BAB 2
LANDASAN TEORI

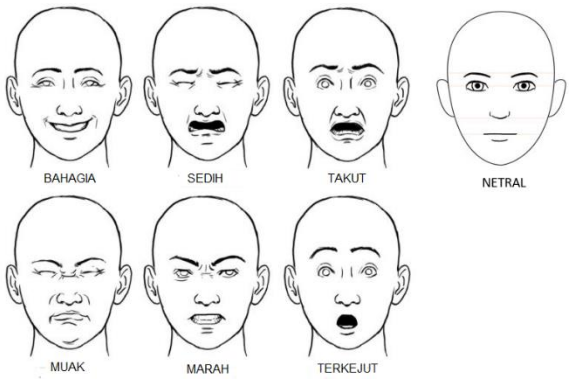
2.1. Ekspresi Wajah

Ekspresi wajah berperan penting dalam berkomunikasi dengan orang lain sebagai mengungkapkan perasaan atau emosi, berikut ciri-ciri dari ekspresi wajah:

Tabel 2. 1. Ciri-ciri ekspresi wajah

Ekspresi	Ciri-ciri
Bahagia	Pada bagian bawah kelopak mata sedikit terangkat kemudian terlihat ada kerutan pada bagian bawah mata dan mata menyipit, sedangkan pada bagian mulut dan bibir melebar, kadang-kadang gigi terlihat.
Sedih	Pada bagian dalam alis terangkat, kemudian dahi berkerut dan bagian mulut tertarik kebawah.
Terkejut	Pada bagian alis seluruhan terangkat kemudian mata membesar, sedangkan dahi berkerut dan pada bagian bibir ditarik atau sedikit terbuka.
Takut	Bagian kelopak mata bagian atas terangkat kemudian bagian putih mata terlihat jelas sedangkan kelopak mata bagian bawah menegang dan terangkat kemudian bagian dahi berkerut dan bagian bibir ditarik kebawah.
Marah	Bagian alis ditarik ke dalam sedangkan mata menyipit kemudian bagian bibir tertutup rapat.
Muak	Pada bagian kelopak mata bagian bawah terangkat dan berkerut sedangkan mulut dirapatkan dan kedua bibir terangkat atau cemberut.
Netral	Seluruh otot wajah dalam kondisi rileks kemedian kelopak mata bersinggungan dengan retina dan pada bagian bibir atas dan bawah saling bersentuhan kemudian garis bibir berbentuk horisontal dan ujung bibir rata dan mulut tertutup.

Dari ciri-ciri ekspresi wajah diatas dapat digambarkan dalam sebuah ilustrasi sebagai berikut:



Gambar 2. 1. Bentuk Ekspresi Wajah

2.2. *Resize Image*

Resize Image atau perubahan citra merupakan proses untuk mengubah ukuran citra menjadi ukuran citra lebih kecil. Proses ini dilakukan supaya pemprosesan citra oleh komputer akan lebih cepat dan tidak banyak menghabiskan memori penyimpanan di dalam memori sementara atau RAM (*Read Only Memory*). Skala *resize image* yang digunakan adalah sebesar 50% dari ukuran citra asli artinya ukuran citra akan lebih kecil setengah dari citra aslinya.

Untuk merubah ukuran citra asli menjadi citra lebih kecil menggunakan perhitungan sebagai berikut:

$ResizeW = w \times 0.5$ .....(2.1).

$ResizeH = h \times 0.5$ .....(2.2).

Keterangan:

ResizeW = ukuran lebar citra baru (*width*)

ResizeH = ukuran panjang citra baru (*Height*)

w = ukuran lebar citra asli (*width*)

h = ukuran panjang citra asli (*Height)*

2.3. *Grayscale*

Proses *grayscale* atau konversi citra berwarna (RBG) menjadi citra keabuan bertujuan untuk mengubah citra RGB yang tersusun dari tiga nilai yaitu: *red, green* dan *blue* dikonversi menjadi satu nilai saja.

Untuk merubah citra RGB menjadi citra *grayscale* menggunakan perhitungan sebagai berikut:

***Grayscale* = R × 0.29 + G × 0.59 + B × 0.11** .....(2.3).

Keterangan:

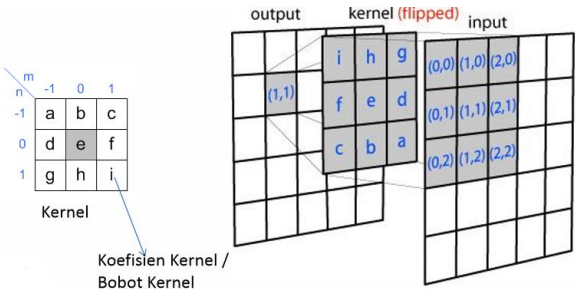
R = nilai piksel berwarna merah (*Red*)

G = nilai piksel berwarna hijau (*Green*)

B = nilai piksel berwarna biru (*Blue*)

2.4. *Image Smoothing*

Proses *image smoothing* atau penghalusan citra merupakan proses yang digunakan untuk merubah citra menjadi lebih halus. Teknik *Image smoothing* yang akan digunakan adalah menggunakan metode konvolusi. Metode konvolusi merupakan suatu proses untuk memperoleh suatu piksel didasarkan pada nilai piksel itu sendiri dan tetangganya, dengan melibatkan suatu matriks yang disebut karnel yang merepresentasikan pembobotan. Operasi yang digunakan dengan menumpangkan suatu jendela atau karnel yang berisi dengan angka-angka pengali pada setiap piksel yang ditimpali, kemudian nilai jumlah diambil dari hasil-hasil kali tersebut.



Gambar 2. 2. Simulasi konvolusi citra

Pada proses *image smoothing* yang menggunakan metode konvolusi bertujuan untuk memperbaiki citra supaya citra keluaran lebih halus. Pada proses perhitungan *image smoothing* menggunakan filter rata-rata. Pada filter rata-rata nilai intensitas setiap piksel diganti dengan rata-rata dari nilai intensitas piksel tersebut dengan piksel-piksel tetangganya. Filter rata-rata yang akan digunakan berupa filter *h* dalam bentuk matriks berukuran *m*×*n*, biasanya *m*=*n* dan nilainya sama untuk setiap elemen. Ukuran *m*×*n* ini yang menentukan jumlah tetangga yang harus dilibatkan dalam perhitungan. Secara matematis filter rata-rata dapat dituliskan sebagai berikut:

$$h(x,y) = \frac{1}{mn}\sum_{x=0}^{m-1} \quad \sum_{y=0}^{n-1} \quad \text{.....(24).}$$

Keterangan:

h(x,y) = filter *h* (filter rata-rata)

n = jumlah baris pada filter *h* (filter rata-rata)

m = jumlah kolom pada filter *h* (filter rata-rata)

x = koordinat letak citra pada titik *x*

y = koordinat letak citra pada titik *y*

Operasi pada filter rata-rata yang akan digunakan adalah metode konvolusi yaitu perkalian fungsi diskrit antara citra *f(x,y)* dan filter *g(x,y)* (filter *h(x,y)* dimisalkan sebagai *g(x,y)*).

