Lapooran Tugas Histogram dan Konvolusi
IF-604 Pengolahan Citra

INSTITUT TEKNOLOGI HARAPAN BANGSA

Veritas vos liberabit

1119007 - Timothy Ray
1119023 - Jedediah Fanuel
1119033 - Fedly Septian
1119038 - Elangel Neilea Shaday

Institut Teknologi Harapan Bangsa

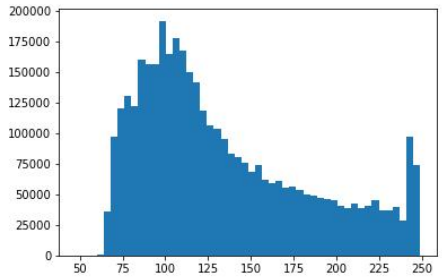
2022

I. Histogram

Source code

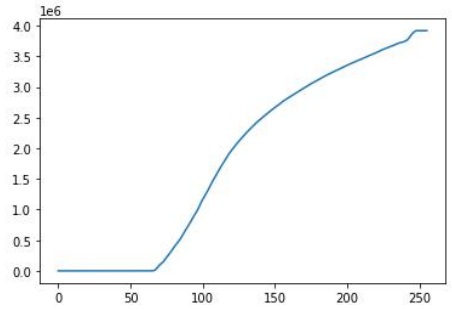
```
%matplotlib inline
from IPython.display import display, Math, Latex
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

img = Image.open('grayscale.png')

# tampilkan citra asal
plt.imshow(img, cmap='gray')
```


```
# ubah citra menjadi array
img = np.asarray(img)

# mengubah array 2D menjadi 1D
## Nilai yang ada pada array flat adalah nilai intensitas ( dari 0 [hitam] ~ 255 [putih] )
flat = img.flatten()

# kita plot histogram citra asal
plt.hist(flat, bins=50)
```

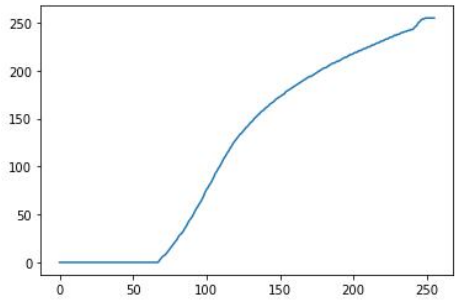



```
def get_histogram(image, bins):
    # array seukuran bin hitogram dan diisikan dengan nilai nol
    histogram = np.zeros(bins)

    # loop menelusuri pixel dan jumlahkan jumlah pixel
    for pixel in image:
        histogram[pixel] += 1

    # return hasil akhir
    return histogram

# buat fungsi cumulative sum (kalau di kelas disebut Sk)
def cumsum(a):
    a = iter(a)
    b = [next(a)]
    for i in a:
        b.append(b[-1] + i)
    return np.array(b)
```

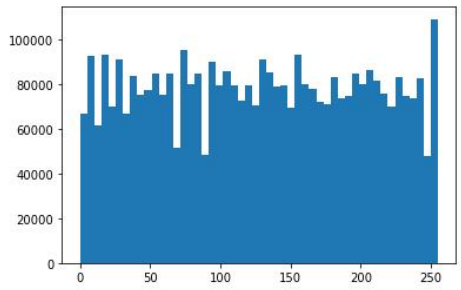
```
# eksekusi fungsi
cs = cumsum(hist)

# tampilkan hasilnya
plt.plot(cs)plt.plot(cs)
```




```
# pembilang & penyebut
nj = (cs - cs.min()) * 255
N = cs.max() - cs.min()

# normalisasi jumlah kumulatif ke range (0 ~ 255)
cs = nj / N

# mengubahnya kembali ke tipe uint8
# karena tidak dapat menggunakan nilai floating point dalam citra
cs = cs.astype('uint8')

plt.plot(cs)
```

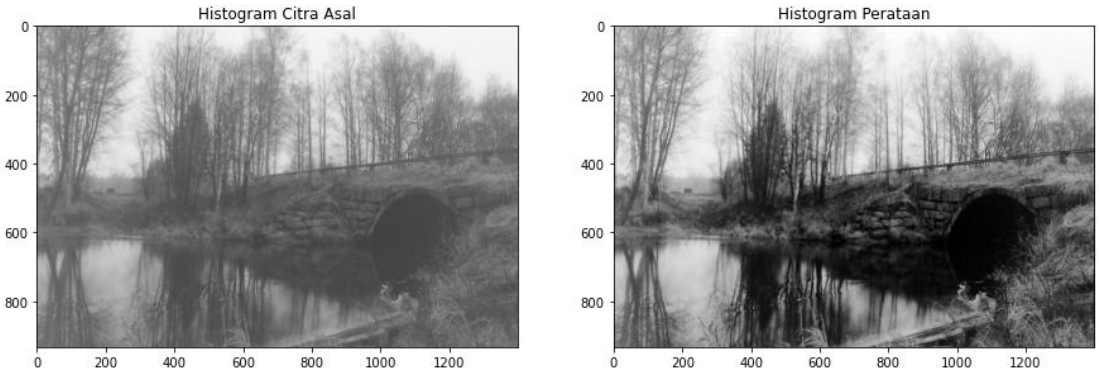



```
# dapatkan nilai dari jumlah kumulatif untuk setiap indeks di flat ke variable baru
img_new = cs[flat]

# kita plot histogram perataan
plt.hist(img_new, bins=50)
```

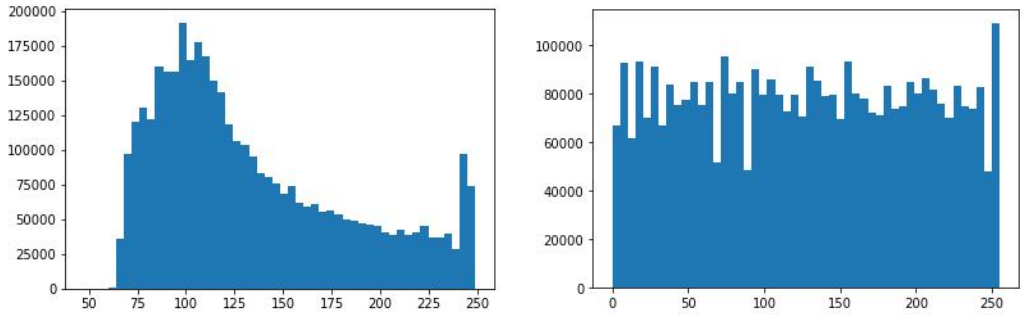



```
# kembalikan array 1D ke bentuk mula-mula citra
img_new = np.reshape(img_new, img.shape)

# mengatur citra agar bersebelahan
fig = plt.figure()
fig.set_figheight(15)
fig.set_figwidth(15)
```

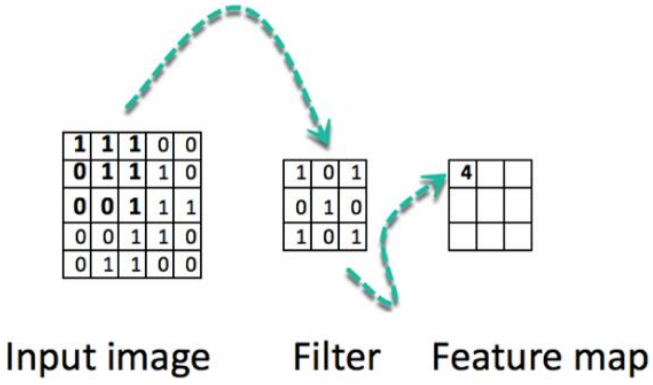
```
fig.add_subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title('Histogram Citra Asal')

# tampilkan citra baru
fig.add_subplot(1,2,2)
plt.imshow(img_new, cmap='gray')
plt.title('Histogram Perataan')

plt.show(block=True)
```



Seperti yang dapat dilihat, citra yang dihasil oleh perataan histogram lebih jelas.



II. Konvolusi



Konvolusi dalam pengolahan citra secara sederhana adalah mencari nilai dari sebuah pixel berdasarkan nilai pixel tetangganya menggunakan filter (kernel). Pada tugas kali ini kami menggunakan ukuran kernel 3x3.

III. Konvolusi untuk Pengurangan Noise

Pada konvolusi untuk pengurangan noise, kami mencoba tiga metode, yaitu *mean*, *median*, dan combined *balanced*.

