|
| -1 | 0 | 0 | -1 |
| 0 | 1 | 1 | 0 |

Roberts

| | | | | | |
|----|----|----|----|---|---|
| -1 | -1 | -1 | -1 | 0 | 1 |
| 0 | 0 | 0 | -1 | 0 | 1 |
| 1 | 1 | 1 | -1 | 0 | 1 |

Prewitt

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

Sobel

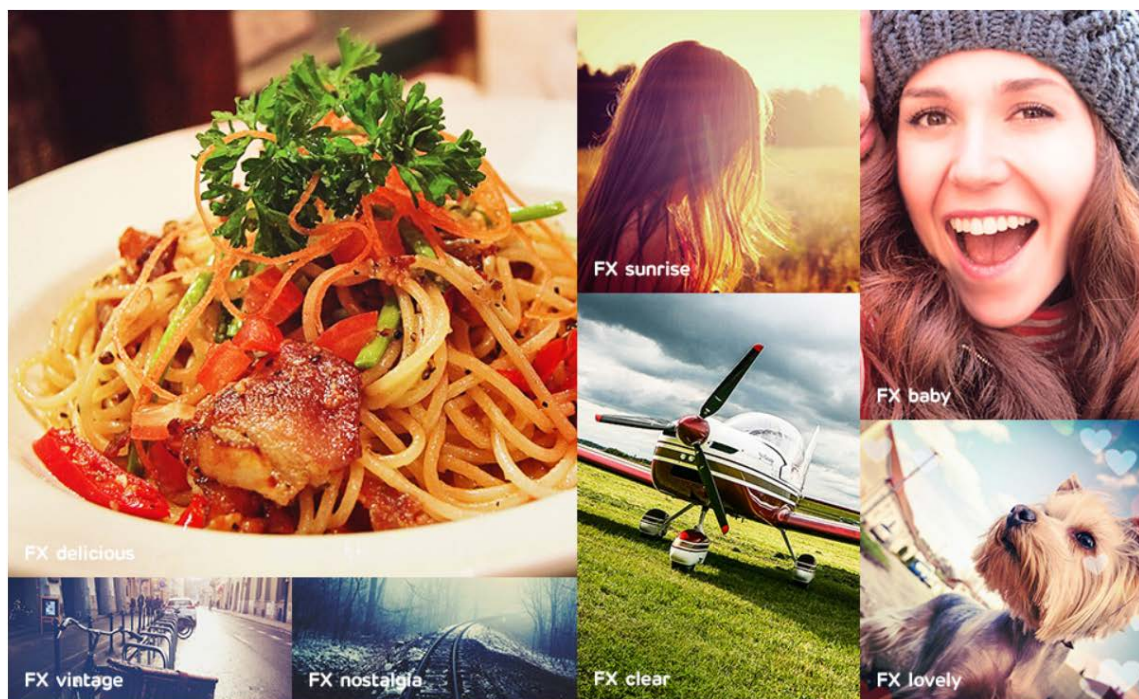
| | |
|---|---|
| a | b |
| c | d |

FIGURE 10.16

(a) Original image of size 834×1114 pixels, with intensity values scaled to the range $[0, 1]$.
 (b) $|g_x|$, the component of the gradient in the x -direction, obtained using the Sobel mask in Fig. 10.14(f) to filter the image.
 (c) $|g_y|$, obtained using the mask in Fig. 10.14(g).
 (d) The gradient image, $|g_x| + |g_y|$.

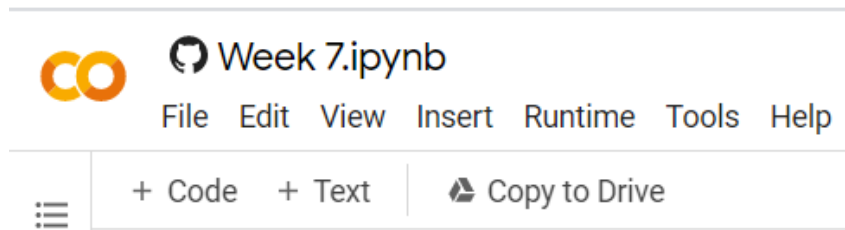


When this almost all application processing image have filter specifically each of which is very attractive to users . Figure below is an example of a filter that is owned by mobile Line Camera Application.



D. PRAKTIKUM FILTER

1. Make convolution function using algorithms that have been described in Part C, without using a library or method of convolution of OpenCV.
2. The following are steps that can be taken :
 - a. Create a notebook just on google colab, and give the name Week7.ipynb. Save a copy on the github account as in the previous module .



