BAB 2
LANDASAN TEORI

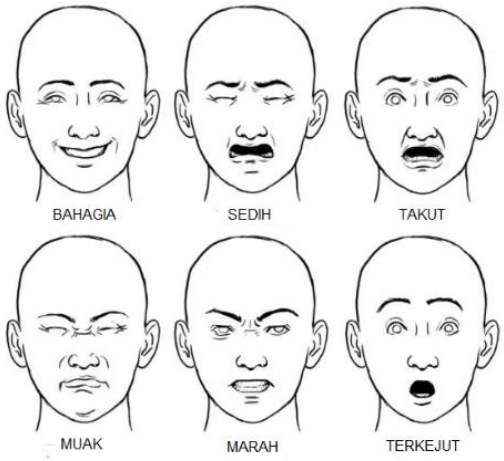
2.1. Ekspresi Wajah

Ekspresi Wajah atau mimik adalah hasil dari satu atau lebih dari gerakan atau posisi otot pada wajah. Ekspresi wajah merupakan salah satu bentuk komunikasi nonverbal, dan dapat menyampaikan keadaan emosi kepada orang yang mengamati. Adapun gerakan atau posisi otot pada ekspresi wajah, bisa dilihat seperti pada tabel berikut[1] :

Tabel 2. 1. Ciri Ekspresi Wajah

Emosi	Ciri	Dahi	Mulut
Bahagia	Bagian bawah kelopak mata agak terangkat, terlihat ada kerutan, dan mata menyipit.		Bibir dan mulut melebar, kadang-kadang gigi terlihat.
Sedih	Ujung dalam alis terangkat	Berkerut	Sudut mulut tertarik kebawah dan bibir gemetar
Terkejut	Seluruh alis terangkat dan mata membesar.		Rahang menurun dan mulut terbuka perlahan
Takut	Kelopak mata bagian atas terangkat, bagian putih mata terlihat jelas, kelopak mata bagian bawah menegang dan terangkat.	Berkerut	Bibir ditarik.
Marah	Alis ditarik ke dalam, mata menyipit.		Bibir tertutup rapat.
Muak	Kelopak mata bagian bawah terangkat dan berkerut.		Merapat. Kedua bibir terangkat atau cemberut

Dari Tabel 2. 1 bisa digambarkan dalam sebuah ilustrasi yang dapat dilihat pada Gambar 2. 1.



Gambar 2. 1. Bentuk Ekspresi Wajah

2.2. Deteksi Wajah (*Viola-Jones*)

Deteksi Wajah pada penelitian ini menggunakan metode yang dikemukakan oleh Viola & Jones [8], terbagi menjadi 4 komponen utama: *Grayscale*, *Haar Like Feature*, *Integral Image*, *Adaptive Boosting* dan *Cascade of Classifier.*

2.2.1. Citra *Grayscale*

Citra Asli memiliki tiga komponen utama yaitu *Red, Green, dan Blue* (RGB). Grayscale adalah citra yang hanya memiliki satu buah kanal sehingga yang ditampilkan hanyalah intensitas atu dikenal juga dengan istilah drajat keabuan, untuk mendapatkan citra *grayscale* dari citra berwarna dapat dikonversi dengan rumus[9]. :

$$y=0.29R+0.59G+11B \quad (2.11)$$

2.2.2. Haar Like Feature

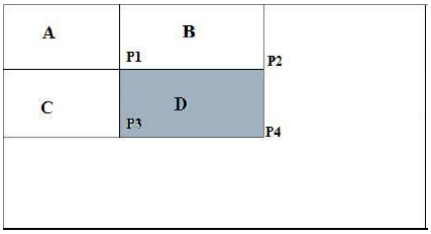
Teknik yang dilakukan yaitu dengan cara mengkotak-kotakkan setiap daerah pada citra dari mulai ujung kiri atas sampai kanan bawah. Proses ini dilakukan untuk mencari apakah ada fitur wajah pada area tersebut. Dalam

algoritma Viola-Jones, ada beberapa jenis fitur yang bisa digunakan seperti *Edge-feature*, *Line feature*, dan *Four-rectangle feature*.

Gambar 2. 2. Ilustrasi *Haar Like Feature*

2.2.2.1. Integral Image

Integral image sering digunakan pada algoritma untuk pendeteksian wajah. Dengan menggunakan *integral image* proses perhitungan bisa dilakukan hanya dengan satu kali scan dan memakan waktu yang cepat dan akurat. *Integral image* digunakan untuk menghitung hasil penjumlahan nilai piksel pada daerah yang dideteksi oleh fitur haar. Berikut adalah simulasi dari integral image.



Gambar 2. 3. Ilustrasi *Integral Image*

$$P_1 = A, P_2 = A, P_3 = A + C, P_4 = A + B + C + D \quad \text{..... (2.1)}$$

$$P_1 + P_4 - P_3 = A + A + B + C + D - A - B - A - C = D \quad \text{..... (2.2)}$$

2.2.3. Adaptive Booster (AdaBoost)

Adaptive Booster (AdaBoost) adalah Mesin belajar yang digunakan metode AdaBoost digunakan untuk meningkatkan performa pengklasifikasian fitur. Algoritma tersebut mengkombinasikan *performance* banyak *weak classifier* untuk menghasilkan *strong classifier*. *Weak classifier* dalam hal ini adalah nilai dari *haar-like feature*[12].

Bobot awal = $wj_{1_{y_i}} = \frac{1}{2m}, wj_{1_{y_i}} = \frac{1}{2l}$ ..... (2.3)

$h_t(x)$ merupakan nilai fitur gambar positif. ..... (2.4)

$h_j(x)$ merupakan nilai fitur gambar negatif. ..... (2.5)

Untuk citra positif : $\epsilon_t = \left(\sum_t^T w_{t,i}\right)|h_t(x) - y_i|$ ..... (2.6)

Untuk citra negatif: $\epsilon_j = \left(\sum_j^J w_{t,i}\right)|h_j(x) - y_i|$ ..... (2.7)

Jika $\epsilon_t \vee \epsilon_j < 0$, hentikan iterasi. ..... (2.8)

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

$$H(x) = \begin{cases} 1 \ \sum_{j=1}^{J} \alpha_j h_j \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ \\ 0 \qquad bukan\ objek. \end{cases}$$ ..... (2.9)

Dimana :

$\alpha_j = \log \frac{1}{\beta_t}, \alpha_t = \log \frac{1}{\beta_t}$ ..... (2.10)

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

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

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

αj = Tingkat pembelajaran gambar positif.

αt = Tingkat pembelajaran gambar negatif.

βj=Nilai bobot setelah error rate pada gambar negatif

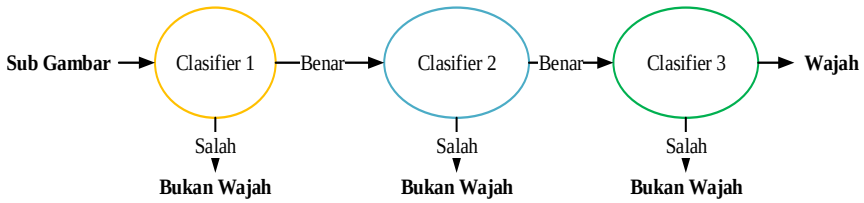
βt= Nilai bobot setelah error rate pada gambar positif

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

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

2.2.4. Cascade Classifier

Cascade Classifier adalah turunan dari *decision tree* dimana bagian clasifier dilatih hampir semua *object of interest* dan menolak bagian tertentu yang tidak termasuk dalam *object pattern.* Ilustrasi dari *Cascade Clasifier* dapat dilihat pada Gambar 2. 4.



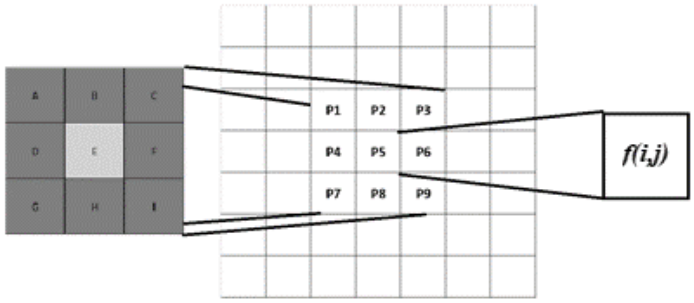
Gambar 2. 4. *Cascade Classifier* dengan *N stages*.

2.3. Lowpass Filter

Lowpass Filter atau penapisan lolos rendah adalah suatu proses pada citra untuk meloloskan data pada frekuensi rendah dan akan mengurangi atau menolak data pada frekuensi tinggi. Berikut adalah aturan yang digunakan untuk lowpass filter[9]. Dalam implementasinya *lowpass filter* menggunakan konvolusi untuk komputasinya, Konvolusi citra adalah operasi pengolahan citra yang mengalikan citra dengan sebuah *mask* atau *kernel* [9].

Konvolusi 2 fungsi $f(x)$ dan $g(x)$.

$$f(x,y) * g(x,y) = f(x,y) \otimes g(x,y) \quad \text{..... (2.14)}$$



Gambar 2. 5. Proses Konvolusi Citra

$f(i,j)$ = (A×P1) + (B×P2) + (C×P3) + (D×P4) + (E×P5) + (F×P6) + (G×P7) + (H×P8) + (I×P9) .....

$\sum_i \sum_j g(i,j)\, A = 1,$

dimana $g(i,j) > 0$ ..... (2.16)

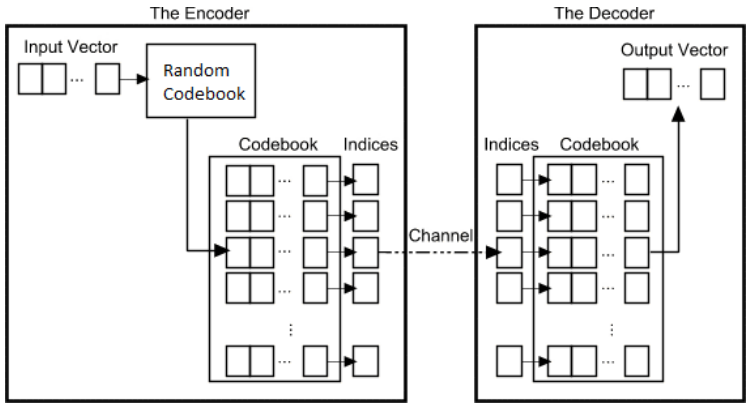
Kernel yang digunakan untuk penelitian adalah *mean filter* dengan *kernel* sebagai berikut.

$$g = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \quad \text{..... (2.17)}$$

Tujuan dilakukan *lowpass filter* yaitu untuk menghilangkan *noise* dan membuat citra menjadi lebih halus.

2.4. *Vector Quantization* (VQ)

Vector Quantization (VQ) atau kuantisasi vektor adalah algoritma yang bekerja dengan memetakan vektor dari ruang vektor yang besar menjadi bentuk terbatas[9]. Pada awalnya VQ dikembangkan oleh Yoseph Linde, Andres Buzo, dan Robert M Gray dan metode untuk membangkitkan *codebook* adalah *Ludo Buzo Gray* (LBG). [12] VQ diilustrasikan pada Gambar 2. 6.



Gambar 2. 6. *Vector Quantization*

Tahapan dari VQ adalah :

1. Buat citra berukuran N×N menjadi blok-blok berukuran $n \times n$,. Blok-blok citra ini dinotasikan sebagai $X = \{x_1, x_2, x_3, \dots, c_{Nb}\}$, dimana nilai *input vector* $x_i = \{x_{i1}, x_{i2}, x_{i3}, \dots, c_{(n\times n)}\}$.
2. Lakukan bangkitkan *codebook* dengan nilai random yang bernilai antara 0-255 berukuran Nc. *Codebook* didefinisikan sebagai $C = \{c_1, c_2, c_3, \dots, c_{Nc}\}$ *codebook* memiliki komponen yaitu *codeword* yang didefinisikan sebagai $c_i = \{c_{i1}, c_{i2}, c_{i3}, \dots, c_{(n\times n)}\}$.
3. Membangkitkan *Codebook* baru yang akan digunakan pada proses pembentukan citra terkompresi, dengan mengimplementasikan metode LBG, adapun tahapannya sebagai berikut :
    a. Mencari indeks terdekat

    Dilakukan perulangan dari 0 sampai dengan Nb, dimana merupakan jumlah dari blok-blok yang terdapat di dalam citra, lalu dilakukan lagi perulangan dari 0 sampai dengan Nc

    Untuk menghitung nilai terdekat maka digunakan rumus :

$$d(x, c_i) = \sqrt{\sum_{t=0}^{n-1}(x_t - c_t)^2} \quad \text{..... (2.18)}$$

b. *Update Codebook*

Update codebook adalah tahapan untuk membuat *codebook* baru yang akan digunakan untuk melakukan mapping terhadap *codebook* baru sehingga menghasilkan citra baru. Dari indeks yang telah didapat maka centroid yang didapat dari nomor $indeks$, gunakan rumus sebagai berikut :

$$c_t = \frac{x_t}{N_j} \quad \text{.....(2.19)}$$

Setelah didapatkan *codebook* baru didapatkan cari nilai distorsi antara nilai dengan menggunakan persamaan berikut :

$$d(X,C) = \sqrt{\sum_{b=0}^{N_b-1}\sum_{t=0}^{n-1}(c_{index,t} - x_{b,t})^2} \quad \text{.....(2.20)}$$

Setelah didapatkan *codebook* penjumlahan jarak sampai setiap blok dan jumlahkan.

$$D_m = \frac{d(X,C)}{Nb \times n} \quad \text{.....(2.21)}$$

Hentikan proses jika $D_m < \varepsilon$, dimana $\varepsilon = 0.001$.

4. Bentuk Citra Terkompresi

Setelah didapatkan *codebook* baru maka dilakukan pembentukan kembali citra wajah dengan melakukan pemetaan terhadap nilai indeks. Dari proses sebelumnya diketahui ada kemunculan indeks, pada setiap blok kemunculan indeks ini ditampung kemudian dipetakan dengan *codebook* baru, simulasinya sebagai berikut :

		baris = indeks terdekat				
Blok Citra			Codebook			
35	32	baris 1	35	32	33	37
33	37	baris 2				
		baris 3				
		baris 4				
		baris 5				
		baris 6				
		baris 7				

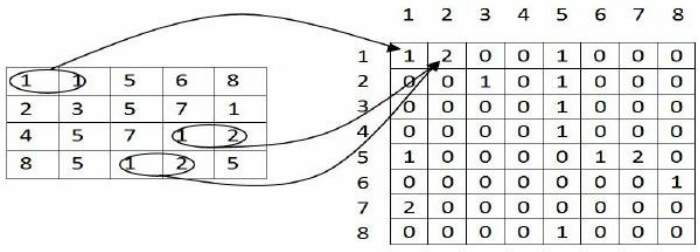
Gambar 2. 7. Pemetaan *Codebook*

2.5. *Markov Stationary Feature* (MSF)

Markov Stationary Feature (MSF) algoritma *extend* dari *VQ*, dimana algoritma ini dapat menangani tiga level histogram yang berbeda, yang dapat meringkankan keterbatasan histogram, adapun cara kerja MSF adalah [6]:

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 matrix*:



Gambar 2. 8. 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(2.20), dimana p_{ij} menunjukan kemungkinan perubahan status c_i menjadi c_j.[13]

$$p_{ij} = \frac{c_{ij}}{\sum_{j=1}^{K} c_{ij}} \quad \text{..... (2.22)}$$

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

$\pi \approx \frac{1}{K}\sum_{i=1}^{K} \vec{\alpha}_{ij}$, dimana $A_n = [\overrightarrow{a_1}, \dots, \overrightarrow{a_n}]^T$ ..... (2.23)

Disini π adalah distribusi *stationary* yang memenuhi π = πP.

$$A_n = \frac{1}{n+1}\ (I + P + \cdots + P^n) \quad \text{..... (2.24)}$$

4. Fitur lengkap yang mencakup kombinasi distribusi awal didefinisikan oleh formula (2.37) dan *stationary distribution* diperoleh dengan mengunakan formula(2.38).

$$\pi(0) = \frac{c_{ii}}{\sum_{i=1}^{K} c_{ii}} \quad \text{..... (2.25)}$$

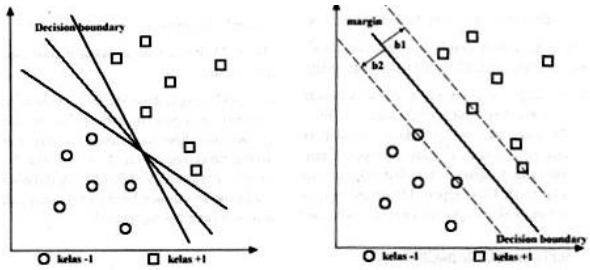
$$\overrightarrow{MSF} = [\pi(0), \pi]^T \quad \text{..... (2.26)}$$

2.6. *Support Vector Machine* (SVM)

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 SVM [14].

2.6.1. Konsep SVM

Ide dasar SVM adalah memaksimalkan batas *hyperplane*, yang diilustrasikan pada Gambar 2. 9. Pada gambar (a) ada sejumlah pilihan *hyperplane* yang mungkin untuk set data, sedangkan gambar (b) merupakan *hyperplane* dengan margin yang paling maksimal. Meskipun sebenarnya pada gambar (a) bisa juga menggunakan *hyperplane* sembarang, tetapi *hyperplane* dengan marign yang maksimal akan memberikan generalisasi yang lebih baik pada metode klasifikasi.

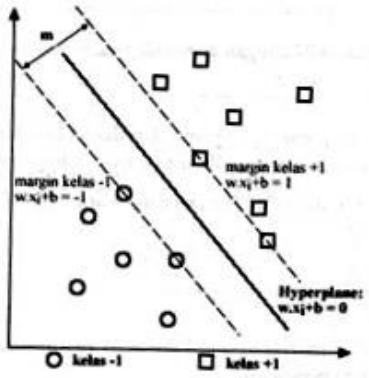


Gambar 2. 9. Batas keputusan yang mungkin untuk set data.

Konsep klasifikasi dengan SVM dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* terbaik yang berfungsi sebagai pemisah dua buah kelas data pada input space. Gambar 2.8. memperlihatkan beberapa data yang merupakan anggota dari dua buah kelas data, yaitu +1 dan -1. Data yang tergabung pada kelas -1 disimbolkan dengan bentuk lingkaran, sedangkan data pada kelas +1, disimbolkan dalam bentuk bujur sangkar.

Hyperplane (batas keputusan) 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. 9 (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. 10. *Margin hyperplane*

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

$$\vec{w} \cdot \vec{x} + b = 0 \quad \text{.....(2.27)}$$

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

$$\vec{w} \cdot \vec{x} + b \leq -1 \quad \text{.....(2.28)}$$

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

$$\vec{w} \cdot \vec{x} + b \geq +1 \quad \text{.....(2.29)}$$

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 (2.31)

$$\min_{\vec{w}} \tau(w) = \frac{1}{2} \|\vec{w}\|^2 \quad \text{.....(2.30)}$$

$y_i(\vec{x_i}.\vec{w}+b)-1 \geq 0, \forall i$ ..... (2.31)

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

$L(\vec{w},b,\alpha) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l}\alpha_i\left(y_i\left((\vec{x}_i.\vec{w}+b)-1\right)\right),\ i = 1,2\ldots l$ (2.32)

α_i adalah *Langrange multipliers*, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan (2.22) 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 (2.51) dapat dimodifikasi sebagai maksimalisasi *problem* yang hanya mengandung α_i, sebagaimana persamaan (2.53) dibawah

Maximize :

$\sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j \vec{x}_i.\vec{x}_j$ ..... (2.33)

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 :

$$\varphi\begin{bmatrix}x\\y\end{bmatrix} = \begin{cases}\sqrt{x_n^2+y_n^2} > 2, maka & \begin{bmatrix}\sqrt{x_n^2+y_n^2}-x & |x-y|\\ \sqrt{x_n^2+y_n^2}-x & |x-y|\end{bmatrix}\\ \sqrt{x_n^2+y_n^2} < 2, maka & \begin{bmatrix}x\\y\end{bmatrix}\end{cases} \quad \text{..... (2.34)}$$

Kemudian untuk mencari nilai α_i didapatkan dari persamaan sebagai berikut :

$\sum_{i=1,j=1}^{n} a_i T_i^T T_j$ ..... (2.35)

Dan selanjutnya menggunakan persamaan sebagai berikut :

$W = \sum_{i=1}^{n} a_i T_i \ dan \ \sum_{i=1,j=1}^{n} a_i T_i = y_i$ ..... (2.36)

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

$y = wx + b\ ,\ w = \sum_{i}^{n} a_i s_i$ ..... (2.37)

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

2.6.2. Kernel Trick

Feature space dalam prakteknya biasanya memiliki 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*

   $$K(x_i, x) = x_i^T x \quad \text{.....(2.38)}$$

2. *Polynomial kernel*

   $$K(x_i, x) = (\gamma . x_i^T x + r)^p, \gamma > 0 \quad \text{.....(2.39)}$$

   Radial basis function (RBF)

   $$K(x_i, x) = \exp(-\gamma |x_i - x|^2), \gamma > 0 \quad \text{.....(2.40)}$$

   Sigmoid kernel

   $$K(x_i, x) = \tanh(\gamma x_i^T x + r) \quad \text{.....(2.41)}$$

2.6.3. *Multiclass* SVM

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 *multiclass* SVM 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*, Dengan menggunakan metode ini, akan dibangun k buah model SVM biner (k adalah jumlah kelas). Contohnya, terdapat permasalahan klasifikasi dengan 4

buah kelas. Untuk pelatihan digunakan 4 buah SVM biner seperti pada Tabel 2. 2. dan penggunanya dalam mengklasifikasi kelas pada data baru dapat dilihat pada persamaan sebagai berikut [14]:

$$Kelas\ x = \arg\max_{i=1..k}\left(\left(w^{(i)}\right)^{T}.\varphi(x) + b^{(i)}\right) \quad \text{..... (2.42)}$$

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

Tabel 2. 2. Contoh 4 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	$f^4(x) = (w^4)x + b^4$

2.7. NetBeans IDE

Bagi pengembang aplikasi berbasis Java mungkin sudah familiar dengan menggunakan NetBeans IDE. Biasanya NetBeans IDE ini digunakan oleh pengembang aplikasi untuk melakukan pemrograman, kompilasi, mencari kesalahan, dan menjalankan aplikasi yang telah dibuat. NetBeans IDE sendiri dibuat dengan menggunakan bahasa pemrograman Java, namun untuk membuat aplikasi yang dapat digunakan dalam sebuah perangkat komputer, maupun *mobile*. Selain itu, pembaca dapat memasang *plugin*, maupun modul yang bisa didapatkan di komunitas untuk mendukung bahasa lain agar dapat dijalankan di NetBeans IDE ini.

NetBeans IDE terbaru telah memasuki versi 8.2, dan telah mendukung Java 8. Selain itu, IDE ini bisa berjalan di sistem operasi Windows, Mac OS, dan Linux 32/64 bit. [15].

2.8. *Open Source Computer Vision Library* (OpenCV)

Open Source Computer Vision Library (OpenCV) adalah sebuah *library* perangkat lunak yang ditujukan untuk pengolahan citra yang biasa digunakan secara *real-time* (pada waktu itu juga). *Library* ini dibuat oleh Intel dan merupakan *library* yang bebas digunakan dan berada dalam naungan sumber terbuka (*Open source*) dari lisensi. *Library* ini juga bisa digunakan diberbagai *platform* dan didedikasikan sebagian besar untuk pengolahan citra secara *real-time*. Umumnya *library* ini menggunakan bahasa pemrograman C/C++, tetapi akhir – akhir ini sudah dikembangkan keberbagai bahasa pemrograman seperti Phyton, Javascript dan Java [16]. Adapun pustaka yang digunakan pada penelitian ini adalah:

1. *CascadeClassifier*

   Merupakan pustaka untuk untuk menggunakan algoritma haar cascade.
2. *Mat*

   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*

   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 mesin belajar SVM.

2.9. *Unified Modelling Language* (UML)

Diagram UML adalah sekumpulan alat yang digunakan untuk melakukan abstraksi terhadap sebuah sistem atau perangkat lunak berbasis objek. Dalam UML sendiri terdapat beberapa diagram yang wajib dikuasai yaitu [17]:

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.
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.

2.10. Vector Quantization Library

Vector Quantization Library merupakan pustaka pada bahasa pemograman java yang dibangun oleh Sherif Abdul Navi untuk menggunakan *Vector Quantization*[18], 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 indeks.
2. `Decompress(VectorQuantization.getCompressedPath(path));`
   Kegunaan dari fungsi ini adalah mengkonversi *codebook* yang telah di-*update* dan nomor indeks menjadi bentuk gambar.