Algoritmanya secara umum adalah sebagai berikut:
1. Input citra asal
2. Ubah menjadi grayscale
3. Ubah citra menjadi array
4. Lakukan konvolusi metode masing-masing (akan dijelaskan pada bagian masing-masing)

1. Mean

Pada metode mean, kernel yang kami gunakan sebagai berikut:

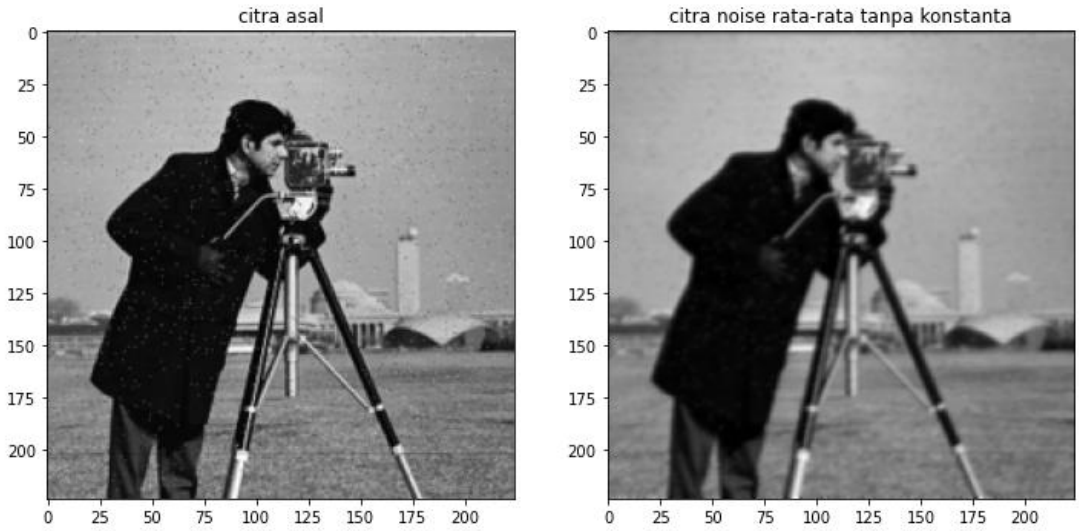
$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

```
noise_mean = np.array([
      [1/9, 1/9, 1/9],
      [1/9, 1/9, 1/9],
      [1/9, 1/9, 1/9]
])
```

Kernel tersebut akan menelusuri di atas seluruh pixel gambar, kemudian nilai dikalikan dan dijumlahkan, yang digunakan untuk menentukan nilai cell tengah. Dengan cara ini, nilai yang dihasilkan menghilangkan pixel yang tidak sesuai dengan lingkungannya, sehingga mengurangi noise.

Source code

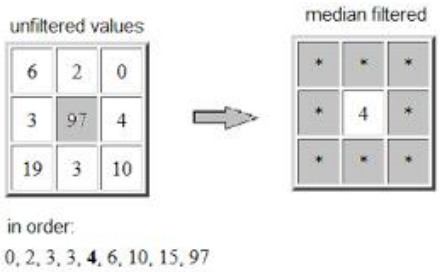
```
# Mendapatkan kernel 3x3
def get_kernel():
    return np.ones((3, 3), np.float32) / 9

def get_mean_with_kernel(filter_area, kernel):
    # Fastest solution to multiply the matrices and get the result.
    return np.sum(np.multiply(kernel, filter_area))

def mean_filter(image, height, width):
    # Set the kernel.
    kernel = get_kernel()

    for row in range(1, height + 1):
        for column in range(1, width + 1):
            # Get the area to be filtered with range indexing.
            filter_area = image[row - 1:row + 2, column - 1:column + 2]
            res = get_mean_with_kernel(filter_area, kernel)
            image[row][column] = res

    return image


cameraman = Image.open('cameraman.jpeg')
cameraman = ImageOps.grayscale(cameraman)
cameraman = cameraman.resize(size=(224, 224))
plot_image(img=cameraman)

cameraman_arr = np.array(cameraman)
meanF2 = mean_filter(cameraman_arr,222,222)
plot_two_images(
    img1=cameraman,
    img2=meanF2,
    name1='citra asal',
    name2='citra noise rata-rata tanpa konstanta'
)
```



