

## BAB I PENDAHULUAN

### 1.1. Latar Belakang

Pemrosesan citra digital digunakan dengan tujuan komputer yang dapat mengenali suatu bidang dengan penglihatan layaknya manusia. Penglihatan manusia telah diciptakan sedemikian rupa sehingga dapat mengenali objek sekitar, salah satunya ialah tekstur obyek. Tekstur merupakan atribut yang paling penting dalam aplikasi citra digital. Tekstur pada suatu obyek dapat dikenali manusia menggunakan indera penglihatan. Indera penglihatan mengenali tekstur dengan melihat pola (corak) yang ada pada permukaan obyek. Pada hal ini, sistem pemrosesan citra digital berusaha menganalisis citra agar dapat dikenali teksturnya. Tekstur adalah salah satu konsep yang sulit direpresentasikan dalam pengolahan citra digital. Pada umumnya, untuk mengidentifikasi tekstur suatu citra dilakukan pemodelan tekstur sebagai variasi skala keabuan 2 dimensi. Tidak seperti penglihatan manusia yang dapat dengan mudah mengenali tekstur, algoritma pengolahan citra digital rawan dengan kesalahan. Berbagai faktor yang ada serta keterbatasannya menjadi penyebab sering terjadinya kesalahan. Maka dari itu, pemrosesan citra digital masih terus dalam pengembangan mengenai algoritma yang digunakan. Dengan bantuan *software* pengolah citra, data digital dari citra dapat dianalisis menggunakan rumus tertentu agar didapat nilai angka yang menjadi hasil dari ekstraksi ciri suatu citra. Ekstraksi ciri ini yang nantinya akan menjadi kunci untuk mengenali bidang bertekstur.

Ada beberapa metode yang biasa diterapkan pada pemrosesan citra digital yakni salah satunya metode GLCM (*Gray Level Co-occurrence Matrix*) merupakan salah satu metode untuk ekstraksi tekstur pada citra. Ekstraksi tekstur dilakukan untuk mengambil informasi pokok dari suatu citra sebelum digunakan ke proses berikutnya. Metode GLCM menggunakan beberapa fitur pendekatan statistik seperti energi, entropi, kontras, dan sebagainya. GLCM dapat menentukan ciri dalam merepresentasikan karakteristik tekstur dari sebuah citra.

## 1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah penulis uraikan di atas, maka rumusan masalah yang akan dibahas dalam makalah ini adalah bagaimana merancang sistem temu kembali atau *information retrieval system* berbasis citra gambar menggunakan metode GLCM (*Gray Level Co-occurrence Matrix*).

## 1.3. Batasan Masalah

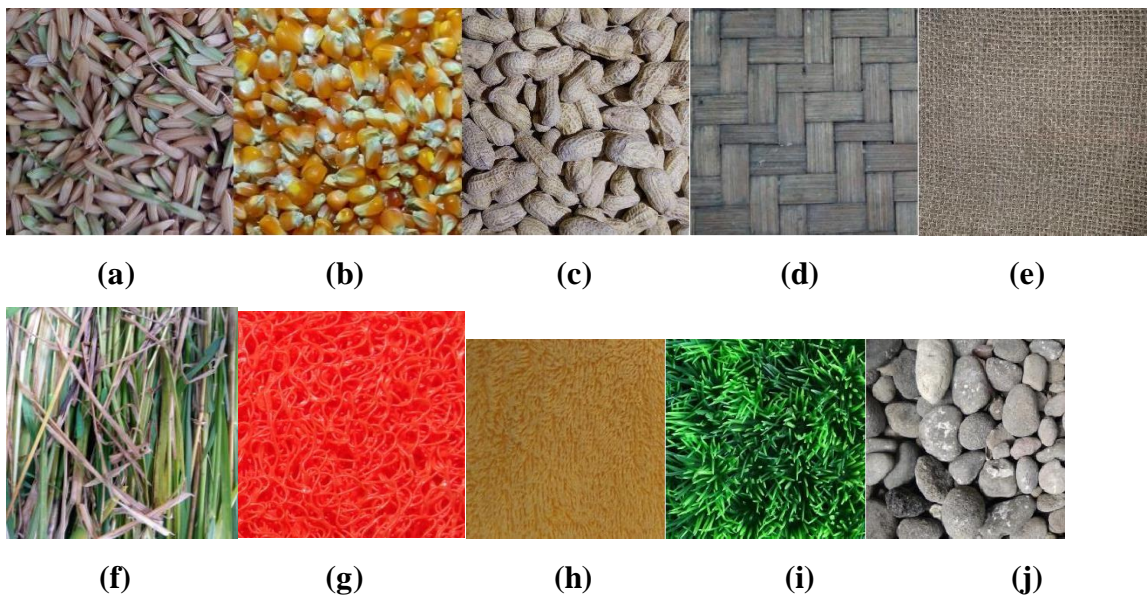
Berikut adalah batasan masalah dari proyek kami :

1. Menggunakan Metode GLCM dengan lima fitur tekstur; *Contrast*, *Dissimilarity*, *Asm*, *Homogeneity*, dan *Energy*.
2. Input data minimal satu gambar berekstensi .jpg
3. Information Retrieval sebatas menampilkan hasil matrix GLCM pada terminal dan hasil kalkulasi kelima fitur tekstur dalam aplikasi GUI berbasis bahasa pemrograman Python.

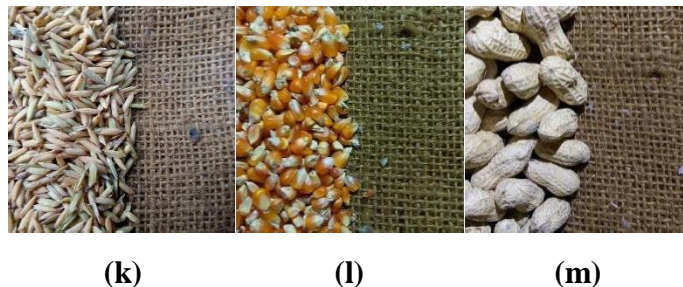
## BAB II LANDASAN TEORI

### 2.1. Tekstur

Menurut KBBI, tekstur adalah ukuran dan susunan (jaringan) bagian suatu benda. Dalam pengertian lain, tekstur merupakan jalinan atau penyatuan bagian-bagian sehingga membentuk suatu benda. Menurut Bhosle, tekstur merupakan pengulangan dari elemen atau pola pada suatu permukaan. Adanya pengulangan elemen pada suatu permukaan membuat suatu permukaan bisa dikatakan memiliki tekstur.

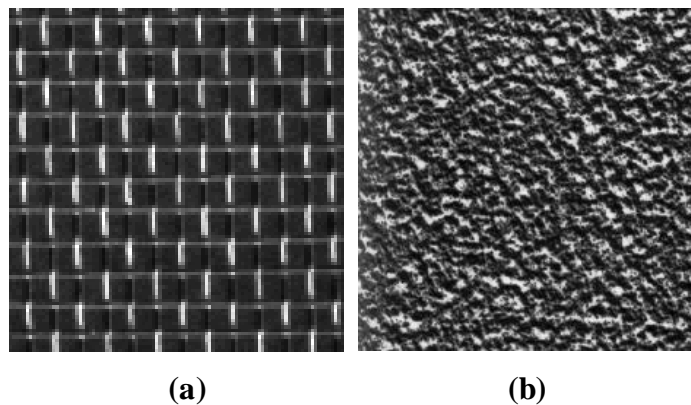


Gambar 2.1. Sepuluh tekstur tunggal: (a) gabah (b) jagung (c) kacang tanah (d) anyaman bambu (e) karung goni (f) keset (g) jerami (h) handuk (i) rumput, dan (j) bebatuan,



Gambar 2.2. Tiga tekstur ganda: (k) gabah dan karung goni (l) jagung dan karung goni, dan (m) kacang dan karung goni.

Dalam bukunya, Kadir mengambil definisi tekstur sebagai hubungan mutual antara nilai intensitas piksel ketetanggaan yang berulang di suatu area yang lebih luas daripada jarak hubungan ketetanggaan tersebut. Kadir juga membagi tekstur menjadi 2 kategori berdasarkan keteraturan pengulangan polanya, yaitu teratur dan tidak teratur. Tekstur teratur memiliki pola berulang yang selalu sama dan cenderung konstan. Tekstur tidak teratur memiliki pola berulang yang berbeda dan cenderung acak. Tekstur teratur dan tidak teratur dapat dilihat pada Gambar 2.3.



Gambar 2.3. Tekstur teratur (a) dan tekstur tidak teratur atau acak (b)  
(Gambar (a) dan (b) diambil dari Brodatz Textures)

## 2.2. Pengolahan Citra Digital

### 2.2.1. Citra Digital

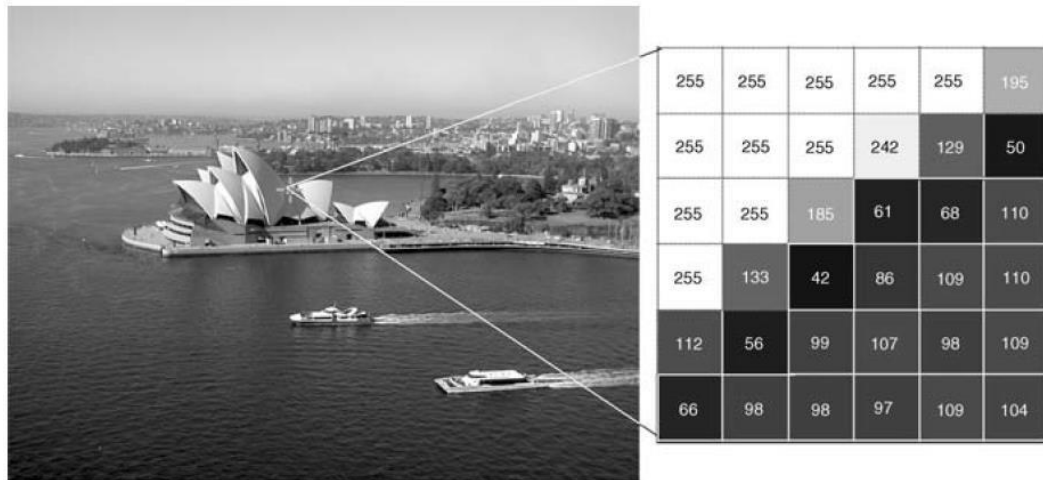
Citra digital dapat direpresentasikan sebagai matriks 2 dimensi dengan angka riil. Angka riil pada citra digital biasa disebut sebagai *picture elements* atau piksel. Tiap piksel direpresentasikan dengan satu atau lebih nilai numerik. Untuk citra monokrom (grayscale), nilai piksel tunggal menunjukkan intensitas piksel dengan rentang pada umumnya  $[0, 255]$ . Untuk citra berwarna, terdapat 3 nilai dalam satu piksel menunjukkan jumlah merah (R), hijau (G), dan biru (B). Matriks citra digital dengan ukuran  $M \times N$  piksel, dapat ditulis dengan fungsi  $f(x, y)$ . Pada fungsi ini,  $x$  adalah angka baris dari 0 hingga  $M-1$  dan  $y$  adalah angka kolom dari 0 hingga  $N-1$  (Gambar 2.7)

$$f(x, y) = \begin{bmatrix} f(0, 0) & f(0, 1) & \dots & f(0, N-1) \\ f(1, 0) & f(1, 1) & \dots & f(1, N-1) \\ \vdots & \vdots & \dots & \vdots \\ f(M-1, 0) & f(M-1, 1) & \dots & f(M-1, N-1) \end{bmatrix}$$

Gambar 2.7. Matriks 2 Dimensi Representasi Citra Digital

### 2.2.2. Citra *Grayscale*

Pada citra *Grayscale*, tiap koordinat piksel  $f(x, y)$  mencerminkan intensitas tingkat keabuan (*gray level*). Nilai maksimal dan minimal dari intensitas piksel bervariasi tergantung dari jenis data yang digunakan. Rentang yang biasa digunakan direpresentasikan: 0.0 (hitam) hingga 1.0 (putih) untuk tipe data *double* dan 0 (hitam) hingga 255 (putih) untuk tipe data *uint8* (*unsigned integer, 8 bits*). Sebagai contoh, nilai piksel citra *grayscale* dengan 6x6 ketetanggaan dapat dilihat pada Gambar 2.8



Gambar 2.8. Nilai piksel citra *grayscale* dengan 6x6 ketetanggaan

### 2.2.3. Citra RGB

Citra RGB merupakan citra berwarna yang representasinya lebih kompleks dan bervariasi. Citra RGB terdiri dari 3 dimensi array, yaitu  $M \times N \times 3$ .  $M$  dan  $N$  adalah dimensi panjang dan lebar citra sedangkan 3 adalah jumlah kanal warna. Terdapat 3 kanal warna, yaitu merah (R), hijau (G), dan biru (B). Setiap kanal warna mengandung nilai 8-bit, yang mengindikasikan skala merah, hijau, dan biru antara 0 hingga 255.

Tiap piksel memiliki 3 komponen kanal warna untuk merepresentasi warna. Kombinasi dari 3 kanal warna 8-bit menghasilkan angka 24-bit sehingga terdapat  $2^{24}$  (16.777.216) kombinasi warna. Representasi alternatif ialah menggunakan 32-bit per piksel dengan memasukkan kanal ke-4 yang disebut kanal *alpha*. Kanal tersebut digunakan untuk ukuran transparansi tiap piksel yang biasa digunakan dalam penyuntingan citra. Citra RGB dengan komponen R, G, dan B dapat dilihat pada Gambar 2.9.



Gambar 2.9. Citra RGB (a) dengan komponen R (b), G (c), dan B (d)

#### 2.2.4. Konversi Citra RGB menjadi Citra *Grayscale*

Pada banyak sistem pemrosesan citra, seringkali citra berwarna (RGB) perlu dikonversi menjadi citra dengan skala keabuan (*grayscale*). Citra berwarna pada dasarnya memiliki 3 layer warna yaitu merah, hijau, dan biru. Citra *grayscale* hanya memiliki 1 layer warna untuk skala keabuan. Untuk mengkonversi citra RGB menjadi citra *grayscale* menggunakan persamaan 2.1.

$$I = 0.2989 * R + 0.5870 * G + 0.1141 * B \quad (2.1)$$

Berdasarkan persamaan 2.1, I menyatakan citra *grayscale* hasil konversi, R menyatakan nilai kanal warna merah, G menyatakan nilai kanal warna hijau, dan B menyatakan nilai kanal warna biru.

### 2.3. Machine Vision Systems

*Machine Vision Systems* atau *Computer Vision Systems* ialah suatu sistem bagaimana komputer dapat mengenali objek layaknya penglihatan manusia (*Human Visual System*). Di dalam *Machine Vision Systems* secara umum, terdapat beberapa tahap dalam pengolahan suatu citra agar dapat diklasifikasikan, yaitu :

a. Tahap Akuisisi

Pada tahap ini, objek ditangkap menggunakan kamera dengan pengaturan pencahayaan untuk memastikan bahwa citra yang diambil dapat diproses lebih lanjut. Beberapa pertimbangan yang diambil pada tahap ini, yakni seperti jarak antara kamera dengan objek, pergeseran kamera terhadap objek, dan rotasi kamera terhadap objek.

b. Tahap *Preprocessing*

Tujuan dari tahap ini ialah untuk meningkatkan kualitas dari citra, seperti peningkatan kontras, koreksi kecerahan, dan *noise removal*.

c. Tahap Segmentasi

Pada tahap ini bertujuan untuk membagi citra menjadi beberapa bagian ke dalam komponen utama, yaitu objek relevan dan latar belakang. Tahap ini adalah salah satu tantangan dalam *Machine Vision System*.

d. Tahap Ekstraksi Fitur

Dalam tahap ini terdapat algoritma yang berfungsi untuk encode konten citra menjadi bentuk informasi yang deskriptif. Fitur yang paling relevan setelah citra melalui tahap segmentasi ialah ukuran distribusi (intensitas) warna, tekstur, dan bentuk. Fitur ini biasanya disebut dengan vektor fitur (*feature vector*) berupa indikator numerik dari konten citra. Fitur tersebut yang akan menjadi modal suatu konten citra dikenali (diklasifikasi).

e. Tahap Klasifikasi

Setelah citra melalui tahap ekstraksi dan didapat vektor fitur, selanjutnya ialah tahap pengklasifikasian atau pengenalan. Pemrosesan citra digunakan untuk mengenali pola (*pattern recognition*).

#### 2.4. Gray Level Co-occurrence Matrix

Metode *Gray Level Co-occurrence Matrix* (GLCM) adalah salah satu ekstraksi order kedua pada fitur statistik tekstur. Ekstraksi order kedua menunjukkan hubungan statistik antara 2 piksel. GLCM adalah sebuah matriks dengan jumlah baris dan kolom sebanding dengan jumlah *gray level* (G) dalam suatu citra. Metode GLCM menggunakan citra berskala keabuan (*grayscale*). Rumus matriks GLCM dapat dilihat pada Persamaan 2.2.

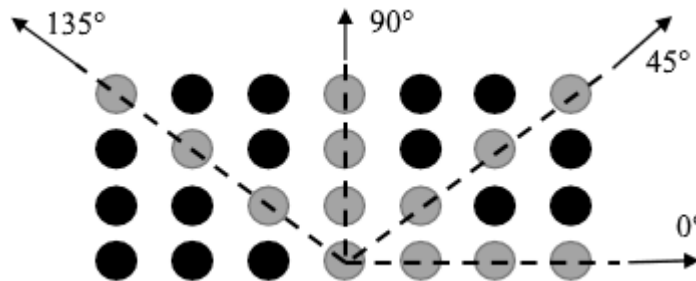
$$GLCM_{\vec{r}}(i, j) = \frac{|\{(x_1, y_1), (x_2, y_2) \in (N_x, N_y) \times (N_x, N_y) \mid f(x_1, y_1) = i \wedge f(x_2, y_2) = j \wedge \vec{r} = (x_2 - x_1, y_2 - y_1)\}|}{N_x N_y} \quad (2.2)$$

Sebagai contoh, matriks G adalah matriks referensi yang berpacitra *grayscale*. Nilai (i,j) merupakan nilai intensitas piksel. Dengan demikian, matriks GLCM adalah matriks frekuensi dengan elemen (i,j) merupakan jumlah dari hubungan ketetanggaan nilai piksel i dengan jarak dan sudut tertentu ( $\vec{r}$ ) terhadap nilai piksel j. Jumlah baris dan kolom matriks GLCM bergantung pada tingkat keabuan suatu citra. Karena nilai tingkat keabuan (*grayscale*) suatu citra antara 0 hingga 255, matriks GLCM bisa memiliki baris dan kolom sebesar 256 x 256.

Menurut Newsam [16], untuk mendapat hasil ekstraksi ciri GLCM ada 2 hal perlu yang dilakukan. Pertama ialah memasang piksel *co-occurences* spasial yang dipisahkan oleh sudut dan jarak tertentu yang ditabulasi menggunakan GLCM. Kedua ialah GLCM digunakan untuk menghitung kuantitas skalar yang memiliki karakteristik dengan aspek yang berbeda sesuai teksturnya. Nilai kuantitas ini merupakan hasil ekstraksi ciri dari GLCM yang digunakan untuk menginterpretasikan suatu tekstur.



GLCM memiliki 4 arah sudut dalam ketetanggaan antar piksel, yaitu  $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ , dan  $135^\circ$ . Sebagai ilustrasi, arah sudut ketetanggaan piksel GLCM dapat dilihat pada Gambar 2.10. Untuk sudut  $0^\circ$ , ketetanggaan piksel dihitung dengan jarak 1 piksel ke kanan. Untuk sudut  $45^\circ$ , ketetanggaan piksel dihitung dengan jarak 1 piksel ke kanan atas. Untuk sudut  $90^\circ$ , ketetanggaan piksel dihitung dengan jarak 1 piksel ke atas. Untuk sudut  $180^\circ$ , ketetanggaan piksel dihitung dengan jarak 1 piksel ke atas.



Gambar 2.10. Arah sudut ketetanggaan piksel pada metode GLCM

Contoh matriks berikut memperlihatkan penjelasan cara kerja metode GLCM serta hubungan ketetanggaan antar piksel.

0	1	0	1
2	3	1	1
0	2	3	2
2	1	3	1

Gambar 2.11. Contoh matriks 4x4 dengan variasi angka 0 sampai 3

Matriks disusun ulang berdasarkan hubungan antar piksel berdasarkan matriks *framework* di bawah. Matriks dengan hubungan antar piksel disebut matriks GLCM. Matriks tersebut terdiri dari 4 baris, yaitu baris 0, 1, 2, dan 3 serta 4 kolom, yaitu kolom 0, 1, 2, dan 3. Dimensi matriks GLCM sesuai dengan angka minimum pada matriks hingga angka maksimum pada matriks. Pada citra *grayscale*, angka minimum pada matriks ialah 0 dan angka maksimum pada matriks ialah 255. Dengan demikian pada matriks GLCM dengan citra *grayscale*, memiliki dimensi matriks 256x256. Matriks GLCM mula-mula diisi angka 0.

	0	1	2	3
0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)
3	(3,0)	(3,1)	(3,2)	(3,3)

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Gambar 2.12. Matriks *Framework* GLCM

Matriks di bawah (Gambar 2.13) memperlihatkan hubungan antar piksel dengan sudut ketetanggaan  $0^\circ$  yaitu piksel baris ke-1 kolom ke-1 dengan baris ke-1 kolom ke-2. Pada piksel tersebut terdapat angka 0 dan 1. Pasangan ketetanggaan (0,1) pada matriks *framework* GLCM bertambah 1.

0	1	0	1
2	3	1	1
0	2	3	2
2	1	3	1

	0	1	2	3
0	0	1	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Gambar 2.13. Ketetanggaan piksel (0,1) bertambah 1

Matriks di bawah (Gambar 2.14) memperlihatkan hubungan antar piksel dengan sudut ketetanggaan  $0^\circ$  yaitu baris ke-1 kolom ke-2 dengan baris ke-1 kolom ke-3. Pada piksel tersebut terdapat angka 1 dan 0. Pasangan ketetanggaan (1,0) pada matriks *framework* GLCM bertambah 1.

				0	1	2	3
	0			0	1	0	0
	1			1	0	0	0
	2			0	0	0	0
	3			0	0	0	0
0	1	0	1				
2	3	1	1				
0	2	3	2				
2	1	3	1				

Gambar 2.14. Ketetanggaan piksel (1,0) bertambah 1

Matriks di bawah (Gambar 2.15) memperlihatkan hubungan antar piksel dengan sudut ketetanggaan  $0^\circ$  yaitu baris ke-1 kolom ke-3 dengan baris ke-1 kolom ke-4. Pada piksel tersebut terdapat angka 0 dan 1. Pasangan ketetanggaan (1,0) pada matriks *framework* GLCM bertambah 1 menjadi 2.

				0	1	2	3
	0			0	2	0	0
	1			1	0	0	0
	2			0	0	0	0
	3			0	0	0	0
0	1	0	1				
2	3	1	1				
0	2	3	2				
2	1	3	1				

Gambar 2.15. Ketetanggaan piksel (0,1) bertambah 1

Demikian proses tersebut diulang hingga baris ke-4 sehingga didapat matriks GLCM sudut  $0^\circ$  seperti Gambar 2.16. Kemudian matriks GLCM dijumlah dengan *transpose* dari matriks itu sendiri agar matriks menjadi simetris. Setelah matriks GLCM simetris selanjutnya matriks GLCM dinormalisasi agar dapat dianalisis untuk menghitung masing-masing ekstraksi fitur menggunakan rumus. Penjumlahan matriks GLCM dengan matriks GLCM *transpose* dapat dilihat pada Gambar 2.17.

	0	1	2	3
0	0	2	1	0
1	1	1	0	1
2	0	1	0	2
3	0	2	1	0

Gambar 2.16. Matriks GLCM 4x4 sudut 0°

	Matriks GLCM sudut 0°					Matriks GLCM <i>transpose</i>			
	0	2	1	0		0	1	0	0
	1	1	0	1		2	1	1	2
+	0	1	0	2		1	0	0	1
	0	2	1	0		0	1	2	0
	0	3	1	0		0	3	1	0
	3	2	1	3		3	2	1	3
	1	1	0	3		1	1	0	3
	0	3	3	0		0	3	3	0

Matriks GLCM Simetris sudut 0°

Gambar 2.17. Matriks GLCM 4x4 sudut 0° setelah dibuat simetris

Setelah didapat matriks GLCM yang simetris, matriks GLCM melalui tahap normalisasi. Tiap nilai piksel matriks akan dibagi dengan jumlah piksel ketetanggaan. Pada Gambar 2.17, jumlah piksel ketetanggaan ialah 24. Untuk itu, tiap nilai piksel matriks GLCM akan dibagi dengan 24. Setelah matriks GLCM dinormalisasi, selanjutnya dapat dihitung menggunakan rumus untuk mendapat nilai ekstraksi ciri.

=	0	3/24	1/24	0
	3/24	2/24	1/24	3/24
	1/24	1/24	0	3/24
	0	3/24	3/24	0

0	0.125	0.042	0
0.125	0.083	0.042	0.125
0.042	0.042	0	0.125
0	0.125	0.125	0

Gambar 2.18. Matriks GLCM 4x4 sudut 0°

Setelah dinormalisasi Metode GLCM dapat menghasilkan setidaknya 5 ekstraksi ciri dari suatu citra digital tiap sudut ketetanggaan pikselnya. Lima besaran yang digunakan untuk mendapat hasil ekstraksi ciri tersebut antara lain: *angular second moment (ASM)*, kontras, *inverse different moment (IDM)*, entropi, dan korelasi. ASM merupakan ukuran homogenitas citra. Kontras merupakan ukuran keberadaan aras keabuan dalam citra. IDM juga digunakan untuk mengukur homogenitas. Entropi merupakan ukuran ketidakteraturan aras keabuan dalam citra. Korelasi merupakan ukuran ketergantungan linear antar nilai aras keabuan dalam citra.

ASM bisa disebut juga *uniformity* atau *energy*. ASM adalah penjumlahan pangkat dari elemen matriks GLCM. ASM memiliki nilai yang tinggi ketika citra memiliki homogenitas yang baik atau nilai piksel yang hampir serupa. Untuk mencari ekstraksi fitur ASM dapat digunakan rumus sebagai berikut.

$$ASM = \sum_{i=1}^L \sum_{j=1}^L (GLCM(i, j))^2 \quad (2.3)$$

Kontras bisa disebut juga *inertia*. Kontras adalah suatu ukuran intensitas aras keabuan antara piksel dengan piksel lainnya dengan lokasi relatif. Kontras memiliki batas nilai dari 0 hingga pangkat 2 dari panjang matriks GLCM simetris. Pada citra dengan elemen piksel yang bernilai sama secara keseluruhan, kontras bernilai 0. Untuk mencari ekstraksi fitur kontras dapat digunakan rumus sebagai berikut.

$$Kontras = \sum_{i=1}^L \sum_{j=1}^L (i - j)^2 GLCM(i, j) \quad (2.4)$$

IDM bisa disebut juga *homogeneity*. IDM adalah homogenitas lokal. IDM

berkaitan dengan kontras. Bobot IDM merupakan kebalikan dari bobot kontras. Untuk mencari ekstraksi fitur IDM dapat digunakan rumus sebagai berikut.

$$IDM = \sum_{i=1}^L \sum_{j=1}^L \frac{GLCM(i, j)}{1 + (i - j)^2} \quad (2.5)$$

Entropi adalah ukuran ketidakaturan aras keabuan dalam suatu citra. Nilai entropi akan semakin tinggi jika nilai elemen piksel citra semakin acak. Untuk mencari ekstraksi fitur entropi dapat digunakan rumus sebagai berikut.

$$Entropy = - \sum_{i=1}^L \sum_{j=1}^L GLCM(i, j) \log GLCM(i, j) \quad (2.6)$$

Korelasi dalam GLCM mengukur ketergantungan linear dari aras keabuan dalam ketetanggaan piksel citra. Untuk mencari ekstraksi fitur korelasi dapat digunakan rumus sebagai berikut.

$$Korelasi = \sum_{i=1}^L \sum_{j=1}^L \frac{(i - \mu_i) * (j - \mu_j) * GLCM(i, j)}{\sigma_i * \sigma_j} \quad (2.7)$$

dengan:

$$\mu_i = \sum_{i=1}^L \sum_{j=1}^L i * GLCM(i, j) \quad (2.8)$$

$$\mu_j = \sum_{i=1}^L \sum_{j=1}^L j * GLCM(i, j) \quad (2.9)$$

$$\sigma_i^2 = \sum_{i=1}^L \sum_{j=1}^L GLCM(i, j) (i - \mu_i)^2 \quad (2.10)$$

$$\sigma_j^2 = \sum_{i=1}^L \sum_{j=1}^L GLCM(i, j) (j - \mu_j)^2 \quad (2.11)$$

Rumus-rumus ekstraksi ciri tersebut dapat diterapkan pada matriks GLCM 4x4 sudut 0° (Gambar 2.19). Nilai piksel matriks GLCM 4x4 dapat dilihat melalui Tabel 2.3. Nilai piksel 0 tidak dihitung dalam rumus.

0	0.125	0.042	0
0.125	0.083	0.042	0.125
0.042	0.042	0	0.125
0	0.125	0.125	0

Gambar 2.19. Matriks GLCM 4x4 sudut 0°

Tabel 2.3. Tabel Nilai Pixel Matriks GLCM

GLCM	Nilai	(2,4)	0.125
(1,2)	0.125	(3,1)	0.042
(1,3)	0.042	(3,2)	0.042
(2,1)	0.125	(3,4)	0.125
(2,2)	0.083	(4,2)	0.125
(2,3)	0.042	(4,3)	0.125

Berikut ini adalah perhitungan kelima fitur untuk mendapat hasil ekstraksi ciri.

a. ASM

$$\begin{aligned}
 & \sum_{i=1}^L \sum_{j=1}^L GLCM(i, j)^2 \\
 &= \{ GLCM(1,2)^2 + GLCM(1,3)^2 + GLCM(2,1)^2 + GLCM(2,2)^2 + GLCM(2,3)^2 \\
 &\quad + GLCM(2,4)^2 + GLCM(3,1)^2 + GLCM(3,2)^2 + GLCM(3,4)^2 \\
 &\quad + GLCM(4,2)^2 + GLCM(4,3)^2 \} \\
 &= \{ 0.125^2 + 0.042^2 + 0.125^2 + 0.083^2 + 0.042^2 + 0.125^2 + 0.042^2 + 0.042^2 \\
 &\quad + 0.125^2 + 0.125^2 + 0.125^2 \} \\
 &= \mathbf{0.1076}
 \end{aligned}$$

b. Kontras

$$\begin{aligned}
 & \sum_{i=1}^L \sum_{j=1}^L (i-j)^2 GLCM(i, j) \\
 &= \{(1-2)^2 GLCM(1,2) + (1-3)^2 GLCM(1,3) + (2-1)^2 GLCM(2,1) \\
 &\quad + (2-2)^2 GLCM(2,2) + (2-3)^2 GLCM(2,3) + (2-4)^2 GLCM(2,4) \\
 &\quad + (3-1)^2 GLCM(3,1) + (3-2)^2 GLCM(3,2) + (3-4)^2 GLCM(3,4) \\
 &\quad + (4-2)^2 GLCM(4,2) + (4-3)^2 GLCM(4,3)\} \\
 &= \{0.125 + 4 * 0.042 + 0.125 + 0 + 0.042 + 4 * 0.125 + 4 * 0.042 + 0.042 + \\
 &\quad 0.125 \\
 &\quad + 4 * 0.125 + 0.125\} \\
 &= \mathbf{0.192}
 \end{aligned}$$

c. IDM

$$\begin{aligned}
 & \sum_{i=1}^L \sum_{j=1}^L i * GLCM(i, j) \\
 &= \{GLCM(1,2)/(1 + (1-2)^2) + GLCM(1,3)/(1 + (1-3)^2) + GLCM(2,1)/(1 \\
 &\quad + (2-1)^2) + GLCM(2,2)/(1 + (2-2)^2) + GLCM(2,3)/(1 + (2 \\
 &\quad - 3)^2) + GLCM(2,4)/(1 + (2-4)^2) + GLCM(3,1)/(1 + (3 \\
 &\quad - 1)^2) + GLCM(3,2)/(1 + (3-2)^2) + GLCM(3,4)/(1 + (3 \\
 &\quad - 4)^2) + GLCM(4,2)/(1 + (4-2)^2) + GLCM(4,3)/(1 + (4 \\
 &\quad - 3)^2)\} \\
 &= \{0.125/2 + 0.042/5 + 0.125/2 + 0.083/1 + 0.042/2 + 0.125/5 + 0.042/5 \\
 &\quad + 0.042/2 + 0.125/2 + 0.125/5 + 0.125/2\} \\
 &= \mathbf{0.441}
 \end{aligned}$$



d. Entropi

$$\begin{aligned}
 & \sum_{i=1}^L \sum_{j=1}^L GLCM(i, j) \log GLCM(i, j) \\
 &= -\{GLCM(1,2) \log GLCM(1,2) + GLCM(1,3) \log GLCM(1,3) \\
 &\quad + GLCM(2,1) \log GLCM(2,1) + GLCM(2,2) \log GLCM(2,2) \\
 &\quad + GLCM(2,3) \log GLCM(2,3) + GLCM(2,4) \log GLCM(2,4) \\
 &\quad + GLCM(3,1) \log GLCM(3,1) + GLCM(3,2) \log GLCM(3,2) \\
 &\quad + GLCM(3,4) \log GLCM(3,4) + GLCM(4,2) \log GLCM(4,2) \\
 &\quad + GLCM(4,3) \log GLCM(4,3)\} \\
 &= -\{0.125 \log 0.125 + 0.042 \log 0.042 + 0.125 \log 0.125 + 0.083 \log 0.083 \\
 &\quad + 0.042 \log 0.042 + 0.125 \log 0.125 + 0.042 \log 0.042 \\
 &\quad + 0.042 \log 0.042 + 0.125 \log 0.125 + 0.125 \log 0.125 \\
 &\quad + 0.125 \log 0.125\} \\
 &= \mathbf{0.9973}
 \end{aligned}$$

e. Korelasi

$$\begin{aligned}
 & \sum_{i=1}^L \sum_{j=1}^L i * GLCM(i, j) \\
 &= \{1 * GLCM(1,2) + 1 * GLCM(1,3) + 2 * GLCM(2,1) + 2 * GLCM(2,2) + 2 \\
 &\quad * GLCM(2,3) + 2 * GLCM(2,4) + 3 * GLCM(3,1) + 3 * \\
 &\quad GLCM(3,2) \\
 &\quad + 3 * GLCM(3,4) + 4 * GLCM(4,2) + 4 * GLCM(4,3)\} \\
 &= \{0.125 + 0.042 + 2 * 0.125 + 2 * 0.083 + 2 * 0.042 + 2 * 0.125 + 3 * 0.042 + \\
 &\quad 3 \\
 &\quad * 0.042 + 3 * 0.125 + 4 * 0.125 + 4 * 0.125\} \\
 &= \mathbf{2.554}
 \end{aligned}$$

$$\begin{aligned}
& \sigma^2 = \sum_{i=1}^L \sum_{j=1}^L GLCM(i, j)(i - \mu_{i'})^2 \\
& = \{GLCM(1,2) * (1 - 2.554)^2 + GLCM(1,3) * (1 - 2.554)^2 + GLCM(2,1) \\
& \quad * (2 - 2.554)^2 + GLCM(2,2) * (2 - 2.554)^2 + GLCM(2,3) \\
& \quad * (2 - 2.554)^2 + GLCM(2,4) * (2 - 2.554)^2 + GLCM(3,1) \\
& \quad * (3 - 2.554)^2 + GLCM(3,2) * (3 - 2.554)^2 + GLCM(3,4) \\
& \quad * (3 - 2.554)^2 + GLCM(4,2) * (4 - 2.554)^2 + GLCM(4,3) \\
& \quad * (4 - 2.554)^2\} \\
& = \{0.125 * (1 - 2.554)^2 + 0.042 * (1 - 2.554)^2 + 0.125 * (2 - 2.554)^2 + 0.083 \\
& \quad * (2 - 2.554)^2 + 0.042 * (2 - 2.554)^2 + 0.125 * (2 - 2.554)^2 \\
& \quad + 0.042 * (3 - 2.554)^2 + 0.042 * (3 - 2.554)^2 + 0.125 \\
& \quad * (3 - 2.554)^2 + 0.125 * (4 - 2.554)^2 + 0.125 * (4 - 2.554)^2\} \\
& = 1.0827
\end{aligned}$$

$$\begin{aligned}
& \mu_{j'} = \sum_{i=1}^L \sum_{j=1}^L j * GLCM(i, j) \\
& = \{2 * GLCM(1,2) + 3 * GLCM(1,3) + 1 * GLCM(2,1) + 2 * GLCM(2,2) + 3 \\
& \quad * GLCM(2,3) + 4 * GLCM(2,4) + 1 * GLCM(3,1) + 2 * \\
& \quad GLCM(3,2) \\
& \quad + 4 * GLCM(3,4) + 2 * GLCM(4,2) + 3 * GLCM(4,3)\} \\
& = \{2 * 0.125 + 3 * 0.042 + 0.125 + 2 * 0.083 + 3 * 0.042 + 4 * 0.125 + 0.042 + \\
& \quad 2 \\
& \quad * 0.042 + 4 * 0.125 + 2 * 0.125 + 3 * 0.125\} \\
& = 2.554
\end{aligned}$$

$$L \quad L$$

$$\begin{aligned}
i' \quad \sigma^2 &= \sum_{i=1} \sum_{j=1} GLCM(i, j)(i - \mu_{i'})^2 \\
&= \{GLCM(1,2) * (2 - 2.554)^2 + GLCM(1,3) * (3 - 2.554)^2 + \\
&\quad GLCM(2,1) \\
&\quad * (1 - 2.554)^2 + GLCM(2,2) * (2 - 2.554)^2 + GLCM(2,3) \\
&\quad * (3 - 2.554)^2 + GLCM(2,4) * (4 - 2.554)^2 + GLCM(3,1) \\
&\quad * (1 - 2.554)^2 + GLCM(3,2) * (2 - 2.554)^2 + GLCM(3,4) \\
&\quad * (4 - 2.554)^2 + GLCM(4,2) * (2 - 2.554)^2 + GLCM(4,3) \\
&\quad * (3 - 2.554)^2\} \\
&= \{0.125 * (2 - 2.554)^2 + 0.042 * (3 - 2.554)^2 + 0.125 * (1 - 2.554)^2 + 0.083 \\
&\quad * (2 - 2.554)^2 + 0.042 * (3 - 2.554)^2 + 0.125 * (4 - 2.554)^2 \\
&\quad + 0.042 * (1 - 2.554)^2 + 0.042 * (2 - 2.554)^2 + 0.125 \\
&\quad * (4 - 2.554)^2 + 0.125 * (2 - 2.554)^2 + 0.125 * (3 - 2.554)^2\} \\
&= 1.0827 \\
&= \{((1 - 2.554) * (2 - 2.554) * GLCM(1,2))/(1.0827 * 1.0827) + ((1 - 2.554) * (3 \\
&\quad - 2.554) * GLCM(1,3))/(1.0827 * 1.0827) + ((2 - 2.554) * (1 \\
&\quad - 2.554) * GLCM(2,1))/(1.0827 * 1.0827) + ((2 - 2.554) * (2 \\
&\quad - 2.554) * GLCM(2,2))/(1.0827 * 1.0827) + ((2 - 2.554) * (3 \\
&\quad - 2.554) * GLCM(2,3))/(1.0827 * 1.0827) + ((2 - 2.554) * (4 \\
&\quad - 2.554) * GLCM(2,4))/(1.0827 * 1.0827) + ((3 - 2.554) * (1 \\
&\quad - 2.554) * GLCM(3,1))/(1.0827 * 1.0827) + ((3 - 2.554) * (2 \\
&\quad - 2.554) * GLCM(3,2))/(1.0827 * 1.0827) + ((3 - 2.554) * (4 \\
&\quad - 2.554) * GLCM(3,4))/(1.0827 * 1.0827) + ((4 - 2.554) * (2 \\
&\quad - 2.554) * GLCM(4,2))/(1.0827 * 1.0827) + ((4 - 2.554) * (3 \\
&\quad - 2.554) * GLCM(4,3))/(1.0827 * 1.0827)\} \\
&= \{((1 - 2.554) * (2 - 2.554) * 0.125)/(1.0827 * 1.0827) + ((1 - 2.554) * (3 \\
&\quad - 2.554) * 0.042)/(1.0827 * 1.0827) + ((2 - 2.554) * (1 - \\
&\quad 2.554)
\end{aligned}$$

$$\begin{aligned}
& * 0.125)/(1.0827 * 1.0827) + ((2 - 2.554) * (2 - 2.554) \\
& * 0.083)/(1.0827 * 1.0827) + ((2 - 2.554) * (3 - 2.554) \\
& * 0.042)/(1.0827 * 1.0827) + ((2 - 2.554) * (4 - 2.554) \\
& * 0.125)/(1.0827 * 1.0827) + ((3 - 2.554) * (1 - 2.554) \\
& * 0.042)/(1.0827 * 1.0827) + ((3 - 2.554) * (2 - 2.554) \\
& * 0.042)/(1.0827 * 1.0827) + ((3 - 2.554) * (4 - 2.554) \\
& * 0.125)/(1.0827 * 1.0827) + ((4 - 2.554) * (2 - 2.554) \\
& * 0.125)/(1.0827 * 1.0827) + ((4 - 2.554) * (3 - 2.554) \\
& * 0.125)/(1.0827 * 1.0827)
\end{aligned}$$

$$= \mathbf{0.1047}$$

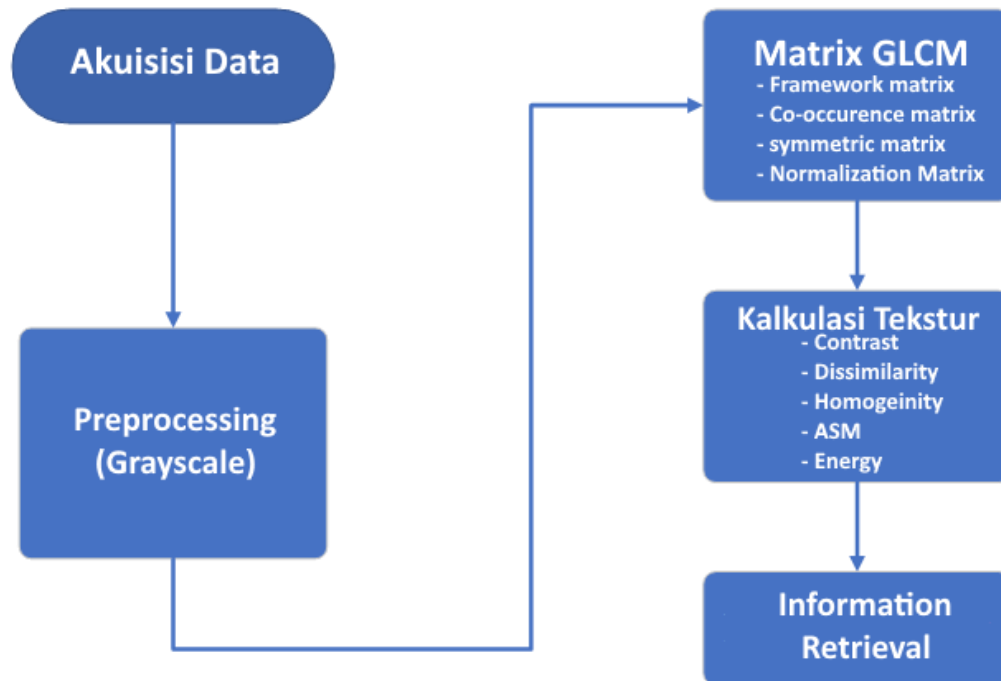
Dengan demikian, ekstraksi ciri matriks GLCM 4x4 sudut  $0^\circ$  ialah sebagai berikut:

- i. ASM = 0.1076
- ii. Kontras = 0.192
- iii. IDM = 0.4418
- iv. Entropi = 0.9973
- v. Korelasi = 0.1047

Tiap sudut GLCM memiliki 5 ekstraksi fitur. Dalam satu citra dapat digunakan 4 sudut ketetanggaan piksel. Dengan demikian, dalam satu citra dapat menghasilkan 20 ekstraksi fitur yang memiliki karakternya masing-masing. Ekstraksi fitur tersebut kemudian disimpan dalam bentuk matriks sendiri yang bisa disebut matriks ekstraksi ciri GLCM.

## BAB III PEMBAHASAN SISTEM

### 3.1. Workflow Sistem



Pada sistem kami terdapat 5 alur kerja (*WorkFlow*) :

1. Akuisisi Data

Pada Proses ini kami mencari file sumber berupa gambar berekstensi .jpg untuk diuji coba, dilanjutkan dengan studi kasus dan pembelajaran pemrograman Python untuk nantinya bisa melakukan implementasi.

2. Preprocessing (*Grayscale*)

Pada proses ini kami melakukan preprocessing file inputan berupa gambar untuk diubah menjadi format grayscale dan mendapatkan array matrixnya.

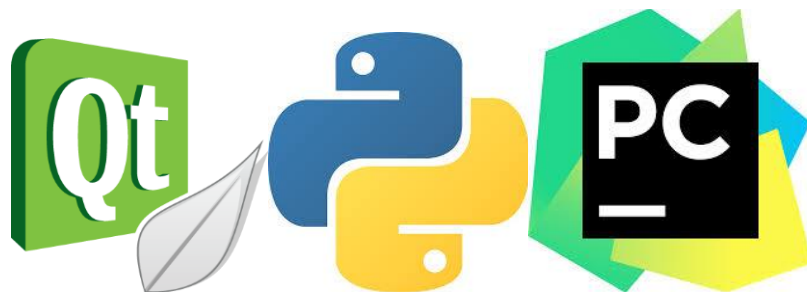
3. Matrix GLCM

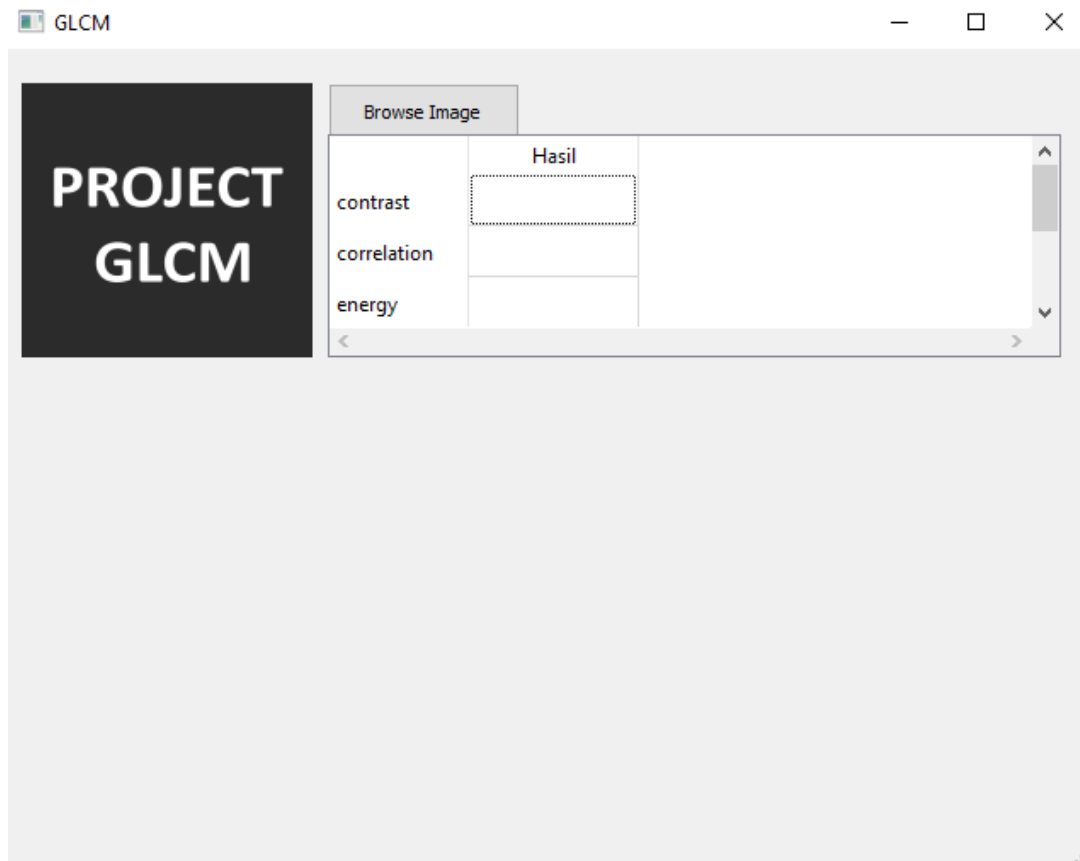
Setelah melewati proses *Preprocessing* dan mendapatkan array matrix dari gambar yang diinput, kami melanjutkan pada proses Pembentukan Matrix GLCM yang dibagi menjadi empat tahapan :

- Pembuatan framework matrix
  - Pembuatan co-occurrence matrix (mengisi framework matrix)
  - Pembuatan symmetric matrix (penjumlahan co-occurrence matrix dengan transpose matrix)
  - Matrix normalization yang akan menghasilkan nilai matrix antara 0–1
4. Kalkulasi Tekstur
- Pada tahap ini setelah mendapatkan matrix GLCM normalisasi, kami lanjutkan dengan kalkulasi lima fitur tekstur yang meliputi *Contrast*, *Dissimilarity*, *Homogeneity*, *ASM*, dan *Energy* yang masing-masing ada formulanya tersendiri.
5. Information Retrieval
- Pada tahap terakhir kami bisa menampilkan hasil dari kalkulasi tekstur pada aplikasi GUI.

### 3.2. Kebutuhan Sistem dan Tampilan Antarmuka

Implementasi GUI pada sistem kami menggunakan bantuan aplikasi QtDesigner yang nantinya bisa dihubungkan dengan backend python menggunakan library PyQt5. IDE yang kami gunakan adalah JetBrains Pycharm.





Gambar diatas adalah hasil GUI yang kami desain di dalam aplikasi QtDesigner.

### 3.3. Penjelasan Koding

#### 3.3.1. Library

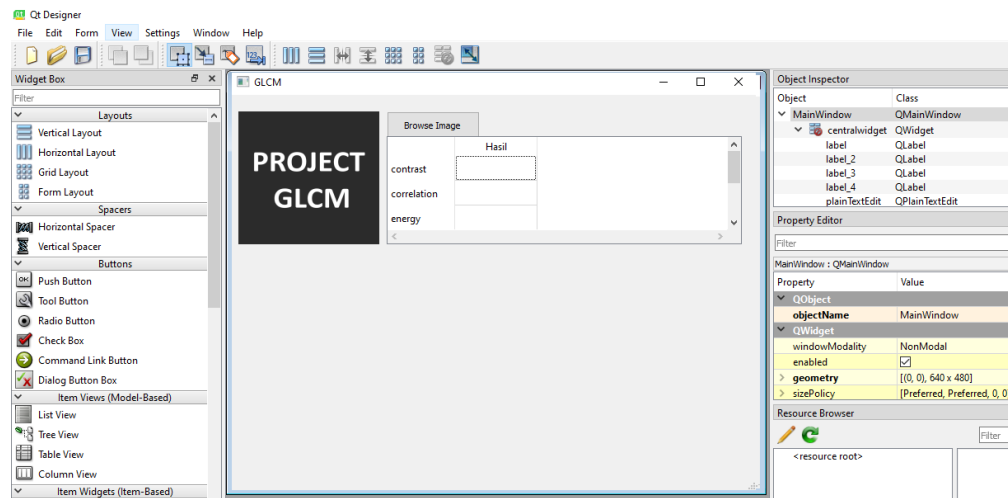
```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog
from PyQt5.QtGui import QPixmap
import numpy as np
import math
from PIL import Image
```

Penjelasan Kode Program Diatas :

#### 1. Import Modul Library PyQt5

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog
```

Keterangan : PyQt5 adalah modul library yang digunakan untuk memfasilitasi pemanggilan platform GUI perangkat Qt (toolkit untuk pengembangan aplikasi grafis yang bersifat lintas-platform.) untuk bahasa pemrograman Python. Karena GUI yang kami implementasikan menggunakan Aplikasi QtDesigner sebagai berikut :



Sehingga untuk dapat menghubungkan desain Frontend GUI diatas dengan backend pemrograman Python, kami memerlukan Library **PyQt5**. Library ini memiliki properti seperti `QtCore`, `QtGui`, `QtWidgets` yang bisa dipanggil sebagai object class disetiap pemanggilan properti GUI seperti *Button*, *Plaintext*, dan *label*. Pada modul properti `QtWidgets` kami juga memanggil properti lain seperti `QFileDialog` yang berguna untuk memanggil objek *opendialog* saat user melakukan input data. Modul `QPixmap` yang kami import dari `QtGui` berguna untuk menampilkan file gambar.

## 2. Import Modul Library *numpy* as *np* dan *math*

```
import numpy as np
```

```
import math
```

Keterangan : **NumPy** (*Numerical Python*) adalah library Python yang fokus pada *scientific computing*. NumPy memiliki kemampuan untuk membentuk objek N-dimensional *array*, yang mirip dengan *list* pada Python.

Keunggulan NumPy array dibandingkan dengan *list* pada Python adalah konsumsi *memory* yang lebih kecil serta *runtime* yang lebih cepat. NumPy juga



memudahkan kita pada Aljabar Linear, terutama operasi pada Vector (1-d array) dan Matrix (2-d array). Pada proyek ini kami menggunakan modul *Numpy* untuk mempermudah perhitungan array dimensional pada matrix. Penggunaan *as* akan memanggil *numpy* dengan prefix *np* pada proses berikutnya. Kemudian kami menggunakan Modul *math* untuk memanggil fungsi-fungsi matematika seperti *pow*, *sqrt* atau *log*.

### 3. Import Modul *from PIL import Image*

Keterangan : Modul *Pillow* digunakan untuk bisa melakukan pemrosesan dengan file gambar berekstensi “.jpg, .png, .gif, dll..” kami menggunakan modul ini untuk bisa melakukan input file gambar ke dalam sistem.

## 3.3.2. Fungsi

### 3.3.2.1. Class *Ui\_MainWindow*

Ini merupakan class yang dipergunakan sebagai inisialisasi awal dari operasi array dan matrix, di class ini semua array dan matrix diberikan inisialisasi nilai 0.

```
class Ui_MainWindow(object):
    arrFromImg = 0
    maxVal = 0
    occMatrix = 0
    transMatrix = 0
    sumMatrix = 0
    normMatrix = 0
    asm = 0
```

### 3.3.2.2. Fungsi *SetupUi*

Fungsi ini adalah markup atau setup GUI yang diambil dari library PyQt5.

```
def setupUi(self, MainWindow):
    arrFromImg = 0
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(640, 480)
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setGeometry(QtCore.QRect(10, 20, 171, 161))
    self.label.setText("")
```

```

self.label.setPixmap(QtGui.QPixmap("kalehub-2.png"))
self.label.setScaledContents(True)
self.label.setObjectName("label")
self.pushButton = QtWidgets.QPushButton(self.centralwidget)
self.pushButton.setGeometry(QtCore.QRect(190, 20, 113, 32))
self.pushButton.setObjectName("pushButton")
self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)
self.tableWidget.setGeometry(QtCore.QRect(190, 50, 431, 131))

self.tableWidget.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)

self.tableWidget.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
self.tableWidget.setRowCount(4)
self.tableWidget.setColumnCount(1)
self.tableWidget.setObjectName("tableWidget")
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setVerticalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setVerticalHeaderItem(1, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setVerticalHeaderItem(2, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setVerticalHeaderItem(3, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(1, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(2, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(3, item)
item = QtWidgets.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(4, item)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 640, 24))
self.menubar.setObjectName("menubar")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)

```

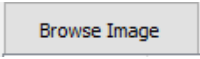
### 3.2.2.3. Fungsi retranslateUi

Fungsi ini digunakan untuk mengubah fungsi *setupUI* sebelumnya (diatas) menjadi sebuah objek di python.

```
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("Image", "GLCM"))
    self.pushButton.setText(_translate("MainWindow", "Browse
Image"))
    # ketika button di click
    self.pushButton.clicked.connect(self.browseImage)
    item = self.tableWidget.verticalHeaderItem(0)
    item.setText(_translate("MainWindow", "contrast"))
    item = self.tableWidget.verticalHeaderItem(1)
    item.setText(_translate("MainWindow", "correlation"))
    item = self.tableWidget.verticalHeaderItem(2)
    item.setText(_translate("MainWindow", "energy"))
    item = self.tableWidget.verticalHeaderItem(3)
    item.setText(_translate("MainWindow", "homogeneity"))
    item = self.tableWidget.horizontalHeaderItem(0)
    item.setText(_translate("MainWindow", "Hasil"))
```

### 3.2.2.4. Fungsi browseImage

Fungsi ini berfungsi untuk mendapatkan informasi input file image dari *user*.

Ketika *user* mengakses *button* , nama *button* tersebut adalah *pushbutton2* mengacu pada kode program

```
self.pushButton.clicked.connect(self.browseImage)
```

pada fungsi *SetupUi* sebelumnya, maka program akan memanggil Fungsi *browseImage*.

```
def browseImage(self):
    img, _ = QFileDialog.getOpenFileName(
        MainWindow,
        "Open image file",
        "/Users/firyan2903/Documents/imgProc",
        "Image files (*.jpg)",
    )
    loadImg = Image.open(img).resize((200, 200))
    arrFromImg = toGrayScale(loadImg)
    print(arrFromImg)

    # update pixmap
```

```

self.updatePixmap(arrFromImg, img)

# nilai terbesar
# maxVal = np.amax(imgArr)
maxVal = np.amax(arrFromImg)
maxVal = int(maxVal)
print(maxVal)

# membuat matriks baru
newMatrix = self.getNewMatrix(maxVal)
print("Matrix baru : ", newMatrix)

# find matrix occurences
occMatrix = self.findOccurences(arrFromImg, newMatrix)
print("Matrix Occurence : ", occMatrix)

# matrix tranpose
transMatrix = np.transpose(occMatrix)
print("Matrix transpose : ", transMatrix)

# sum of matrix occurences and matrix transpose
sumMatrix = np.add(occMatrix, transMatrix)
print("Matrix penjumlahan : ", sumMatrix)

# matrix normalization
normMatrix = self.normalizeMatrix(sumMatrix)
print("Matrix normalisasi : ", normMatrix)

# -----calculating texture-----

# contrast
self.calcContrast(normMatrix)
# dissimilarity
self.calcDiss(normMatrix)
# homogeneity
self.calcHomo(normMatrix)
# asm
asm = self.cac1Asm(normMatrix)
# energy
self.cac1Energy(asm)

```

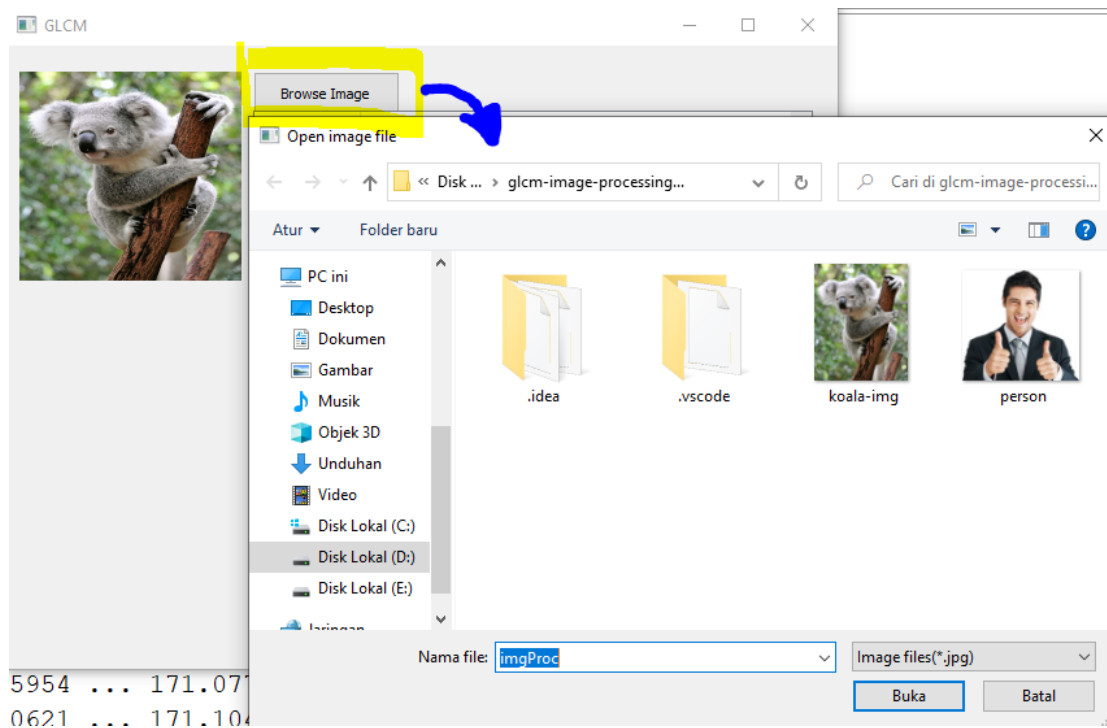
#### Keterangan Kode Program Diatas :

Untuk menampung file yang dipilih *user* sebagai input, file akan disimpan/diinisialisasikan sebagai dua variabel *img* dan notasi *underscore \_* yang akan diteruskan ke `QFileDialog.getOpenFileName()`. didalamnya membawa dua parameter, pertama dari *MainWindow* untuk membuka jendela windows untuk memudahkan kita

memilih gambar dan terakhir ada parameter yang mengarahkan direktori file *image* yang akan dipilih.

```
img, _ = QFileDialog.getOpenFileName (
    MainWindow,
    "Open image file",
    "/Users/firyan2903/Documents/imgProc",
    "Image files (*.jpg)",
)
```

Berikut adalah jendela windows yang mengarahkan pada direktori yang sudah dicantumkan pada parameter kedua :



Kemudian untuk bisa memuat file gambar dan memprosesnya menjadi text retrieval, kita perlu *function Image.open* dari library PILLOW atau ditulis PIL diawal deklarasi import library, berikut adalah potongan kode programnya :

```
loadImg = Image.open(img).resize((200, 200))
    arrFromImg = toGrayScale(loadImg)
    print(arrFromImg)
```

Potongan kode program diatas akan memuat parameter *img* sebagai file yang diinputkan dan ukuran filenya akan disesuaikan mengikuti lebar kali panjang 200 pixel untuk mempersingkat proses pemuatan. Setelahnya *img* akan dimasukkan pada *array* baru dalam format *GrayScale* dengan memanggil *function toGrayScale* yang memuat parameter *loadImg*.

*function toGrayScale* diletakkan di *function main* hal ini disebabkan karena function ini tidak terikat class tertentu.

```
if __name__ == "__main__":
    import sys

    def toGrayScale(img):
        rgb = np.array(img)
        return np.dot(rgb[... , :3], [0.2989, 0.5870,
0.1140])

    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

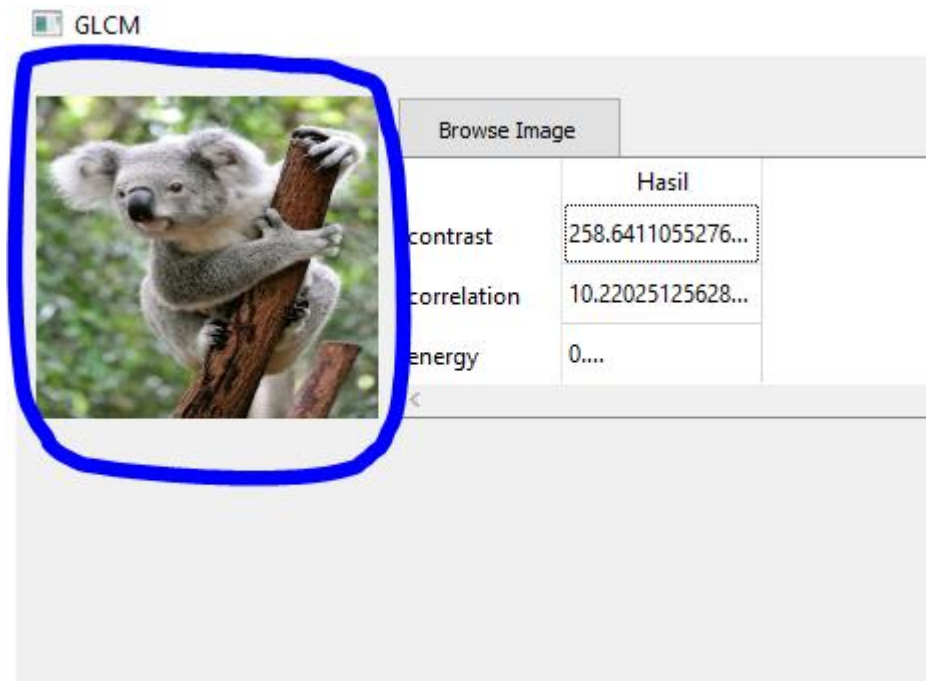
Selanjutnya melalui parameter *img* akan dibuatkan array dengan bantuan *inheritance np* dari *function library numpy*, array tersebut akan direturn dengan perkalian array dari *rgb* dan *np* yang membawa parameter *float grayscale :3*, *[0.2989, 0.5870, 0.1140]* yang merupakan array default dari pembentukan sebuah image ke format abu-abu atau *grayscale*.

Terakhir hasil dari array akan di print pada terminal dengan kode program `print(arrFromImg)`, seperti inilah array yang dihasilkan :

```
Run Run
[[117.6788 122.612 124.0679 ... 184.0757 178.8374 179.8373]
 [ 99.6806 110.6132 118.5954 ... 171.077 171.2511 174.8378]
 [ 91.327 103.4937 120.0621 ... 171.1048 174.0937 175.7515]
 ...
 [ 72.3936 77.3931 83.1645 ... 122.5979 115.6695 108.6702]
 [ 84.3016 91.002 95.5994 ... 128.8253 118.794 107.7951]
 [ 93.0188 99.0182 100.072 ... 118.2501 103.0945]
IDE and Plugin Updates: PyCharm is ready to update. (yesterday 23.16) 49 chars 89:58 LF+ UTF-8
```

### 3.2.2.5. Fungsi updatePixmap

Fungsi ini digunakan untuk merubah gambar pada GUI dan memastikan gambar yang ditampilkan sesuai dengan inputan pengguna, berikut adalah penampakan pixmap pada GUI :



Dan dibawah ini adalah potongan kode yang berfungsi merubah pixmap :

```
# update pixmap
self.updatePixmap(arrFromImg, img)
```

Kode diatas disertai dengan parameter `arrFromImg` yakni variabel yang membawa value array RGB dan `img` adalah variabel yang membawa gambar asli sesuai dengan inputan user.

Potongan kode diatas akan dilanjutkan dengan potongan kode dibawah ini sebagai *Class method* yang akan memproses kinerja dari update pixmap, dimana pada method ini akan dilewatkan 3 parameter, yang pertama ada *self* karena kita ingin memanggil sebuah variabel atau metode di class ini juga, maka kita pakai kata “self” di depan nama variabel atau metodenya, pada kasus ini kita akan memanggil variabel pm dan img.

```
def updatePixmap(self, pm, img):
    saveImg = Image.fromarray(pm)
    saveImg = saveImg.convert("RGB")
    saveImg.save("pm.jpeg")
    pixmap = QPixmap(img)
    self.label.setPixmap(QPixmap(pixmap))
    self.label.setScaledContents(True)
```

Penjelasan detail dari kode diatas :

1. `saveImg = Image.fromarray(pm)`

Pertama kita deklarasikan variabel baru bernama *saveImg*, pada baris kode ini berfungsi untuk menerjemahkan kembali gambar yang sudah didapat dalam bentuk array *GrayScale*, untuk diubah kembali menjadi file gambar.

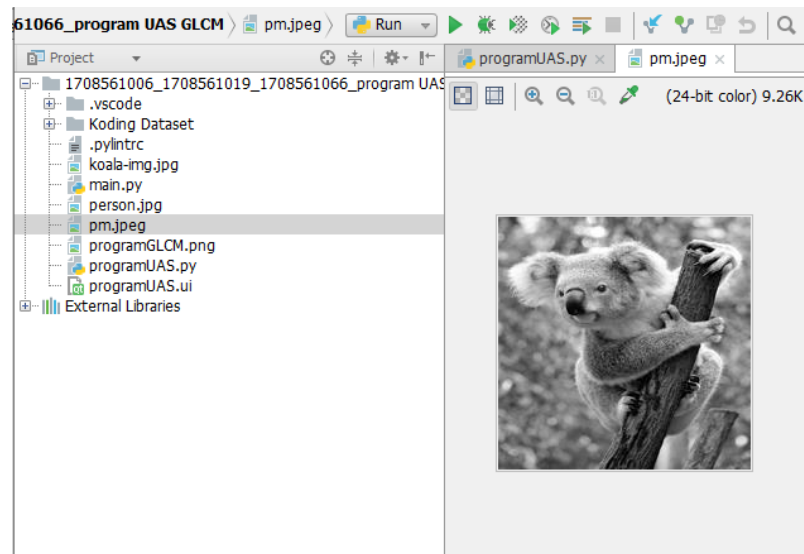
2. `saveImg = saveImg.convert("RGB")`

Kemudian file gambar akan dikonversi ke bentuk desimal RGB

3. `saveImg.save("pm.jpeg")`

file gambar yang sudah dikonversi ke bentuk desimal RGB akan disimpan menjadi pm.jpeg, hasilnya akan menjadi seperti ini :





```
4. pixmap = QPixmap(img)
    self.label.setPixmap(QPixmap(pixmap))
    self.label.setScaledContents(True)
```

variabel *pixmap* adalah bagian dari *library PyQt5* yang akan memanggil method *Qpixmap* yang membawa parameter file gambar untuk nantinya ditampilkan dalam format yang sudah disesuaikan pixelnya di GUI, sehingga seberapaapun besar pixel gambar asli akan selalu ditetapkan dalam ukuran pixel GUI.

### 3.2.2.6. Fungsi `getNewMatrix`, `findOccurences`, dan `normalizeMatrix`

Ketiga fungsi ini adalah langkah pembuatan matrix GLCM :

- Pertama pembuatan framework matrix, yang dimana pertama kali, kita tentukan nilai terbesar dari array gambar yang sudah didapatkan sebelumnya, untuk bagian ini akan diwakili oleh potongan kode program dibawah ini :

```
# nilai terbesar
# maxVal = np.amax(imgArr)
maxVal = np.amax(arrFromImg)
maxVal = int(maxVal)
print(maxVal)
```

`maxVal` tipe data aslinya adalah float, namun kita ubah ke integer dengan bantuan `maxVal = int(maxVal)` hal ini untuk mempermudah perulangan pada *findOccurence Matrix*, sehingga pada terminal akan tercetak salah satu angka terbesar dari seluruh elemen matriks :



```
# find matrix occurences
occMatrix = self.findOccurences(arrFromImg, newMatrix)
print("Matrix Occurence : ", occMatrix)
```

Matrixnya dinamai *occMatrix* yang akan memanggil *method findOccurences* yang membawa parameter array gambar yang sudah diubah ke *grayScale* dan *newMatrix* sebelumnya, selanjutnya akan diproses oleh kode dibawah ini

```
def findOccurences(self, ia, nm):
    print(ia)
    for i in range(0, len(ia)):
        for j in range(0, len(ia[i]) - 1):
            findindex = ia[i][j]
            sindex = ia[i][j + 1]
            # print(ia[i][j],ia[i][j+1])
            nm[int(findindex)][int(sindex)] += 1
    return nm
```

Method Class diatas membawa 3 parameter, pertama ada *self* karena hanya memanggil atau memproses variabel yang berlaku di cakupan method ini saja yakni *ia* yang merepresentasikan *imageArray* atau array gambar dan *nm* merepresentasikan *newMatrix* sebelumnya.

Selanjutnya dilakukan dua kali perulangan untuk mendapatkan tiap posisi pada *framework matrix* menjadi kombinasi nilai pixel pada matrix input, perulangan pertama, yakni : `for i in range(0, len(ia)):`

Yang pertama untuk index *i*, selama range nya dari 0 sampai `len(ia)` atau panjang matrix arrayImage yang berformat *grayScale*, kemudian untuk perulangan yang kedua `for j in range(0, len(ia[i]) - 1):` yang rangenya dari 0 sampai `len(ia)-1` dengan index *j*,

Pada citra digital 8-bit akan memiliki *quantization level* 256, mengingat *gray tone*-nya antara 0 -255, pada kasus ini *gray tone* kami adalah 254, maka selanjutnya kita akan menggunakan *distance* =1 dan *angel* =0°, untuk membentuk *co-occurrence matrix*,

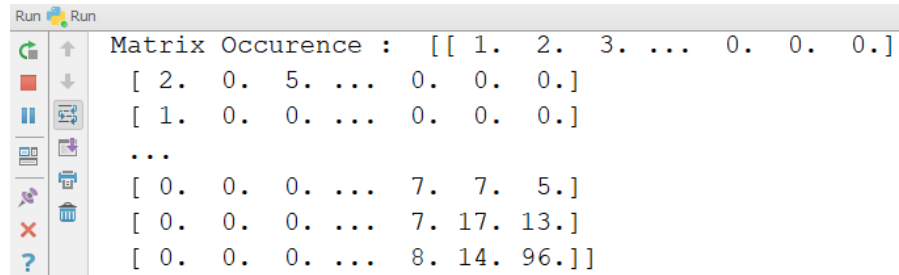
Kombinasi pixel berpasangan dipilih mulai dari kiri-atas terus sampai kanan-bawah.

Konsep jalannya perulangan untuk penempatan tiap lokasi *pixel* bisa digambarkan seperti gambar dibawah ini, yang dicontohkan dengan matrix ordo 3x3, karena tidak mungkin kami bisa menggambarkan keseluruhan matrix berordo 254+1 x 254+1 :



- Berikut contoh ilustrasi keseluruhan step pencarian *co-occurrence matrix* pada input matrix

Matrix co-occurrence akan ditampilkan di terminal :

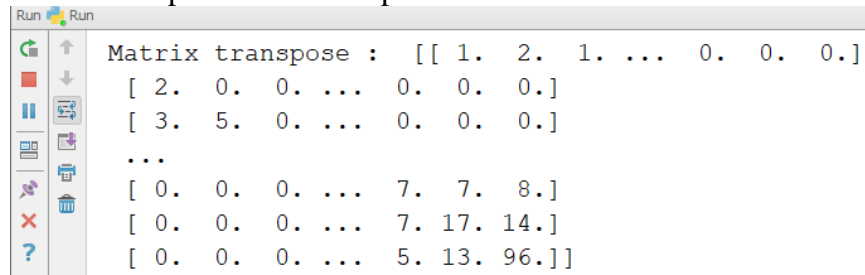


```
Run Run
Matrix Occurence : [[ 1.  2.  3. ... 0.  0.  0.]
[ 2.  0.  5. ... 0.  0.  0.]
[ 1.  0.  0. ... 0.  0.  0.]
...
[ 0.  0.  0. ... 7.  7.  5.]
[ 0.  0.  0. ... 7. 17. 13.]
[ 0.  0.  0. ... 8. 14. 96.]]
```

- Pembuatan symmetric matrix (penjumlahan co-occurrence matrix dengan transpose matrix), pada langkah ini pertama kita perlu untuk melakukan transpose matrix dari co-occurrence matrix yang sudah kita dapat sebelumnya dengan bantuan kode berikut :

```
# matrix tranpose
transMatrix = np.transpose(occMatrix)
print("Matrix transpose : ", transMatrix)
```

`np.transpose` sangat membantu mempersingkat waktu dalam membentuk matriks transpose dari co-occurrence matrix  
matrix transpose akan ditampilkan di terminal :



```
Run Run
Matrix transpose : [[ 1.  2.  1. ... 0.  0.  0.]
[ 2.  0.  0. ... 0.  0.  0.]
[ 3.  5.  0. ... 0.  0.  0.]
...
[ 0.  0.  0. ... 7.  7.  8.]
[ 0.  0.  0. ... 7. 17. 14.]
[ 0.  0.  0. ... 5. 13. 96.]]
```

Selanjutnya akan dilakukan penjumlahan matrix antara co-occurrence matrix dengan transpose matrix yang dipermudah lagi dengan library *numpy* pada kode dibawah ini :

```
# sum of matrix occurences and matrix transpose
sumMatrix = np.add(occMatrix, transMatrix)
print("Matrix penjumlahan : ", sumMatrix)
```

- Terakhir adalah melakukan Matrix normalization yang akan menghasilkan nilai matrix antara 0–1, dengan bantuan kode dibawah ini :

```
# matrix normalization
normMatrix = self.normalizeMatrix(sumMatrix)
print("Matrix normalisasi : ", normMatrix)
```

Kode diatas akan memanggil *method* `normalizeMatrix` yang membawa parameter `sumMatrik` sebelumnya, dibawah ini adalah *method* `normalizeMatrix`

```
def normalizeMatrix(self, m):
    pass # formula : glcmNorm = glcmVal/sum of all glcm
    value
    sumOfAll = np.sum(m)
    for i in range(0, len(m)):
        for j in range(0, len(m[i])):
            m[i][j] = m[i][j] / sumOfAll
    return m
```

Kode diatas adalah adaptasi dari rumus aritmatika matriks normalisasi GLCM berikut :

$$glcmNorm = \frac{glcmValue}{\sum_i^N glcmValue}$$

Keterangan :

`glcmNorm` = Normalisasi GLCM

`glcmValue` = nilai GLCM

$\sum_i^N glcmValue$  = seluruh nilai GLCM dari i sampai N

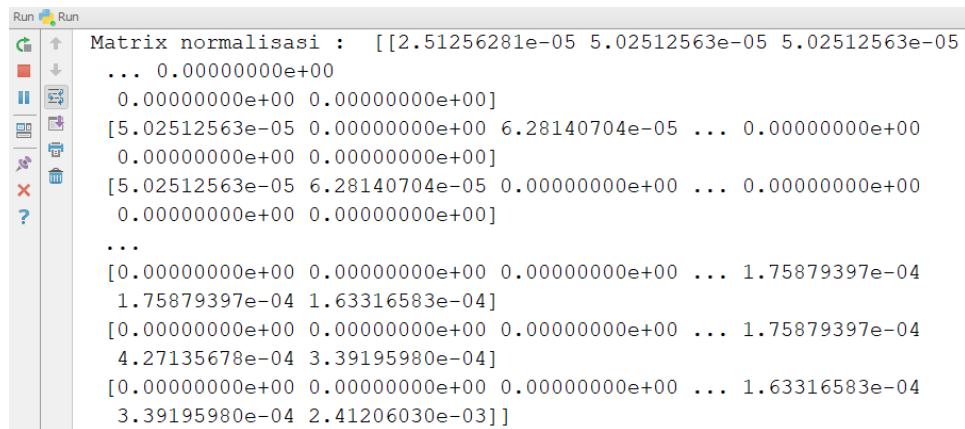
$\sum_i^N glcmValue$  didapatkan dengan bantuan *library numpy* `sumOfAll = np.sum(m)` dan kemudian kita mengambil tiap elemen matrix dengan perulangan :

```
for i in range(0, len(m)):
    for j in range(0, len(m[i])):
```

terakhir masing-masing elemen akan ditempatkan berdasarkan index kolom i dan baris j, lalu dihitung normalisasinya dengan kode ini

```
m[i][j] = m[i][j] / sumOfAll
```

Matrix normalisasi akan ditampilkan di terminal :



```

Run Run
Matrix normalisasi : [[2.51256281e-05 5.02512563e-05 5.02512563e-05
... 0.00000000e+00
0.00000000e+00 0.00000000e+00]
[5.02512563e-05 0.00000000e+00 6.28140704e-05 ... 0.00000000e+00
0.00000000e+00 0.00000000e+00]
[5.02512563e-05 6.28140704e-05 0.00000000e+00 ... 0.00000000e+00
0.00000000e+00 0.00000000e+00]
...
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 1.75879397e-04
1.75879397e-04 1.63316583e-04]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 1.75879397e-04
4.27135678e-04 3.39195980e-04]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 1.63316583e-04
3.39195980e-04 2.41206030e-03]]

```

### 3.2.2.7. Fungsi CalcContrast, calcDisc, calcHomo, calcAsm, calcEnergy

Kelima fungsi ini mengadaptasi rumus dari fitur tekstur dan membawa nilai dari matriks Normalisasi.

1. **Fungsi CalcContrast** adalah implementasi dari Nilai kontras yang digunakan untuk mengukur kekuatan perbedaan intensitas dalam citra. Nilai kontras membesar jika variasi intensitas dalam citra tinggi, dan sebaliknya jika variasinya rendah. Dibawah ini adalah rumus dari fitur *Contrast* :

$$\begin{aligned}
 contrast &= \sum_{i,j}^{levels-1} P_{i,j} (i-j)^2 \\
 &= \sum_{i,j}^3 P_{i,j} (i-j)^2 \\
 &= (0.16 \times [0-0]^2 + 0.16 \times [0-1]^2 \dots 0 \times [0-2]^2)
 \end{aligned}$$

Dan berikut adalah implementasi rumus *Contrast* dalam python yang membawa parameter nm atau matriks normalisasi :

```

def calcContrast(self, nm):
    contrastRes = 0
    for n in range(0, len(nm)):
        for m in range(0, len(nm[n])):
            # print(nm[n][m], n, m)
            temp = nm[n][m] * math.pow((n - m), 2)

```

```

        contrastRes += temp
    print("Contrast : ", contrastRes)
    self.tableWidget.setItem(0, 0,
QtWidgets.QTableWidgetItem(str(contrastRes)))

```

2. **Fungsi CalcDiss** adalah implementasi dari Nilai *dissimilarity* yang digunakan untuk mengukur perbedaan tekstur dalam citra. Dibawah ini adalah rumus dari fitur *Dissimilarity* :

$$\text{'dissimilarity': } \sum_{i,j=0}^{levels-1} P_{i,j} |i - j|$$

Dan berikut adalah implementasi rumus *Dissimilarity* dalam python yang membawa parameter nm atau matriks normalisasi :

```

def calcDiss(self, nm):
    dissRes = 0
    for n in range(0, len(nm)):
        for m in range(0, len(nm[n])):
            # print(nm[n][m], n, m)
            temp = nm[n][m] * abs(n - m)
            dissRes += temp
    print("Dissimilarity : ", dissRes)
    self.tableWidget.setItem(1, 0,
QtWidgets.QTableWidgetItem(str(dissRes)))

```

3. **Fungsi CalcHomo** adalah implementasi dari Nilai *Homogeneity* yang digunakan untuk mengukur kehomogenan variasi intensitas dalam citra. Nilai homogenitas membesar jika variasi intensitas dalam citra rendah, dan sebaliknya jika variasinya tinggi. Dibawah ini adalah rumus dari fitur *Homogeneity* :

$$\text{'homogeneity': } \sum_{i,j=0}^{levels-1} \frac{P_{i,j}}{1+(i-j)^2}$$

Dan berikut adalah implementasi rumus *Homogeneity* dalam python yang membawa parameter nm atau matriks normalisasi :

```

def calcHomo(self, nm):
    homoRes = 0
    for n in range(0, len(nm)):
        for m in range(0, len(nm[n])):
            # print(nm[n][m], n, m)
            temp = nm[n][m] / (1 + math.pow((n - m), 2))

```



```

        homoRes += temp
    print("Homogeneity : ", homoRes)
    self.tableWidget.setItem(2, 0,
        QtWidgets.QTableWidgetItem(str(homoRes)))

```

4. **Fungsi CalcAsm** adalah implementasi dari Nilai *Angular Second Moment* yang didapat dari penjumlahan pangkat elemen matriks GLCM untuk mengukur kehomogenan variasi intensitas dalam citra. ASM memiliki nilai yang tinggi ketika citra memiliki *Homogeneity* yang baik atau nilai pixel yang hampir serupa, dan sebaliknya. Dibawah ini adalah rumus dari fitur *ASM* :

$$\text{'ASM': } \sum_{i,j=0}^{levels-1} P_{i,j}^2$$

Dan berikut adalah implementasi rumus ASM dalam python yang membawa parameter nm atau matriks normalisasi :

```

def caclAsm(self, nm):
    asmRes = 0
    for n in range(0, len(nm)):
        for m in range(0, len(nm[n])):
            # print(nm[n][m], n, m)
            temp = math.pow(nm[n][m], 2)
            asmRes += temp
    print("ASM : ", asmRes)
    self.tableWidget.setItem(3, 0,
        QtWidgets.QTableWidgetItem(str(asmRes)))
    return asmRes

```

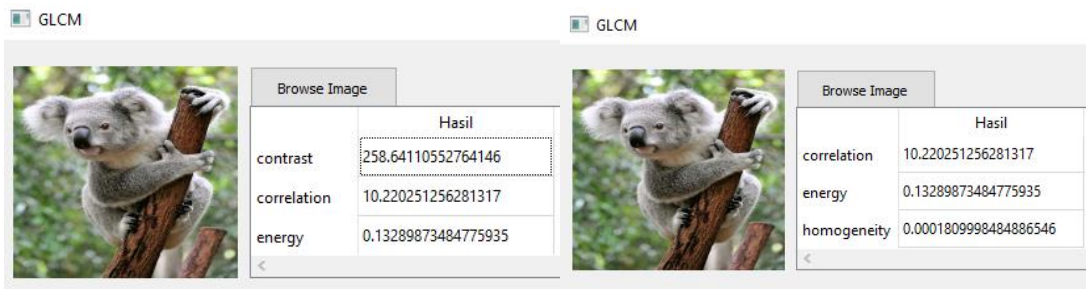
5. **Fungsi CalcEnergy** adalah implementasi dari Nilai *Energy* yang digunakan untuk mengukur konsentrasi pasangan intensitas pada matriks co-occurrence. Nilai energi membesar apabila pasangan piksel terkonsentrasi pada beberapa koordinat, dan sebaliknya akan mengecil jika letaknya menyebar. Dibawah ini adalah rumus dari fitur *Energy* :

$$\text{'energy': } \sqrt{ASM}$$

Dan berikut adalah implementasi rumus *Energy* dalam python yang membawa parameter nm atau matriks normalisasi :

```
def caclEnergy(self, asm):
    eng = math.sqrt(asm)
    self.tableWidget.setItem(4, 0,
    QtWidgets.QTableWidgetItem(str(eng)))
    print("Energy : ", eng)
```

Kelima fitur ini akan ditampilkan pada antarmuka aplikasi :



## BAB IV KESIMPULAN

Dapat disimpulkan bahwa Metode Gray Level Co-occurrence Matrix (GLCM) adalah suatu metode yang digunakan untuk analisis tekstur/ekstraksi ciri pada pengolahan citra digital. Pada proyek ini kami sudah berhasil mengimplementasikan metode GLCM dengan file sumber berupa gambar berekstensi .jpg untuk diuji coba pada pemrograman Python berbasis GUI. Pada tahap preprocessing gambar input diubah menjadi format grayscale dan mendapatkan array matrixnya. Setelah melewati proses *Preprocessing* dan mendapatkan array matrix dari gambar yang diinput, kami melanjutkan pada proses Pembentukan Matrix GLCM yang dibagi menjadi empat tahapan, meliputi pembuatan framework matrix, pembuatan co-occurrence matrix (mengisi framework matrix), pembuatan symmetric matrix (penjumlahan co-occurrence matrix dengan transpose matrix), matrix normalization yang akan menghasilkan nilai matrix antara 0–1, setelah mendapatkan matrix GLCM normalisasi, kami lanjutkan dengan kalkulasi lima fitur tekstur yang meliputi *Contrast*, *Dissimilarity*, *Homogeneity*, *ASM*, dan *Energy* yang masing-masing ada formulanya tersendiri. Tahap terakhir kami bisa menampilkan hasil dari kalkulasi tekstur pada aplikasi GUI.

### DAFTAR PUSTAKA

1. Yunus Muhammad. 2020. Feature Extraction : Gray Level Co-occurrence Matrix (GLCM) di <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcmm-10c45b6d46a1> (di akses 29 Desember).
2. L. Veronica, I. H. Al Amin, B. Hartono, and T. Kristianto, “Ekstraksi Fitur Tekstur Menggunakan Matriks GLCM pada Citra dengan Variasi Arah Obyek,” Pros. SENDI\_U, pp. 978–979, 2019.
3. R. Widodo, A. W. Widodo, and A. Supriyanto, “Pemanfaatan Ciri Gray Level Co-Occurrence Matrix ( GLCM ) Citra Buah Jeruk Keprok ( Citrus reticulata Blanco ) untuk Klasifikasi Mutu,” J. Pengemb. Teknol. Inf. dan Ilmu Komput., vol. 2, no. 11, pp. 5769–5776, 2018.
4. Suhendri and P. Rahayu, “Metode Grayscale Co-occurrence Matrix (GLCM) Untuk Klasifikasi Jenis Daun Jambu Air Menggunakan Algoritma Neural Network,” J. Inf. Technol., vol. 1, no. 1, pp. 15–22, 2019.