**Lapooran Tugas Histogram dan Konvolusi**

**IF-604 Pengolahan Citra**



1119007 - Timothy Ray

1119023 - Jedediah Fanuel

1119033 - Fedly Septian

1119038 - Elangel Neilea Shaday

Institut Teknologi Harapan Bangsa

2022

1. **Histogram**

**Source code**

%matplotlib inline

from IPython.display import display, Math, Latex

import numpy as np

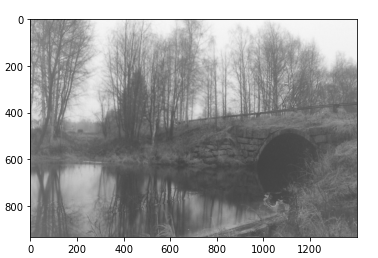
import matplotlib.pyplot as plt

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)

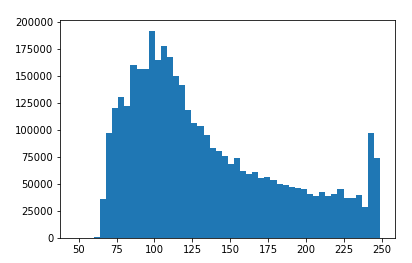
# mengubah array 2D menjadi 1D

## 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)

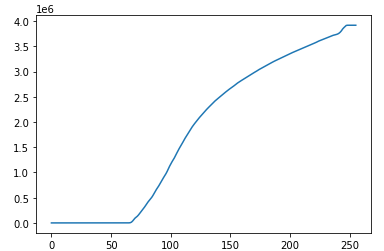
return np.array(b)

# 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)

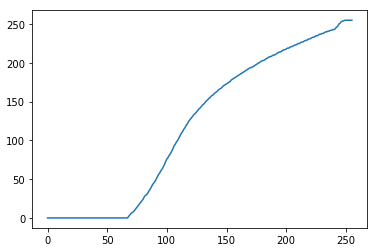
cs = nj / N

# mengubahnya kembali ke tipe uint8

# karena tidak dapat menggunakan nilai floating point dalam citra

cs = cs.astype('uint8')

plt.plot(cs)

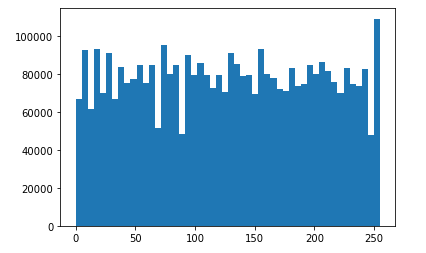


# dapatkan nilai dari jumlah kumulatif untuk setiap indeks di flat ke variable baru

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')

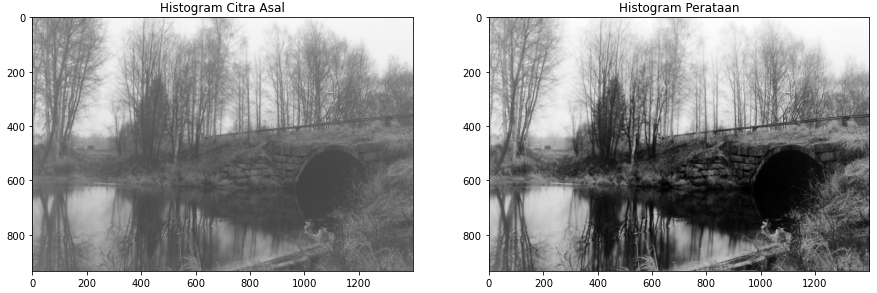
# tampilkan citra baru

fig.add\_subplot(1,2,2)

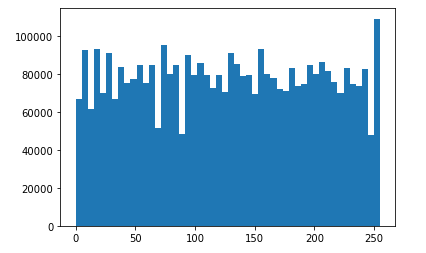
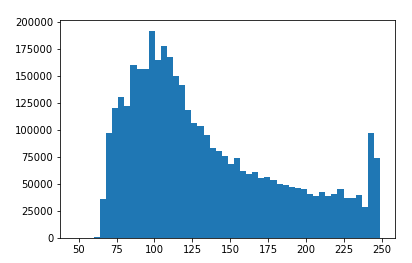
plt.imshow(img\_new, cmap='gray')

plt.title('Histogram Perataan')