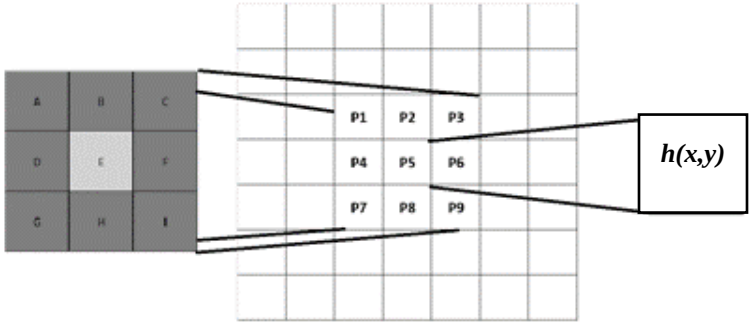
$$h(x,y) = f(x,y) \times g(x,y) \text{ .....(2.5).}$$

Pada filter *h(x,y)* yang akan digunakan adalah karnel berbasis 3×3 sebagai proses penghalusan citra dan disebut sebagai konvolusi dari *f(x,y)* dengan respon *h(x,y)*, berikut adalah karnel yang digunakan dalam proses perhalusan citra:

$$h(x,y) = 1/9 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{.....(2.6).}$$

f(x,y) adalah piksel yang dikenai operasi beserta tetangganya, maka *h(x,y)* adalah hasil dari perhitungan adalah sebagai berikut:

$$h(x,y) = (\mathrm{A} \times \mathrm{P1}) + (\mathrm{B} \times \mathrm{P2}) + (\mathrm{C} \times \mathrm{P3}) + (\mathrm{D} \times \mathrm{P4}) + (\mathrm{E} \times \mathrm{P5}) + (\mathrm{F} \times \mathrm{P6}) + (\mathrm{G} \times \mathrm{P7}) + (\mathrm{H} \times \mathrm{P8}) + (\mathrm{I} \times \mathrm{P9}) \quad \text{..... (2.7).}$$



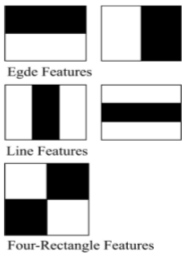
Gambar 2. 3. Proses Konvolusi Citra

2.5. Segmentasi Citra

Segmentasi citra adalah proses membagi sebuah citra ke dalam beberapa wilayah atau objek. Hasil dari segmentasi citra adalah memberikan label atau tanda seperti lingkaran, kurva, atau garis atau mengenali dan memotong tiap objek pada citra. Pada penelitian ini segmentasi citra pada wajah memanfaatkan fitur *Haar like Feature* pada metode *viola and jones* untuk menentukan objek mata, objek hidung dan objek mulut dan dibantu dengan metode *thresholding*. Adapun tahapan yang akan dilakukan adalah sebagai berikut:

2.5.1. *Haar Like Feature*

Haar Like Feature adalah teknik yang dilakukan dengan cara meyeleksi citra dengan cara mengkotak-kotakkan setiap daerah pada citra dari mulai ujung kiri atas sampai kanan bawah. Proses ini dilakukan untuk mencari apakah ada fitur wajah pada area tersebut. Dalam algoritma *Viola and Jones*, ada beberapa jenis fitur yang bisa digunakan seperti *Edge-feature, Line feature,* dan *Four-rectangle feature*. dan berikut adalah contoh fitur yang terdapat dalam fitur *haar like feature:*



Gambar 2. 4. Macam-Macam Variasi *Feature* pada *Haar*

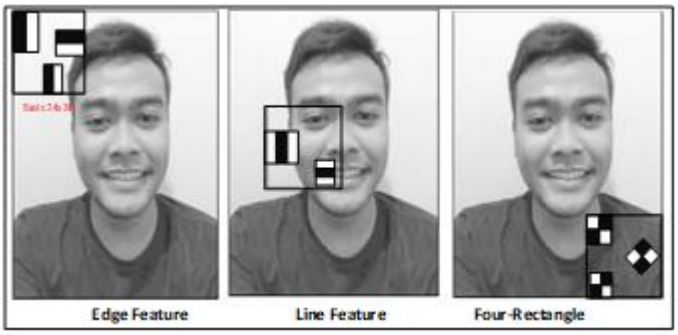
3 tipe kotak(rectangular) feature :

1. Tipe *two-rectangle feature (horisontal/vertikal)* adalah perbedaan antara jumlah nilai piksel pada kotak terang dengan nilai piksel pada kotak gelap dam untuk mendapatkan fitur dengan cara menjumlahkan nilai piksel pada kotak terang dan kotak gelap.
2. Tipe *three-rectangle feature* adalah dengan mempertimbangkan tiga nilai piksel pada kotak terang pertama dijumlahlam dengan nilai piksel pada kotak terang kedua dan kemudian ditambahkan dengan nilai piksel kotak gelap antara kotak terang kesatu dan kotak terang kedua.
3. Tipe *four-rectangle feature* adalah dengan mempertimbangkan nilai piksel kotak terang dan nilai piksel kotak gelap sebesar 2×2 yang tersusun secara diagonal dan menjumlahkan seluruh nilai piksel kotak terang dan kotak hitam.

Pada proses pemilihan fitur *Haar*, fitur-fitur tersebut digunakan untuk mencari fitur wajah seperti mata, hidung, dan mulut pada citra. Pada setiap kotak-kotak fitur tersebut terdiri dari beberapa piksel dan akan dihitung selisih antara nilai piksel pada kotak terang dengan nilai piksel pada kotak gelap. Apabila nilai selisih antara daerah terang dengan daerah gelap di atas nilai ambang (*threshold*), maka daerah tersebut dinyatakan memiliki fitur.

Berikut adalah contoh proses dalam pendeteksian fitur pada citra wajah dengan melakukan pencarian fitur dari ujung kiri atas sampai kanan bawah menggunakan basis

24×24 pada fitur *haar*, apakah terdapat fitur pada citra tersebut, dapat dilihat pada gambar berikut ini:



Gambar 2. 5. Ilustrasi *Haar Like Feature*

Adanya fitur Haar ditentukan dengan cara mengurangi rata-rata piksel pada daerah gelap dari rata-rata piksel pada daerah terang. Jika nilai perbedaannya itu diatas nilai ambang atau *threshold*, maka dapat dikatakan bahwa fitur tersebut ada. Nilai *haar like feature* diperoleh dari selisih jumlah nilai piksel daerah gelap dengan jumlah nilai piksel daerah terang :

$$F(Haar) = \sum F_{White} - \sum F_{Black} \quad \text{.....(2.8).}$$

Keterangan:

F(Haar) = Nilai fitur total

$\sum F_{White}$ = Nilai fitur pada daerah terang

$\sum F_{Black}$ = Nilai fitur pada daerah gelap

2.5.2. *Integral Image*

Integral Image digunakan pada algoritma untuk pendeteksian objek dimana proses perhitungan dengan menggunakan *intgral image* memerlukan waktu yang singkat atau cepat dan hasil yang akurat. Proses untuk menghitung hasil penjumlahan nilai piksel pada daerah yang dideteksi oleh fitur *haar*. Nilai-nilai piksel yang akan dihitung adalah nilai-nilai piksel dari sebuah citra masukan yang dilalui oleh fitur *haar*

pada saat pencarian fitur wajah. Pada setiap jenis fitur yang digunakan, pada setiap kotak-kotaknya terdiri dari beberapa piksel.

Dari nilai-nilai piksel yang didapatkan pada fitur tersebut, maka akan dihitung nilai *integral image* pada fitur tersebut dengan rumus sebagai berikut:

$$\boldsymbol{s(x,y) = i(x,y) + s(x,y) + s(x,y-1) + s(x-1,y) - s(x-1,y-1)} \text{ ..... (2.9).}$$

Dimana:

s(x,y) = merupakan nilai hasil penjulahan dari tiap-tiap piksel

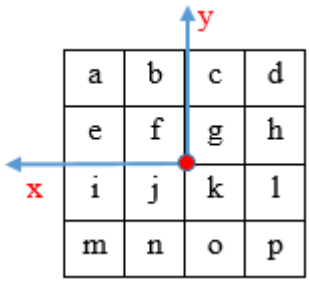
i(x,y) = merupakan nilai intensitas diperoleh dari nilai piksel dari citra masukan

s(x-1,y) = merupakan nilai piksel pada sumbu x

s(x,y-1) = merupakan nilai piksel pada sumbu y

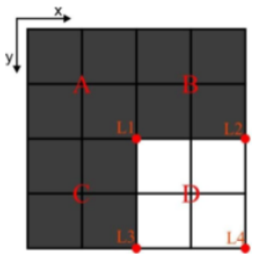
s*(x-1,y-1)* = merupakan nilai piksel diagonal

berikut adalah ilustrasi perhitngan *integral image* dalam proses pendeteksian objek, adalah sebagai berikut:



Gambar 2. 6. Ilustrasi arah perhitungan *integral Image*

Selanjutnya dilakukan perhitungan untunk mendapatkan nilai pada daerah tertentu, ilustrasinya sebagai berikut:



Gambar 2. 7. Ilustrasi perhitungan piksel pada daerah tertentu

Untuk menghitung jumlah piksel pada daerah tertentu, menggunakan rumus sebagai berikut:

D = L1 + L4 – (L2 + L3) .....(2.10).

Apabila sudah didapatkan nilai *integral image* dari sebuah citra masukan dan nilai jumlah piksel pada daerah tertentu, maka hasil tersebut akan dibandingkan antara nilai piksel pada daerah terang dan daerah gelap. Jika selisih nilai piksel pada daerah terang dengan nilai piksel pada derah gelap di atas nilai ambang (*threshold*) maka daerah tersebut dinyatakan memiliki fitur.

2.5.3. *Adaptive Booster*

Adaptive Booster adalah proses untuk memperbaharui bobot dengan melakukan perhitungan menggunakan persamaan jumlah gambar negatif dan jumlah gambar positif. Algoritma tersebut mengkombinasikan *performance* banyak *weak classifier* untuk menghasilkan *strong classifier*. *Weak classifier* dalam hal ini adalah nilai dari *haar-like feature*.

Bobot awal = $wj_{1_{y_i}} = \frac{1}{2m}, wj_{1_{y_i}} = \frac{1}{2l}$ ..... (2.11).

Untuk citra positif : $\in_t = \left(\sum_t^T w_{t,i}\right)|h_t(x) - y_i|$ ..... (2.12).

Untuk citra negatif: $\in_j = \left(\sum_j^J w_{t,i}\right)|h_j(x) - y_i|$ ..... (2.13).

Jika $\in_t \vee \in_j < 0$, hentikan iterasi. ..... (2.14).

Dimana:

w = *week classifier*

m = jumlah citra positif.

l = jumlah citra negatif.

t = indeks iterasi dari citra positif.

j = indeks iterasi dari citra negatif.

$h_t(x)$ = nilai fitur citra positif.

$h_j(x)$ = nilai fitur citra negatif.

Hasil Akhir klasifikasi yang diharapkan pada citra positif adalah sebagai berikut :

$$H(x) = \begin{cases} 1 \ \sum_{j=1}^{J} \alpha_j h_j \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ \\ 0 \qquad bukan\ objek. \end{cases} \quad \text{.....(2.15).}$$

Dimana :

$$\alpha_j = \log \frac{1}{\beta_t}, \alpha_t = \log \frac{1}{\beta_t} \quad \text{.....(2.16).}$$

Kondisi:

Jika posisi H(x) = Ketentuan 1 maka citra tersebut merupakan objek

Jika posisi H(x) = Ketentuan 0 maka citra tersebut merupakn bukan objek

Keterangan:

$H(x)$ = Strong Classifier atau kelasifikasi yang menyatakan objek atau bukan

αj = Tingkat pembelajaran citra positif.

αt = Tingkat pembelajaran citra negatif.

βj = Nilai bobot setelah *error rate* pada citra negatif

βt = Nilai bobot setelah *error rate* pada citra positif

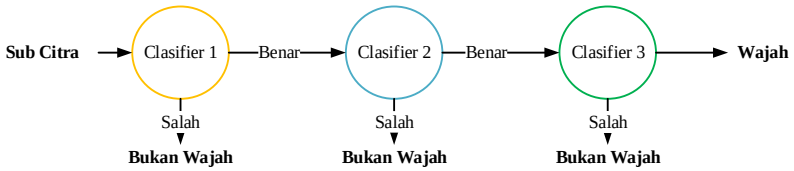
Hj = *weak* atau *basic classifiers* (awal dari klasifikasi) citra negatif.

Ht = *weak* atau *basic classifiers* (awal dari klasifikasi) citra positif.

2.5.4. *Cascade Clasifier*

Pada proses *cascade classifier* yang merupakan metode untuk mengkombinasikan *classifier* yang kompleks dalam sebuah struktur yang bertingkat

dan dapat meningkatkan kecepatan pendeteksian sebuah objek pada citra yang mefokuskan pada daerah citra yang berpeluang saja, dan berikut adalah proses dari *cascade classifier*:



Gambar 2. 8. *Cascade Clasifier* dengan N *stages*.

2.5.5. *Thresholding*

Untuk mengilangkan objek-bojek yang tidak diperlukan dilakukan tahap *Thresholding* dengan memisahkan latar belakang dari citra dengan memberi warna hitam, dalam tahap ini menggunakan metode *thresholding* untuk menjadikan citra biner.