- b. Access the files contained on the drive and import some of the libraries that are needed

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[ ] import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
import math
from google.colab.patches import cv2_imshow
from PIL import Image as im
```

- c. Create a convolutional function . Note : the parameters used may be modified . For example , only uses parameters and kernel image only , or image, kernel, and padding.

▼ Konvolusi tanpa Library

Membuat fungsi konvolusi

Fungsi konvolusi yang dibuat memiliki parameter berupa:

1. citra masukan,
2. kernel berupa matriks untuk memfilter citra,
3. nilai stride / besarnya pergeseran untuk setiap konvolusi,
4. nilai pad yang akan ditambahkan pada citra

```
[ ] def convolution2d(image, kernel, stride, padding):
```

- d. Load the image to be processed and turn it into a gray image

Load Image yang akan diproses

```
[ ] img = cv.imread('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/mandrill.tiff')
img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
```

- e. Determine the kernel that will be used , for example, the kernel to filter sharpening as follows :

Menentukan kernel yang akan digunakan

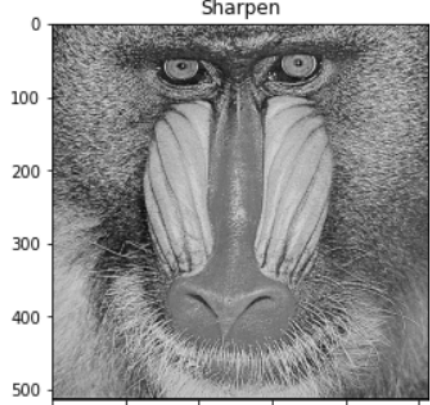
```
#image sharpen
kernel_sharpen = np.array([[0,-1,0],
                           [-1,5,-1],
                           [0,-1,0]])
```

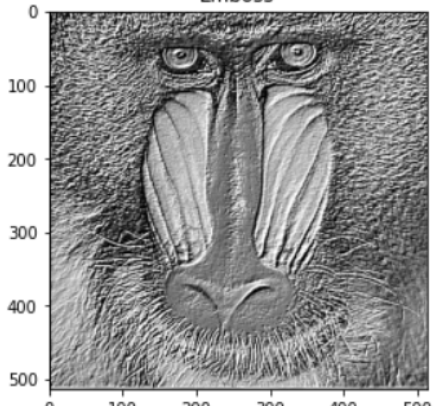
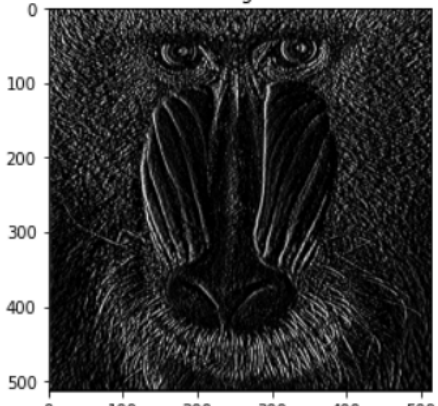
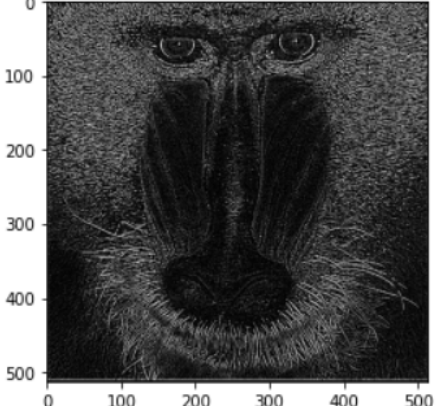
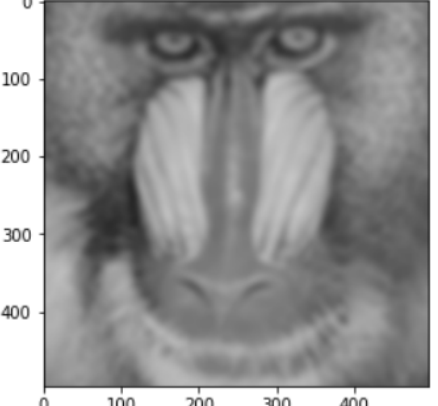
- f. Calling a function convolution that has been created previously , and displays the results konvolusinya :

Memanggil fungsi konvolusi dan menerapkan setiap filter yang telah ditentukan

```
convolution2d(img_gray,kernel_sharpen,1,2)
```

3. Create Image Filter for Average filter, low pass filter, high pass filter, and some of the following filters :

| Operasi | Kernel | Hasil |
|---------|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ | <p>Sharpen</p>  |

| | | |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <p>Emboss</p> | $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ | <p>Emboss</p>  |
| <p>Left Sobel Edge Detection</p> | $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ | <p>Left Sobel Edge Detection</p>  |
| <p>Canny Edge Detection</p> | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ | <p>Canny Edge Detection</p>  |
| <p>21x21 Gaussian Blur</p> | <p>To generate a Gaussian kernel, you can use the following code :</p> <pre>sigma=math.sqrt(kernel_size) gaussian_kernel = cv.getGaussianKernel(kernel_size , sigma) gauss_kernel = gaussian_kernel @ gaussian_kernel.transpose()</pre> | <p>21x21 Gaussian Blur</p>  |