

MODUL 6 – Histogram, Histogram Equalization, Dithering

A. TUJUAN

1. Mahasiswa mampu memahami mengenai histogram citra dan menerapkannya dalam program Python
2. Mahasiswa mampu memahami mengenai *histogram equalization* dan menerapkannya dalam program Python
3. Mahasiswa mampu memahami mengenai dithering citra dan menerapkannya dalam program Python

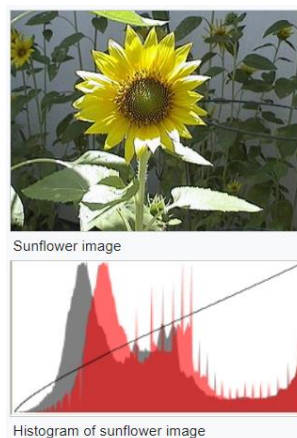
B. ALAT DAN BAHAN

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

C. ULASAN TEORI

C.1 Histogram Image

Citra Histogram adalah tipe histogram yang berfungsi sebagai representasi grafis dari distribusi intensitas warna pixel pada citra digital. Grafik histogram seperti pada gambar 1 akan menampilkan jumlah pixel dari tiap intensitas warna pixel. Dengan memperhatikan grafik histogram, seseorang dapat menilai distribusi warna pixel secara cepat. Untuk gambar berwarna histogram mewakili setiap channel warna yang ada, sedangkan untuk citra greyscale hanya terdiri dari satu histogram saja. Axis horizontal dari grafik histogram mewakili variasi nilai pixel, sedangkan axis vertikalnya mewakili jumlah pixel. Sebelah kiri dari axis horizontal mewakili area hitam dan gelap, bagian tengah mewakili area abu-abu, dan bagian kanan mewakili area terang dan putih.



Gambar 1. Contoh Citra Asli dan Histogram Citra Tersebut

Berikut adalah algoritma dalam membuat histogram dari citra digital

1	Siapkan dua form, form 1 berisi 1 chart yang digunakan untuk menampilkan histogram citra grayscale, form 2 berisi 3 chart yang digunakan untuk menampilkan histogram citra R, G, dan B.
2	Buat variable penyimpan histogram R, G, dan B Variabel dapat disimpan dalam array 2 dimensi atau menggunakan Dictionary pada Visual Studio. Tipe data untuk axis horizontal dapat menggunakan int atau Byte, sedangkan pada axis vertical, disarankan menggunakan tipe data Double. Hal ini dilakukan karena nilai histogramnya akan kita normalisasi. Pada percobaan ini data histogram kita normalisasi karena selanjutnya akan dilakukan proses perbaikan citra menggunakan histogram. Percobaan perbaikan citra menggunakan histogram akan kita lakukan pada jobsheet berikutnya.
3	Buat variable bertipe Bitmap untuk menyimpan nilai citra masukan
4	Beri nilai awal untuk tiap intensitas pixel dengan nilai 0
5	Untuk setiap baris dan kolom citra, periksa nilai pixel dan tambahkan 1 untuk tiap nilai pixel yang diperiksa
6	Periksa nilai dictionary, jika histogram dari R sama persis dengan G //Tampilkan form 1
7	Simpan nilai histogram dari dictionary ke dalam variable <i>kunci</i> bertipe data List
8	Tampilkan form 1
9	Untuk setiap nilai kunci
10	Normalisasi nilai histogram
11	Tambahkan data histogram ke dalam chart
12	Selesai
13	Jika tidak sama antara R dan G //Tampilkan form 2
14	Lakukan langkah yang sama seperti ketika menampilkan form 1
15	selesai

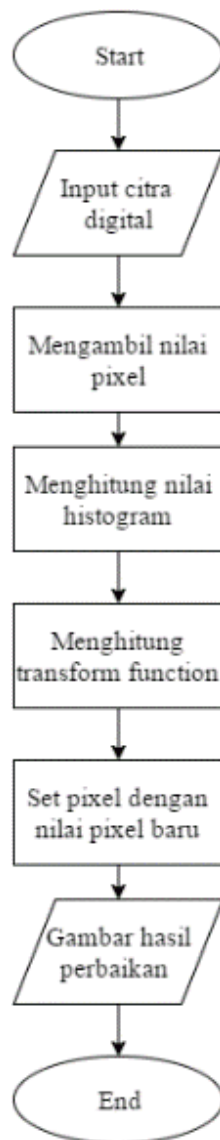
C.2 Histogram Equalization

Histogram Equalization adalah persamaan yang diimplementasikan pada sebuah citra *digital* di mana distribusi histogramnya akan lebih menyebar, dalam hal ini walaupun tidak dapat dibuktikan bahwa bentuk histogram-nya akan seragam namun dengan *Histogram Equalization* dapat dipastikan histogram-nya akan lebih merata. Perataan histogram diperoleh dengan mengubah derajat keabuan sebuah piksel (r) dengan derajat keabuan yang baru (s) dengan sebuah fungsi transformasi (T) (Gonzalez & Woods, 2002). *Histogram Equalization* dilakukan dengan membuat transform function dari histogram image yang telah dinormalisasi. Berikut adalah persamaan untuk transform function:

$$G(z_q) = (L - 1) \sum_{i=0}^q p_z(z_i)$$

Perhitungan manual untuk *Histogram Equalization* kali ini menggunakan 3-bit *image* ($L = 8$) dengan size 64x64 piksel ($MN = 4096$) dan distribusi intensitas akan diperlihatkan pada Tabel 1, dimana level intensitas adalah berupa nilai integer pada jarak $[0, L - 1] = [0, 7]$.