Thresholding digunakan untuk mengatur jumlah derajat keabuan yang ada pada citra. Pada dasarnya proses *thresholding* adalah merubah kuantisasi pada citra, sehingga untuk melakukan *thresholding* dengan derajat keabuan menggunakan rumus sebagai berikut:

$$\boldsymbol{x = \frac{w}{b}} \quad \text{.....(2.17).}$$

Keterangan:

w = nilai derajat keabuan sebelum *thresholding.*

b = jumlah derajat keabuan yang diinginkan.

x = nilai derajat keabuan setelah *thresholding.*

Dengan menggunakan fungsi *THRESH_TOZERO_INV* pada pustaka *openCV* [12], fungsi ini akan menerapkan *thresholding* ke *single-channel array*. Fungsi ini biasanya digunakan untuk mendapatkan citra *bit-level* atau *binary* dari gambar *grayscale* atau untuk menghilangkan *noise*, yaitu dengan cara menyaring piksel yang

memiliki nilai terlalu kecil atau terlalu besar. Dan berikut adalah gambaran fungsi dari *THRESH_TOZERO_INV*:

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases} \quad \text{.....(2.18).}$$

Dimana:

src = input array (berupa *single-channel* antara 8-bit atau 32 bit).

dst = output array yang memiliki nilai yang sama dengan src.

thresh = nilai *thresholding*.

2.6. *Vector Quantization*

Vector Quantization (VQ) atau kuantisasi vektor adalah algoritma yang bekerja dengan memetakan vektor dari ruang vektor yang besar menjadi bentuk terbatas [13]. Pada awalnya *Vector Quantization* dikembangkan oleh Yoseph Linde, Andres Buzo, dan Robert M Gray.

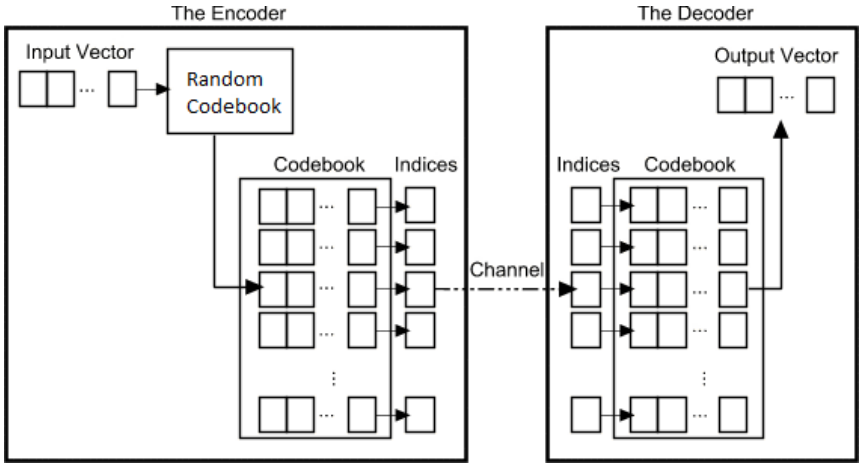
VQ adalah salah satu metode kompressi *lossy* yang paling kuat yang menghasilkan gambar berkualitas baik dengan kompleksitas perhitungan rendah. VQ memiliki penerapan yang paling besar dibidang kompresi citra dan video, *patte recognition* dan *watermaking.* Namun VQ memiliki beberapa kompleksitas yang tergantung dalam memilih sejumlah klaster dan pemilihan acak pada data latih baru. Pemilihan data latih baru yang salah dapat menghasilkan kualitas yang buruk dan proses pengujian memakan waktu yang lama. VQ memiliki tiga prosedur utama diantaranya pembuatan *coodebook, encoder* dan *decoder*.

Dalam pembuatan *codebook* berbagai citra dibagi menjadi beberapa vektor pelatihan kedalam *k-dimension,* dari perwakilan *codebook* dihasilkan dari vektor citra latih atau citra uji dengan mengklompokannya.

Encoder digunakan untuk membagi citra latih atau citra uji menjadi beberapa vektor dan setiap vektor dikodekan oleh indeks *codeword* kedalam tabel dan untuk membandingkan pencari vektor masukan dengan setiap vektor dalam *codebook* dan

menghasilkan indeks vektor antara vektor yang mewakili perbedaan minimum dari input vektor.

Sedangkan *decoder* adalah mengambil indeks untuk menempatkan kode vektor dalam *codebook* dan menghasilkan vektor baru serta menggunakan *codebook* yang sama sebagai penerjemah indeks untuk menghasilkan *codeword* yang sesuai untuk merekonstruksi citra, prinsip dari *decoder* adalah melibatkan pengkodean blok piksel yang disebut vektor daripada *encoding* yang hanya melibatkan satu piksel. Vector Quantization diilustrasikan pada gambar berikut ini:



Gambar 2. 9 *Vector Quantization*

Codebook didefinisikan sebagai fungsi pemetaan terhadap *k-dimension* kapasitas vektor sehingga tidak terbatas, $\mathbf{CB = \{C_1, C_2, C_3, ...C_{Nc}\}}$ dimana **CB** adalah *codebook* terdiri dari nilai ***Nc, Nc*** didefinisikan banyaknya nilai *codebook*. dan *codevector* didefinisikan $\mathbf{C_i = \{ C_{i1}, C_{i2}, C_{i3}, ... C_{ij} \}}$ dimana C_i didigunakan sebagai *codeword* dan C_{ij} adalah nilai dari *codeword*. Diasumsikan citra asli dengan ukuran $N \times N$, dan citra alsi dibagi menjadi beberapa sub-citra atau blok dengan ukuran $n \times n$, dengan persamaan $\mathbf{N_b = [N/n] \times [N/n]}$. N_b adalah nilai blok citra atau sub-citra.

Untuk membuat codebook harus memenuhi syarat sebagai berikut, dimana nilai $\mathbf{(N_b >> N_c)}$. Dalam metode VQ dalam satu vektor *codeword* dimasukan kedalam

codebook berdasarkan persamaan terbesar. Berdasarkan fungsi distorsi didefinisikan sebagai D sebagai berikut:

$$D(C) = \frac{1}{N_b} \sum_{j=1}^{N_c} \sum_{i=1}^{N_b} \mu_{ij} \left\| x_i - c_j \right\|^2 \quad \text{.....(2.19).}$$

Dimana μij disebut dengan koefisien indeks yang harus memenuhi dua kondisi sebagai berikut:

$$\sum_{j=1}^{N_c} \mu_{ij} = 1, \quad \forall i \in \{1,2 \ldots., N_b\} \quad \text{.....(2.20).}$$

$$\mu_{ij} = \begin{cases} 1 & if\ x_i\ is\ the\ \ jth\ region\ (R_i) \\ 0 & else \end{cases} \quad \text{.....(2.21).}$$

Untuk merancang *codebook* secara optimal, harus mempertimbangkan poin-poin berikut:

a. Region R_j = 1,2,...,N_c, yang mewakili vektor input pada *codeword*, dan c_j harus dipenuhi.

$$R_j = \{x \in X : d(x, c_j) < d(x, c_k), \quad \forall k \neq j\} \quad \text{.....(2.22).}$$

Dimana d(x,c_j) adalah jarak *Euclidean* antara vektor x dan c_j.

b. *Codeword* c_j adalah pusat dari region j pada region(R_j) dihitung sebagai

$$c_j = \frac{1}{N_j} i \quad for\ x_i \in R_i \quad \text{.....(2.23).}$$

Dimana, N_j adalah jumlah dari anggota vektor R_j.

