

# Image Enhancement

## Convolution and Correlation

Nama : Muhamad Hilmy Haidar
NIM : G64170030
Tanggal : 20 Februari 2020
Nama Asisten : <ol style="list-style-type: none"><li>1. Hilmi Farhan Ramadhani G64160048</li><li>2. Kautsar Ibrahim Hilmi G64160073</li></ol>

1. Buatlah fungsi untuk melakukan median filter!
  - a. Pertama-tama, lakukan looping baris serta kolom pada gambar asli
  - b. Jika titik tengah (anchor) tidak berada di wilayah batas (border) maka lakukan kalkulasi median. Apabila berada di border maka nilai pixel dianggap tetap untuk menjaga keberagaman nilai pixel image seperti sebelumnya
  - c. Saat kalkulasi, ambil seluruh nilai yang masuk pada wilayah matriks kernel dan lakukan sorting setelahnya. Setelah diurutkan maka berikan nilai kembalian berupa median dari data yang telah di sorting tersebut.

```
def find_median(self, data, x, y):
    result = np.array([])
    for i in range(self.size):
        for j in range(self.size):
            result = np.append(result, data[(x + i - self.border), (y + j - self.border)])
    n = self.size * self.size
    result = sorted(result)
    return int(result[int(n / 2) + 1]) if n % 2 == 0 else int((result[int(n / 2)] + result[int(n / 2) + 1]) / 2)

def median_filter(self, image):
    row, col = image.shape
    canvas = np.zeros((row, col), np.uint8)
    for i in range(row):
        for j in range(col):
            if (self.border < i < row - self.border) and (self.border < j < col - self.border):
                canvas[i, j] = self.find_median(image, i, j)
            else:
                canvas[i, j] = image[i, j]
    return canvas
```

2. Buatlah fungsi untuk melakukan low pass filter!

- Pertama-tama, lakukan looping baris serta kolom pada gambar asli
- Jika titik tengah (anchor) tidak berada di wilayah batas (border) maka lakukan kalkulasi low pass filter. Apabila berada di border maka nilai pixel dianggap tetap untuk menjaga keberagaman nilai pixel image seperti sebelumnya
- Normalisasi terlebih dahulu kernel yang digunakan agar nilai nya sesuai dengan syarat LPF yaitu  $H(x,y) \geq 0$  dan  $\sum H(x,y) = 1$
- Kali kan nilai pixel dari gambar asli yang bersesuaian dengan posisi nilai kernel yang telah dinormalisasi tadi. Selanjutnya kembalikan hasil penjumlahan dari nilai-nilai pixel yang telah dihitung tadi

```
def low_pass_calculation(self, data, x, y):
    result = np.array([], np.int32)
    for i in range(self.size):
        for j in range(self.size):
            result = np.append(result, data[(x + i - self.border), (y + j - self.border)] *
self.normalize[i, j])
    result = result.astype('uint8')
    return sum(result)

def low_pass_filter(self, image):
    row, col = image.shape
    canvas = np.zeros((row, col), np.uint8)
    for i in range(row):
        for j in range(col):
            if (self.border < i < row - self.border) and (self.border < j < col - self.border):
                canvas[i, j] = self.low_pass_calculation(image, i, j)
            else:
                canvas[i, j] = image[i, j]
    return canvas
```

- Tentukan ukuran kernel (dan nilai kernel untuk LPF) serta banyaknya proses filtering yang dibutuhkan. Berikan alasan terhadap pilihan tersebut!

Ukuran yang sering digunakan untuk kernel adalah  $3 \times 3$  karena ukuran tersebut memiliki border yang kecil (hanya satu row dan satu kolom di bagian pinggir) semakin besar ukuran kernel maka semakin besar pula pixel yang tidak bisa dijadikan titik tengah (anchor) sehingga kita harus menggunakan beberapa cara lain untuk mengolahnya.

Untuk LPF, nilai kernel yang sering digunakan yaitu :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Sebenarnya terdapat bentuk kernel lain, hanya saja, kernel di atas menurut saya merupakan kernel yang paling sesuai untuk blurring karena masing-masing cell pada kernel memiliki nilai yang sama. Itu artinya setiap tetangga ikut menyumbang nilai pixel yang baru secara merata tanpa ada yang berpengaruh lebih besar terhadap yang lain (terkecuali kalau nilai awal pixel tengah nya memang besar dan nilai tetangganya kecil maka nilai pixel di titik tengah tidak akan terpengaruh terhadap tetangganya atau sebaliknya jika nilai seluruh tetangganya lebih besar maka akan berpengaruh kepada nilai pixel di titik tengah).

Banyak proses filtering yang dibutuhkan agar filter LPF berpengaruh optimal terhadap gambar adalah dengan memaksimalkan jumlah titik tengah (anchor) yang dikalkulasi menjadi nilai baru dengan perhitungan LPF yang sebelumnya sudah dijelaskan.

4. Terapkan kedua fungsi tersebut pada citra *LennaInput.jpg* sehingga dapat menghasilkan citra sedekat mungkin dengan *LennaOutput.jpg*
  - a. Langkah pertama melakukan blurring agar mengurangi noise sehingga noise tidak akan mengganggu proses edge detection
  - b. Lakukan edge detection dengan LPF

Kode yang digunakan :

```
import cv2
import numpy as np

class Kernel:
    def __init__(self, size, matrix=np.zeros((3, 3), np.uint8)):
        self.size = size
        self.border = int(self.size / 2)
        self.matrix = matrix
        self.total = self.matrix.sum()
        self.normalize = np.zeros((self.size, self.size))
        for i in range(self.size):
            for j in range(self.size):
                if self.total > 0:
                    self.normalize[i, j] = self.matrix[i, j] / self.total
                else:
                    self.normalize[i, j] = self.matrix[i, j]

    def set_size(self, value):
        self.size = value
        self.border = int(self.size / 2)

    def get_size(self):
        return self.size

    def set_matrix(self, value):
        self.matrix = value
```

```

        self.total = self.matrix.sum()
        for i in range(self.size):
            for j in range(self.size):
                self.normalize[i, j] = self.matrix[i, j] / self.total

    def get_matrix(self):
        return self.matrix

    def find_median(self, data, x, y):
        result = np.array([])
        for i in range(self.size):
            for j in range(self.size):
                result = np.append(result, data[(x + i - self.border), (y + j - self.border)])
        n = self.size * self.size
        result = sorted(result)
        return int(result[int(n / 2) + 1]) if n % 2 == 0 else int((result[int(n / 2)] +
result[int(n / 2) + 1]) / 2)

    def median_filter(self, image):
        row, col = image.shape
        canvas = np.zeros((row, col), np.uint8)
        for i in range(row):
            for j in range(col):
                if (self.border < i < row - self.border) and (self.border < j < col -
self.border):
                    canvas[i, j] = self.find_median(image, i, j)
                else:
                    canvas[i, j] = image[i, j]
        return canvas

    def low_pass_calculation(self, data, x, y):
        result = np.array([], np.int32)
        for i in range(self.size):
            for j in range(self.size):
                result = np.append(result, data[(x + i - self.border), (y + j - self.border)] *
self.normalize[i, j])
        result = result.astype('uint8')
        return sum(result)

    def low_pass_filter(self, image):
        row, col = image.shape
        canvas = np.zeros((row, col), np.uint8)
        for i in range(row):
            for j in range(col):
                if (self.border < i < row - self.border) and (self.border < j < col -
self.border):
                    canvas[i, j] = self.low_pass_calculation(image, i, j)
                else:
                    canvas[i, j] = image[i, j]
        return canvas

def main():
    image_raw = cv2.imread('LennaInput.png', 0)
    kernel = Kernel(size=3, matrix=np.array([
        [1, 1, 1],

```


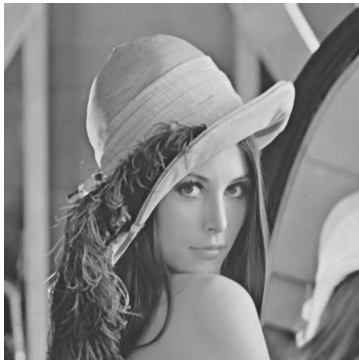

```

    [1, 1, 1],
    [1, 1, 1],
  ]))
  image_low_pass_filter = kernel.low_pass_filter(image=image_raw)
  image_median_filter = kernel.median_filter(image=image_raw)
  cv2.imwrite('LennaOutput_median.png', image_median_filter)
  cv2.imwrite('LennaOutput_low_pass.png', image_low_pass_filter)

if __name__ == '__main__':
  main()

```

Hasil :

Gambar Asli	Median Filter (Lebih Halus)	Low Pass Filter (Lebih Kasar)
		

5. Filter manakah yang lebih baik? Berikan alasan mengapa memilih filter tersebut!

Filter ini tidak lebih baik karena pada algoritma nya seluruh nilai tetangga memiliki pengaruh (baik besar maupun kecil terhadap nilai titik tengah) sehingga nilai pada titik tengah merupakan akumulasi yang mendekati rata-rata dari keseluruhan nilai tetangga yang akhirnya menyebabkan banyak nilai yang beragam (bisa memunculkan noise). Sedangkan untuk filter median, nilai pixel yang diambil hanyalah dari 1 nilai yang berada pada median saat data tetangga dan nilai pixel titik tengah diurutkan (hasil nya menjadi lebih halus karena banyak nilai yang seragam atau bisa dibilang tidak ada nilai baru yang dihasilkan).