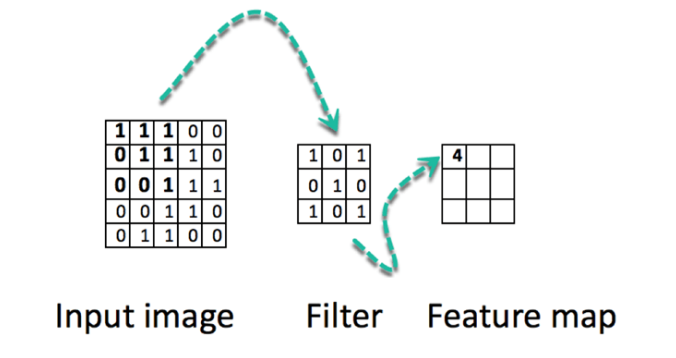
plt.show(block=True)



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



1. **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.

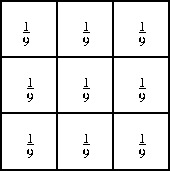
1. **Konvolusi untuk Pengurangan Noise**

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)
5. **Mean**

Pada metode mean, kernel yang kami gunakan sebagai berikut:



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(

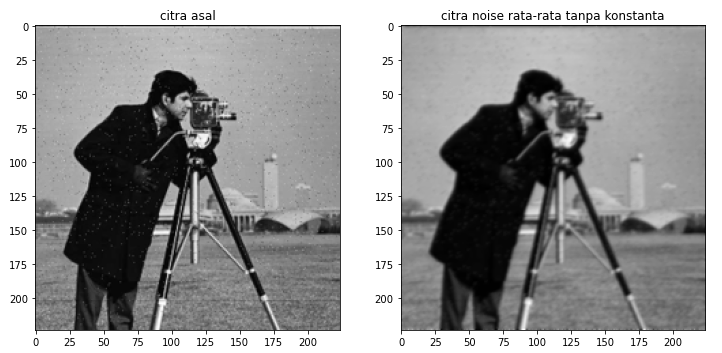
img1=cameraman,

img2=meanF2,

name1='citra asal',

name2='citra noise rata-rata tanpa konstanta'

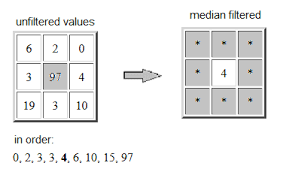
)



1. **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(

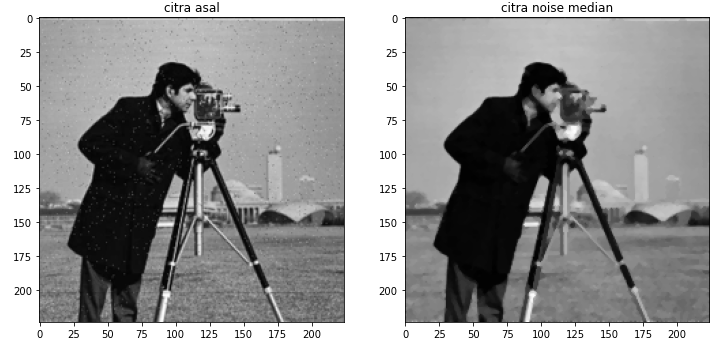
img1=cameraman,

img2=medianF,

name1='citra asal',

name2='citra noise median'

)



