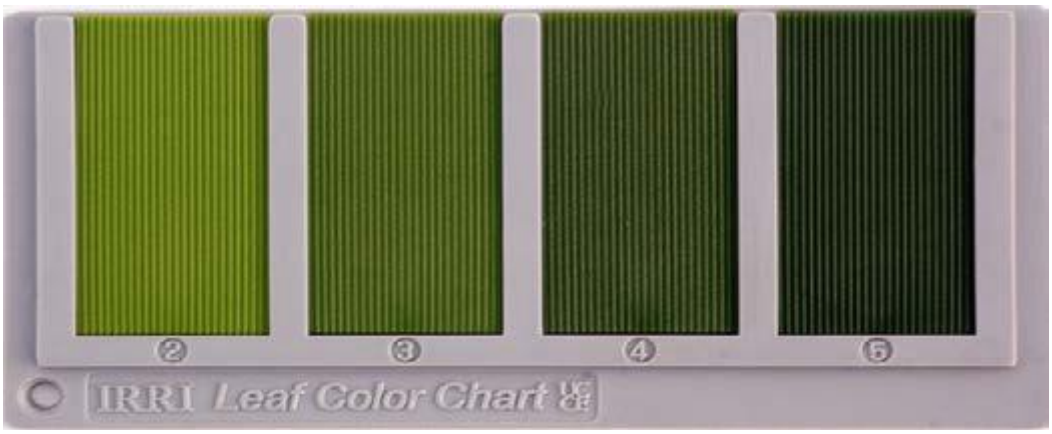
BAB 2
LANDASAN TEORI

2.1 Bagan Warna Daun (BWD)

BWD adalah salah satu alat yang digunakan untuk mengevaluasi keadaan tanaman padi, seperti kadar pupuk nitrogen yang dibutuhkan oleh tanaman padi. BWD dikembangkan oleh International *Rice Research Institute*(IRRI) dengan tujuan memantau pertumbuhan tanaman padi. BWD awalnya dikembangkan di jepang untuk membantu petani menentukan intensitas warna daun yang berhubungan langsung dengan kandungan klorofil dan status nitrogen dalam daun [5]. Cara penggunaan BWD adalah dengan mencocokan warna daun padi dengan skala warna yang ada pada BWD pada **Error! Reference source not found.**. nilai skala warna daun dimulai dari 2 sampai 5.



Gambar 2.1 Bagan Warna Daun (BWD)

Sampel bagan warna daun yang akan digunakan minimal 10 daun, dan semua hasil nilai pencocokan warna lalu dirata -ratakan. Selanjutnya nilai rata rata tersebut cocokan dengan tabel rekomendasi pupuk pada **Error! Reference source not found.**

Nilai warna daun dengan BWD	Hasil (t/ha GKG)			
	~ 5,0	~ 6,0	~ 7,0	~ 8,0
	Takaran pupuk urea yang digunakan (kg/ha), berdasarkan waktu yang telah ditetapkan			
2 – 3	7 5	100	125	150
Antara 3 dan 4	5 0	7 5	100	125
4 – 5	0	0–50	5 0	5 0
	Takaran pupuk urea yang digunakan (kg/ha), berdasarkan kebutuhan riil tanaman			
Di bawah 4	5 0	7 5	100	125

Gambar 2.2 Rekomendasi pemberian pupuk nitrogen (N).

2.2 Pengolahan Citra Digital

Secara umum, pengolahan citra digital menunjuk pada pemrosesan gambar 2 dimensi menggunakan komputer. Dalam konteks yang lebih luas, pengolahan citra digital mengacu pada pemrosesan setiap data 2 dimensi. Citra digital merupakan sebuah larik (array) yang berisi nilai-nilai real maupun komplek yang direpresentasikan dengan deretan bit tertentu. Suatu citra dapat didefiisikan sebagai fungsi f(x,y) berkuran M baris dan N kolom, dengan x dan y adalah koordinat spasial, dan *amplitude* f di titik koordinat (x,y) dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x, y, dan nilai *amplitude* f secara keseluruhan berhingga (*finite*) dan bernilai diskrit maka dapat dikatan bahwa citra tersebut adalah citra digital. [7]

Ada tiga jenis citra yang umum digunakan dalam pemrosesan citra. Ketiga jenis citra tersebut yaitu citra berwarna, citra berskala keabuan, dan citra biner. [8]

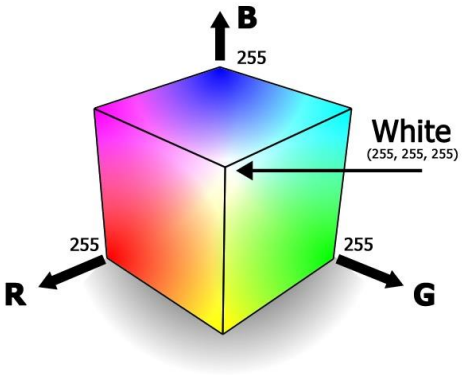
2.2.1 Citra Berwarna

Citra berwarna pada umumnya memiliki beberapa model warna diantaranya model warna RGB dan HSV :

2.2.1.1 Model Warna RGB

Citra berwarna, pada umumnya menggunakan model warna RGB yang terdiri dari 3 komponen warna primer, yakni R (merah), G (hijau), dan B (biru). Setiap komponen warna menggunakan delapan bit. (nilainya berkisar antara 0

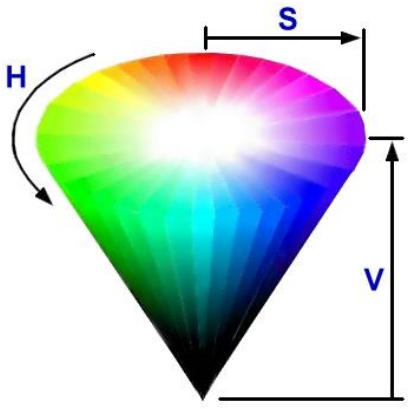
sampai dengan 255). Dengan demikian, kemungkinan warna yang dapat disajikan mencapai 255 x 255 x 255 atau 16.581.375 warna.



Gambar 2.3 Model Warna RGB

2.2.1.2 Model Warna HSV

HSV adalah model warna yang lebih baik untuk digunakan untuk berbagai keperluan pengolahan citra dan *computer vision*. Misalnya saja pada object tracking berdasarkan warna, segmentasi citra dsb. *Hue* (H) adalah ukuran dari jenis warna seperti warna merah, kuning, hijau dan seterusnya. Representasinya dalam bentuk deraja dengan nilai 0 – 360. *Saturation* (S) adalah keberwarnaan suatu warna. Semakin berwarna sebuah warna berarti semakin besar nilai saturasinya. Namun apabila suatu warna pucat, itu berarti saturasinya rendah. *Value* (V) adalah nilai kecerahan sebuah warna. Warna cerah memiliki nilai *Value* tinggi dan sebaliknya untuk warna yang gelap [9]



Gambar 2.4 Model Warna HSV

Berikut ini merupakan perhitungan untuk konversi dari citra RGB menjadi citra HSV

$$V = \max R, G, B$$

$$V_m = \min R, G, B$$

$$S = \begin{cases} 0 \; jika \; V = 0 \\ \dfrac{V_m}{V} \quad jika \; V > 0 \end{cases}$$

$$H = \begin{cases} 0° \;\; jika \; S = 0 \\ 60° \times \left(\dfrac{G - B}{V_m} \; mod 6 \right) \;\; jika \; V = R \\ 60° \times \left(2 + \dfrac{B - R}{V_m} \right) jika \; V = G \\ 60° \times \left(4 + \dfrac{R - G}{V_m} \right) jika \; V = B \end{cases} \qquad (2.1)$$

2.2.2 Citra Berskala Keabuan

Sesuai dengan nama yang melekat, citra jenis ini menangani gradasi warna hitam dan putih, yang tentu saja menghasilkan efek warna abu-abu. Pada jenis gambar ini, warna dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar antara 0 sampai 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih.

2.2.3 Citra Biner

Citra biner adalah citra dengan setiap piksel hanya dinyatakan dengan sebuah nilai dari dua kemungkinan (yaitu nilai 0 dan 1). Nilai 0 menyatakan warna hitam dan nilai 1 menyatakan warna putih. Citra jenis ini banyak dipakai dalam pemrosesan citra, misalnya untuk kepentingan memperoleh tepi bentuk suatu objek.

2.3 Morfologi Citra

Operasi morfologi merupakan operasi yang digunakan pada citra biner (hitam-putih) untuk mengubah struktur bentuk objek yang terkandung dalam citra [12]. Selain itu terdapat operasi morfologi lainnya yaitu seperti *Erode* (Erosi), *Dilate* (Dilasi), *Opening* (Erosi-Dilasi), *Closing*(Dilasi-Erosi).

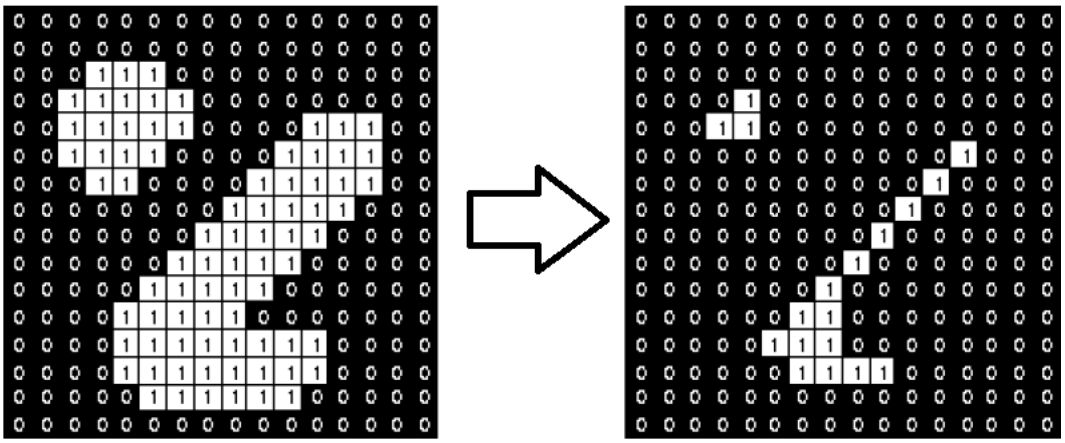
2.3.1 *Erode*

Operasi *erode* (Erosi) mempunyai efek memperkecil struktur citra. Operasi erosi dapat dirumuskan sebagai berikut [8] :

$$A \ominus B = \{ p \in z2 \mid (a + b) \in |, untuk\ setiap\ b \in B \} \quad (2.2)$$

Dimana A merupakan f(x,y) dari citra asli dan B adalah elemen penstruktur atau biasa disebut strel. Elemen penstruktur yang biasa digunakan dalam operasi erosi adalah bentuk kotak. Bentuk elemen penstruktur lainnya ada yang berupa elipse, garis, piringan dan lainnya.

Hasil erosi biasanya merupakan operasi nalar *AND* dari setiap koordinat A dan B. Berikut merupakan hasil dari operasi *erode* terdapat pada **Error! Reference source not found.**



Gambar 2.5 Hasil Operasi *Erode* / Erosi

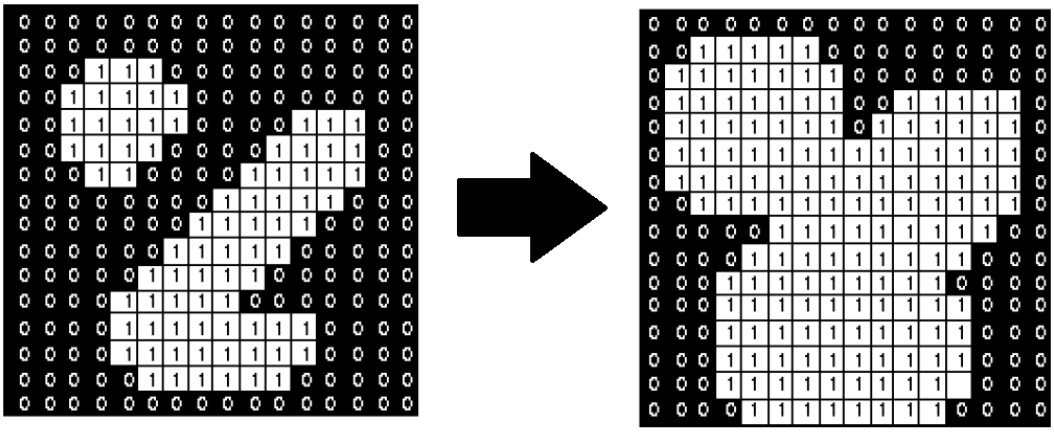
2.3.2 Dilate

Operasi dilate (Dilasi) biasa dipakai untuk mendapatkan efek pelebaran terhadap piksel bernilai 1. Operasi dilasi dapat dirumuskan sebagai berikut [8]:

$$A \oplus B = \{ z \mid z = a + b, dengan\ a\ \in A\ dan\ b\ \in B \} \quad (2.3)$$

Dimana A merupakan *f*(x,y) dari citra asli dan B adalah elemen penstruktur atau biasa disebut *strel*. Elemen penstruktur yang biasa digunakan dalam operasi dilasi juga biasanya adalah berbentuk kotak.

Hasil dilasi berupa penjumlahan seluruh pasangan koordinat dari A dan B. Berikut merupakan hasil dari operasi dilate terdapat pada **Error! Reference source not found.**



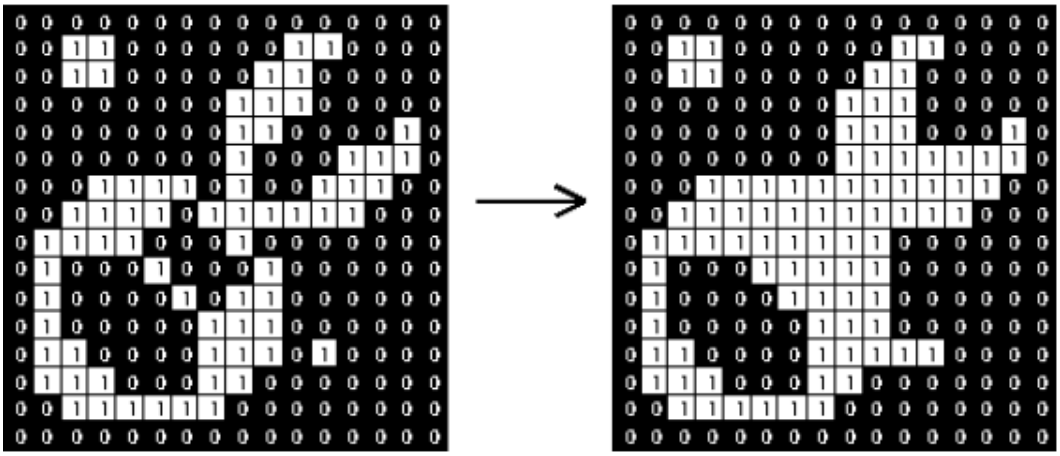
Gambar 2.6 Hasil Operasi Dilate/Dilasi

2.3.3 *Closing* (Erosi-Dilasi)

Operasi *closing* (Dilasi-Erosi) adalah kombinasi antara operasi dilasi dan erosi yang dilakukan secara berurutan. Citra asli didilasi terlebih dahulu, kemudian hasilnya dierosi. Operasi ini digunakan untuk menutup atau menghilangkan lubang-lubang kecil yang ada dalam segmen objek, menggabungkan objek yang berdekatan dan secara umum membuat smooth batas dari objek besar tanpa mengubah objek secara signifikan. Operasi closing dapat dirumuskan sebagai berikut [8]:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.4}$$

Berikut merupakan hasil dari operasi closing terdapat pada **Error! Reference source not found.**



Gambar 2.7 Hasil Operasi Closing

2.4 Segmentasi Warna HSV

Dalam pengolahan citra, terkadang kita menginginkan pengolahan hanya pada obyek tertentu. Oleh sebab itu, perlu dilakukan proses segmentasi citra yang bertujuan untuk memisahkan antara objek (*foreground*) dengan *background*. Pada umumnya keluaran hasil segmentasi citra adalah berupa citra biner di mana objek (foreground) yang dikehendaki berwarna putih (1), sedangkan background yang ingin dihilangkan berwarna hitam (0). Secara garis besar proses segmentasi warna hsv adalah sebagai berikut [10] :

1. Tentukan citra RGB yang menjadi obyek deteksi, nilai warna HSV yang menajdi acuan dan nilai toleransi HSV yang digunakan .
2. Transpose citra RGB ke HSV.
3. Lakukan filter warna pada citra berdasarkan nilai acuan (T) dan nilai toleransi (tol). Dengan x sebagai warna HSV pada pixel yang ada maka warna yang tidak termasuk dalam rentang T-tol < x < T+tol diberi warna hitam.
4. Transpose kembali citra RGB, tampilkan hasil filter.

2.5 Histogram Warna

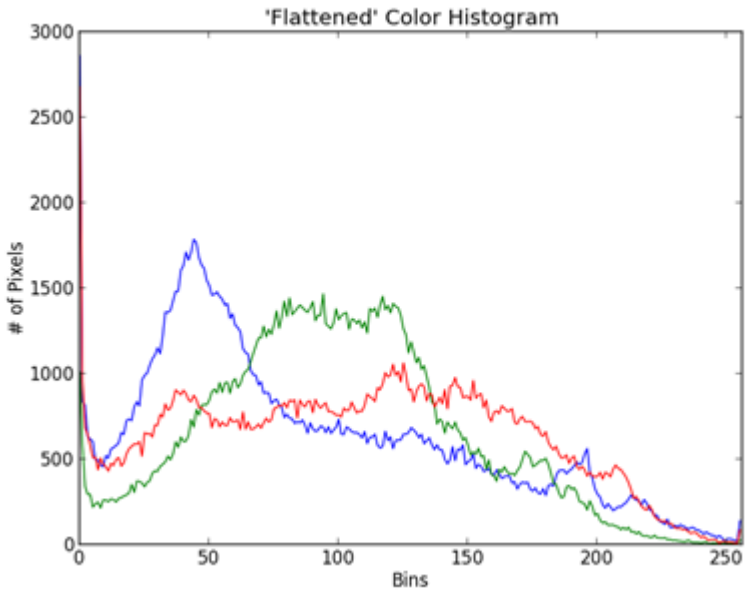
Histogram warna merupakan fitur yang paling banyak digunakan untuk merepresentasikan ciri warna suatu citra. Citra pada umumnya dikonversi ke dalam suatu ruang warna tertentu, kemudian setiap komponen ruang warna dibuat histogramnya. Ruang warna HSV pada umumnya sering digunakan karena ruang warna tersebut dekat dengan persepsi manusia. Namun demikian akan lebih baik pemilihan ruang warna didasari pada objek yang dihadapi [7]. Secara matematis histogram citra dihitung dengan rumus :

$$h_1 = \frac{n_i}{n} \qquad ,i = 0,1 \dots , L-1 \tag{2.5}$$

yang dalam hal ini ,

n_i = jumlah pixel yang memiliki derajat keabuan i

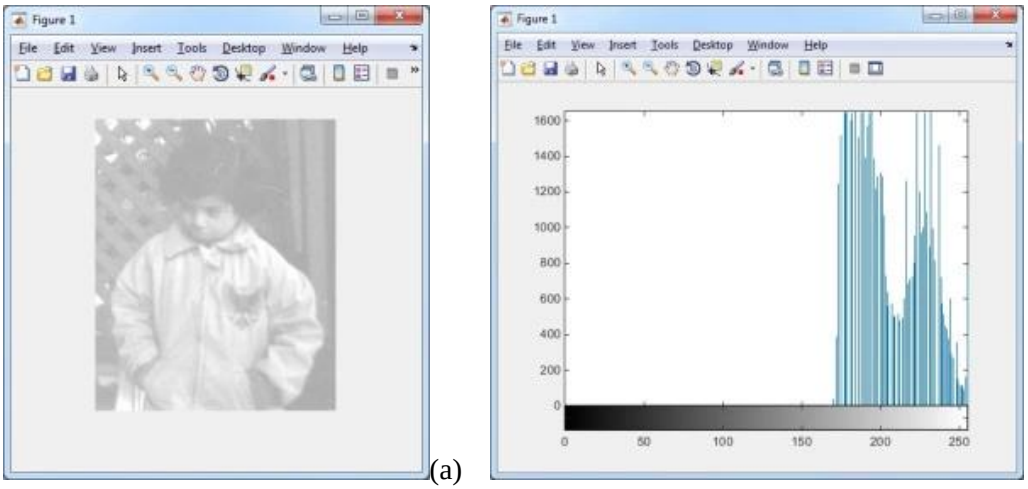
n = jumlah seluruh pixel di dalam citra



Gambar 2.8 Histogram Warna

2.5.1 Intensitas citra keseluruhan

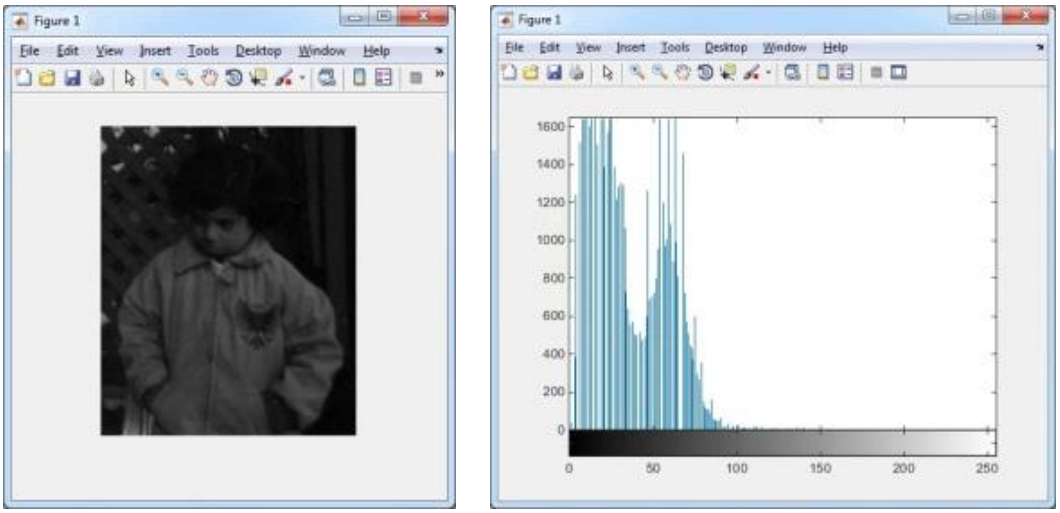
Pada sebuah histogram dapat diketahui intensitas citra secara keseluruhan sehingga bisa disimpulkan citra tersebut gelap atau terang. Jika diamati ilustrasi pada **Error! Reference source not found.** dapat disimpulkan bahwa citra yang bernuansa terang akan memiliki histogram cenderung ke kanan sedangkan pada **Error! Reference source not found.** citra yang bernuansa gelap akan memiliki histogram cenderung ke kiri.



(a)

Citra terang (b) Histogram citra terang

Gambar 2.9 Citra Terang



(b) Citra gelap (b) Histogram citra gelap

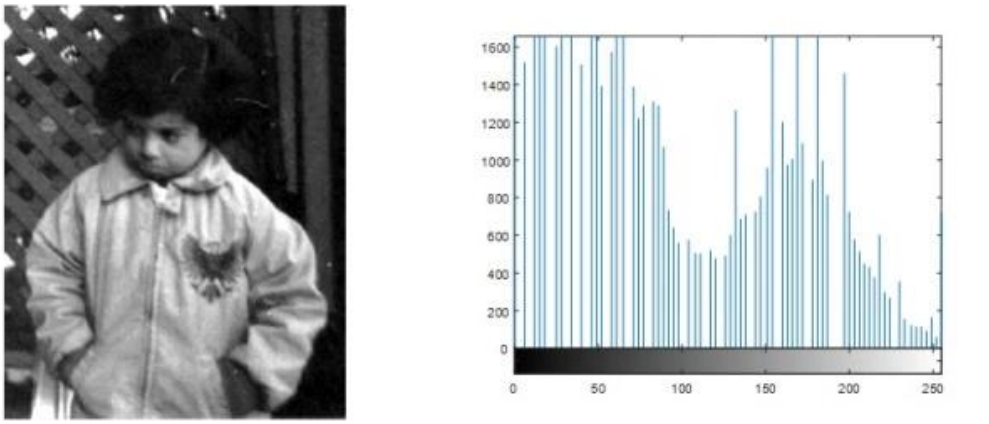
Gambar 2.10 Citra Gelap

Luminosity (terangnya cahaya) dari sebuah citra dapat diukur dengan menghitung nilai rata-rata (*mean*) dari histogram sesuai dengan persamaan . Semakin tinggi nilai rata-rata, maka semakin tinggi kecerahan suatu citra.

$$mean = \sum \mathrm{i} \frac{\mathrm{H(i)}}{\mathrm{N}} \tag{2.6}$$

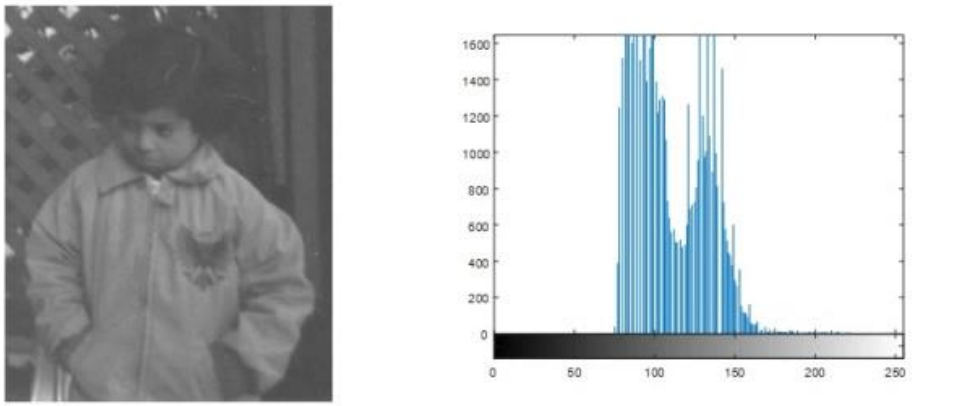
2.5.2 Kontras citra

Sebuah citra yang memiliki kontras baik/ tinggi maka nilai-nilainya terdistribusi dengan merata seperti pada Gambar 2.8 sedangkan citra yang memiliki kontras buruk/rendah maka nilai-nilainya berkumpul pada satu sisi tertentu seperti pada **Error! Reference source not found.**



(a) Citra dengan kontras tinggi (b) Histogram citra dengan kontras tinggi

Gambar 2.11 Citra Kontras Tinggi



(a) Citra dengan kontras rendah (b) Histogram citra dengan kontras rendah

Gambar 2.12 Citra Kontras Rendah

Secara kuantitatif, nilai kontras suatu citra dapat dihitung berdasarkan standar deviasinya dengan menggunakan persamaan (2.8) dan (2.10)

$$\text{var} = \sum (\text{i} - \text{mean})^2 \frac{\text{H(i)}}{\text{N}} \tag{2.7}$$

$$\text{std} = \sqrt{\text{var}} \tag{2.8}$$

2.6 Open CV *(Open Source Computer Vision Library)*

OpenCV *(Open Source Computer Vision Library)* 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.

Secara garis besar OpenCV mempunyai modul/subrutin yang ada pada *library* yang akan dijelaskan sebagai berikut [11]:

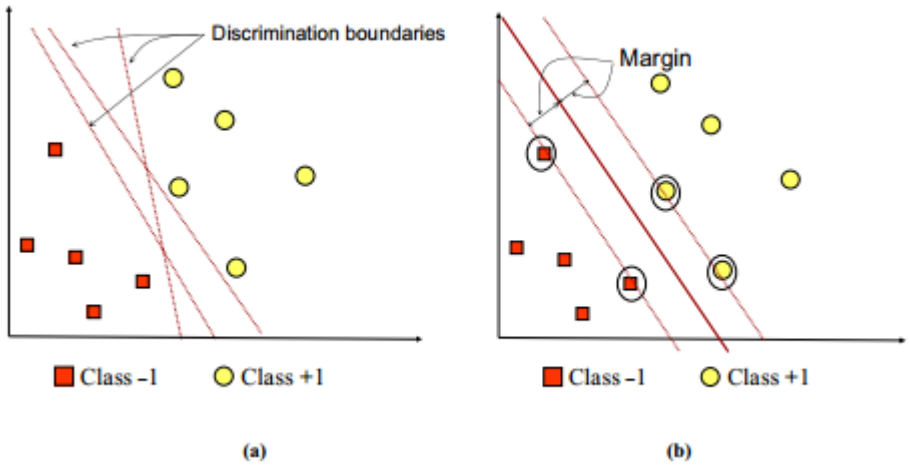
1. *Core* – sebuah modul/subrutin dasar dari struktur data, sudah termasuk array multi dimensi dan matriks.
2. *Imgproc* – sebuah modul/subrutin untuk pemprosesan citra seperti image filtering, geometrical image, image transformation dan color space conversion.
3. *Video* – sebuah modul/subrutin untuk analisis video termasuk motion, background subtraction dan object tracking algorithm.
4. *Calib3d* – sebuah modul/subrutin untuk geometry algorithm, kalibrasi kamera dan elemen untuk membangun gambar 3D (3-Dimensional)..
5. *Features2d* – sebuah modul/subrutin untuk perhitungan konvolusi dan ekstraksi fitur.

6. *Objdetect* – sebuah modul/subrutin untuk deteksi objek dari kelas yang sudah ditentukan
7. *Highgui* – sebuah modul/subrutin untuk menangkap kamera webcam dan image and video codecs.
8. *Ml* – sebuah modul/subrutin tambahan untuk melakukan perhitungan Machine learning seperti K-nearest Neighbors, Support Vector Machine dan Decision tree.

2.7 *Support Vector Machine*

Support Vector Machine (SVM) diperkenalkan oleh Vapnik pada tahun 1992 sebagai suatu teknik klasifikasi yang efisien untuk masalah nonlinier. SVM berbeda dengan teknik klasifikasi di era 1980-an, seperti decision tree dan ANN, yang secara konsep kurang begitu jelas dan seringkali terjebak pada optimum lokal. SVM memiliki konsep yang jauh lebih matang, lebih jelas secara matematis, dibanding teknik – teknik klasifikasi sebelum era 1990-an. SVM berusaha menemukan hyperplane dengan memaksimalkan jarak antar kelas. Dengan cara ini, SVM dapat menjamin kemampuan generalisasi yang tinggi untuk data – data yang akan datang .

Menurut penelitian yang dilakukan oleh Anto Satriyo Nugroho, Arief Budi Witarto, dan Dwi Handoko [12] konsep SVM dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* terbaik yang berfungsi sebagai pemisah dua buah class pada *input space*. Gambar 2.12-a memperlihatkan beberapa *pattern* yang merupakan anggota dari dua buah *class* : +1 dan –1. *Pattern* yang tergabung pada *class* –1 disimbolkan dengan warna merah (kotak), sedangkan pattern pada class +1, disimbolkan dengan warna kuning(lingkaran).



Gambar 2.13 SVM Mencari Hyperplane Terbaik

Masalah klasifikasi dapat diterjemahkan dengan usaha menemukan garis (*hyperplane*) yang memisahkan antara kedua kelompok tersebut. Berbagai alternatif garis pemisah (*discrimination boundaries*) ditunjukkan pada Gambar 2.15 *Hyperplane* pemisah terbaik antara kedua class dapat ditemukan dengan mengukur margin *hyperplane* tsb. dan mencari titik maksimalnya. Margin adalah jarak antara *hyperplane* tersebut dengan *pattern* terdekat dari masing-masing *class*. *Pattern* yang paling dekat ini disebut sebagai *support vector*.

Garis solid pada Gambar 2.15 menunjukkan *hyperplane* yang terbaik, yaitu yang terletak tepat pada tengah-tengah kedua *class*, sedangkan titik merah dan kuning yang berada dalam lingkaran hitam adalah *support vector*. Usaha untuk mencari lokasi *hyperplane* ini merupakan inti dari proses pembelajaran pada SVM. 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 :

$$\vec{w}.\vec{x} + b = 0 \quad (2.9)$$

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

$$\vec{w}.\vec{x} + b \leq -1 \quad (2.10)$$

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

$$\vec{w}.\vec{x} + b \geq -1 \quad (2.11)$$

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 (2.21), dengan memperhatikan *constraint* persamaan (2.11)

$$\min_{\vec{w}} \tau(w) = \frac{1}{2}\|\vec{w}\|^2 \quad (2.12)$$

$$y_i(\vec{x}_i.\vec{w} + b) - 1 \geq 0, \forall i \quad (2.13)$$

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

$$L(\vec{w}, b, \alpha) = \frac{1}{2}\|\vec{w}\|^2 - \Sigma_{i=1}^{l} a_i(y_i((x_i.\vec{w} + b) - 1)), i = 1,2 \ldots l \quad (2.14)$$

a_i adalah *Langrange multipliers*, yang bernilai nol atau positif ($\alpha i \geq 0$). Nilai optimal dari persamaan (2.4) dapat dihitung dengan meminimalkan L terhadap $\vec{w}$

dan b, dan memaksimalkan L terhadap a_i. Dengan memperhatikan sifat bahwa pada titik optimal *gradient* L = 0, persamaan (2.11) dapat dimodifikasi sebagai maksimalisasi problem yang hanya mengandung a_i, sebagaimana persamaan (2.13) dibawah

Maximize :

$$\Sigma_{i=1}^{l} a_i - \frac{1}{2}\Sigma_{i,j=1}^{l} a_i a_j y_i y_j \vec{x}_i . \vec{x}_j \qquad (2.15)$$

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

Untuk mendapatkan nilai a_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 :

$$\varphi\left[\frac{x}{y}\right] = \begin{cases} \dfrac{\sqrt{x_n^2 + y_n^2} > 2, \text{maka } \left[\frac{4 - x + |x - y|}{4 - x + |x - y|}\right]}{\sqrt{x_n^2 + y_n^2} > 2, \text{maka } \left[\frac{x}{y}\right]} \end{cases} \qquad (2.16)$$

Kemudian untuk mencari nilai a_i didapatkan dari persamaan sebagai berikut :

$$\sum_{i=1,j=1}^{n} a_i T_i^T T_j \qquad (\text{Error}$$

Dan selanjutnya menggunakan persamaan sebagai berikut :

$$\sum_{i=1,j=1}^{n} a_i T_i \text{ dan } \sum_{i=1,j=1}^{n} a_i T_i = yi \qquad (\text{Error}$$

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

$$y = wx + b, \qquad w = \sum_{i}^{n} a_i s_i \qquad \text{(Error}$$

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

2.7.1 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 [13]:

Kernel linier

$$K(x_i, x) = x_i^T x \qquad \text{(Error}$$

Polynomial kernel

$$K(x_i, x) = (y.x_i^T x + r)^p, y > 0 \qquad \text{(Error}$$

Radial basis function (RBF)

$$K(x_i, x) = \exp(-y\,|x_i - x|^2), y > 0 \qquad \text{(Error}$$

Sigmoid kernel

$$\ldots K(x_i, x) = \tanh(yx_i^T x + r) \qquad \text{(Error}$$

2.7.2 Multiclass Support Vector Machine

SVM saat pertama kali diperkenalkan oleh Vapnik, 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 umum digunakan untuk mengimplementasikan multiclass SVM dengan pendekatan sebagai berikut [13]:

1. 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 2.1 dan penggunanya dalam mengklasifikasi kelas pada data baru dapat dilihat pada persamaan sebagai berikut [13]:

$$Kelas\ x = (x_i, x) = \arg\max_{i=1..k}((w^{(i)})^T.\varphi(x) + b^{(i)}) \quad \text{(Error}$$

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

Tabel 2.1 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	$f^1(x) = (w^2)x + b^2$
Kelas 3	Bukan kelas 3	$f^1(x) = (w^3)x + b^3$

Kelas 4	Bukan kelas 4	$\boldsymbol{f^1(x) = (w^4)x + b^4}$

Selanjutnya untuk menentukan hyperplane persamaan Lagrange Multiplier dapat dijabarkan menjadi :

$$Lp = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} a_i y_i (w.x_i) - b \sum_{i=1}^{N} a_i y_i + \sum_{i=1}^{N} a_i \qquad (2.17)$$