Gambar 2 menunjukkan proses perbaikan citra digital menggunakan *Histogram Equalization*.



Gambar 2. Flowchart jalannya perbaikan citra digital dengan HE

Pada Gambar 3 ditunjukkan original histogram sebelum dilakukan proses ekualisasi. Pada Gambar 4 menunjukkan hasil transform function Pada Gambar 5 ditunjukkan histogram hasil penghitungan dengan metode HE. Penghitungan *Histogram Equalization* menggunakan rumus:

$$s_0 = T(r_0) = 7 \sum_{j=0}^1 p_r(r_j) = 7 p_r(r_0) = 1.33$$

$$s_1 = T(r_1) = 7 \sum_{j=0}^1 p_r(r_j) = 7 p_r(r_0) + 7 p_r(r_1) = 3.08$$

$$\text{dan } s_2 = 4.55, s_3 = 5.67, s_4 = 6.23, s_5 = 6.65, s_6 = 6.86, s_7 = 7.00$$

Tabel 1 Distribusi intensitas dan nilai histogram untuk 3-bit *image*, 64 x 64 citra *digital*

r_k	n_k	$P_r(r_k) = n_k/MN$
$r_{0=0}$	790	0.19
$r_{1=1}$	1023	0.25
$r_{2=2}$	850	0.21
$r_{3=3}$	656	0.16
$r_{4=4}$	329	0.08
$r_{5=5}$	245	0.06
$r_{6=6}$	122	0.03
$r_{7=7}$	81	0.02

Dimana n_k = nilai piksel k , dan n adalah seluruh jumlah pixel pada suatu citra. Rumus lain yang dapat digunakan untuk menghitung *histogram equalization* adalah sebagai berikut :

$$K_o = \text{round} \left(\frac{C_i \cdot (2^k - 1)}{w \cdot h} \right)$$

Dimana :

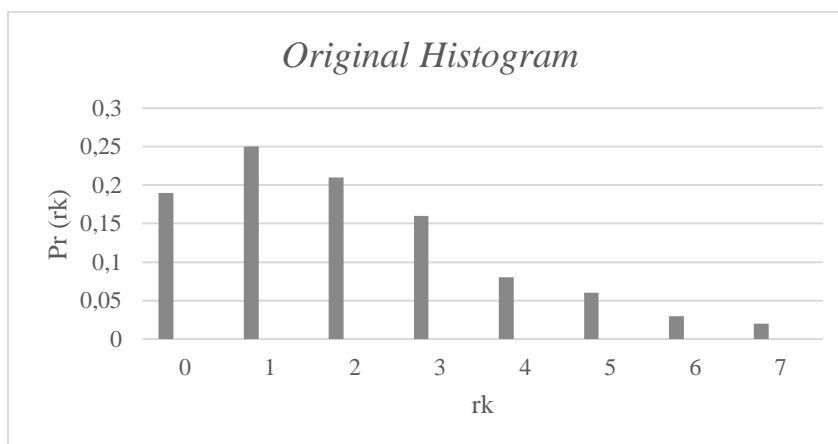
C_i = Distribusi Kumulatif nilai pixel

round = Fungsi pembulatan terdekat

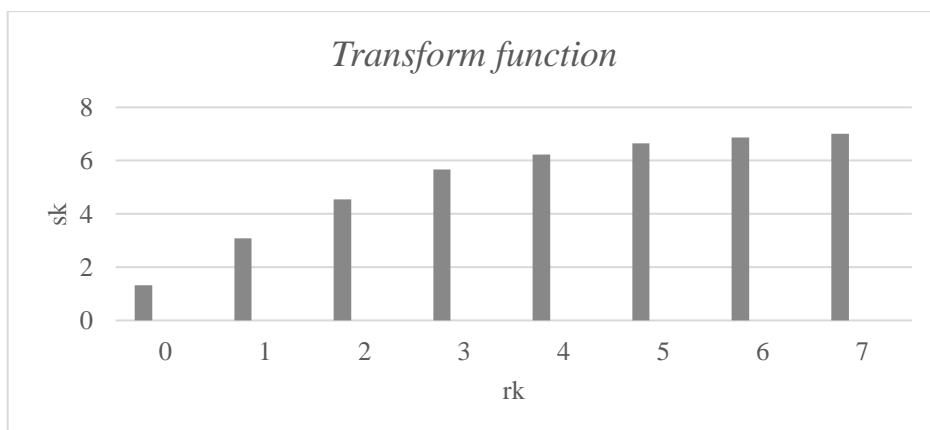
K_o = Nilai intensitas warna pixel hasil *Histogram Equalization*

w = lebar citra

h = tinggi citra



Gambar 3. Original histogram sebelum dilakukan proses ekualisasi



Gambar 4. Transform function



Gambar 5. Histogram hasil penghitungan dengan metode *Equalization Histogram*

Pada point ini, s memiliki nilai berbentuk pecahan maka nilai tersebut dibulatkan pada nilai integer yang paling dekat.

$$s_0 = 1.33 \rightarrow 1$$

$$s_0 = 6.23 \rightarrow 6$$

$$s_1 = 3.08 \rightarrow 3$$

$$s_0 = 6.65 \rightarrow 7$$

$$s_2 = 4.55 \rightarrow 5$$

$$s_0 = 6.86 \rightarrow 7$$

$$s_3 = 5.67 \rightarrow 6$$

$$s_0 = 7.00 \rightarrow 7$$

C.3 Image Dithering (*Error Difussion*)

Error Diffusion adalah salah satu tipe halftoning dimana proses kuantisasi didistribusi pada piksel tetangga yang belum diproses. Penggunaan utamanya adalah untuk mengubah kedalaman image menjadi lebih kecil, tetapi kualitas gambar tidak terlalu buruk. Proses ini

biasanya digunakan pada teknologi cetak, karena jumlah warna tinta yang tidak terlalu banyak sehingga ilusi warna terbentuk dari teknik ini.

Error diffusion digolongkan sebagai area operation, karena yang dilakukan oleh error diffusion pada satu pixel akan mempengaruhi hasil perhitungan operasi pixel lainnya. Error diffusion juga memiliki kemampuan untuk memperkuat tepi dari citra. Hal ini dapat membuat teks dalam citra menjadi lebih mudah terbaca daripada teknik halftoning yang lain.

Beberapa metode Error Diffusion diantaranya adalah:

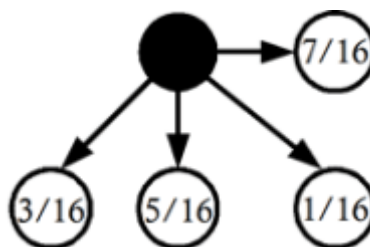
1. Floyd and Steinberg

Floyd dan Steinberg mendeskripsikan system pembentuk error diffusion pada citra digital dengan menggunakan kernel sederhana:

$$\frac{1}{16} \begin{bmatrix} - & \# & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Dimana – menunjukkan pixel yang sudah diproses, # menunjukkan pixel saat ini yang sedang diproses.

Atau dalam bentuk diagram node ditunjukkan dengan Gambar 6 berikut:



Gambar 6. Diagram Node Dithering Floyd and Steinberg

Rumus dithering Floyd and Steinberg memiliki persamaan sebagai berikut :

$$\text{Rumus pertama (R1)} \rightarrow I_{acc}(i+1, j) = I_{acc}(i+1, j) + \frac{7}{16}e$$

$$\text{Rumus pertama (R2)} \rightarrow I_{acc}(i+1, j+1) = I_{acc}(i+1, j+1) + \frac{3}{16}e$$

$$\text{Rumus pertama (R3)} \rightarrow I_{acc}(i, j+1) = I_{acc}(i, j+1) + \frac{5}{16}e$$

$$\text{Rumus pertama (R4)} \rightarrow I_{acc}(i-1, j+1) = I_{acc}(i-1, j+1) + \frac{1}{16}e$$

Dimana :

I_{acc} = Target pixel (pixel yang sedang diproses)

i = baris

j = kolom

e = error

Misalkan saja, untuk citra berukuran 3 x 3 pixel, maka dithering dilakukan sesuai dengan tahapan rumus yang sesuai dengan posisi yang akan dihitung seperti gambar 7 berikut.

X	R1		-	-	-	-	-	-
R3	R4		X	R1		-	-	-
			R3	R4		X	R1	
-	X	R1	-	-	-	-	-	-
R2	R3	R4	-	X	R1	-	-	-
			R2	R3	R4	-	X	R1
-	-	X	-	-	-	-	-	-
	R2	R3	-	-	X	-	-	-
				R2	R3	-	-	-

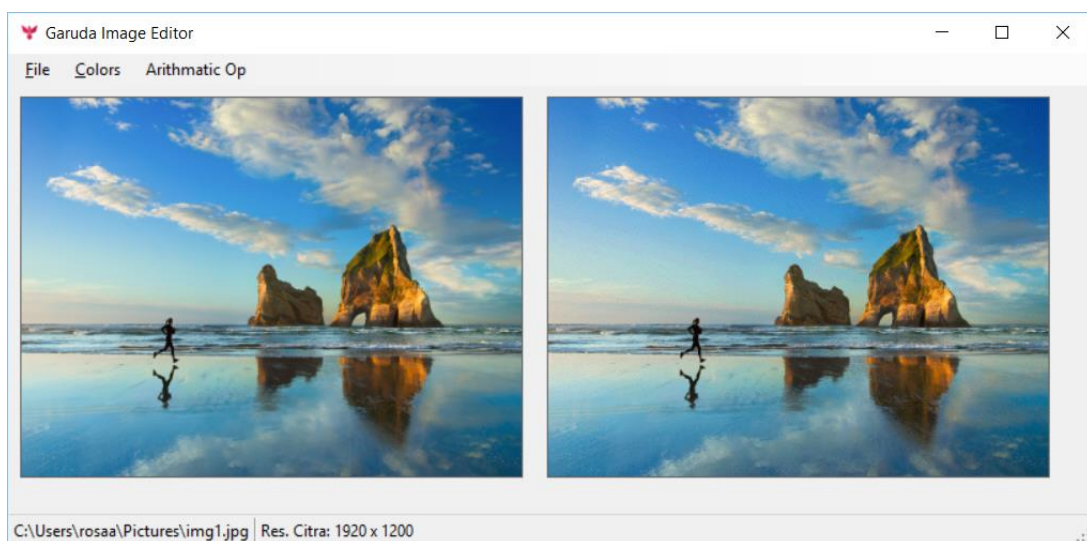
Gambar 7. Posisi Impelementasi rumus Dithering Floyd and Steinberg

- Jarvis, Judice dan Ninke dari Bell Labs mendeskripsikan system yang sama yang mereka istilahkan “minimized average error”, menggunakan kernel dengan ukuran yang lebih besar:

$$\frac{1}{48} \begin{bmatrix} - & - & \# & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

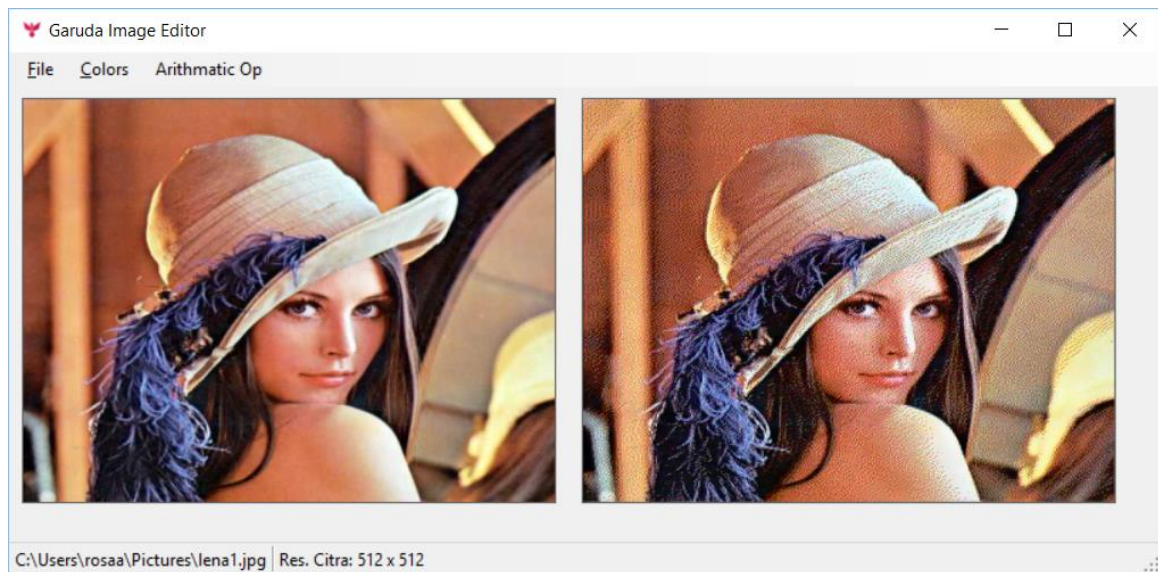
Jika ingin menyederhanakan jumlah warna tanpa error diffusion, maka cara yang termudah adalah dengan mencari warna terdekatnya, Jika input warna lebih dekat ke merah, maka warnanya akan diubah kemerah. Jika lebih dekat ke cyan, maka warnanya diubah ke cyan, dan seterusnya. Hasilnya terlihat pada gambar pada sub bab warna terdekat.

Perhatikan gambar 8 berikut sebelum dan sesudah disederhanakan kedalam 8 warna.



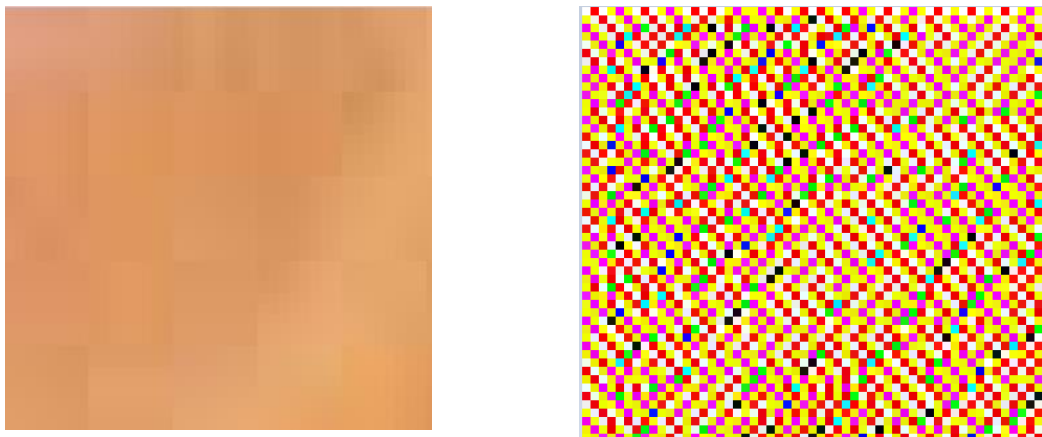
Gambar 8. Penyederhaan Warna Gambar Menjadi 8 Warna

Berikut adalah contoh yang lain:



Gambar 9. Contoh lain Penyederhaan Warna Gambar Menjadi 8 Warna

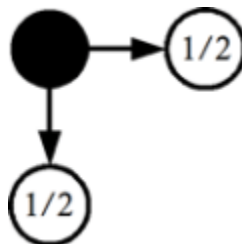
Dari gambar tersebut diambil bagian pojok kiri atas dan kemudian diperbesar sebagai berikut :



Gambar 10. Hasil Zoom Penyederhanaan Gambar Menjadi 8 Warna

Error diffusion bekerja dengan membandingkan warna asli pixel dengan warna terdekat yang ditentukan dan menghitung selisihnya. Selisih ini dinamakan dengan error. Porsi error ini kemudian dibagikan ke pixel tetangga sehingga menyebabkan errornya terdifusi dan dinamakan "Error Diffusion".

Bentuk paling sederhana dari error diffusion terlihat dari gambar berikut:



Gambar 10. Bentuk Sederhana Error Difussion

Dengan bentuk error diffusion seperti diatas, setengah error dari pixel yang diproses (ditunjukkan dengan titik hitam) terdifusi setengahnya ke pixel sebelah kanan dan setengahnya lagi ke pixel bawahnya. Pada error diffusion warna, proses ini harus dilakukan disemua channel red, green, dan blue.

Bagian penting yang perlu digarisbawahi adalah jumlah total error yang terdifusi tidak boleh melebihi satu. Hal penting lain yang perlu dicatat adalah ketika porsi error terdifusi ke tetangganya, tetangga tersebut memiliki nilai pixel antara 0 – 255. Jika nilainya diluar 0 – 255, maka nilai akhir perlu di truncate.

Berikut adalah pseudo-code untuk error difusi sederhana yang harus dilakukan setelah sebuah pixel(x,y) diubah nilainya ke warna terdekat:

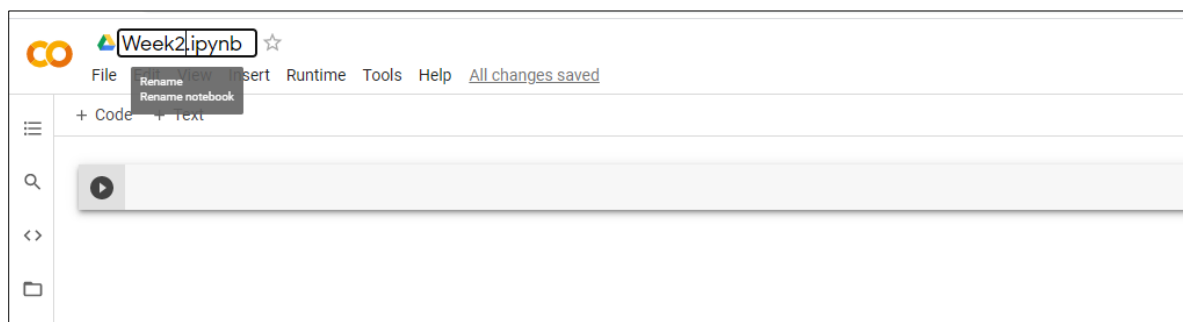
1	Error = warna asli - warnaTerdekat
2	SetPixel(x+1, y) = Truncate(GetPixel(x+1,y) + 0.5 * error)
3	SetPixel(x, y+1) = Truncate(GetPixel(x, y+1) + 0.5 * error)

Berikut adalah pseudo-code untuk error difusi Floyd and Steinberg yang harus dilakukan setelah sebuah pixel(x,y) diubah nilainya ke warna terdekat:

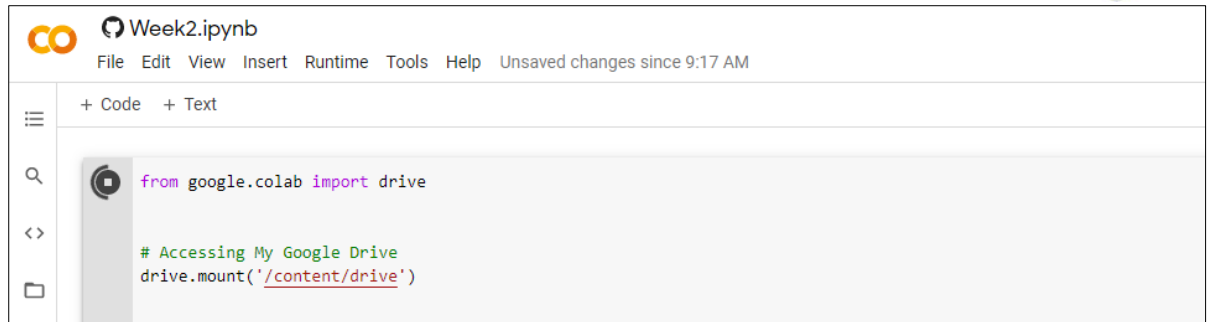
1	Error = warna asli - warnaTerdekat
2	SetPixel(x+1, y) = Truncate(GetPixel(x+1, y) + 7/16 * error)
3	SetPixel(x-1, y+1) = Truncate(GetPixel(x-1, y+1) + 3/16 * error)
4	SetPixel(x , y+1) = Truncate(GetPixel(x , y+1) + 5/16 * error)
5	SetPixel(x+1, y+1) = Truncate(GetPixel(x+1, y+1) + 1/16 * error)

D. TUGAS PRAKTIKUM

1. Buka <https://colab.research.google.com/>. Setelah dipastikan bahwa google Colab terhubung dengan Github Anda, lanjutkan dengan memilih repository yang telah digunakan pada praktikum minggu lalu, rename file menjadi “Week6.ipynb”.



Kemudian import folder yang ada di Drive Anda dengan cara sebagai berikut.



```

Week2.ipynb
File Edit View Insert Runtime Tools Help Unsaved changes since 9:17 AM

+ Code + Text

from google.colab import drive

# Accessing My Google Drive
drive.mount('/content/drive')
    
```

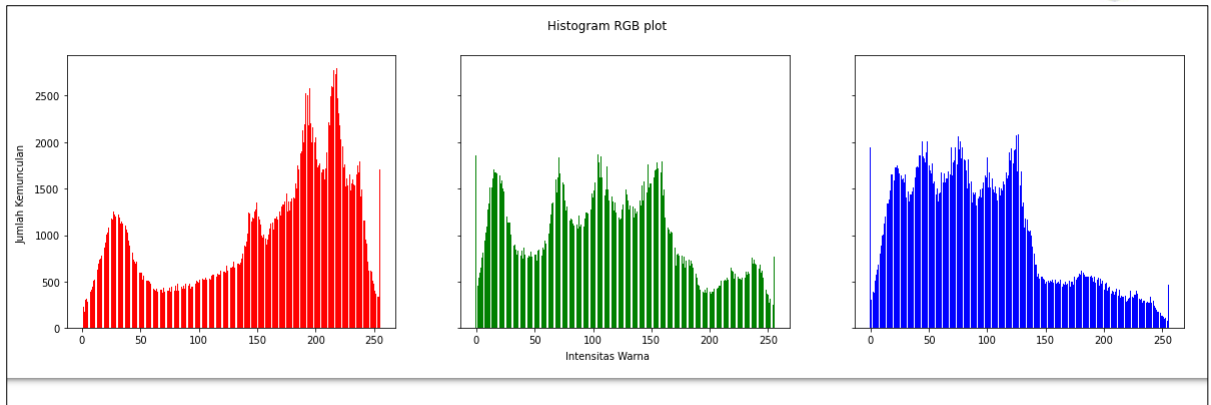
2. Import beberapa library berikut yang akan digunakan selama uji coba praktikum minggu ke-6 berikut.

```

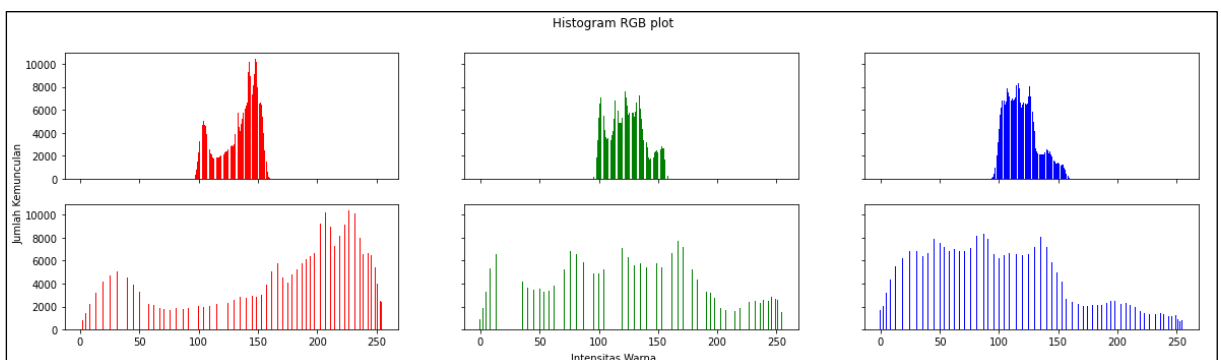
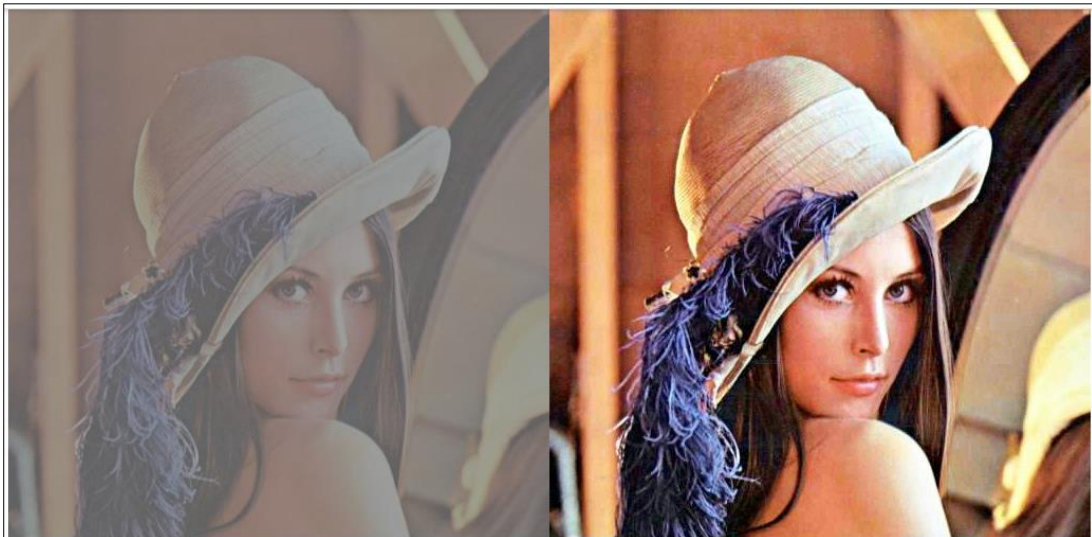
import cv2 as cv
from google.colab.patches import cv2_imshow
from skimage import io
import matplotlib.pyplot as plt
import numpy as np
import math
import os
import glob
    
```