1. **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(

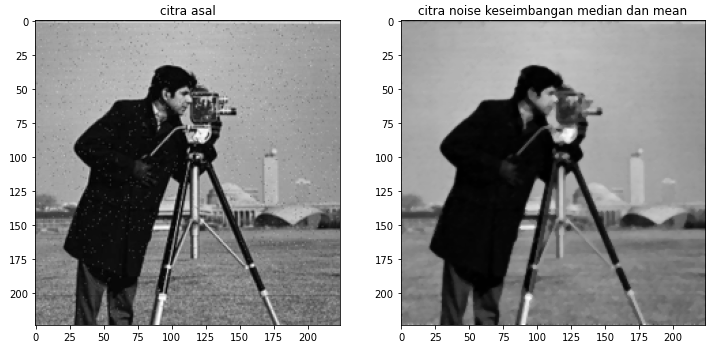
img1=cameraman,

img2=mean\_median\_balanced,

name1='citra asal',

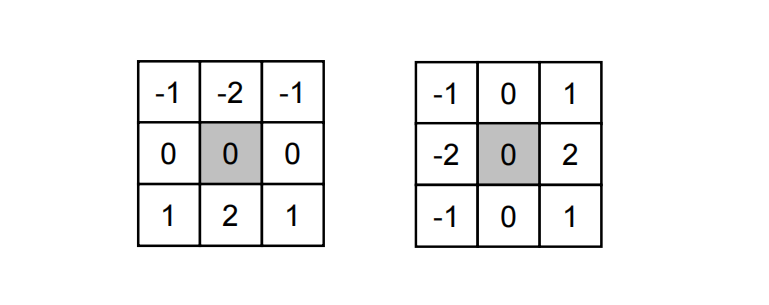
name2='citra noise keseimbangan median dan mean'

)



1. **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:



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
   1. Gunakan persamaan berikut:

*G=√Gx2+Gy2*

**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(

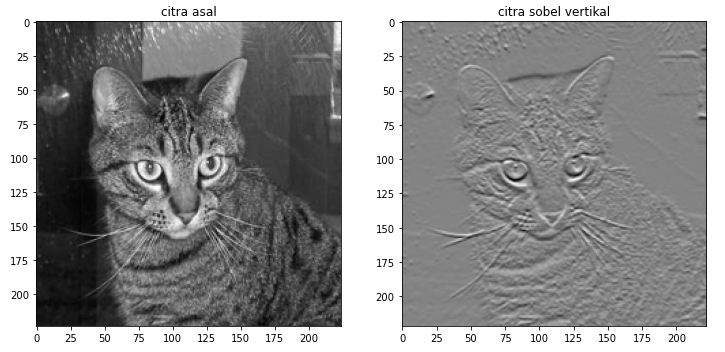
img1=cat,

img2=cat\_outlinedV,

name1='citra asal',

name2='citra sobel vertikal'

)



cat\_outlinedH = convolve(img=np.array(cat), kernel=edgesH)

plot\_two\_images(

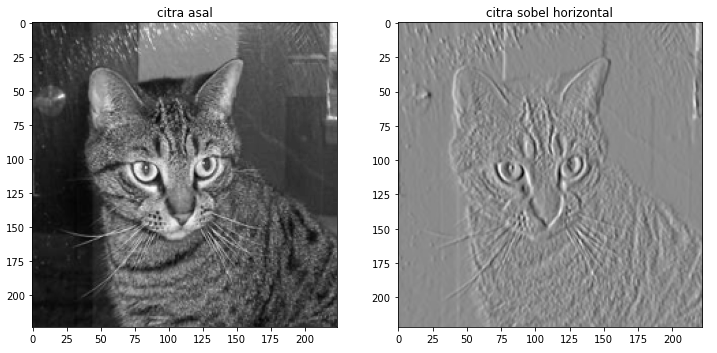
img1=cat,

img2=cat\_outlinedH,

name1='citra asal',

name2='citra sobel horizontal'

)



# Gabungkan sobel vertical & sobel horizontal

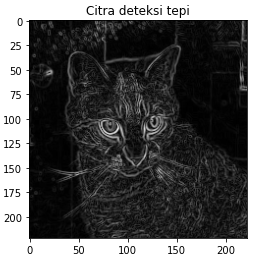
gradient\_magnitude = np.sqrt(np.square(cat\_outlinedH) + np.square(cat\_outlinedV))

gradient\_magnitude\_unnormalize = gradient\_magnitude

plt.imshow(gradient\_magnitude\_unnormalize, cmap='gray')

plt.title("Citra deteksi tepi")

plt.show()



Contoh Lain:

