MODUL 7 – Filter Spasial Low Pass Filter, High Pass Filter, Point Detection, Line Detection, Edge Detection

A. TUJUAN

- Mahasiswa mampu memahami konsep Filter Spasial
- Mahasiswa dapat mengetahui beberapa jenis Filter Spasial
- Mahasiswa dapat membuat filter sederhana menggunakan filter Kernel yang tersedia dan melakukan perhitungan konvolusi.

B. ALAT DAN BAHAN

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

C. ULASAN TEORI

C.1. Proses Filter

Secara umum, proses filter sebuah citra adalah suatu mekanisme yang dapat mengubah sinyal-sinyal optis, elektronis ataupun digital, sesuai dengan kriteria tertentu. Pemfilteran merupakan cara untuk mengekstraksi bagian tertentu dari suatu himpunan data, dengan menghilangkan bagian-bagian data yang tidak diinginkan. Tujuan dari dilakukan proses filter pada sebuah citra adalah membuat citra menjadi tampak lebih baik, atau tampak lebih jelas untuk dianalisis. Pada pengolahan citra digital, filter digunakan untuk:

1. Menekan frekuensi tinggi pada citra, memperhalus citra (smoothing)
2. Menekan frekuensi rendah seperti, memperjelas atau mendeteksi tepi pada citra.

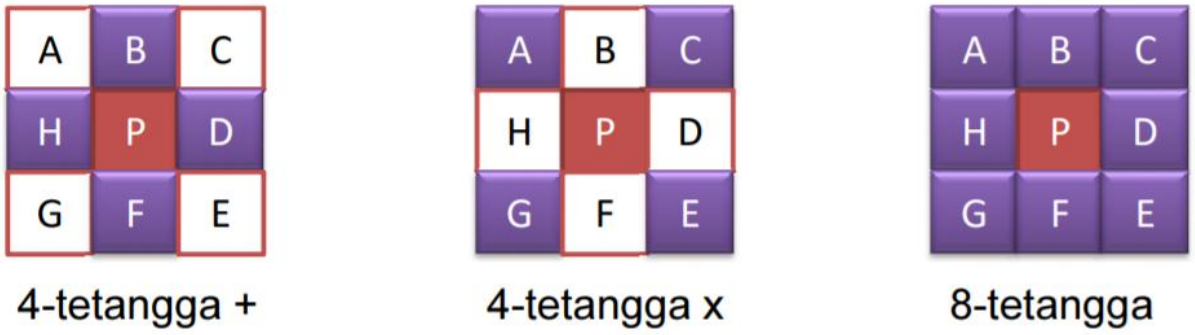
Filter pada pengolahan citra dapat dilakukan pada domain frekuensi dan domain spasial.

a. **Domain frekuensi**. Pemfilteran citra pada ruang lingkup frekuensi akan mengubah citra ke domain frekuensi, mengalikannya dengan sebuah fungsi filter frekuensi, kemudian mentransformasikan kembali hasilnya ke ruang lingkup spasial. Fungsi filter bekerja untuk mengurangi (attenuates) frekuensi tertentu dan memperkuat frekuensi lainnya, atau dapat juga diartikan meloloskan (menerima) komponen frekuensi tertentu dan menghilangkan (menolak) frekuensi yang lain. Misalnya fungsi lowpass filter berarti meloloskan komponen frekuensi yang rendah. Low Pass Filter menghasilkan nilai 1 untuk frekuensi kurang dari ambang (*threshold*) dan 0 untuk lainnya sehingga didapatkan citra blur (lembut/halus).
b. **Domain spasial**. Pemfilteran domain spasial adalah proses manipulasi kumpulan piksel dari sebuah citra untuk menghasilkan citra baru. Pemfilteran domain spasial merupakan salah satu metode yang digunakan dalam banyak bidang untuk berbagai aplikasi, khususnya untuk peningkatan kualitas citra dan perbaikan citra. Pemfilteran citra pada ruang lingkup spasial adalah melakukan konvolusi citra input *f(i,j)* dengan fungsi filter *h(i,j)*, dimana fungsi filter yang digunakan harus disimulasikan dalam bentuk kernel diskret tertentu.

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

C.2. Proses Konvolusi

Konvolusi merupakan penjumlahan dari perkalian setiap titik pada kernel dengan setiap titik pada fungsi masukan. Kernel dioperasikan secara bergeser pada fungsi masukan/citra input f(x). Jumlah perkalian setiap titik pada kedua fungsi tersebut merupakan hasil konvolusi yang dinyatakan dengan g(x). Kernel menggunakan konsep piksel tetangga (*neighbouring pixels*), dimana matriks kernel dibuat dengan asumsi bahwa nilai sebuah piksel bisa dipengaruhi oleh piksel-piksel tetangganya. Piksel tetangga adalah sejumlah piksel yang bersebelahan langsung (adjacent) dengan sebuah piksel pusat. Ilustrasi dari piksel tetangga dapat dilihat pada gambar 1.

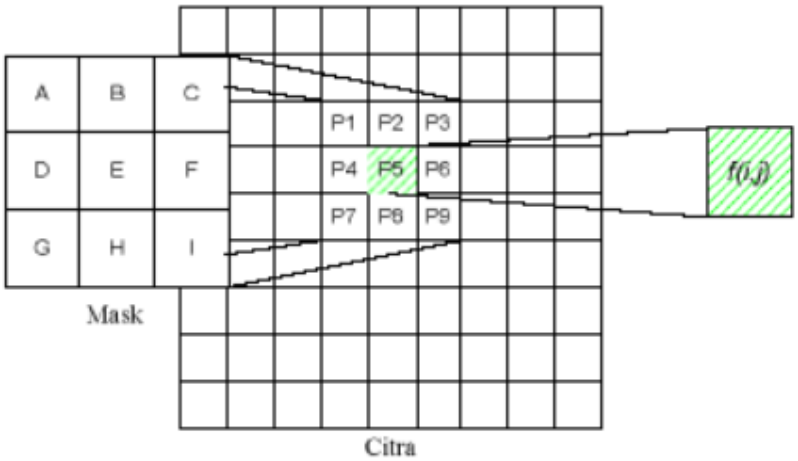


Gambar 1. Ilustrasi piksel tetangga dari piksel P

Pada pengolahan citra digital, konvolusi dilakukan secara dua dimensi pada sebuah citra, seperti ditunjukkan oleh persamaan:

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

Ilustrasi proses konvolusi dari Persamaan di atas ditunjukkan pada gambar2.



$$f(i,j) = AP_1 + BP_2 + CP_3 + DP_4 + EP_5 + FP_6 + GP_7 + HP_8 + IP_9$$

Gambar 2. Ilustrasi proses konvolusi

Misal terdapat matriks citra input dan matriks kernel sebagai berikut:

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

(a) Citra input

¼	¼	¼
¼	¼	¼
¼	¼	¼

(b) Kernel

Maka hasil perhitungan konvolusi daerah di atas adalah:

g(2,2) = 5×¼ + 5×¼ + 6×¼ + 5×¼ + 4×¼ + 4×¼ + 0×¼ + 0×¼ + 2×¼

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

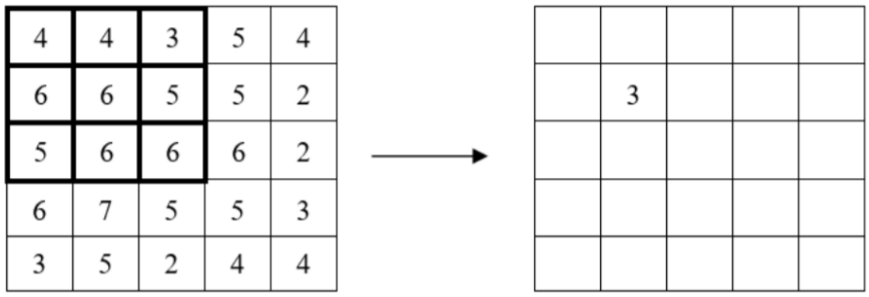
Konvolusi dilakukan sampai semua piksel citra input terkena perhitungan konvolusi. Berikut ini merupakan contoh tahapan proses konvolusi pada sebuah matriks citra berukuran 5x5 dengan sebuah kernel berukuran 3x3.

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

catatan: tanda * menunjukkan posisi (0,0) dari kernel

Operasi konvolusi $f(x,y) \odot h(x,y)$ adalah sebagai berikut:

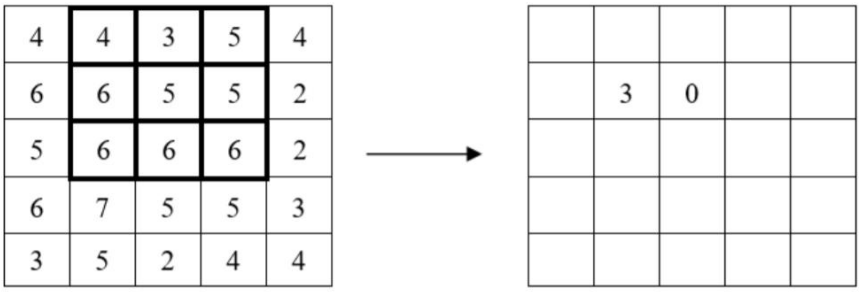
1. Tempatkan kernel pada sudut kiri atas, kemudian hitung nilai piksel pada posisi (0,0) dari kernel.



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

(0 × 4) + (-1 × 4) + (0 × 3) + (-1 × 6) + (4 × 6) + (-1 × 5) + (0 × 5) + (-1 × 6) + (0 × 6) = 3

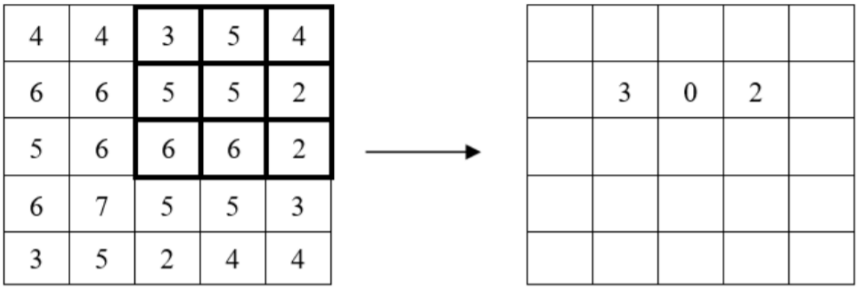
2. Geser kernel satu piksel ke kanan, kemudian hitung nilai piksel pada posisi (0,0) dari kernel



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

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

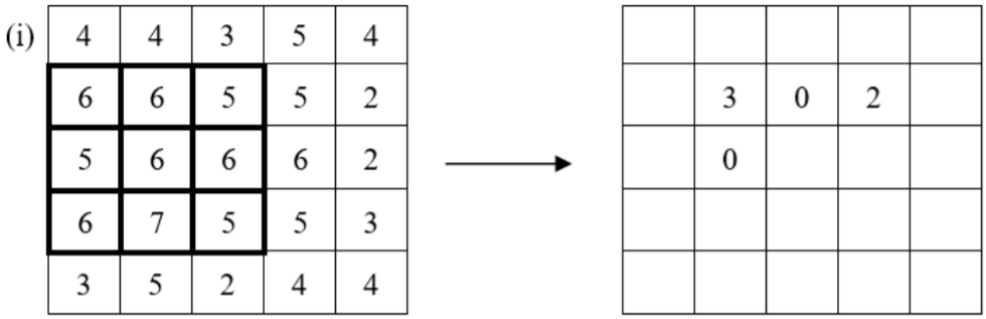
3. Geser kernel satu piksel ke kanan, kemudian hitung nilai piksel pada posisi (0,0) dari kernel



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. Selanjutnya geser kernel satu piksel ke bawah, lalu mulai lagi melakukan konvolusi dari sisi kiri citra. Setiap kali konvolusi, geser kernel satu piksel ke kanan dan hitung nilai piksel pada posisi (0,0) dari kernel.



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

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

(ii)

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

→

	4	0	8	
	0	2		

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

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

(iii)

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

→

	4	0	8	
	0	2	6	

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. Dengan cara yang sama seperti di atas, maka piksel-piksel baris ketiga dikonvolusi sehingga menghasilkan:

	4	0	8	
	0	2	6	
	6	0	2	

Masalah timbul bila piksel yang dikonvolusi adalah piksel pinggir (*border*), karena beberapa koefisien konvolusi tidak dapat dapat diposisikan pada piksel-piksel citra (efek "menggantung") seperti pada gambar 3. Masalah "menggantung" seperti ini selalu terjadi pada pixel-pixel pinggir kiri, kanan, atas, dan bawah.

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	

Gambar 3. Contoh efek menggantung pada proses konvolusi

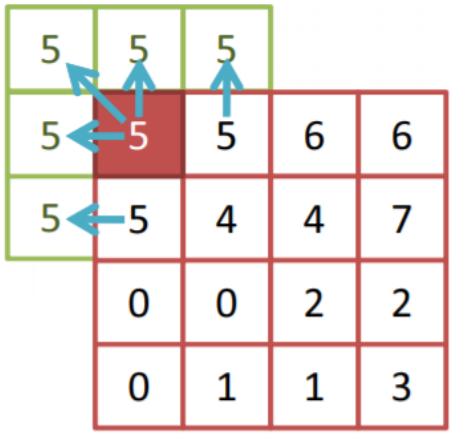
Solusi untuk permasalahan ini adalah:

1. Piksel-piksel pinggir diabaikan, tidak dikonvolusi. Solusi ini banyak dipakai di dalam pustaka fungsi-fungsi pengolahan citra. Dengan cara seperti ini, maka piksel-piksel pinggir nilainya tetap sama seperti citra asal, seperti contoh pada gambar 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

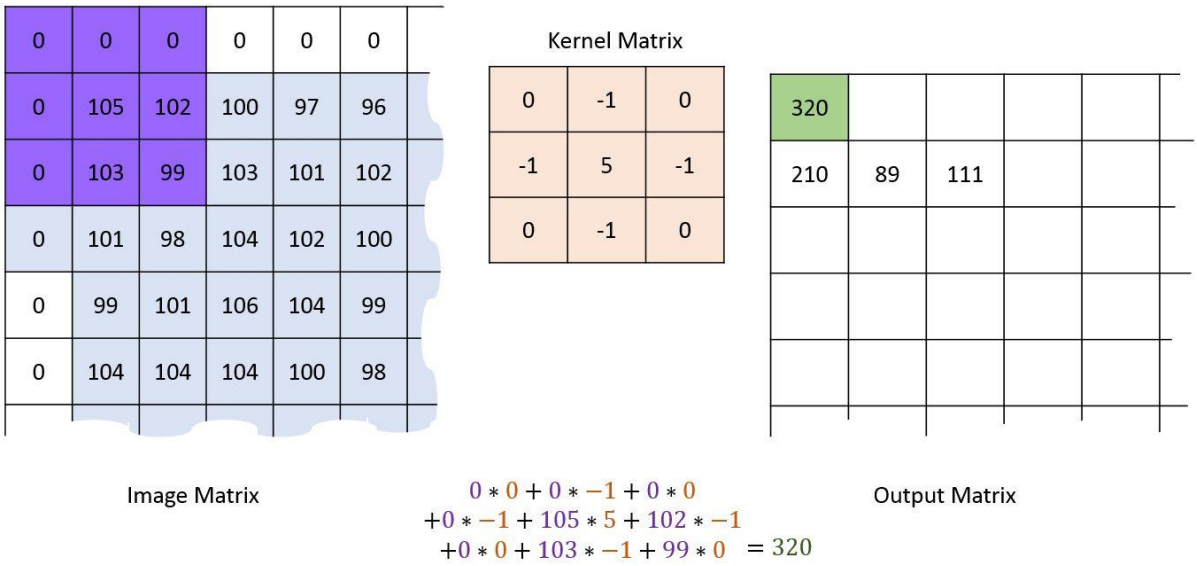
Gambar 4. Piksel-piksel pinggir (yang tidak diarsir) tidak dikonvolusi

2. Duplikasi elemen citra, misalnya elemen kolom pertama disalin ke kolom M+1, begitu juga sebaliknya, lalu konvolusi dapat dilakukan terhadap pixel-pixel pinggir tersebut, seperti ilustrasi pada gambar 5.



Gambar 5. Ilustrasi proses duplikasi nilai piksel

3. Elemen yang ditandai dengan "?" pada gambar 3 diasumsikan bernilai 0 atau konstanta yang lain, sehingga konvolusi pixel-pixel pinggir dapat dilakukan, seperti ilustrasi pada gambar 6.



Gambar 6. Ilustrasi pemberian nilai 0 pada citra input

Solusi dengan ketiga pendekatan di atas mengasumsikan bagian pinggir citra lebarnya sangat kecil (hanya satu pixel) relatif dibandingkan denagn ukuran citra, sehingga pixel-pixel pinggir tidak memperlihatkan efek yang kasat mata.

Berikut ini merupakan algoritma proses konvolusi

```
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
```

Pseudocode untuk proses konvolusi adalah sebagai berikut:

```
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];
}
```

Pixel yang dikonvolusi adalah elemen *(i, j)*. Delapan buah pixel yang bertetangga dengan pixel *(i, j)* adalah sebagai berikut:

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. Jenis Filter Spasial

Hasil dari filter spasial ditentukan oleh elemen matriks kernel, dan dapat menghasilkan efek yang berbeda-beda terhadap citra input. Beberapa jenis filter spasial adalah:

- Sharpening (penajaman)
- Blurring (pengaburan)
- Emboss
- Edge detection (deteksi tepi)

C.3.1. Sharpening

Kernel untuk sharpening/penajaman citra menggunakan prinsip bahwa intensitas piksel pusat harus lebih diperkuat pada arah yang berlawanan terhadap tetangganya. Contoh kernel untuk penajaman citra:

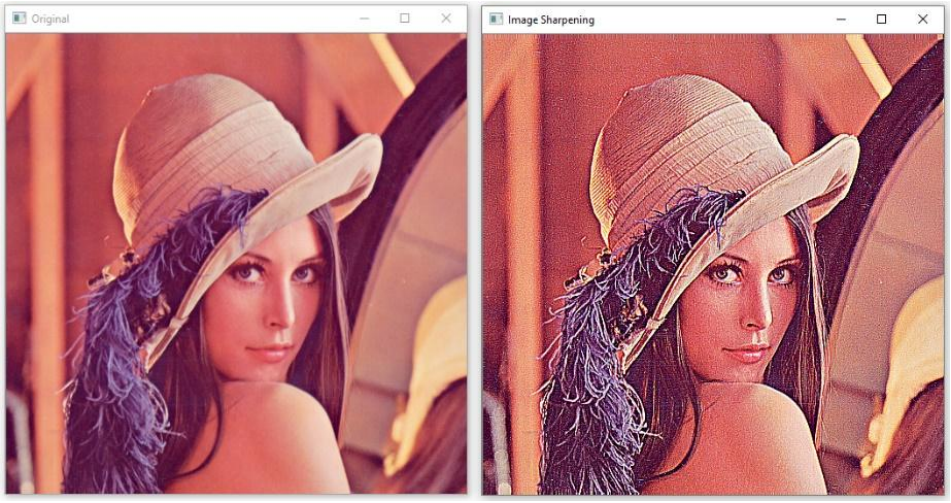
0	-1	0
-1	5	-1
0	-1	0

courtesy
http://docs.gimp.org/en/plug-in-convmatrix.html

1	1	1
1	-8	1
1	1	1

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

Sedangkan contoh hasil sharpening dapat dilihat pada gambar 7.



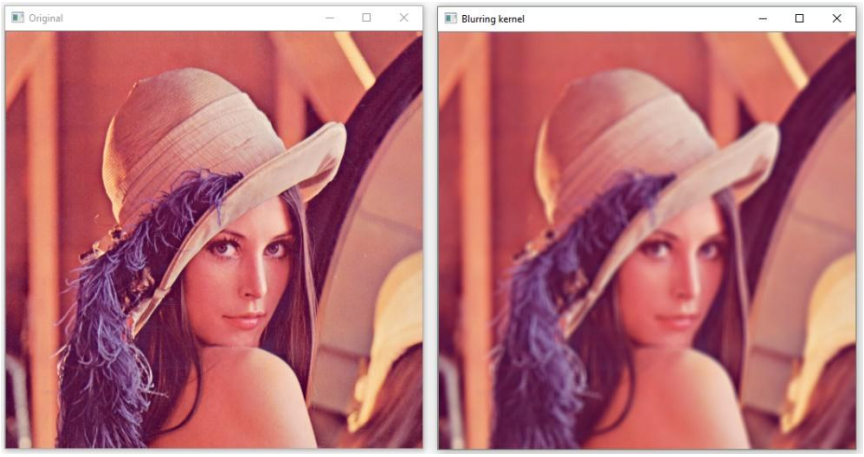
Gambar 7. Contoh Filter Sharpening

C.3.2. Blurring

Kernel untuk blurring/pengaburan menggunakan prinsip bahwa nilai piksel pusat harus dibuat mendekati piksel tetangganya (mengurangi perbedaan). Contoh kernel untuk pengaburan citra dapat menggunakan Gaussian Blurring:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$ **atau** $$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

Sedangkan contoh hasil sharpening dapat dilihat pada gambar 9.



Gambar 9. Contoh Filter Gaussian Blurring

C.3.3. Emboss

Kernel untuk emboss didasarkan pada prinsip memperkuat edge pada satu arah tertentu, tanpa menghilangkan warna lainnya. Arah penguatan ditunjukkan perubahan dari elemen negatif ke positif. Contoh kernel untuk emboss:

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

courtesy
http://docs.gimp.org/en/plug-in-convmatrix.html

1	0	0
0	0	0
0	0	-1

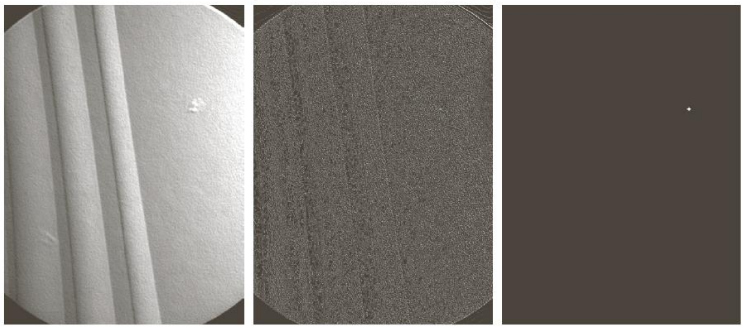
courtesy
http://www.imagesincontext.com/IICFeatures/convolution-filter.htm

Sedangkan contoh hasil emboss dapat dilihat pada gambar 10.

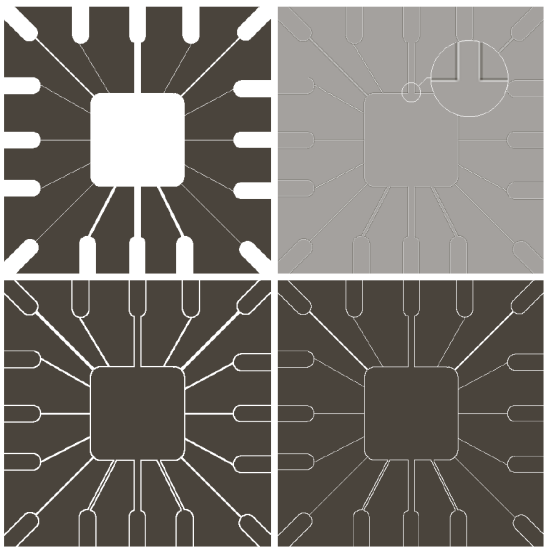
Berikut ini merupakan beberapa contoh kernel filter dasar lainnya:

- $Average\ Filter\ 3x3 = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$
- $Low\ Pass\ Filter = \frac{1}{12}\begin{bmatrix}1 & 1 & 1\\ 1 & 4 & 1\\ 1 & 1 & 1\end{bmatrix}$
- $High\ Pass\ Filter = \begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 3\\ -3 & 0 & 1\end{bmatrix}$
- **Point Detection**

1	1	1
1	−8	1
1	1	1



- **Line Detection**



−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

2	−1	−1
−1	2	−1
−1	−1	2

+45°

−1	2	−1
−1	2	−1
−1	2	−1

Vertical

−1	−1	2
−1	2	−1
2	−1	−1

−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

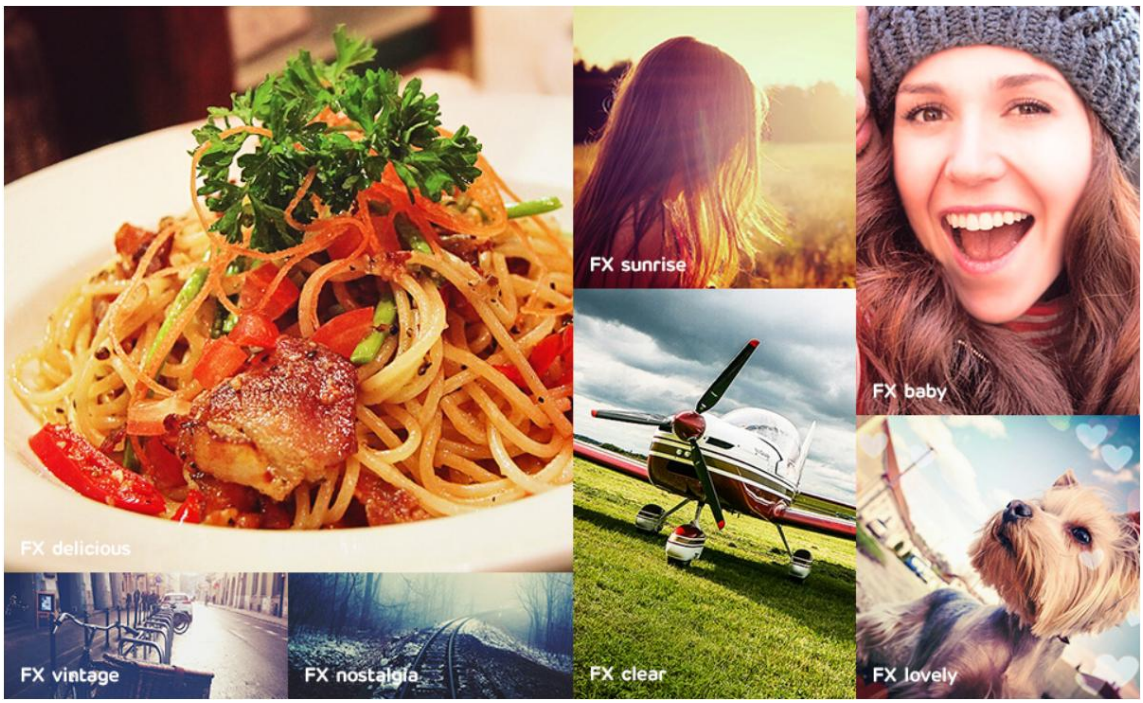
−1	−2	−1		−1	0	1		
0	0	0		−2	0	2		
1	2	1		−1	0	1		

Sobel

a b
c d

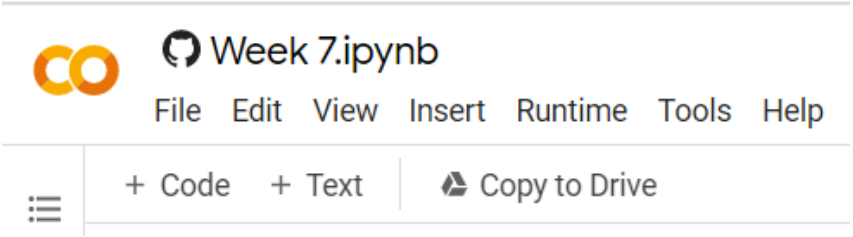
FIGURE 10.16
(a) Original image of size 834 × 1114 pixels, with intensity values scaled to the range [0, 1]. (b) $|g_x|$, the component of the gradient in the x-direction, obtained using the Sobel mask in Fig. 10.14(f) to filter the image. (c) $|g_y|$, obtained using the mask in Fig. 10.14(g). (d) The gradient image, $|g_x| + |g_y|$.

Saat ini hampir semua aplikasi pengolah citra memiliki filter khusus masing-masing yang sangat menarik bagi penggunanya, Gambar berikut adalah contoh filter yang dimiliki aplikasi mobile Line Camera



D. PRAKTIKUM FILTER

1. Buat fungsi konvolusi menggunakan algoritma yang telah dijelaskan pada Bagian C, tanpa menggunakan library atau metode konvolusi dari OpenCV.
2. Berikut merupakan langkah-langkah yang dapat dilakukan:
   a. Buat notebook baru pada google colab, dan beri nama Week7.ipynb. Simpan Salinan pada akun github seperti pada modul sebelumnya.



   b. Akses file yang terdapat pada drive dan import beberapa library yang dibutuhkan

```
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. Buatlah fungsi konvolusi. Catatan: parameter yang digunakan boleh dimodifikasi. Misal, hanya menggunakan parameter image dan kernel saja, atau image, kernel, dan 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 citra yang akan diproses dan ubah menjadi citra keabuan

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. Tentukan kernel yang akan digunakan, contohnya kernel untuk filter sharpening sebagai berikut:

Menentukan kernel yang akan digunakan

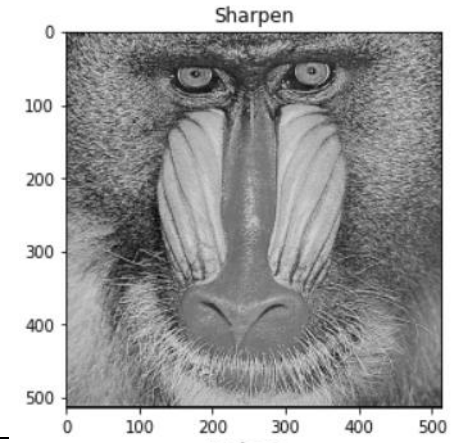
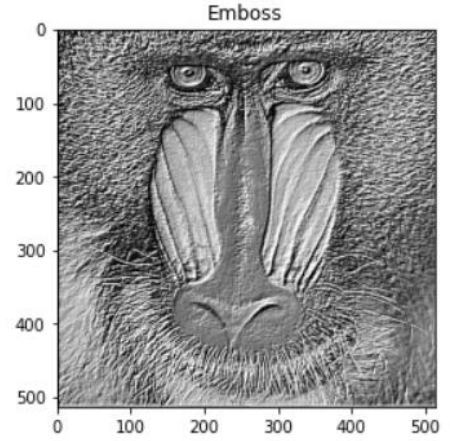
```
#image sharpen
kernel_sharpen = np.array([[0,-1,0],
                           [-1,5,-1],
                           [0,-1,0]])
```

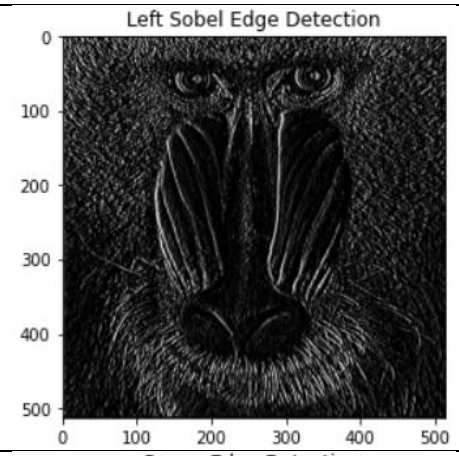
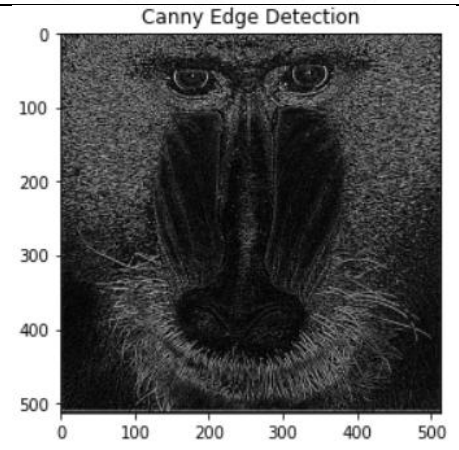
f. Memanggil fungsi konvolusi yang telah dibuat sebelumnya, dan menampilkan hasil konvolusinya:

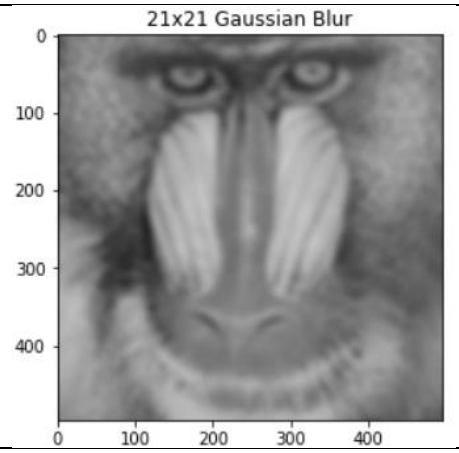
Memanggil fungsi konvolusi dan menerapkan setiap filter yang telah ditentukan

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

3. Buat Image Filter untuk Average filter, low pass filter, high pass filter, dan beberapa filter berikut:

Operasi	Kernel	Hasil
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Emboss	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	

Left Sobel Edge Detection	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	
Canny Edge Detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

21x21 Gaussian Blur	Untuk generate kernel gaussian bisa menggunakan kode berikut:
	`sigma=math.sqrt(kernel_size)`
	`gaussian_kernel = cv.getGaussianKernel(kernel_size, sigma)`
	`gauss_kernel = gaussian_kernel @ gaussian_kernel.transpose()`
		

E. FILTER LIBRARY DAN FILTER MODERN

Pada beberapa percobaan berikut ini kita akan melihat beberapa filter menggunakan library, filter modern yang digunakan pada CNN, filter modern dengan kombinasi beberapa filter tradisional (tanpa Deep Learning).

Percobaan 1:

Pada percobaan 1 ini, kita akan membuat Filter Gaussian, Sharpen, dan Canny menggunakan library filter2d dari OpenCV. Filter ini akan kita terapkan pada Image RGB. Pada bagian awal kode terdapat fungsi show_side_by_side yang digunakan untuk menampilkan gambar secara berdampingan.

```
# Fungsi tampil berdampingan
def show_side_by_side(images, titles, figsize=(15,5)):
    plt.figure(figsize=figsize)
    for i, (img, title) in enumerate(zip(images, titles)):
        if len(img.shape) == 2:  # grayscale
            plt.subplot(1, len(images), i+1)
            plt.imshow(img, cmap="gray")
        else:  # color
            plt.subplot(1, len(images), i+1)
            plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis("off")
    plt.show()

img = cv.imread("/content/drive/MyDrive/Kuliah/PCVK Genap 2023/Images/lena.jpg")
img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

blur = cv.GaussianBlur(img, (7,7), 1)
edges = cv.Canny(cv.cvtColor(img, cv.COLOR_BGR2GRAY), 100, 200)
sharpen_kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
sharpened = cv.filter2D(img, -1, sharpen_kernel)
show_side_by_side([img, blur, sharpened, edges],
                  ["Asli", "Gaussian Blur", "Sharpened", "Canny Edge Detection"])
```

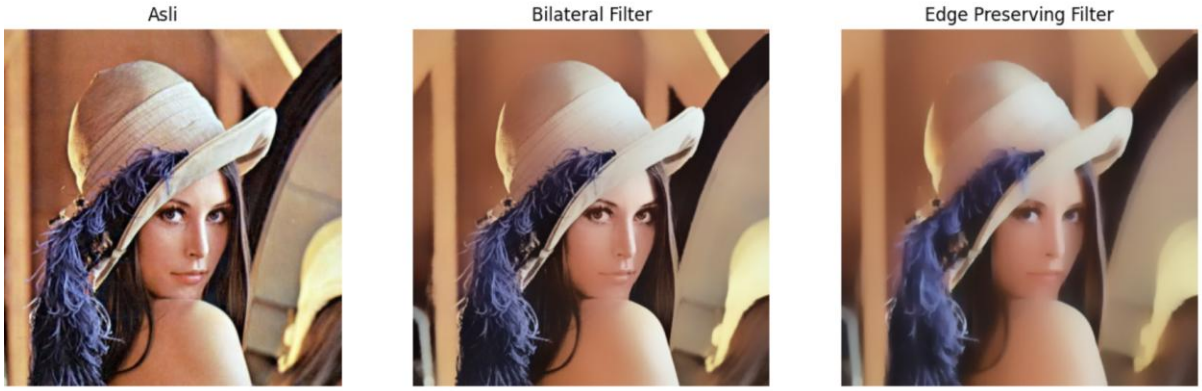
Berikut ini adalah hasil tampilannya:

Percobaan 2:

Pada percobaan 2 berikut ini akan dilakukan filtering modern dari Library OpenCV. Dua filter yang akan digunakan adalah Bilateral Filtering dan Guided Filter. **Bilateral filtering** adalah sebuah metode penyaringan non-linear yang banyak digunakan untuk menghaluskan citra sekaligus tetap mempertahankan ketajaman tepi. Berbeda dengan Gaussian blur biasa yang hanya memperhitungkan jarak spasial antar piksel, bilateral filter juga mempertimbangkan perbedaan intensitas warna. Dengan cara ini, piksel-piksel yang letaknya dekat dan memiliki warna mirip dengan pusat jendela akan mendapat bobot lebih besar dalam perhitungan, sedangkan piksel dengan perbedaan warna kontras (misalnya di sisi tepi) akan tereduksi pengaruhnya. Akibatnya, citra menjadi lebih halus pada area datar, namun garis batas dan detail penting tetap terjaga. Meskipun menghasilkan kualitas yang baik, bilateral filter tergolong lambat karena perhitungan bobot yang cukup kompleks. Dalam praktiknya, filter ini banyak dipakai untuk keperluan seperti perbaikan kualitas foto, pengurangan noise, maupun sebagai dasar dari efek *beauty filter* untuk melembutkan tekstur kulit tanpa mengaburkan kontur wajah.

Guided filtering merupakan teknik yang lebih modern dan efisien. Filter ini didasarkan pada asumsi bahwa, dalam sebuah jendela lokal, hasil penyaringan dapat direpresentasikan sebagai fungsi linear dari citra pemandu (*guide image*). Artinya, setiap piksel keluaran dihitung dengan mempertimbangkan hubungan linier antara nilai piksel di citra masukan dengan nilai piksel pada citra pemandu. Jika citra masukan dan citra pemandu sama, guided filter akan berperan mirip bilateral filter namun dengan perhitungan yang jauh lebih cepat dan hasil yang lebih halus. Keunggulan lainnya, guided filter bisa menggunakan citra yang berbeda sebagai pemandu sehingga mampu mengarahkan proses filtering sesuai kebutuhan. Berkat sifat ini, guided filter sering dipakai dalam berbagai aplikasi lanjutan seperti *HDR tone mapping*, peningkatan detail, *image matting*, *feathering*, serta pemurnian *depth map* pada sistem stereo vision. Secara umum, guided filter menawarkan keseimbangan antara preservasi tepi yang baik, kualitas visual yang halus, dan efisiensi komputasi, sehingga dianggap sebagai penyempurnaan dari pendekatan bilateral filtering.

```
#Filter Modern dari OpenCV
# Bilateral Filter (edge-preserving)
bilateral = cv.bilateralFilter(img, 50, 100, 100)

# Edge Preserving Filter (alternatif Guided Filter)
edge_preserve = cv.edgePreservingFilter(img, flags=1, sigma_s=100, sigma_r=0.9)

show_side_by_side([img, bilateral, edge_preserve],
                  ["Asli", "Bilateral Filter", "Edge Preserving Filter"])
```



Percobaan 3:

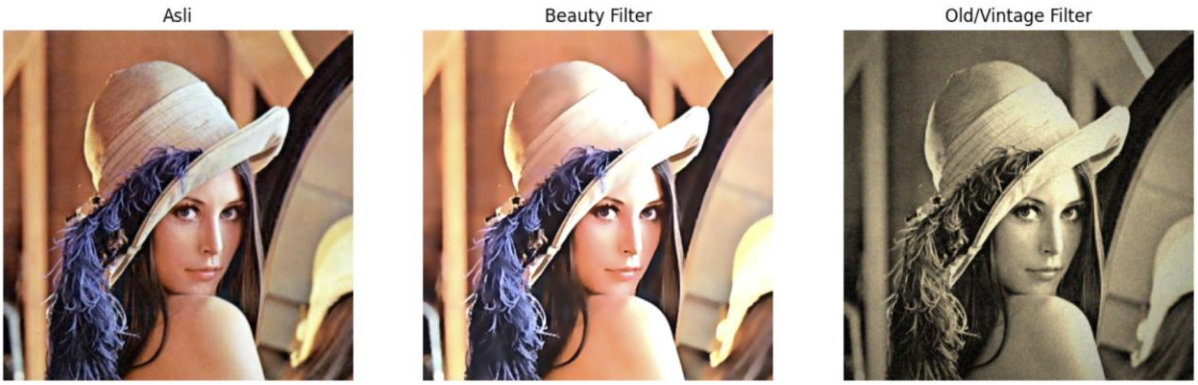
Percobaan kali ini akan mencoba melihat proses Filtering pada CNN (bagian Feature Map), lakukan running code beberapa kali dan perhatikan hasil outputnya. Apa yang dapat kamu simpulkan dari hasil keluaran tersebut.

```python
#Filter Feature Map yang digunakan pada CNN, Lakukan running code bagian ini beberapa kali dan perhatikan hasilnya
import torch
import torch.nn as nn

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 4, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        return self.conv1(x)

model = SimpleCNN()

# Ubah gambar ke tensor
img_tensor = torch.tensor(img_gray, dtype=torch.float32).unsqueeze(0).unsqueeze(0) / 255.0

# Hasil CNN
with torch.no_grad():
    features = model(img_tensor)

# Visualisasi feature maps
feature_maps = [features[0,i].numpy() for i in range(features.shape[1])]
show_side_by_side([img_gray] + feature_maps, ["Asli (Gray)"] + [f"Feature {i+1}" for i in range(len(feature_maps))])
```

Percobaan 4:

Percobaan kali ini akan melakukan efek Beauty dan Vintage yang biasanya digunakan pada Aplikasi popular saat ini. Filter yang digunakan merupakan kombinasi dari filter tradisional. Perlu diketahui untuk filter aplikasi popular bisa jadi tidak menggunakan metode yang sama. Pada Aplikasi popular bisa jadi menggunakan model GenAI dengan data Training untuk memberikan hasil yang lebih akurat.

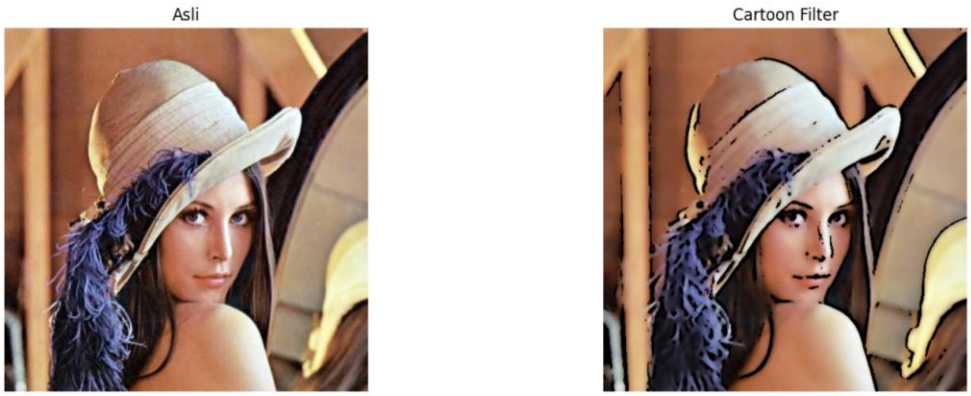
```
# ====================
# 1. Beauty Filter
# ====================
# Step 1: Smoothing kulit dengan bilateral filter
smooth = cv.bilateralFilter(img, d=15, sigmaColor=75, sigmaSpace=75)

# Step 2: Unsharp masking (pertajam mata/bibir)
gaussian = cv.GaussianBlur(smooth, (0,0), 3)
sharpened = cv.addWeighted(smooth, 1.5, gaussian, -0.5, 0)

# Step 3: Brightness & contrast
alpha = 1.2  # contrast
beta = 15    # brightness
beauty = cv.convertScaleAbs(sharpened, alpha=alpha, beta=beta)

# ====================
# 2. Old/Vintage Filter
# ====================
# Step 1: Sepia tone
sepia_kernel = np.array([[0.272, 0.534, 0.131],
                         [0.349, 0.686, 0.168],
                         [0.393, 0.769, 0.189]])
sepia = cv.transform(img, sepia_kernel)
sepia = np.clip(sepia, 0, 255).astype(np.uint8)

# Step 2: Vignette
rows, cols = img.shape[:2]
kernel_x = cv.getGaussianKernel(cols, cols*0.6)
kernel_y = cv.getGaussianKernel(rows, rows*0.6)
kernel = kernel_y * kernel_x.T
mask = kernel / kernel.max()
vignette = np.copy(sepia)
for i in range(3):
    vignette[:,:,i] = vignette[:,:,i] * mask

# Step 3: Noise/Grain
noise = np.random.normal(0, 15, vignette.shape).astype(np.int16)
old_img = np.clip(vignette.astype(np.int16) + noise, 0, 255).astype(np.uint8)
```



Percobaan 5:

Percobaan 5 akan menunjukkan pada anda filter anime / cartoon menggunakan kombinasi filter tradisional.

```
#Filter Anime / Cartoon
# Step 1: Edge detection (pakai median blur dulu agar gambar menjadi lebih halus)
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray_blur = cv.medianBlur(gray, 7)
edges = cv.adaptiveThreshold(gray_blur, 255,
                             cv.ADAPTIVE_THRESH_MEAN_C,
                             cv.THRESH_BINARY, 9, 9)

# Step 2: Bilateral filter untuk smoothing warna
color = cv.bilateralFilter(img, d=9, sigmaColor=200, sigmaSpace=200)

# Step 3: Gabungkan (cartoonize)
cartoon = cv.bitwise_and(color, color, mask=edges)

# Tampilkan
show_side_by_side([img, cartoon], ["Asli", "Cartoon Filter"])
```



Percobaan 6:

Pada Percobaan 6 akan ditunjukkan contoh Filter Malam.

```
#Night Filter
img = cv.imread("/content/drive/MyDrive/Kuliah/PCVK Genap 2023/Images/djawatan.jpg")
img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

# Step 1: Gelapkan (contrast turun, brightness negatif)
night = cv.convertScaleAbs(img, alpha=0.6, beta=-40)

# Step 2: Tambah bias biru
blue_tint = np.full_like(night, (50, 0, 100))  # BGR
night = cv.addWeighted(night, 0.8, blue_tint, 0.2, 0)

# Step 3: Efek glow di area terang dengan filter2D (blur kernel)
kernel = np.ones((15,15), np.float32) / 225
glow = cv.filter2D(night, -1, kernel)

# Kombinasikan asli + glow
night_glow = cv.addWeighted(night, 0.7, glow, 0.3, 0)

show_side_by_side([img, night, night_glow],
                  ["Asli", "Night Filter", "Night + Glow (filter2D)"])
```

Percobaan 7

Percobaan 7 menunjukkan Filter Pagi dan Pagi ditambahkan efek kabut.

```python
#Filter Suasana pagi dan Kabut
# =========================
# Step 1: Kurangi kontras & cerahkan
# =========================
alpha = 0.9   # contrast
beta = 20     # brightness
soft = cv.convertScaleAbs(img, alpha=alpha, beta=beta)

# =========================
# Step 2: Tambahkan warm tone (kemerahan / oranye)
# =========================
warm_tint = np.full_like(soft, (40, 70, 120))  # BGR
pagi = cv.addWeighted(soft, 0.8, warm_tint, 0.2, 0)

# =========================
# Step 3: Tambahkan haze (kabut tipis) dengan filter2D
# =========================
# Kernel blur Gaussian-like untuk menciptakan efek kabut
kernel = cv.getGaussianKernel(3, 3)
kernel = kernel @ kernel.T  # jadikan 2D kernel
kabut = cv.filter2D(pagi, -1, kernel)

# tambah layer putih untuk kabut lebih nyata
white_layer = np.full_like(pagi, 255)
kabut = cv.addWeighted(kabut, 0.7, white_layer, 0.3, 0)

show_side_by_side([img, pagi, kabut],
                  ["Asli", "Pagi", "Pagi + Kabut"])
```

Selamat Belajar