Untuk menggunakan fungsi pemetaan untuk mempartisi vektor latih kedalam kelompok N. Fungsi pemetaan didefinisikan sebagai berikut $R^k \rightarrow CB$. Misalkan $X = (x_1, x_2, x_3, \ldots, x_k)$ menjadi citra latih dan *d(X, Y)* adalah jarak *Euclidean* antara dua vektor. Berikut tahapan dari metode VQ:

1. Pembuatan *codebook* secara acak, dengan CB_0 sebagai *codebook.*
2. $i = 0$.
3. Lakukan tahap berikut untuk setiap vektor latih:

a. Hitung jarak *Euclidean* antara vektor pelatihan dan *codebook* dalam $CB_{i.}$ Jarak *Euclidean* didefinisikan sebagai berikut:

$$d(X, C) = \sqrt{\sum_{t=1}^{k}(x_t - c_t)^2} \quad \text{.....(2.24).}$$

b. Cari *codeword* terdekat diantara $CB_{i.}$

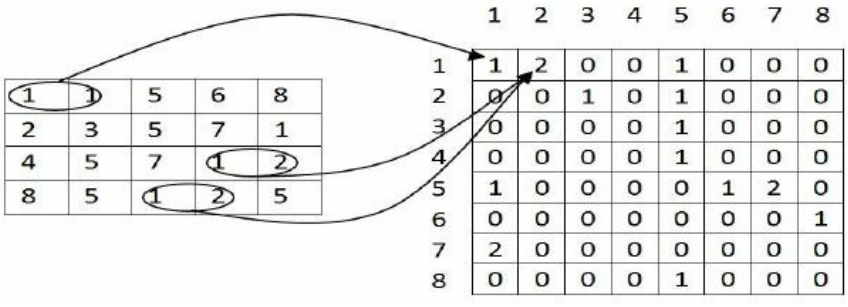
4. Pisahkan *codebook* kedalam kolom *N*.
5. Hitung rata-rata dari setiap kolom untuk mendapatkan *codebook* baru CB_i.
6. Hitung distorsi rata-rata pada $CB_{i+1.}$ Jika diubah dengan jumlah terkecil pada iterasi terakhir, perhitungan *codebook* akan terus diulang sampai prosdur terhenti. Jika tidak *i = i + 1* dan lanjutkan kepada nomor ke-3.

2.7. *Markov Stationary Feature*

Markov Stationary Feature (MSF) algoritma *extend* dari *Vector Quantization histogram*, dimana algoritma ini dapat menangani tiga level histogram yang berbeda, yang dapat meringkankan keterbatasan histogram, adapun cara kerja *Markov Stationary Feature* adalah [14]:

1. Membangkitkan *Spatial Co-Occurance Matrix*

   Tahap ini merupakan tahapan membangkitkan co-occurance matriks p_i merupakan tahap menghitung jumlah nilai matriks berdasarkan kemunculan koordinat, koordinat dipilih berdasarkan nilai ketetanggaan 0^0, 45^0, 90^0, dan 135^0, matriks *co-occurrence* dinotasikan sebagai C = $(C_{ij})_{KxK}$, berikut adalah ilustrasi *dari co-occurrence* matriks:



Gambar 2. 10. Ilustrasi *Co-occurance Matrix*

2. Setelah meperoleh *Co-occurance Matrix*, maka *Matrix Transition* yang sesuai P = $(p_{ij})_{KxK}$ diturunkan dari *Co-occurance* matrix C = $(C_{ij})_{KxK}$ dapat dengan mudah dikomputasi menggunakan formula sebagai berikut:

$$p_{ij} = \frac{c_{ij}}{\sum_{j=1}^{K} c_{ij}} \quad \text{..... (2.25).}$$

dimana p_{ij} menunjukan kemungkinan perubahan status c_i menjadi c_j.

3. Diharuskan status distribusi setelah langkah n adalah π(n) dan inisial distribusinya adalah ukuran invarian dari *markov chain*, yang dapat diakumulasikan.

$\pi \approx \frac{1}{K}\sum_{i=1}^{K} \vec{\alpha}_{ij}$, dimana $A_n = [\overrightarrow{a_1}, \dots, \overrightarrow{a_n}]^T$ .....(2.26).

dimana π adalah distribusi stationary yang memenuhi π = πP.

$$A_n = \frac{1}{n+1}\,(I + P + \cdots + P^n) \quad \text{..... (2.27).}$$

4. Fitur lengkap yang mencakup kombinasi distribusi awal didefinisikan oleh formula sebagai berikut:

$$\pi(0) = \frac{c_{ii}}{\sum_{i=1}^{K} c_{ii}} \quad \text{.....(2.28).}$$

dan stationary distribution diperoleh dengan mengunakan formula sebagai berikut:

$$\overrightarrow{MSF} = [\pi(0), \pi]^T \quad \text{..... (2.29).}$$

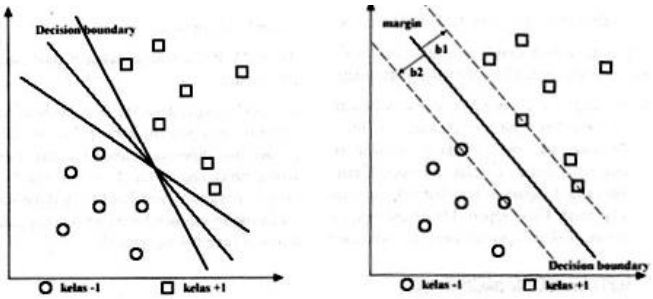
2.8. *Support Vector Machine*

Support Vector Machine (SVM) adalah metode yang berakar dari teori pembelajaran statistik, SVM bekerja dengan menyimpan sebagian kecil dari data latih untuk digunakan pada saat prediksi, data-data yang berkontribusi tersebut disebut support vector sehingga metodenya juga disebut *Support Vector Machine*.

2.8.1. Konsep SVM

Ide dasar SVM adalah memaksimalkan batas hyperlane, yang diilustrasikan pada gambar 2.11. Pada gambar (a) ada sejumlah pilihan *hyperlane* yang mungkin

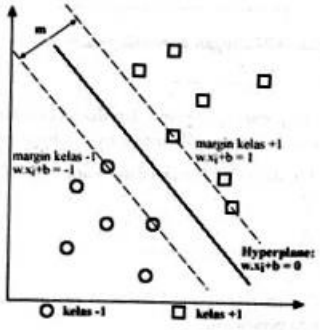
untuk set data, sedangkan gambar (b) merupakan *hyperlane* dengan margin yang paling maksimal. Meskipun sebenarnya pada gambar (a) bisa juga menggunakan *hyperlane* sembarang, tetapi *hyperlane* dengan marign yang maksimal akan memberikan generalisasi yang lebih baik pada metode klasifikasi.



Gambar 2. 11. Batas keputusan yang mungkin untuk set data.

Konsep klasifikasi dengan SVM dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* (batas keputusan) terbaik yang berfungsi sebagai pemisah dua buah kelas data pada input space. Gambar 2.11. memperlihatkan beberapa data yang merupakan anggota dari dua buah kelas data, yaitu positif (+1) dan negatif (-1). Data yang tergabung pada kelas negatif (-1) disimbolkan dengan bentuk lingkaran, sedangkan data pada kelas positif (+1), disimbolkan dalam bentuk bujur sangkar.

Hyperplane pemisah terbaik antara kedua kelas dapat ditentukan dengan mengukur margin *hyperplane* tersebut dan mencari titik maksimalnya. Margin adalah jarak *hyperplane* tersebut dengan data terdekat dari masing-masing kelas. Data yang paling dekat ini disebut dengan *support vector*. Garis solid pada gambar 2.11 (b) sebelah kanan menunjukan hyperplane yang terbaik, yaitu yang terletak pada tengah-tengah kedua kelas, sedangkan data lingkaran dan bujur sangkar yang dilewati batas garis margin (garis putus-putus) adalah *support vector*. Usaha untuk mencari lokasi hyperplane ini merupakan inti dari proses pelatihan pada SVM.



Gambar 2. 12. *Margin hyperplane*

Data yang tersedia dinotasikan sebagai $\overrightarrow{x_i} \in \Re^d$ sedangkan label masing-masing dinotasikan yi ∈ {-1,+1}untuk i = 1,2,...n ,yang mana n adalah banyaknya data. Diasumsikan kedua *class* negatif (–1) dan positif (+1) dapat terpisah secara sempurna oleh *hyperplane* berdimensi *d*, yang didefinisikan sebagai berikut :

$$\vec{w} \,.\, \vec{x} + b = 0 \quad \text{.....(2.30).}$$

Pattern $\overrightarrow{x_i}$ yang termasuk *class* negatif (-1) dapat dirumuskan sebagai *pattern* yang memenuhi pertidaksamaan sebagai berikut :

$$\vec{w} \,.\, \vec{x} + b \leq -1 \quad \text{.....(2.31).}$$

Sedangkan *pattern* $\overrightarrow{x_i}$ yang termasuk *class* positif (+1) adalah sebagai berikut :

$$\vec{w} \,.\, \vec{x} + b \geq +1 \quad \text{.....(2.32).}$$

Margin terbesar dapat ditemukan dengan memaksimalkan nilai jarak antara *hyperplane* dan titik terdekatnya, yaitu $1 / \|\vec{w}\|$. Hal ini dapat dirumuskan sebagai *Quadratic Programming (QP) problem*, yaitu mencari titik minimal persamaan (2.52), dengan memperhatikan *constraint* persamaan sebagai berikut:

$$\min_{\vec{w}} \tau(w) = \frac{1}{2}\|\vec{w}\|^2 \quad \text{.....(2.33).}$$

$$y_i(\overrightarrow{x_i} \,.\, \vec{w} + b) - 1 \geq 0, \forall i \quad \text{.....(2.34).}$$

Problem ini dapat dipecahkan dengan berbagai teknik komputasi, di antaranya *Lagrange Multiplier*.

$$L(\vec{w}, b, \alpha) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i\big((\vec{x}_i \,.\, \vec{w} + b) - 1\big)\right), \; i = 1,2 \ldots l \quad \text{....(2.35).}$$

α_i adalah *Langrange multipliers*, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan (2.35) dapat dihitung dengan meminimalkan *L* terhadap $\vec{w}$ dan *b*, dan memaksimalkan *L* terhadap α_i. Dengan memperhatikan sifat bahwa pada titik optimal *gradient L* = 0, persamaan (2.35) dapat dimodifikasi sebagai maksimalisasi *problem* yang hanya mengandung α_i, sebagaimana persamaan (2.35) dibawah

Maximize :

$$\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \vec{x}_i . \vec{x}_j \quad \text{..... (2.36).}$$

Dimana $\alpha_i \geq 0$ ($i = 1,2, \ldots . l$) dan $\sum_{i=1}^{l} \alpha_i y_i = 0$. Dari hasil perhitungan ini diperoleh α_i yang kebanyakan bernilai positif. Data yang berkorelasi dengan α_i yang positif inilah yang disebut *support vector*.

Untuk mendapatkan nilai α_i, langkah pertama adalah mengubah setiap *feature* menjadi nilai vektor (*support vector*) = $\frac{x}{y}$. Kemudian vektor akan dimasukkan kedalam persamaan *kernel trick phi* φ yaitu sebagai berikut :

$$\varphi \begin{bmatrix} x \\ y \end{bmatrix} = \begin{cases} \sqrt{x_n^2 + y_n^2} > 2, maka \begin{bmatrix} \sqrt{x_n^2 + y_n^2} - x & |x - y| \\ \sqrt{x_n^2 + y_n^2} - x & |x - y| \end{bmatrix} \\ \sqrt{x_n^2 + y_n^2} < 2, maka \begin{bmatrix} x \\ y \end{bmatrix} \end{cases} \text{..... (2.37).}$$

Kemudian untuk mencari nilai α_i didapatkan dari persamaan sebagai berikut :

$$\sum_{i=1,j=1}^{n} a_i T_i^T T_j \quad \text{..... (2.38).}$$

Dan selanjutnya menggunakan persamaan sebagai berikut :

$$W = \sum_{i=1}^{n} a_i T_i \; dan \; \sum_{i=1,j=1}^{n} a_i T_i = y_i \quad \text{..... (2.39).}$$

Terakhir akan dicari nilai w dan b untuk menemukan *hyperplane* sebagai patokan proses klasifikasi dengan persamaan sebagai berikut :

$$y = wx + b \, , \; w = \sum_{i}^{n} a_i s_i \quad \text{..... (2.40).}$$

Nilai s_i merupakan nilai *support vector* yang telah dihitung sebelumnya. Dengan demikian proses klasifikasi selesai dengan memperhatikan *hyperplane* nya.

2.8.2. *Kernel Trick*

Feature space merupakan dimensi yang lebih tinggi dari vektor input (*input space*). Hal ini mengakibatkan komputasi pada *feature space* mungkin sangat besar, karena ada kemungkinan *feature space* dapat memiliki jumlah *feature* yang tidak terhingga. Selain itu, sulit mengetahui fungsi transformasi yang tepat. Untuk mengatasi masalah ini, pada SVM digunakan "*kernel trick*". Fungsi kernel yang umum digunakan adalah sebagai berikut [14]:

1. *Kernel linier*

   $K(x_i, x) = x_i^T x$ .....(2.41).

2. *Polynomial kernel*

   $K(x_i, x) = (\gamma . x_i^T x + r)^p, \gamma > 0$ .....(2.42).

   Radial basis function (RBF)

   $K(x_i, x) = \exp(-\gamma \, |x_i - x|^2) \, , \gamma > 0$ .....(2.43).