2. Median

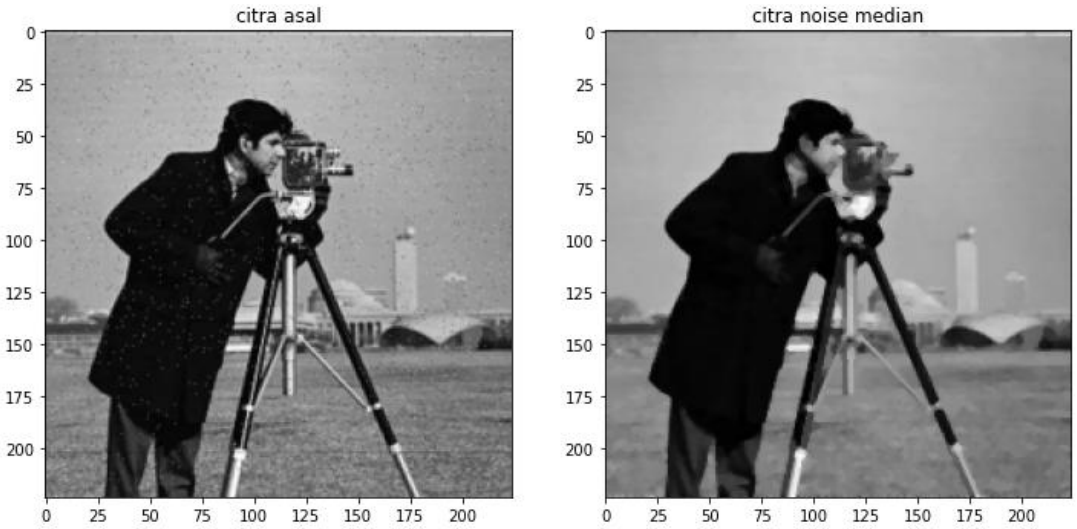
Pada metode median, kernel yang digunakan akan berubah-ubah berdasarkan nilai pixel yang berada dalam jangkauan kernel saat itu. Nilai-nilai yang ada diurutkan dari yang terkecil menuju yang terbesar, kemudian cari nilai mediannya. Ide dari metode ini adalah membuang *outliers* (*noise*) pada citra dengan mengambil nilai median.

Sebagai ilustrasi:



Source code

```
def get_median(filter_area):
    res = np.median(filter_area)
    return res

def median_filter(image, height, width):
    for row in range(1, height + 1):
        for column in range(1, width + 1):
            filter_area = image[row - 1:row + 2, column - 1:column + 2]
            image[row][column] = get_median(filter_area)

    return image

cameraman_arr = np.array(cameraman)
medianF = median_filter(cameraman_arr,222,222)
# plot_image(medianF)
plot_two_images(
    img1=cameraman,
    img2=medianF,
    name1='citra asal',
    name2='citra noise median'
)
```



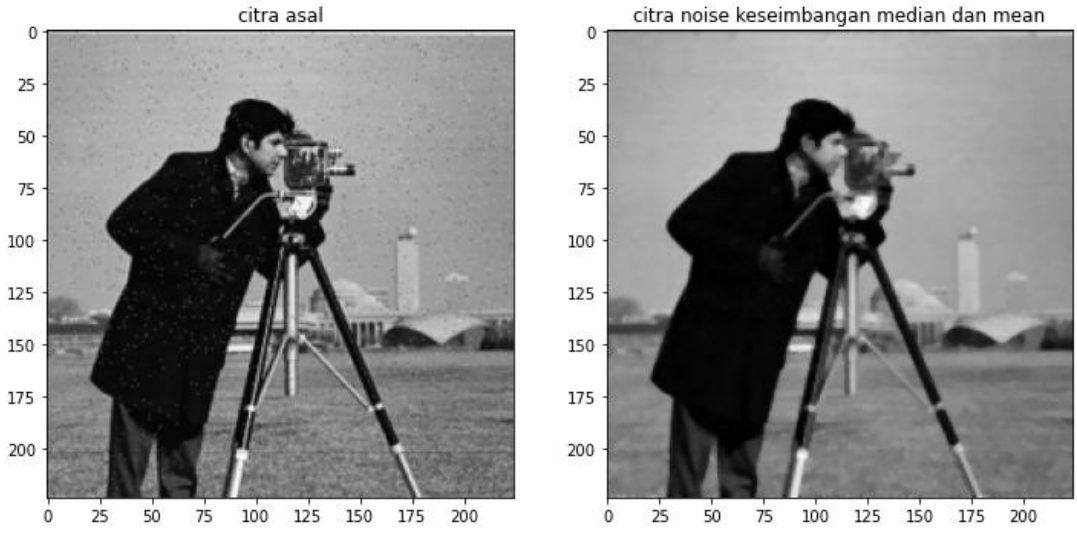
3. Combined Balanced

Pada metode mean median balance, kita gunakan kedua metode sebelumnya, *mean* & *median,* kemudian kita gabungkan dari kedua hasil perhitungan metode tersebut menggunakan rumus:

```
0.2 * meanFilter(pixel) * 0.8 * medianFilter(pixel)
```

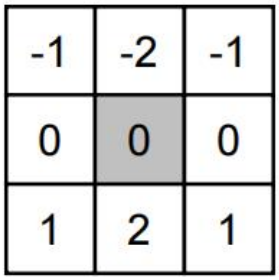
Source code

```
BALANCE_ALPHA = 0.2

def mean_median_balanced_filter(image, height, width):
    for row in range(1, height + 1):
        for column in range(1, width + 1):
            filter_area = image[row - 1:row + 2, column - 1:column + 2]
            mean_filter_vector = get_mean_with_kernel(filter_area, get_kernel())
            median_filter_vector = get_median(filter_area)
            image[row][column] = BALANCE_ALPHA * mean_filter_vector +
                                 (1 - BALANCE_ALPHA) * median_filter_vector
    return image

cameraman_arr = np.array(cameraman)
mean_median_balanced = mean_median_balanced_filter(cameraman_arr,222,222)
plot_two_images(
    img1=cameraman,
    img2=mean_median_balanced,
    name1='citra asal',
    name2='citra noise keseimbangan median dan mean'
)
```

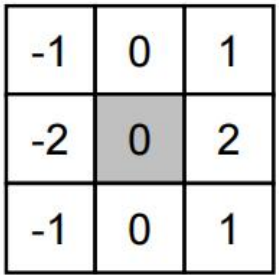


IV. Konvolusi untuk Deteksi Tepi

Pada deteksi tepi, kami menggunakan kernel *sobel*, seperti dijelaskan pada materi *Spatial Filtering 2 hal. 27*, kernel dapat dilihat sebagai berikut:



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



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

```
edgesV = np.array([
      [-1, -2, -1],
      [0,    0, 0],
      [1, 2, 1]
])
```

```
edgesH = np.array([
      [-1, 0, 1],
      [-2, 0, 2],
      [-1, 0, 1]
])
```

Kernel kiri kami sebut edgesV (vertical) dan kernel kanan kami sebut edgesH (horizontal).

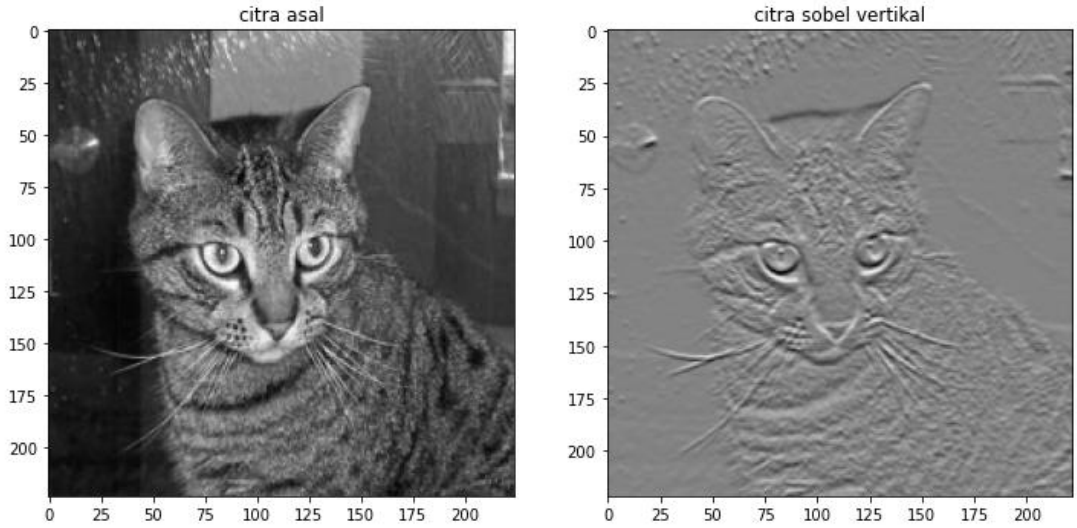
Algoritmanya adalah sebagai berikut:

1. Input citra asal
2. Ubah menjadi citra *grayscale*
3. Ubah citra menjadi array
4. Lakukan konvolusi menggunakan kernel sobel vertical
5. Lakukan konvolusi menggunakan kernel sobel horizontal
6. Gabungkan kedua hasil citra konvolusi sobel vertical dan sobel horizontal
    a) Gunakan persamaan berikut:
    $G=\sqrt{Gx^2+Gy^2}$