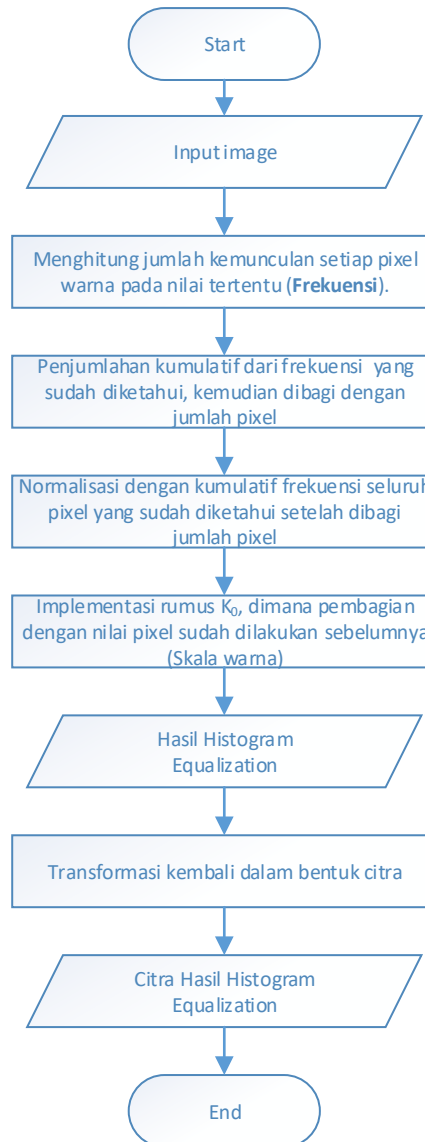
3. Buatlah histogram citra seperti output histogram berikut berdasarkan flowchart di bawah ini : (Gunakan gambar lena.jpg)





4. Setelah mengerjakan soal no. 3, buatlah histogram citra yang sama akan tetapi menggunakan library yang dimiliki oleh NumPy yaitu `"histogram"`. Bandingkan hasilnya. Apakah output muncul sama?
5. Buatlah histogram citra seperti output *histogram equalization* dan juga tampilan gambar sebelum dan sesudah proses *histogram equalization* berikut berdasarkan flowchart di bawah ini : (Gunakan gambar lena_lc.jpg)





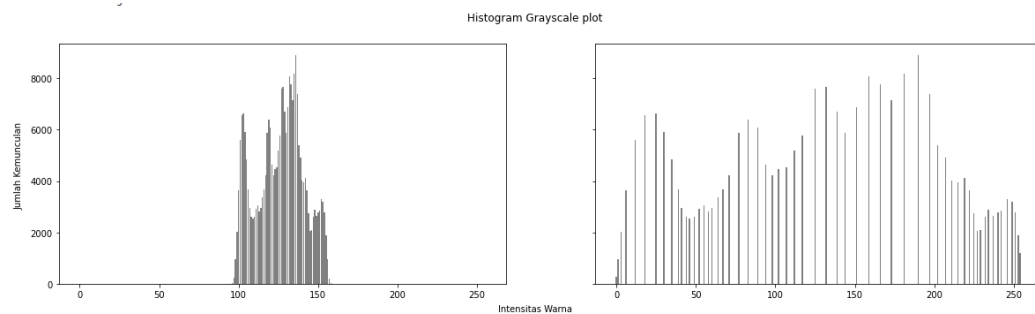
6. Setelah mengerjakan soal no. 5, buatlah histogram citra yang sama akan tetapi menggunakan library yang dimiliki oleh CV2 yaitu "equalizeHist". Bandingkan hasilnya. Apakah output muncul sama?
7. Lakukanlah proses pemetaan 16 juta warna RGB ke dalam 8 warna saja, yaitu hitam, hijau, kuning, biru, cyan, magenta, putih seperti output berikut berdasarkan flowchart di bawah ini! (Gunakan gambar lena.jpg)



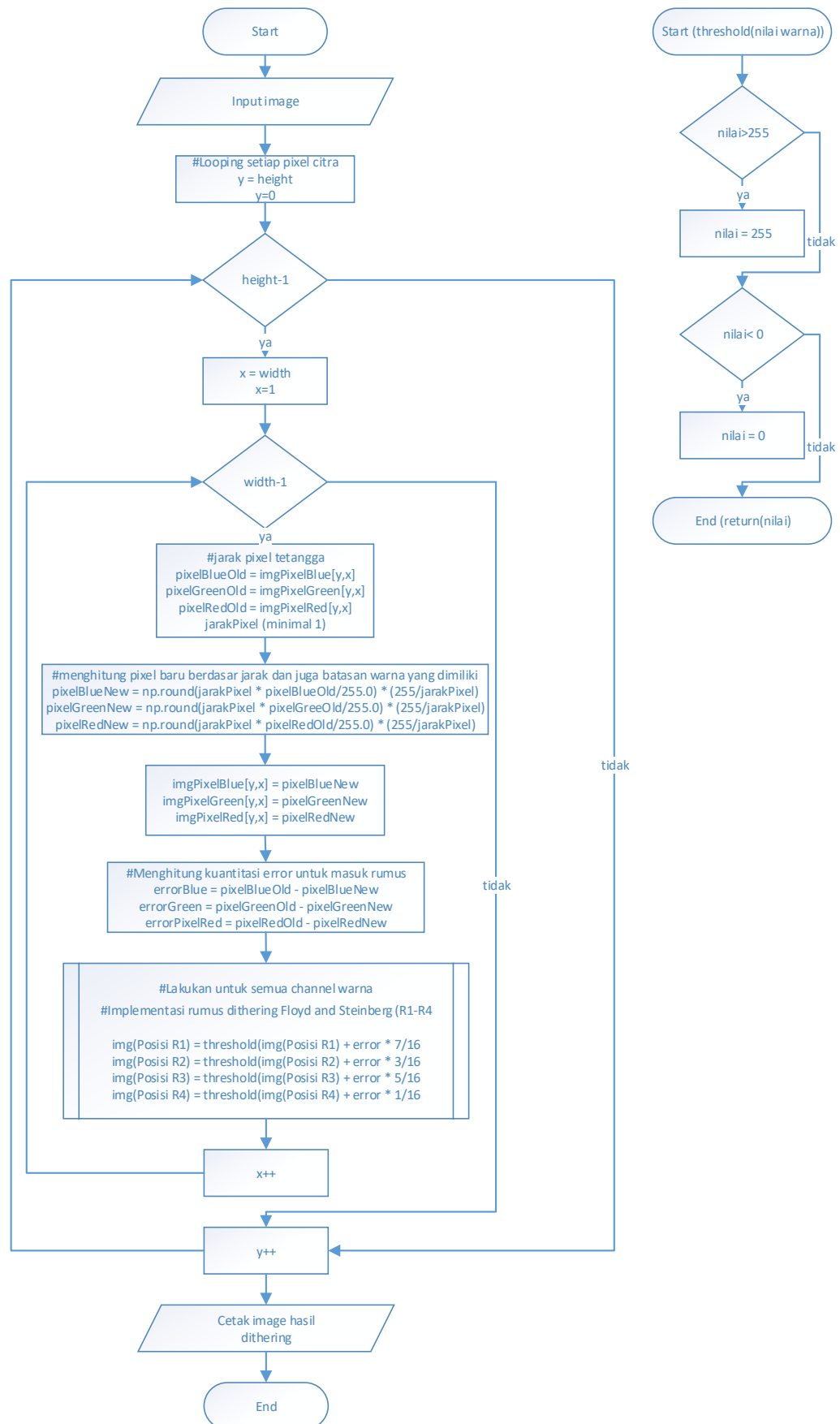
8. Lakukanlah proses dithering Floyd and Steinberg seperti output berikut (tampilan image awal, dan tampilan setelah dithering) berdasarkan flowchart di bagian bawah halaman modul ini! (Gunakan gambar lena.jpg)



9. Menggunakan lena_lc.jpg rubah menjadi grey image, kemudian terapkan *histogram equalization* sehingga terjadi perbaikan sebaran warna pada citra. Setelah itu dari hasil *histogram equalization* implementasikan proses dithering Floyd and Steinberg sehingga menampilkan gambar di bawah ini!







Flowchart dithering Floyd and Steinberg

--- SELAMAT BELAJAR ---