   Sigmoid kernel

   $K(x_i, x) = \tanh(\gamma x_i^T x + r)$ .....(2.44).

2.8.3. *SVM Multiclass*

SVM hanya dapat mengklasifikasikan data ke dalam dua kelas (klasifikasi biner). Namun, penelitian lebih lanjut untuk mengembangkan SVM sehingga bisa mengklasifikasi data yang memiliki lebih dari dua kelas, terus dilakukan. Ada dua pilihan untuk mengimplementasikan SVM *multiclass* yaitu dengan menggabungkan beberapa SVM biner atau menggabungkan semua data yang terdiri dari beberapa kelas ke dalam sebuah bentuk permasalah optimasi. Namun, pada pendekatan yang kedua permasalahan optimasi yang harus diselesaikan jauh lebih rumit. Berikut ini adalah metode yang digunakan untuk mengimplementasikan *multiclass* SVM dengan pendekatan Metode *one-against-all*. Contohnya, terdapat permasalahan klasifikasi dengan 7 buah kelas. Untuk pelatihan digunakan 7 buah SVM biner penggunanya

dalam mengklasifikasi kelas pada data baru dapat dilihat pada persamaan sebagai berikut [15]:

$$Kelas\ x = \arg\max_{i=1..k}(\left(w^{(i)}\right)^{T}.\varphi(x) + b^{(i)}) \quad \text{..... (2.45).}$$

Dengan menentukan *hyperplane* terbesar pada nilai x maka akan mengklasifikasikan kelas tersebut.

Tabel 2. 2. Contoh 7 SVM biner dengan metode *One-against-all*

$y_i = 1$	$y_i = -1$	Hipotesis
Kelas 1	Bukan kelas 1	$f^1(x) = (w^1)x + b^1$
Kelas 2	Bukan kelas 2	$f^2(x) = (w^2)x + b^2$
Kelas 3	Bukan kelas 3	$f^3(x) = (w^3)x + b^3$
Kelas 4	Bukan kelas 4	$f^4(x) = (w^4)x + b^4$
Kelas 5	Bukan kelas 5	$f^5(x) = (w^5)x + b^5$
Kelas 6	Bukan kelas 6	$f^6(x) = (w^6)x + b^6$
Kelas 7	Bukan kelas 7	$f^7(x) = (w^7)x + b^7$

2.9. NetBeans IDE

NetBeans merupakan sebuah proyek kode terbuka yang sukses dengan pengguna yang sangat luas, komunitas yang terus tumbuh, dan memiliki hampir 100 mitra (dan terus bertambah). Sun Microsystems mendirikan proyek kode terbuka

NetBeans pada bulan Juni 2000 dan terus menjadi sponsor utama, Saat ini terdapat dua produk : NetBeans IDE dan NetBeans Platform.

NetBeans IDE adalah sebuah lingkungan pengembangan - sebuah kakas untuk pemrogram menulis, mengompilasi, mencari kesalahan dan menyebarkan program. Netbeans IDE ditulis dalam Java - namun dapat mendukung bahasa pemrograman lain. Terdapat banyak modul untuk memperluas Netbeans IDE. Netbeans IDE adalah sebuah produk bebas dengan tanpa batasan bagaimana digunakan.

Tersedia juga NetBeans Platform; sebuah fondasi yang modular dan dapat diperluas yang dapat digunakan sebagai perangkat lunak dasar untuk membuat aplikasi desktop yang besar. Mitra ISV menyediakan plug-in bernilai tambah yang dapat dengan mudah diintegrasikan ke dalam Platform dan dapat juga digunakan untuk membuat kakas dan solusi sendiri.

Kedua produk adalah kode terbuka (open source) dan bebas (free) untuk penggunaan komersial dan non komersial. Kode sumber tersedia untuk guna ulang dengan lisensi Common Development and Distribution License (CDDL).

2.10. *OpenCV*

OpenCV (*Open Source Computer Vision Library*) adalah sebuah *library* perangkat lunak yang ditujukan untuk pengolahan citra yang biasa digunakan secara *real-time*. *OpenCV* dibuat oleh Intel dan merupakan *library* yang bebas digunakan dan berada dalam naungan sumber terbuka (*Open source*) dari lisensi. Umumnya *library* ini menggunakan bahasa pemrograman C/C++, tetapi akhir – akhir ini sudah dikembangkan keberbagai bahasa pemrograman seperti Phyton, Javascript dan Java. *OpenCV* mempunyai modul/subrutin yang ada pada *library* yang ada pada JAVA akan dijelaskan sebagai berikut [12]:

1. *CascadeClassifier* merupakan pustaka untuk untuk menggunakan algoritma haar cascade.

2. *Mat* merupakan tipe data bentukan opencv untuk menampung array agar bisa diproses secara komputasi oleh operasi-operasi algoritma yang menggunakan pustaka dari opencv.
3. *MatOfRect* Merupakan pustaka untuk mengambil nilai mat yang ada telah diseleksi oleh fitur *haar cascade*.
4. *Rect* merupakan pustaka untuk mendapatkan *rectangle* dari proses proses *haar cascade*.
5. *CvType* adalah library untuk mengambil jenis data dari mat.
6. *Scalar* adalah tipe data bentukan untuk menampung data yang multi channel seperti gambar, sinyal, dan lain sebagainya.
7. *TermCriteria* merupakan pustaka untuk membatasi iterasi.
8. *CvSVM* merupakan pustaka yang digunakan untuk mengkonversi nilai yang ada pada XML menjadi hyperplane yang digunakan oleh SVM.
9. *CvSVMParams* merupakan pustaka yang digunakan untuk memanggil SVM machine learning.

2.11. *Unified Modelling Language* (UML) 2.0

Diagram UML adalah sekumpulan alat yang digunakan untuk melakukan abstraksi terhadap sebuah sistem atau perangkat lunak berbasis objek. UML merupakan singkatan dari Unified Modeling Language. Dalam UML sendiri terdapat beberapa diagram yang wajib dikuasai yaitu:

1. *Structural Diagram*
   a. *Class Diagram*, diagram ini terdiri dari *class*, *interface*, *association*, dan *collaboration*. Diagram ini menggambarkan objek - objek yang ada di sistem.
   b. *Deployment Diagram*, diagram ini menggambarkan kumpulan node dan hubungan antar node. Node adalah entitas fisik dimana komponen di-deploy. Entitas fisik ini dapat berupa server atau perangkat keras lainnya.

2. *Behavioral Diagram*
   a. *Use case Diagram*, diagram ini menggambarkan kumpulan use case, aktor, dan hubungan mereka. *Use case* adalah hubungan antara fungsionalitas sistem dengan aktor internal/eksternal dari sistem.
   b. *Sequence Diagram*, diagram ini menggambarkan interaksi yang menjelaskan bagaimana pesan mengalir dari objek ke objek lainnya.
   c. *Statechart Diagram*, diagram ini menggambarkan bagaimana sistem dapat bereaksi terhadap suatu kejadian dari dalam atau luar. Kejadian (*event*) ini bertanggung jawab terhadap perubahan keadaan sistem.
   d. *Activity Diagram*, menggambarkan aliran kontrol sistem. Diagram ini digunakan untuk melihat bagaimana sistem bekerja ketika dieksekusi.

2.12. *Vector Quantization Library*

Vector Quantization Library merupakan pustaka pada bahasa pemograman java yang dibangun oleh Sherif Abdul Navi untuk menggunakan *Vector Quantization* [13], dalam penggunaanya yang dilakukan adalah memanggil function dengan meng-*import* pustaka sebagai berikut :

```
import VectorQuantization.*;
```

Fungsi yang digunakan adalah :

1. *Compress(vectorHeight, vectorWidth, codebookSize, path)*
   Kegunaan dari fungsi ini adalah melakukan proses yang ada pada tahapan *vector quantization* sampai mendapatkan *codebook* yang telah diupdate dan nomor *index*.
2. *Decompress(VectorQuantization.getCompressedPath(path))*
   Kegunaan dari fungsi ini adalah mengkonversi *codebook* yang telah di-*update* dan nomor *index* menjadi bentuk gambar.

2.13. *ImageJ*

ImageJ adalah sebuah program atau *tools* yang digunakan untuk pemprosesan citra atau gambar berbasis *java open source* yang terinspirasi oleh *NIH Image* pada Macintosh. ImageJ dapat dijalankan pada semua perangkan terutama yang memiliki

mesin virtual Java 1.8 dan tersedia di berbagai flatform seperti Windows, Mac OS, Mac OS X dan Linux [16].

Dapat menampilkan, mengedit, menganalisis, memproses, menyimpan dan mencetak gambar 8-bit, 16-bit dan 32-bit. Dan imageJ dapat membaca banyak format gambar termasuk TIFF, GIF, JPEG, BMP, DICOM, FITS dan "RAW". Ini mendukung "*Stacks*" gambar bertumpuk dan kemudian serangkaian gambar yang berbagi dijadikan satu proses, yang disebut *multithread*, sehingga operasi yang memakan waktu seperti membaca file gambar dapat dilakukan secara *multitasking* dengan operasi lain.

ImageJ dapat menghitung luas dan statistik nilai piksel dari pilihan yang ditentukan pengguna. Dapat mengukur jarak dan sudut. Ini dapat membuat histogram kepadatan dan plot profil garis. ImageJ mendukung fungsi pemrosesan gambar standar seperti manipulasi kontras, penajaman, perataan, deteksi tepi dan penyaringan median.

Data transformasi geometris seperti penskalaan, rotasi dan membalik. Gambar dapat diperbesar hingga 32:1 dan turun ke 1:32. Semua fungsi analisis dan pemrosesan tersedia di setiap faktor pembesaran. Program ini mendukung sejumlah jendela (gambar) secara bersamaan, hanya dibatasi oleh memori yang tersedia.

2.14. Metode Pengujian *Confusion matrix*

Confusion matrix merupakan salah satu metode yang dapat digunakan untuk mengukur kinerja suatu metode klasifikasi. Pada dasarnya *confusion matrix* mengandung informasi yang membandingkan hasil klasifikasi yang dilakukan oleh sistem dengan hasil klasifikasi yang seharusnya [17].

Berdasarkan jumlah keluaran kelasnya, sistem klasifikasi dapat dibagi menjadi 4 (empat) jenis yaitu klasifikasi *binary, multi-class, multi-label* dan *hierarchical* [18].

1. Klasifikasi *binary*, data masukan dikelompokkan ke dalam salah satu dari dua kelas. Jenis klasifikasi ini merupakan bentuk klasifikasi yang paling sederhana dan banyak digunakan.

Pada pengukuran kinerja menggunakan *confusion matrix*, terdapat 4 (empat) istilah sebagai representasi hasil proses klasifikasi. Keempat istilah tersebut adalah *True Positive* (TP), *True Negative* (TN), *False Positive* (FP) dan *False Negative* (FN).

1. *True Negative* (TN) merupakan jumlah data negatif yang terdeteksi dengan benar,
2. *False Positive* (FP) merupakan data negatif namun terdeteksi sebagai data positif,
3. *True Positive* (TP) merupakan data positif yang terdeteksi benar, dan
4. *False Negative* (FN) merupakan kebalikan dari *True Positive*, sehingga data posifit, namun terdeteksi sebagai data negatif.

Pada jenis klasifikasi *binary* yang hanya memiliki 2 keluaran kelas, seperti berikut ini:

Tabel 2. 3. Tabel klasifikasi *binary*

Sebenarnya	Prediksi						
	Bahagia	Sedih	Terkejut	Takut	Marah	Muak	Netral
Bahagia	TP	FN	FN	FN	FN	FN	FN
Sedih	FN	TP	FN	FN	FN	FN	FN
Terkejut	FN	FN	TP	FN	FN	FN	FN
Takut	FN	FN	FN	TP	FN	FN	FN
Marah	FN	FN	FN	FN	TP	FN	FN
Muak	FN	FN	FN	FN	FN	TP	FN
Netral	FN	FN	FN	FN	FN	FN	TP

Berdasarkan nilai *True Positive* (TP) dan *False Negative* (FN) dapat diperoleh nilai akurasi. Nilai akurasi menggambarkan seberapa akurat sistem dapat mengklasifikasikan data secara benar. Dengan kata lain, nilai akurasi merupakan perbandingan antara data yang terklasifikasi benar dengan keseluruhan data. Nilai akurasi dapat diperoleh dengan perhitungan sebagai berikut:

$$\textbf{Akurasi} = \frac{\textbf{TP}}{(\textbf{TP}+\textbf{FN}_1+\cdots+\textbf{FN}_6)} \times \textbf{100}\% \quad \text{.....(2.46).}$$

Keterangan:

True Positive (TP) = merupakan data positif yang terdeteksi benar, dan

False Negative (FN) = merupakan kebalikan dari *True Positive*, sehingga data positif, namun terdeteksi sebagai data negatif.

2.15. *K-Fold Cross-Validation*

K-fold cross validation adalah salah satu metode untuk mengevaluasi kinerja classifier, metode ini dapat digunakan apabila memiliki jumlah data yang terbatas (jumlah instance tidak banyak) [19]. *K-fold cross validation* merupakan salah satu metode yang digunakan untuk mengetahui rata-rata keberhasilan dari suatu sistem dengan cara melakukan perulangan dengan mengacak atribut masukan sehingga sistem tersebut teruji untuk beberapa atribut input yang acak. *K-fold cross validation* diawali dengan membagi data sejumlah *n-fold* yang diinginkan. Dalam proses cross validation data akan dibagi dalam n buah partisi dengan ukuran yang sama D_1, D_2, D_3 .. D_n selanjutnya proses uji dan latih dilakukan sebanyak *n* kali. Dalam iterasi ke-*i* partisi D_i akan menjadi data uji dan sisanya akan menjadi data latih. Untuk penggunaan jumlah fold terbaik untuk uji validitas, dianjurkan menggunakan *10-fold cross validation* dalam model.