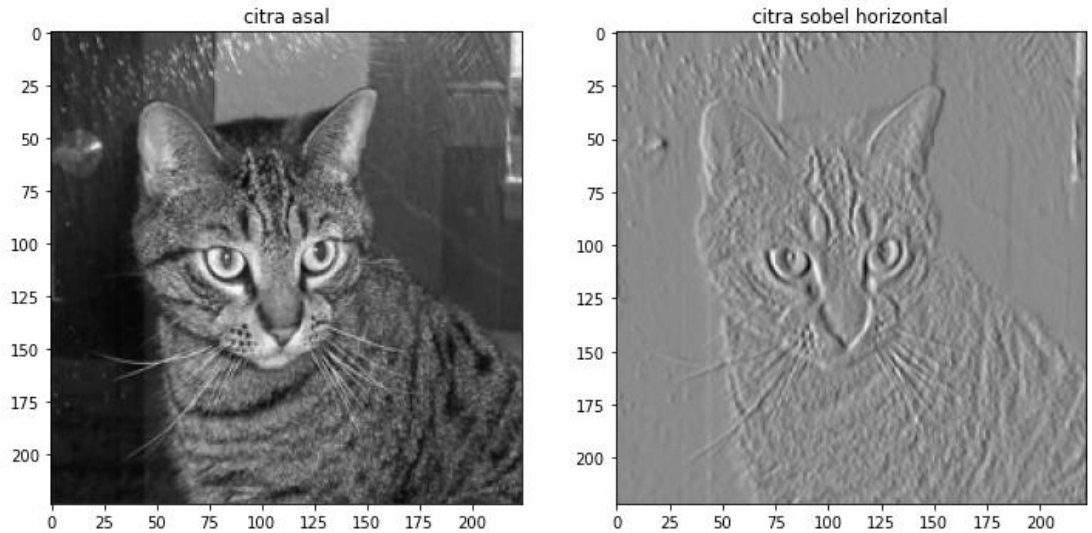
Source Code

```
cat = Image.open('1.jpeg')
cat = ImageOps.grayscale(cat)
cat = cat.resize(size=(224, 224))
plot_image(img=cat)

def convolve(img: np.array, kernel: np.array) -> np.array:
    # Assuming a rectangular image
    tgt_size = calculate_target_size(
        img_size=img.shape[0],
        kernel_size=kernel.shape[0]
    )

    # To simplify things
    k = kernel.shape[0]

    # 2D array of zeros
    convolved_img = np.zeros(shape=(tgt_size, tgt_size))

    # Iterate over the rows
    for i in range(tgt_size):
```

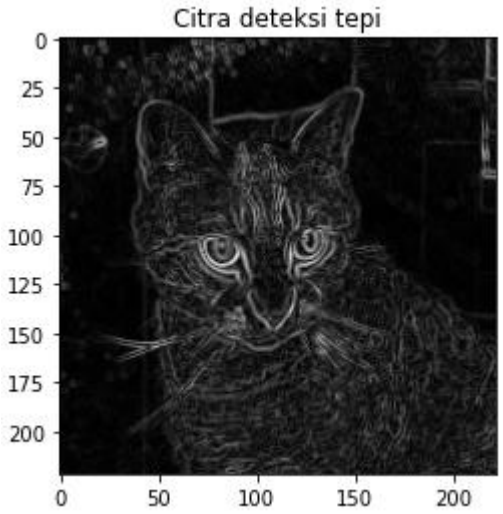
```
        # Iterate over the columns
        for j in range(tgt_size):
            # img[i, j] = individual pixel value
            # Get the current matrix
            mat = img[i:i+k, j:j+k]

            # Apply the convolution - element-wise multiplication and summation of the result
            # Store the result to i-th row and j-th column of our convolved_img array
            convolved_img[i, j] = np.sum(np.multiply(mat, kernel))

    return convolved_img

cat_outlinedV = convolve(img=np.array(cat), kernel=edgesV)
plot_two_images(
    img1=cat,
    img2=cat_outlinedV,
    name1='citra asal',
    name2='citra sobel vertikal'
)
```




```
cat_outlinedH = convolve(img=np.array(cat), kernel=edgesH)
plot_two_images(
    img1=cat,
    img2=cat_outlinedH,
    name1='citra asal',
    name2='citra sobel horizontal'
)
```



```
# Gabungkan sobel vertical & sobel horizontal
gradient_magnitude = np.sqrt(np.square(cat_outlinedH) + np.square(cat_outlinedV))
gradient_magnitude_unnormalize = gradient_magnitude


plt.imshow(gradient_magnitude_unnormalize, cmap='gray')
plt.title("Citra deteksi tepi")
plt.show()
```



Contoh Lain: