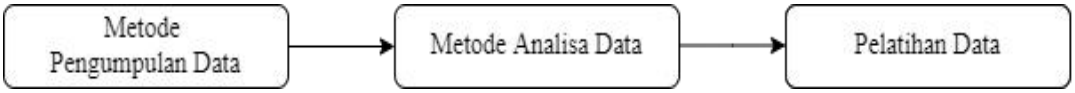
# BAB 3

# METODE DAN IMPLEMENTASI

Pada Bagian ini Bab ini akan membahas tentang metode dan implementasi dalam sistem pengenalan wajah untuk melakukan kehadiran otomatis menggunakan algoritma *Local Binary Pattern Histogram* (LBPH). dalam Bab ini terdapat beberapa tahapan yang dalam pembuatan sistem *face recognition* yaitu analisa data yang akan digunakan, lalu tahap-tahap selanjutnya adalah sebagai berikut. Tahapan yang pertama adalah analisa data yang akan digunakan dengan beberapa tahapan dari *prepprocessing* untuk mengolah citra yang akan digunakan, Melakukan implementasi metode *Haar Cascade Classifier* dan *Local Binary Pattern Histogram* (LBPH) dalam melakukan pengenalan wajah untuk kehadiran secara *real time*. dan melakukan perancangan agar sistem ini dapat berjalan. Pada gambar 3.1 dapat dilihat alur kerja tahapannya.




*Gambar 3. 1 Flowchart Proses Sistem Face Recognition*

## 3.1 Implementasi Perangkat

Pada implementasi perangkat akan dijelaskan mengenai perangkat yang akan digunakan dalam membangun sebuah model sistem pengenalan wajah untuk melakukan kehadiran otomatis. Dengan menggunakan metode *Haar Cascade Classifier* dan *Local Binary Patterns Histogram* (LBPH). Perangkat keras *(hardware)* dan perangkat lunak *(software)* yang akan dijelaskan berikut ini.

### 3.1.1 Perangkat Keras (*Hardware*)

Perangkat keras (*hardware*) yang digunakan untuk membangun sebuah model sistem pengenalan wajah untuk melakukan kehadiran. Secara umum spesifikasi yang perangkat keras yang digunakan pada implementasi tugas akhir ini telah diatas spesifikasi minimun, yang akan ditunjukan pada tabel 3.1 sebagai berikut.

*Tabel 3. 1 Spesifikasi perangkat keras*

| Perangkat Keras | Spesifikasi yang Digunakan |
|---|---|
| Processor | Intel® Core™ i7-8565U |
| Graphic Card | NVIDIA® GeForce® MX150 |
| Hardisk | 512GB |
| RAM | 8GB 2400MHz DDR4 Memory |
| WebCam | 720p HD Camera |

### 3.1.2 Perangkat Lunak (*Software*)

Perangkat lunak *(software)* yang digunakan untuk membangun sebuah model diperlukan kesesuaian metode *face recognition* agar dapat berhasil diimplementasi dan di *install* dengan baik serta dapat dipergunakan. Adapun spesifikasi perangkat lunak yang diimplementasikan akan ditunjukan pada Tabel 3.2 sebagai berikut.

*Tabel 3. 2 Spesifikasi perangkat lunak*

| Perangkat Lunak | Spesifikasi |
|---|---|
| Operating System | Windows 10 64 bit Home |
| Microsoft Visual Code | 1.63 Version |
| Python | 3.9 Version |
| Google Chrome | Version 98.0.4758.82 (Official Build) (64-bit) |

## 3.2 Metode Pengumpulan Data

Pada pembuatan sistem ini, tahapan awal yang harus dilakukan adalah membuat *dataset* yang nantinya akan digunakan dijadikan sebagai data yang akan di latih untuk dapat mendeteksi dan mengenali objek. Metode yang akan digunakan dalam melakukan pengenalan wajah untuk kehadiran otomatis secara *real time* adalah dengan menggunakan metode *Haar Cascade Classifier* dan *Local Binary Pattern Histogram* (LBPH) dengan melewati bebeberapa tahapan. Tahapan-tahapan tersebut akan dimulai dengan membuat data citra. Tahapan selanjutnya adalah tahapan *pre-processing* yang terdiri dari proses *convert to grayscale* dan *resize,* proses ini akan mengubah citra agar proses *training* dapat lebih mudah dilakukan dan dapat dengan cepat untuk mendeteksi wajah dengan menggunakan metode *Haar Cascade Classifier.*

Data yang akan digunakan dalam penelitian ini terdiri dari sampel gambar yang diambil dari hasil *capture* sebuah kamera *webcam* sebanyak 100 *capture* gambar per-wajah yang akan digunakan untuk *dataset*, dengan beberapa batasan parameter yaitu:

Untuk variasi posisi wajah dilakukan beberapa posisi sebagai berikut:

- Menghadap tegak lurus ke depan
- Rotasi 10° ke kanan
- Rotasi 10° ke kiri
- Rotasi 10° ke ke atas
- Mengangkat dagu 10° atas

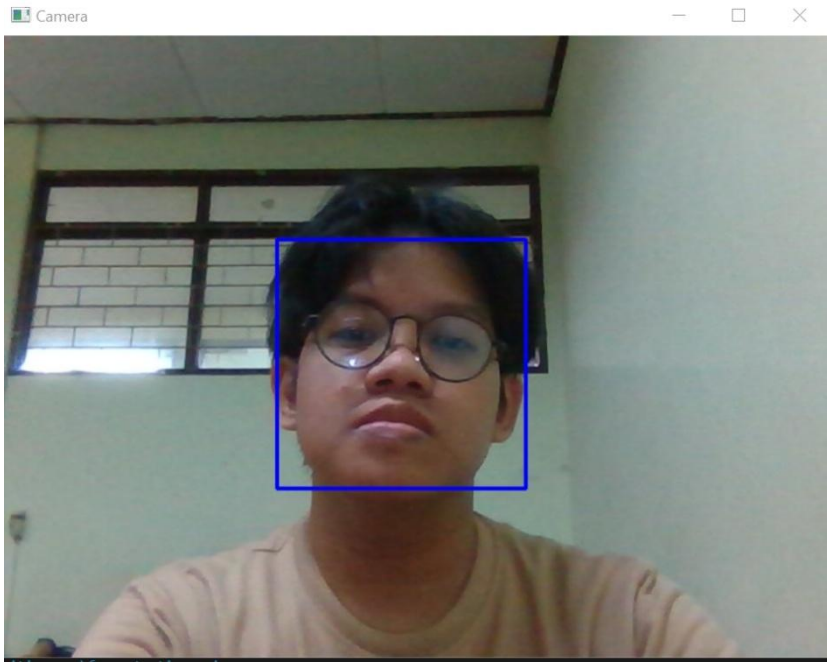
Wajah yang di *capture webcam* tidak terhalangi sebagaian oleh objek lain dan tidak banyak bergerak agar pengambilan gambar dapat dilakukan dengan baik. Untuk aspek jarak wajah terhadap kamera *webcam* akan dicari jarak ideal yaitu 40 cm sampai dengan 100 cm.

## 3.3 Pengolahan Citra

Data yang telah dikumpulkan akan melalui beberapa proses sebelum dilanjutkan ke proses training agar dapat melakukan pengenalan. Data yang sudah didapatkan harus dilakukan perubahan citra *image* dan dilakukan *resize* guna untuk meningkatkan performa model saat dilakukan *training* data dan juga untuk dapat mendeteksi objek. Adapun tahap yang akan dilalui adalah *preprocessing*.

Tahap ini dilakukan untuk mempermudah proses *training* data, Dalam proses *preprocessing* ini akan dilakukan tahap pengolahan citra agar data yang akan diolah mempunyai citra yang lebih baik untuk diproses ke tahapan selanjutnya. Pada tahap proses *preprocessing* ini terdiri dari 2 (dua) proses, yaitu proses *capture* gambar wajah *user* dan dikonversi dari citra RGB menjadi citra *grayscale* lalu dilakukan *resize*.

Hal ini berguna untuk membuat citra dapat diolah dengan menggunakan metode *Haar Cascade*, yang dimana metode ini akan digunakan untuk mendeteksi wajah. Setelah dapat terdeteksi wajah maka sistem akan menandai wajah dengan garis bujur sangkar ROI (*Region of Interest*) berwarna biru dan objek yang digunakan adalah muka. Hal ini dilakukan untuk membedakan dengan objek bukan wajah. Contoh pada tahapan ini akan digambarkan dalam gambar 3.2 sebagai berikut.




*Gambar 3. 2 Pengambilan gambar*

Lalu akan dijelaskan untuk tahapan-tahapan melakukan *preprocessing* seperti contoh pada gambar 3.3 agar dataset yang akan digunakan dapat lebih mudah dan lebih cepat. Untuk merancang sistem pengenalan wajah pada tugas akhir ini digunakan metode yang sangat mudah dan kecepatan pendeteksiannya sangat cepat yaitu metode *Haar Cascade Classifier*. Maka digunakanlah library yang disediakan oleh OpenCV yaitu *haarcascadefrontal_face_default.xml.* yang mana pada *library* tersebut dapat melakukan semua proses pendeteksian wajah.

Berikut ini merupakan tahapan cara kerja pada *library* OpenCV haarcascadefrontal_face_default.xml. yaitu dengan cara berikut ini, Langkah pertama yang harus dilakukan adalah citra harus dikonversikan terlebih dahulu dari RGB menjadi *Grayscale*, Secara teori cara mengkonversikannya adalah dengan cara menggunakan persamaan rumus sebagai berikut 3.1

$$Grayscale = 0.2989\ R + 0.5870\ G + 0.1140\ B \quad (3.1)$$

Pada persamaan 3.1 dapat melihat bahwa setiap nilai R nilai G dan nilai B pada setiap piksel akan dikalikan dengan nilai yang sesuai rumus agar sebuah citra dapat dikonversi menjadi *grayscale*. Maka contoh terdapat pada gambar 3.3 merupakan citra RGB dengan dimensi 200 x 200.

*Gambar 3. 3 Citra RGB*

Kemudian akan menggunakan semua piksel secara acak untuk mengetahui nilai RGB dari salah satu piksel. Lalu pada piksel $x$ = 100 dan $y$ = 100 dapat diketahui RGB = 57 34 16. Dari data tersebut maka dapat dihitung *grayscale* dengan cara sebagai berikut ini :

Diketahui = *Red* (merah) = 57

*Green*(hijau) = 34

*Blue*(biru) = 16

*Grayscale* = 0.2989 R + 0.5870 G + 0.1140 B

*Grayscale* = ( 0.2989 x 57) +( 0.5870 x 34) + (0.1140 x 16)

*Grayscale* = 17.0373 + 19.968 + 1.824

*Grayscale* = 38.8193

Citra *grayscale* merupakan citra dengan ukuran 8 bit. Sehingga didapatkan $(2^8 - 1)$ yang mengubah warna mulai dari 0 hingga 255 , yaitu dimana 0 berarti hitam dan 255 berarti putih. Warna abu memiliki *range* dari 1 hingga 254 dimulai dari abu paling gelap sehingga abu terang dan akhirnya mendekati putih. Maka hasil *grayscale* = 38.8193 merupakan warna abu-abu yang hampir mendekati hitam. Lalu

menggunakan rumus tersebut pada semua piksel dan hasil dari konversi citra RGB pada gambar 3.4 akan menjadi seperti pada gambar 3.4 sebagai berikut :

*Gambar 3. 4 Citra grayscale*

Dari gambar 3.4 dapat diketahui bahwa setelah melakukan konversi menjadi citra *grayscale* sebagian besar daerah pada citra memiliki nilai yang akan lebih hitam. Proses selanjutnya adalah melakukan *Haar feature* yaitu adalah metode yang membutuhkan training terlebih dahulu untuk mendapatkan suatu keputusan apakah di *frame* tersebut dapat mendeteksi objek atau tidak. Selisih dari nilai fitur yang akan diimplementasikan akan dijadikan *threshold* klasifikasi terdeteksi objek atau tidak. Lalu dalam tugas akhir ini objek yang dideteksi adalah wajah. Berikut ini adalah contoh gambar 3.5 yang akan diletakan fitur *haar* dalam sebuah citra.

*Gambar 3. 5 Haar Feature*

Selanjutnya pada gambar 3.5 terdapat sebuah *feature haar* pada gambar hasil konversi *grayscale. Feature* yang digunakan berupa *line feature* yang merupakan salah satu dari banyak *haar feature.* Untuk menentukan nilai *feature* tersebut dapat digunakan persamaan yaitu sebagai berikut ini :

*NILAI FITUR* = |(total piksel hitam) – ( total piksel putih) | (3.2)

Pada persamaan 3.2 dibutuhkan nilai piksel, ada berbagai cara untuk mengetahui nilai piksel. Tetapi dalam kasus ini dibutuhkan teknik yang dapat dilakukan dengan sangat cepat sehingga dapat diimplementasikan pada ratusan fitur *haar* dengan skala yang berbeda-beda. Salah satu teknik yang efisien untuk melakukan itu yaitu *integral image.* Berikut merupakan rumus integral image :

$$S(x,y) = i(x,y) + S\big((x-1),y\big) + S\big(x,(y-1)\big) - S(x-1),(y-1)) \quad (3.3)$$

Pada persamaan 3.3 i(*x,y*) adalah nilai asli matrik citra. Pada gambar 3.5 menunjukan bahwa *feature* diletakan pada tengah wajah yang miliki nilai matriks sebagai berikut :

| | | | | | |
|---|---|---|---|---|---|
| 2 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 1 | 2 | 3 | 5 |
| 3 | 4 | 5 | 5 | 1 | 2 |
| 5 | 1 | 2 | 4 | 3 | 6 |
| 3 | 7 | 4 | 1 | 8 | 2 |
| 6 | 5 | 1 | 2 | 3 | 4 |

Matriks tersebut adalah nilai grayscale dari setiap piksel pada citra. Kemudian fitur *haar* yang akan diimplementasikan ialah sebagai berikut :

| | | | | | |
|---|---|---|---|---|---|
| 2 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 1 | 2 | 3 | 5 |
| 3 | 4 | 5 | 5 | 1 | 2 |
| 5 | 1 | 2 | 4 | 3 | 6 |
| 3 | 7 | 4 | 1 | 8 | 2 |
| 6 | 5 | 1 | 2 | 3 | 4 |

Agar dapat mengetahui nilai fitur *haar* rumus yang digunakan adalah rumus pada persamaan 3.2. Karena itu pertama harus diketahui terlebih dahulu nilai masing-masing bagian fitur. Jika menggunakan penjumlahan untuk menghitung nilai piksel masing-masing fitur yaitu dengan menjumlahkan semuanya seperti pada contoh berikut ini:

*Daerah Hitam* 1 =

*Daerah Putih* 1 =

*Daerah Hitam* 2 =

| | | | | | |
|---|---|---|---|---|---|
| 2 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 1 | 2 | 3 | 5 |
| 3 | 4 | 5 | 5 | 1 | 2 |
| 5 | 1 | 2 | 4 | 3 | 6 |
| 3 | 7 | 4 | 1 | 8 | 2 |
| 6 | 5 | 1 | 2 | 3 | 4 |

*Daerah Hitam* 1 = 2 + 1 + 2 + 3 + 4 + 5 + 4 + 5 + 1 + 2 + 3 + 5 = 37

*Daerah Hitam* 2 = 3 + 7 + 4 + 1 + 8 + 2 + 6 + 5 + 1 + 2 +3 + 4 = 46

*Daerah Putih* 1 = 3 + 4 +5 + 5 + 1 + 2 + 5 + 1 + 2 + 4 + 3 + 6 = 41

*NILAI FITUR* = |(*total piksel hitam*) – (*total piksel putih*)|

*NILAI FITUR* = |(37+46) – (41) |

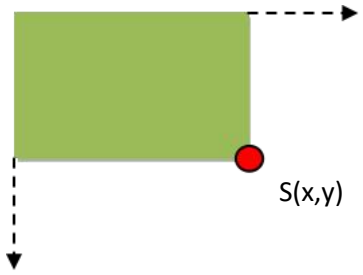
*NILAI FITUR* = 42

Cara perhitungan diatas merupakan cara perhitungan manual untuk mencari nilai fitur. Tetapi cara itu tidak akan efektif dan efisien jika ada ratusan jenis fitur *haar* dan juga ratusan jenis ukuran serta posisi fitur yang acak dalam citra. Jika menghitung dengan cara menggunakan *intergral image* akan lebih efisien dan dapat dengan sekali jalan. Berikut ini merupakan perhitungan menggunakan *integral image*:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

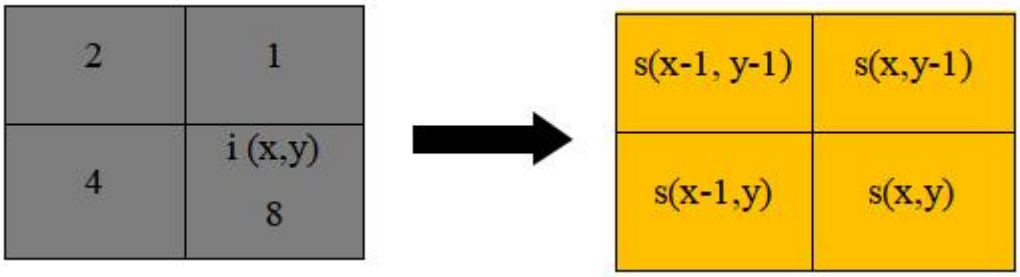
1. Menyiapkan matriks dengan ukuran yang sama untuk digunakan sebagai *buffer.* Fungsi dari matriks *buffer* yaitu sebagai tempat untuk meletakan nilai *integral image* dari setiap piksel.

2. Membuat *summed area table* sebagai contoh seperti pada gambar 3.6 berikut. Fungsi *summed area table* digunakan untuk dapat mempermudah proses perhitungan.




*Gambar 3. 6 Ilustrasi area s(x,y)*

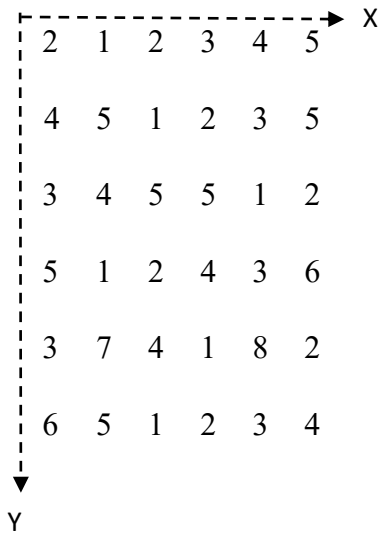
*S(x,y)* adalah nilai *summed area* pada bidang (*x,y*), nilai *i*(x,y) merupakan intensitas dari citra asli, *s* (*x,y-1*) merupakan nilai *summed area* dari nilai piksel tetangga atas atau (*y-1*). Dan *s*(*x-1,y*) merupakan nilai *summed area* dari nilai piksel tetangga x serta *s*(*x-1,y-1*) merupakan nilai *summed area* dari nilai piksel tetangga diagonalnya. Jika diilustrasikan dalam gambar akan menjadi seperti gambar 3.7 berikut .




*Gambar 3. 7 Summed Area*

Gambar 3.7 menunjukan maka matriks dengan warna abu merupakan matriks asli dari citra dan pada warna *orange* merupakan nilai *integral image* yang akan dicari.

3. Selanjutnya adalah mencari nilai *integral image* dengan rumus pada persamaan 3.3. Pada setiap piksel yang berada di citra yang mana ialah setiap nilai pada matriks asli yang telah diketahui dan meletakkan nilainya pada matrik *buffer* yang telah disediakan.



$$S(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$
$$S(1,1) = 2 + 0 + 0 + 0 + 0 + 0 = 2$$
$$S(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$
$$S(2,1) = 2 + 1 + 0 + 0 + 0 + 0 = 3$$
$$S(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$
$$S(1,2) = 2 + 4 + 0 + 0 + 0 + 0 = 6$$

Melakukan perhitungan seperti di atas pada setiap piksel yang ada. Maka hasil setiap perhitungannya dimasukkan ke dalam *buffer* yang telah tersedia sebelumnya. Contoh memasukan *buffer* yang telah dihitung diatas sebagai berikut.

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 0 | 0 | 0 | 0 |
| 6 | 12 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Setelah semua terhitung maka akan ditemukan matriks *integral image* yang telah dimasukkan ke dalam *buffer* sebagai berikut.

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 12 | 17 |
| 6 | 12 | 15 | 20 | 27 | 37 |
| 9 | 19 | 27 | 37 | 45 | 57 |
| 14 | 25 | 35 | 49 | 60 | 78 |
| 17 | 35 | 49 | 64 | 83 | 103 |
| 23 | 46 | 61 | 78 | 100 | 124 |

Setelah didapat nilai dari *integral image*, selanjutnya bagi daerah tersebut menjadi 2 bagian yaitu bagian piksel hitam dan bagian piksel putih sesuai dengan matriks *fitur-haar*.

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 12 | 17 |
| 6 | 12 | 15 | 20 | 27 | 37 |
| 9 | 19 | 27 | 37 | 45 | 57 |
| 14 | 25 | 35 | 49 | 60 | 78 |
| 17 | 35 | 49 | 64 | 83 | 103 |
| 23 | 46 | 61 | 78 | 100 | 124 |

Untuk menentukan jumlah nilai piksel yang dicari pada sebelumnya dengan menggunakan rumus manual, maka dengan hasil matriks yang telah dihitung menggunakan *integral image* dapat dihitung menggunakan persamaan rumus 3.4 sebagai berikut.

$$\text{Luas Area} = L_1 + L_4 - (L_2 + L_3) \quad (3.4)$$

Diketahui = L1 = Orange

L2 = Kuning

L3 = Hijau

L4 = Biru

A. Luas daerah hitam 1

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 12 | 17 |
| 6 | 12 | 15 | 20 | 27 | 37 |
| 9 | 19 | 27 | 37 | 45 | 57 |
| 14 | 25 | 35 | 49 | 60 | 78 |
| 17 | 35 | 49 | 64 | 83 | 103 |
| 23 | 46 | 61 | 78 | 100 | 124 |

Diketahui = $L_1$ = Orange = 0

$L_2$ = Kuning = 0

$L_3$ = Hijau = 0

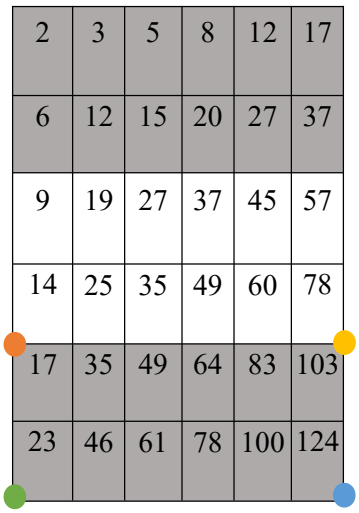
$L_4$ = Biru = 37

Luas area = $(L_1 + L_4) – (L_2+L_3)$

Luas area = 37 – 0

Luas area = 37

B. Luas daerah hitam 2



| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 12 | 17 |
| 6 | 12 | 15 | 20 | 27 | 37 |
| 9 | 19 | 27 | 37 | 45 | 57 |
| 14 | 25 | 35 | 49 | 60 | 78 |
| 17 | 35 | 49 | 64 | 83 | 103 |
| 23 | 46 | 61 | 78 | 100 | 124 |

Diketahui = $L_1$ = Orange = 0

$L_2$ = Kuning = 78

$L_3$ = Hijau = 0

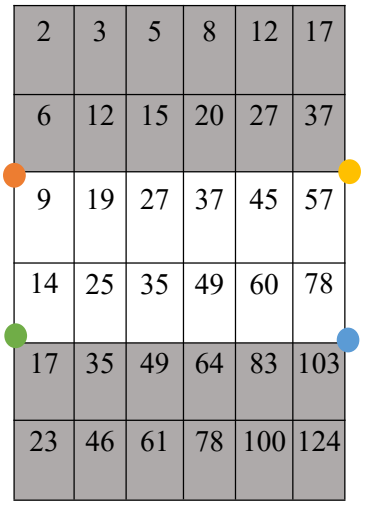
$L_4$ = Biru = 124

Luas area = $(L_1 + L_4) – (L_2+L_3)$

Luas area = 124 – 78

Luas area = 46

C. Luas daerah putih



| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 12 | 17 |
| 6 | 12 | 15 | 20 | 27 | 37 |
| 9 | 19 | 27 | 37 | 45 | 57 |
| 14 | 25 | 35 | 49 | 60 | 78 |
| 17 | 35 | 49 | 64 | 83 | 103 |
| 23 | 46 | 61 | 78 | 100 | 124 |

Diketahui = $L_1$ = Orange = 0

$L_2$ = Kuning = 37

$L_3$ = Hijau = 0

$L_4$ = Biru = 78

Luas area = $(L_1 + L_4) – (L_2+L_3)$

Luas area = 78 – 37

Luas area = 41

*NILAI FITUR* = |(*total piksel hitam*) – (*total piksel putih*)|

*NILAI FITUR* = |(78+37) – (41) |

*NILAI FITUR* = 42

Setelah didapat nilai dari *integral image* akan terbukti sama dengan perhitungan secara manual sebelumnya. Dimana nilai fitur yang terdapat pada image akan digunakan untuk menentukan objek yang ada dalam kamera, yang kemudian objek yang dideteksi tersebut harus mempunyai nilai *threshold* atau batas ambang nilai fitur.

Setelah didapat sebuah nilai fitur proses terakhir adalah membuat *cascade classifier.* untuk dapat menambah akurasi perlu dilakukan proses *haar-like feature* yang dilakukan secara massal dan terorganisir disebut dengan *cascade classifier*. biasanya dilakukan menggunakan *stage* filter dengan jumlah fitur yang berbeda-beda jumlah fiturnya. Jika nilai fitur tidak dapat memenuhi maka hasil langsung ditolak. Contoh *cascade classifier* adalah sebagai berikut :

1. *Stage* Filter 1 (3 fitur *haar*)
2. *Stage* Filter 2 (5 fitur *haar*)
3. *Stage* Filter 3 (10 fitur *haar*)
4. *Stage* Filter 4 (20 fitur *haar*)

Masing-masing *stage* mempunyai *threshold* yang berbeda kemudian kemudian setiap fitur dalam *stage* juga mempunyai *threshold* yang berbeda-beda. Objek yang tidak dapat memenuhi akan ditolak dan yang memenuhi akan melalui *stage* selanjutnya hingga objek ditemukan.

## 3.4 Training Data

Proses selanjutnya adalah mencari pola ciri wajah dari masing-masing individu untuk dapat membedakan antara individu satu dengan yang lainnya. Dibutuhkan operator untuk klasifikasi tekstur dan pada tugas akhir ini menggunakan *Local Binary Patern*. Cara kerjanya adalah dengan mengkonversi seluruh citra *grayscale* yang telah dijadikan *dataset* sebelumnya menjadi *Local Binary Patern* Image dan dijadikan sebuah *Histogram*. Berikut ini adalah langkah-langkah membuat *Local Binary Patern*.

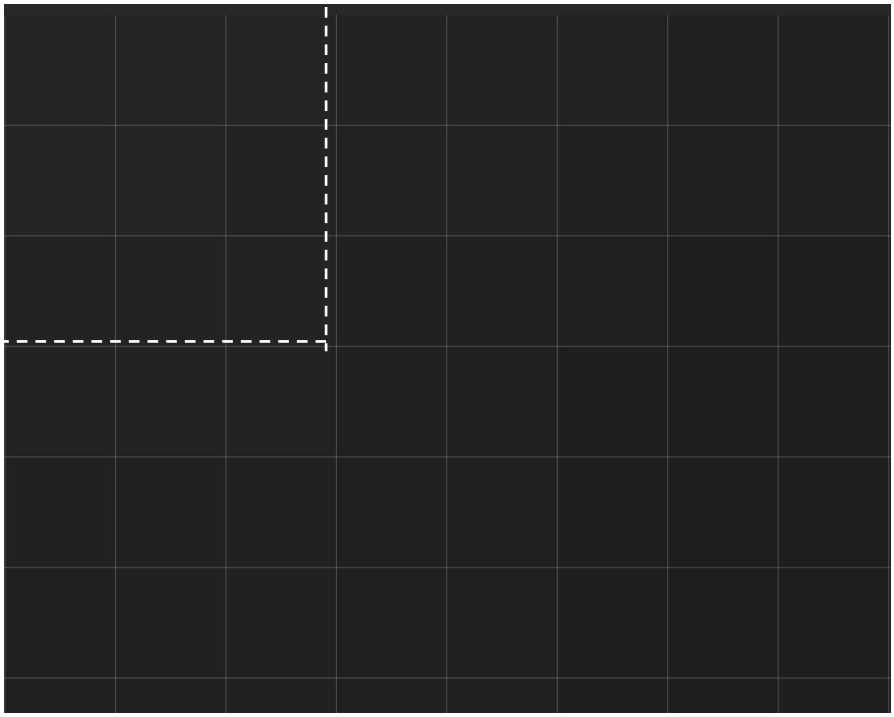
1. Langkah pertama menyiapkan citra *grayscale* yang sudah dijadikan *dataset* sebelumnya lalu akan diubah menjadi *local binary pattern image.* Maka Gambar 3.8 merupakan citra *grayscale.*

*Gambar 3. 8 Citra grayscale*

2. Langkah kedua menggunakan rumus LBP 3 x 3 yang dihitung mulai dari piksel pojok kiri atas. Rumus yang akan digunakan diletakan pada persamaan 3.3

$$LBP = \sum_{n}^{c} = S(i_n - i_c)2^n \quad (3.3)$$

3. Setelah dilakukan *zoom* hingga terlihat jelas setiap piksel yang diambil matrik 3 x 3 pada piksel, maka gambar akan menjadi seperti pada gambar 3.9



*Gambar 3. 9 Matriks 3 x 3 pojok kiri atas*

4. Menghitung nilai matrik 3 x 3 yang telah ditandai tersebut dengan menggunakan rumus 3.3. Nilai matriknya adalah sebagai berikut:

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 6 | **4** | 10 |
| 3 | 2 | 5 |

Titik pusatnya ialah 4 yang menjadi nilai *treshold* kemudian jika nilai pusat lebih tinggi dari nilai yang ada disekitarnya, maka nilai matriks tersebut bandingkan dengan matrik disekitarnya, Dengan aturan jika nilai pusat ≥ nilai piksel sekitar maka beri nilai 0. Dan jika titik pusat < nilai piksel sekitar maka beri nilai 1. Sesudah dilakukan seperti itu kemudian matriknya akan menjadi seperti berikut ini :

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 1 | ***Pusat*** | 1 |
| 0 | 0 | 1 |

Selesai didapat hasil dari biner tersebut, Maka selanjutnya akan diubah ke dalam bentuk desimal dari hasil biner tersebut. Dengan cara sebagai berikut.

| | | |
|---|---|---|
| $0_7$ | $0_6$ | $1_5$ |
| $1_0$ | ***Pusat*** | $1_4$ |
| $0_1$ | $0_2$ | $1_3$ |

$$0_7\ 0_6\ 1_5\ 1_4\ 1_3\ 0_2\ 0_1\ 1_0$$

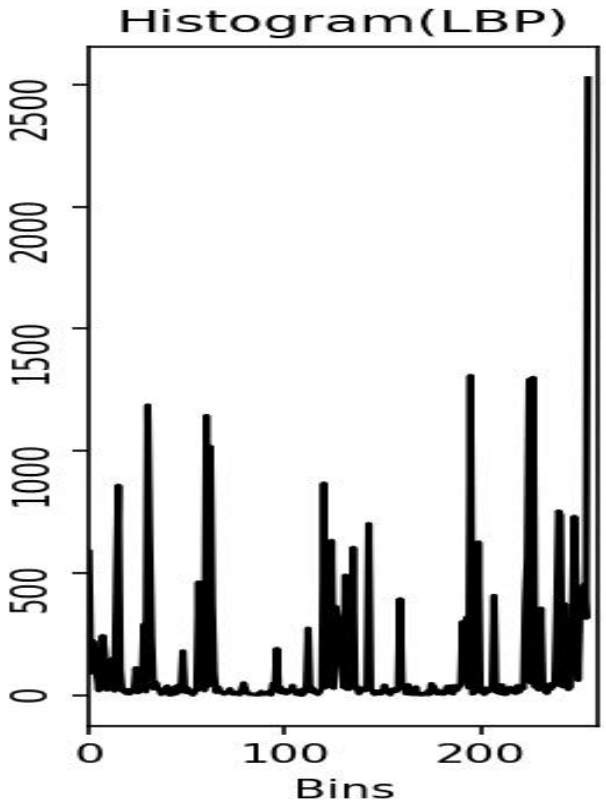
$$2^5\ 2^4\ 2^3\ \qquad\qquad 2^0$$

$$0 + 0 + 32 + 16 + 8 + 0 + 0 + 1 = 57$$

Dari hasil perhitungan tersebut diketahui nilai pusatnya adalah 57. Maka perhitungan tersebut akan diimplementasikan pada semua bagian citra maka citra LBP akan menjadi seperti pada gambar 3.10.

*Gambar 3. 10 LBP image*

Maka hasil perhitungan LBP jika ditampilkan dalam bentuk histogram akan menjadi seperti pada gambar 3.11 sebagai berikut :




*Gambar 3. 11 Local Binary Pattern Histogram*

Sesudah menerapkan proses LBP maka histogram dari setiap gambar akan di ekstrak berdasarkan jumlah *grid* (X dan Y) yang dilewatkan parameter. Setelah histogram dari setiap wilayah di ekstrak maka semua histogram yang ada akan digabungkan dan dibuatlah satu histogram baru yang ada digunakan untuk mempresentasikan gambar. Kemudian setiap hasil tersebut akan disimpan dalam sebuah file .yml.

Setelah melalui proses tersebut maka selanjutnya akan membuat algoritma yang dapat menyimpan hasil LBP dalam bentuk yml. Algoritma akan dirancang dalam bentuk *flowchart*. Untuk fungsi *Local Binary Pattern Histogram* menggunakan library OpenCV yaitu cv2.createRecognizer. pada *library* tersebut sudah mencakup keseluruhan cara kerja *Local Binary Pattern*.