BAB 2
TINJAUAN PUSTAKA

2.1 Grafologi

Grafologi berasal dari Bahasa Yunani, *graph* yang berarti tulisan atau menulis, dan *logos* yang berarti ilmu. Ilmu grafologi telah dikenal sejak berabad-abad yang lalu misalnya di Tiongkok, pengetahuan tulisan tangan sudah digunakan sejak tahun 1000 Masehi, meskipun tidak secara ilmiah. Baru pada tahun 1622, seorang dokter dari Italia mengemukakan penemuan tentang ilmu penulisan tangan yang dibuat secara sistematika dan dimuat ke dalam buku. Dalam buku tersebut, dijelaskan tentang adanya hubungan yang unik antara tulisan tangan seseorang dengan karakter dan kepribadiannya.

Grafologi merupakan cabang dari ilmu psikologi dalam mata kuliah *psikografik* atau *psikodiagnostik*. Grafologi adalah seni dan ilmu yang mempelajari tentang tulisan tangan. Karena tulisan tangan berasal dari otak manusia, maka apa yang dituangkan dalam tulisan itu adalah buah pikirannya. Dari buah pikiran tersebut dapat memberi gambaran atau mencerminkan kepribadian manusia. Tanda tangan juga merupakan unsur penting dalam grafologi. Dari tanda tangan dapat melambangkan nilai dan kepribadian seseorang. Kebanyakan ahli dan pakar grafologi menilai kedua-duanya, baik tulisan maupun tanda tangan untuk mendapatkan gambar atau membuat ramalan mengenai diri dan kehidupan seseorang.

2.1.1 Manfaat Grafologi

Grafologi tidak saja bermanfaat untuk mengetahui karakter seseorang untuk kepentingan pribadi, tetapi ilmu ini banyak digunakan dalam memilih dan menilai calon karyawan, seperti yang banyak dilakukan di Amerika Serikat dan Eropa. Selain itu ada beberapa contoh dari manfaat grafologi sebagai berikut: [12]

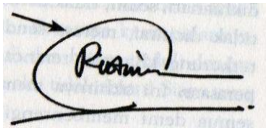
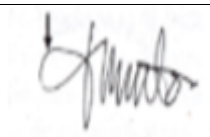
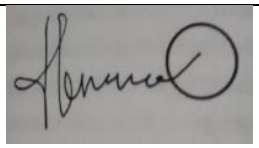
1. Mengetahui *lifetrap*.
2. Mengetahui karakter orang lain.
3. Membantu kemampuan kita dalam berkomunikasi.

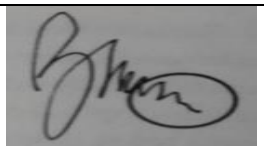
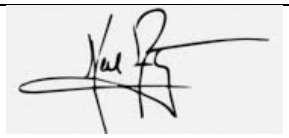
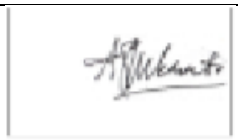
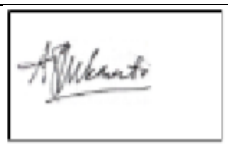
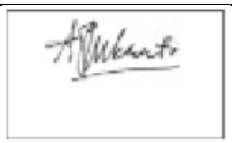
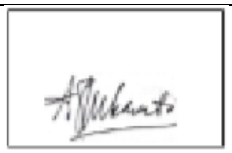
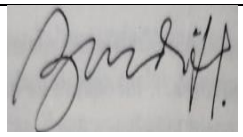
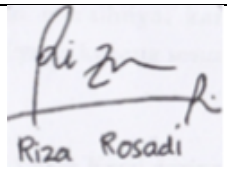
4. Identifikasi.
5. Sarana intropeksi diri.
6. Meningkatkan keahlian.
7. Mengungkap kasus kejahatan.

2.1.2 Dasar Penentuan Analisis Tanda Tangan

Terdapat 9 fitur tanda tangan yang menjadi dasar penetuan analisis, dari sembilan fitur tersebut terdapat ciri berserta kepribadiannya. Dan pada penelitian ini hanya menggunakan 4 fitur yaitu awal kurva, coretan akhir, coretan ditengah, garis bawah itu dikarenakan menurut grafologi hanya ke empat fitur tersebut yang dapat dilakukan komputasi oleh komputer. Tabel 2.1 akan memperlihatkan pola tanda tangan berserta ciri dan kepribadian yang menyertainya.

Tabel 0.1 Fitur-fitur tanda tangan

No	Fitur	Gambar	Ciri	Kepribadian
1	Cakang		Lengkung Tertutup	Ketakutan berlebihan, introvert, tidak memperdulikan sekitar, tidak suka bergaul dan bekerja sama.
2	Awal Kurva		Lengkung Mundur	Nyaman akan masa lalu.
			Lengkung Tajam	Mampu memformulasi pikiran secara tajam.
			Lengkung Lembut	Hati-hati, ramah, diplomasi.
3	Coretan Akhir		Menaik	Terbuka, pandangan ke depan, keinginan maju, percaya diri.

			Menurun	Kurang semangat, berfikir realistis, kurang percaya diri, mudah putus asa.
4	Garis Ditengah		Terdapat Garis Ditengah	Kurang percaya diri dan mudah depresi.
5	Garis Bawah		Adanya garis bawah	Membutuhkan dukungan membuat keputusan, serta memiliki keandalan dalam memimpin.
6	Margin Ekstrim		Cenderung ke kanan	Ceroboh, kurang perhatian.
			Cenderung ke kiri	Takut gagal, takut pada orang lain, kurang percaya diri, pesimis.
			Cenderung di atas	Respek pada diri sendiri, mencerminkan pribadi Bahagia.
			Cenderung ke bawah	Depresi, pemalu, merasa asing.
7	Struktur Titik		Terdapat titik	Pendirian stabil, memiliki rasa curiga, selalu menjaga jarak tidak mudah percaya.
8	Tanda Tangan Terpisah		Terdapat tanda tangan terpisah	Memiliki pengalaman kurang menyenangkan dimasa lalu.
9	Garis Terpisah		Terdapat garis terpisah	Membatasi keinginannya, tidak berani mengambil resiko, sering patah semangat dan ragu

				mengambil keputusan.

2.2 Tanda Tangan

Tanda tangan adalah suatu tulisan tangan yang kadang-kadang diberi gaya tulisan tertentu dari nama seseorang atau identifikasi lainnya yang ditulis pada dokumen sebagai sebuah bukti dari identitas dan kemauan. Menurut Kamus Besar Indonesia (KBBI) tanda tangan sebagai lambing nama yang dituliskan dengan tangan oleh orang itu sendiri sebagai penanda pribadi [13]. Tanda tangan merupakan bentuk yang paling banyak digunakan untuk identifikasi seseorang. Contoh-contoh tanda tangan setiap orang umumnya identik namun tidak sama, itu artinya tanda tangan seseorang dapat berubah-ubah setiap waktu. Perubahan ini menyangkut posisi, ukuran maupun faktor tekanan tanda tangan. Pada perubahan-perubahan tersebut dapat dipengaruhi oleh waktu, umur, kebiasaan dan keadaan mental tertentu [12].

2.3 Pengolahan Citra Digital

Pengolahan citra digital (*Digital Image Processing*) adalah sebuah disiplin ilmu yang mempelajari tentang teknik-teknik mengolah citra. Citra yang dimaksud disini adalah gambar diam (foto) maupun gambar bergerak (yang berasal dari webcam) [14]. Sedangkan digital disini mempunyai maksud bahwa pengolah citra atau gambar dilakukan secara digital menggunakan computer [14].

2.3.1 Jenis Citra

Jenis citra dibagi menjadi tiga jenis yang umum digunakan untuk pemrosesan pada citra, diantaranya yaitu citra berwana (RGB), citra berskala keabuan (*Grayscale*), dan citra biner.

1. Citra Berwarna (RGB)

   Citra berwarna atau biasa dianamakan RGB, merupakan jenis citra yang menyajikan warna dalam bentuk komponen R (merah), G (hijau), B (biru). Setiap piksel dibentuk oleh ketiga komponen tersebut dan setiap komponen warna menggunakan 8 bit, nilainya berkisar antara 0-255. Dengan demikian

warna yang dapat disajikan mencapai 16.581.375 warna. Berikut ini contoh warna RGB yang ditunjukan pada Tabel 2.2 [15].

Tabel 0.2 Warna dan Nilai Penyusun Warna

Warna	R	G	B
Merah	255	0	0
Hijau	0	255	255
Biru	0	0	255
Hitam	0	0	0
Putih	255	255	255
Kuning	0	255	255

2. Citra Berskala Keabuan (*Grayscale*)

   Citra berskalah keabuan atau citra *grayscale* merupakan citra digital yang hanya memiliki satu nilai kanal pada pixel-nya, dengan kata lain nilai bagian Red = Green = Blue. Nilai tersebut digunakan untuk menunjukan tingkat intensitas. Warna yang dimiliki adalah warna dari hitam, keabuan, dan putih. Tingkatan keabuan disini merupakan warna abu dengan berbagai tingkatan dari hitam hingga mendekati putih [16]. Dalam hal ini, intensitas bekisar antara 0 – 255, dimana 0 menyatakan hitam dan nilai 255 menyatakan putih.

3. Citra Biner

   Citra biner merupakan citra digital yang hanya memiliki dua kemungkinan nilai *pixel* yaitu hitam dan putih. Citra biner juga disebut sebagai citra B&W (*black and white*) atau citra monokrom. Hanya dibutuhkan satu bit untuk mewakili nilai setiap *pixel* dari citra biner. Citra biner sering kali muncul sebagai hasil dari proses pengolahan seperti segmentasi, pengambangan, morfologi ataupun *dithering* [16].

2.3.2 *Preprocessing* Citra

Preprocessing citra merupakan penjelasan tahap awal dalam alur kerja untuk menyiapkan citra sebagai data input yang siap diolah lebih lanjut oleh sistem.

Berikut ini tahapan pra-proses citra yang dilakukan pada penelitian ini sebagai berikut:

2.3.2.1 Grayscale

Citra *grayscale* adalah suatu citra yang hanya memiliki warna tingkat keabuan. Warna abu-abu pada citra *grayscale* adalah warna R (*Red*), G (*Green*), B (*Blue*) yang memiliki intensitas yang sama. Sehingga dalam *grayscale image* hanya membutuhkan nilai intensitas tunggal dibandingkan dengan citra berwarna membutuhkan 3 intensitas untuk tiap pikselnya. Untuk melakukan perubahan citra berwarna menjadi citra keabuan dapat menggunakan rumus [17]:

$$I_{[i.j]} = 0.299\ x\ R + 0.587\ x\ G + 0.114\ x\ B \qquad (0.1)$$

Dimana i dan j merupakan titik koordinatnya, dan (R) adalah nilai pada warna merah, (G) adalah nilai pada warna hijau, (B) adalah nilai pada warna biru.

2.3.2.2 Deteksi Tepi Canny

Pendeteksian tepi merupakan langkah pertama untuk melingkupi informasi di dalam citra. Tepi mecirikan batas-batas objek dan karena itu tepi berguna untuk proses segmentasi dan identifikasi di dalam citra. Tujuan pendeteksian tepi adalah untuk meningkatkan permukaan garis batas suatu daerah atau objek di dalam citra [18].

Metode Canny merupakan salah satu algoritma deteksi tepi. Deteksi tepi Canny ditemukan oleh John F. Canny pada tahun 1986. Algoritma ini memberikan tingkat kesalahan yang rendah, melokalisasi titik-titik tepi (jarak piksel-piksel tepi yang ditemukan deteksi dan tepi yang sesungguhnya sangat pendek), dan hanya memberikan satu tanggapan untuk satu tepi.

Ada beberapa kriteria pendeteksi tepian paling optimum yang dapat dipenuhi oleh algoritma canny [18]:

a. Mendeteksi dengan baik (kriteria deteksi)

   Kemampuan semua tepi yang ada sesuai dengan pemilihan parameter-parameter konvolusi yang dilakukan. Sekaligus juga memberikan

fleksibilitas yang sangat tinggi dalam hal menentukan tingkat deteksi ketebalan tepi sesuai yang diinginkan.

b. Melokalisasi dengan baik (kriteria lokalisasi)

Dengan Canny dimungkinkan dihasilkan jarak yang minimum antara tepi yang dideteksi dengan tepi yang asli.

c. Respon yang jelas (kriteria respon)

Hanya ada satu respon untuk tiap tepi. Sehingga mudah dideteksi dan tidak menimbulkan kerancuan pada pengolahan citra selanjutnya.

Berikut ini merupakan langkah-langkah dalam melakukan deteksi tepi canny:

1. *Image Smoothing*

Image Smoothing merupakan proses untuk mengaburkan atau memblurkan gambar untuk menghilangkan derau. Hal ini dimaksudkan untuk mendapatkan tepian citra yang sebenarnya. Bila tidak dilakukan maka garis-garis halus juga akan dideteksi sebagian tepian. Proses ini dapat dilakukan dengan menggunakan filter gausian dapat dilihat pada persamaan 2.2 sebagai berikut:

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \quad (0.2)$$

2. *Finding Gradient*

Fiding Gradient merupakan sebuah proses untuk mendapatkan kekuatan tepi, Tepian harus ditandai pada gambar yang memiliki gradien yang besar. Operator yang digunakan dalam menentukan gradien dapat menggunakan dengan operator sobel. Kelebihan dari metode sobel ini adalah mampu mengurangi *noise* sebelum melakukan perhitung deteksi tepi sehingga tepi-tepi yang dihasilkan lebih banyak. Pada persamaan 2.3 digambarkan bahwa operator sobel terdiri dari kernel 3x3 konvolusi.

-1	0	1
-2	0	2
-1	0	1

Mx

1	2	1
0	0	0
-1	-2	-1

My

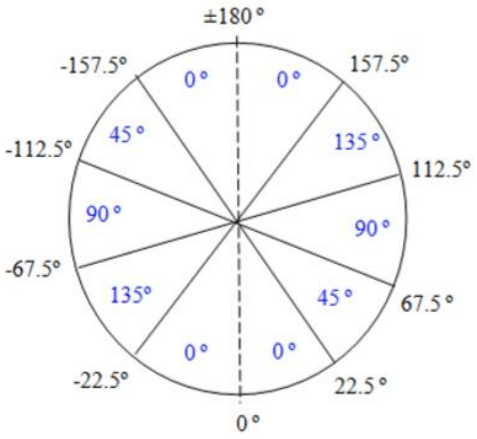
(0.3)

Hasil dari *Gausian Filter* akan dikonvolusi kembali dengan filter operator sobel Mx sehingga akan menghasilkan nilai Gx, sedangkan untuk mengetahui nilai Gy dengan cara mengkonvolusikan kembali hasil dari *gaussian filter* dengan filter operator sobel My. Setalah itu untuk mendapatkan hasil *Magnitudo* gradien atau kekuatan tepi dapat menentukan jarak *Euclidean* yang diukur dengan menerapkan hukum *Pythagoras* seperti yang ditunjukan dalam persamaan 2.4

$$G = \sqrt{G_x^2 + G_y^2} \tag{0.4}$$

Setelah itu menentukan sudut gradien dengan cara membagi kedalam 4 arah, untuk menentukan arah tepian dapat menggunakan persamaan 2.5, sebagai berikut:

$$\theta = tan^{-1}\left(\frac{|G_x|}{|G_y|}\right) \tag{0.5}$$



Gambar 0.1 Pembagian Sudut Gradien Penentu Arah

Aturan konversi yang berlaku sebagai berikut [18]:

a. Jika theta > 0 dan theta < 255 atau theta > 157.5 dan theta < -157.5, maka **0**

b. Jika theta > 22.5 dan theta < 67.5 atau theta > -157.5 dan theta < -112.5, maka **45**
c. Jika theta > 67.5 dan theta < 112.5 atau theta > -112 dan theta < -67.5, maka **90**
d. Jika theta > 112.5 dan theta < 157.5 atau theta > -67.5 dan theta < -22.5, maka **135**

3. *Non-Maximum Supression*

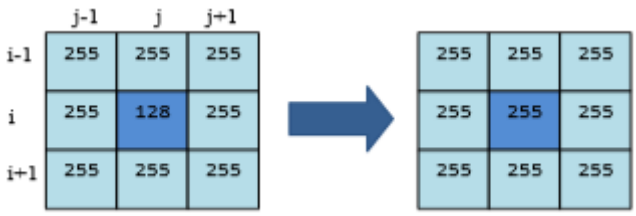
Non-Maximum Supression merupakan proses memperkecil garis tepi yang muncul sehingga menghasilkan garis tepian yang lebih ramping [18]. Penghilangan *non-maximum* dilakukan sepanjang tepi pada arah tepi dan menghilangkan piksel-piksel (piksel diatur menjadi 0) yang tidak dianggap sebagai tepi.

4. *Double Thresholding*

Double Thresholding bertujuan untuk klasifikasi dua buah nilai *High-threshold* (T_2) (ambang atas) dan *Low-threshold* (T_1) (ambang bawah), dengan $(T_2) \approx 2T_1$. Apabila nilai *pixel* lebih besar atau sama dengan T_2 maka diatur nilai 255, apabila nilai *pixel* kurang dari atau sama dengan T_1 maka diatur menjadi 0. *Pixel* diantara T_1 dan T_2 disebut kandidat *pixel* tepi maka sementara diberi nilai 128 [19].

5. *Edge Tracking by Hysteresis Thresholding*

Edge Tracking bertujuan memperoleh tepian final dengan menekan semua sisi yang tidak terhubung pada tepian yang sangat kuat. Nilai 128 selanjutnya dilakukan pengecekan pada dari 8 arah tetangganya, sehingga *pixel* hanya bernilai 0 atau 255. Perubahan nilai 128 menjadi nilai 255 apabila semua kondisi terpenuhi yakni jika salah satu atau semua *pixel* pada 8 arah tentangga bernilai 255. Proses pengujian dilakukan sampai tidak ada lagi perubahan dari nilai 128 menjadi 255. Selanjutnya, semua *pixel* yang bernilai 128 yang tersisa diubah menjadi 0 [19].



Gambar 0.2 Pengujian mengubah nilai 128 menjadi 255

2.3.2.3 Segmentasi Objek

Segmentasi adalah proses pemisahan objek yang satu dengan objek yang lain dalam suatu gambar (citra) menjadi objek-objek berdasarkan karakteristik tertentu. Proses segmentasi berhenti jika objek yang dicari telah ditemukan. Tujuan dari segmentasi ialah untuk menyederhanakan atau mengubah penyajian gambar ke sesuatu yang lebih bermakan dan lebih mudah dalam menganalisisnya. Hasil dari proses segmentasi adalah berupa citra biner dengan objek (*foreground*) yang dikehendaki berwarna putih (1), sedangkan untuk *background* yang ingin dihilangkan berwarna hitam (0). Dari hasil segmentasi tersebut dapat digunakan untuk proses tingkat tinggi lebih lanjut yang dapat dilakukan terhadap suatu citra [20]. Dalam penelitian ini metode *Connected Component Labeling* yang akan digunakan dalam proses segmentasi. Metode ini menerapkan teori *connectivity* piksel dari citra. Seluruh piksel pada sebuah *region* disebut *connected* atau memiliki hubungan bila mematuhi aturan *adjacency* (kedekatan) piksel [21]. Terdapat 2 macam konektivitas yang digunakan pada citra dua dimensi yaitu:

1. 4-*Connected Neighbors*

   Piksel-piksel yang berdekatan dikatakan memiliki hubungan 4 konektivitas jika piksel-piksel tersebut terletak berdampingan secara horizontal dan vertikal N4 (P). Kumpulan dari piksel-piksel ini disebut dengan 4 neighbors of P. Pada konsep ini bila terdapat 2 *pixel* yang saling bersinggungan secara diagonal maka akan dianggap 2 objek.

	P(x, y-1)	
P (x-1, y)	P (x,y)	P(x+1, y)
	P(x, y+1)	

Gambar 0.3 4-*Connected Neighbors*

2. 8-*Connected Neighbors*

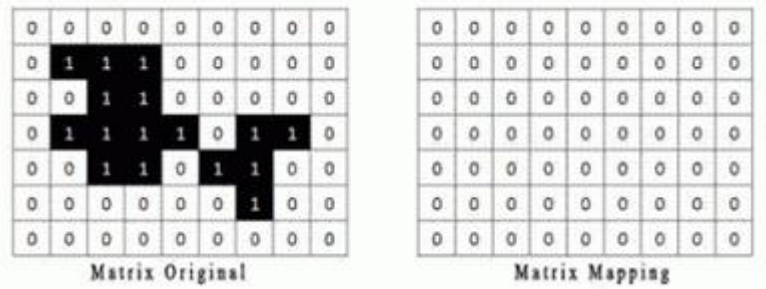
Piksel-piksel yang berdekatan dikatakan memiliki hubungan *8*-konektivitas jika piksel-piksel tersebut terletak berdampingan secara horizontal dan vertikal *N8 (P)* atau disebut juga empat diagonal *neighbors*. Pada ini apabila terdapat 2 pixel yang saling bersinggungan baik secara diagonal ataupun secara horizontal dan vertikal maka akan dianggap 1 objek.

P (x-1, y-1)	P(x, y-1)	P (x+1, y-1)
P (x-1, y)	P (x,y)	P(x+1, y)
P (x-1, y+1)	P(x, y+1)	P (x+1, y+1)

Gambar 0.4 8-*Connected Neighbors*

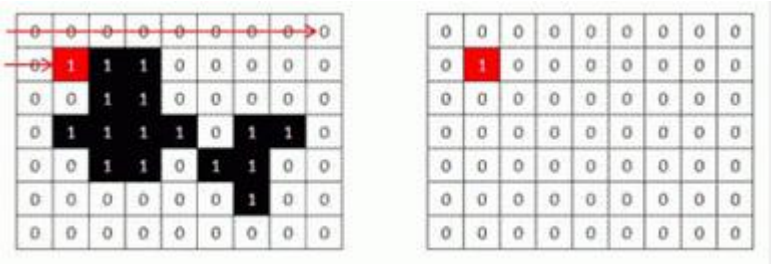
Berikuti ini merupakan langkah-langkah dalam metode *Connected Component Labeling* [22]:

1. Buat dua buah matrix, matrix pertama merepresentasikan objek gambar/citra biner yang akan diolah dan matrix kedua merupakan matrix tempat meletakkan piksel-piksel terpilih yang disebut dengan matrix mapping.



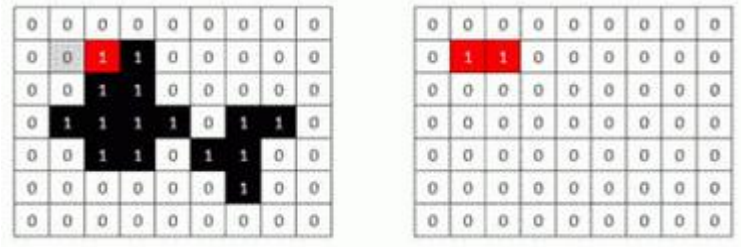
Gambar 0.5 Matriks Original dan Matriks Mapping

2. Lakukan scanning piksel-piksel terhadap citra *foreground* mulai sisi atas *matrix* yaitu dari kiri ke kanan dan dari atas ke bawah. Seperti yang ditunjukan pada Gambar 2.6

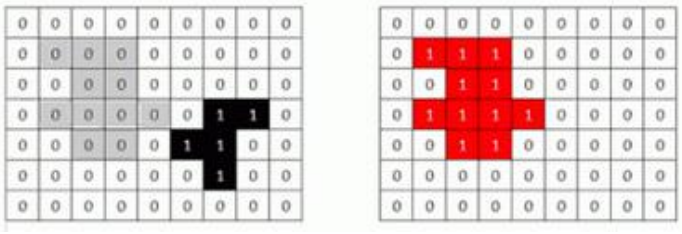


Gambar 0.6 *Scanning Pixel*

3. Lakukan pendekatan *4-connected neighbors* secara berulang-ulang terhadap piksel-piksel yang memiliki kedekatan dan kesamaan nilai intensitas sampai tidak ada lagi kedekatan secara *4-connected neighbors* antara piksel-piksel yang telah dilabeli dengan label satu (1).



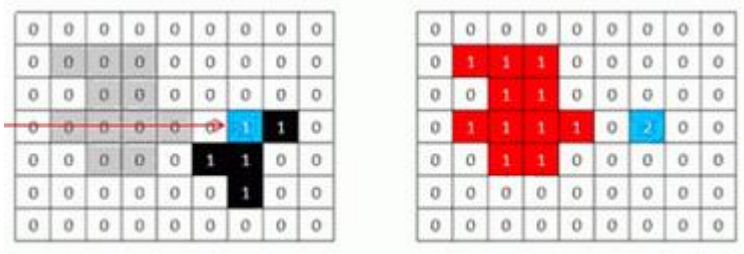
Gambar 0.7 Langkah pertama memindai nilai



Gambar 0.8 Langkah terakhir memindai nilai

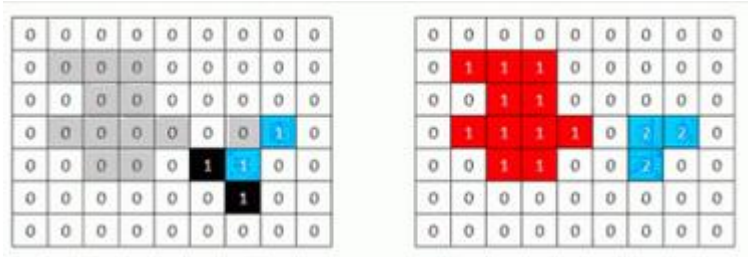
Jika sudah tidak ditemukan lagi piksel-piksel tetangga yang terdapat kedekatan secara *4-connected neighbors*, maka lakukan proses *merging* pada *matrix mapping*.

4. Melakukan scanning lagi terhadap citra *foreground* untuk mendapatkan karakter objek yang lain.

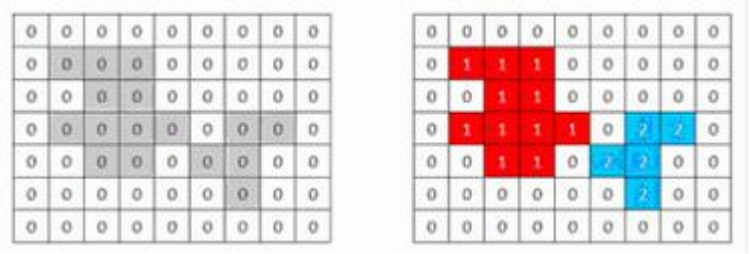


Gambar 0.9 Scanning terhadap citra foreground

5. Lakukan pendekatan *4-connected neighbors* secara berulang-ulang terhadap piksel-piksel yang memiliki kedekatan dan kesamaan nilai intensitas sampai tidak ada lagi kedekatan secara *4-connected neighbors* antara piksel-piksel yang telah dilabeli dengan label (2) dua.



Gambar 0.10 Scanning terhadap citra foreground



Gambar 0.11 Hasil akhir gambar segmentasi

2.3.2.4 Resize

Resize adalah proses untuk memperbesar atau memperkecil ukuran dari sebuah citra sesuai dengan ukuran yang akan dibutuhkan. Proses *resize* bertujuan untuk menyamakan ukuran citra hasil dari segementasi objek agar dapat diproses pada saat melakukan ekstraksi ciri dari hasil citra segmentasi vertikal & horizontal. Berikut ini merupakan rumus yang digunakan untuk meresize pada suatu citra [22]:

$$x = \frac{pb * pp}{pa} \tag{0.6}$$

Keterangan:

x = Nilai piksel baris baru

pb = Ukuran panjang matriks baru

pp = Posisi piksel baris

pa = Ukuran panjang matriks lama

$$y = \frac{lb * lp}{la} \tag{0.7}$$

Keterangan:

y = Nilai piksel kolom baru

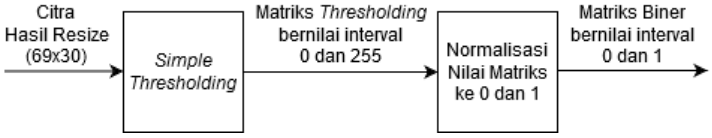
lb = Ukuran lebar matriks baru

lp = Posisi piksel kolom

la = ukuran lebar matriks lama

2.3.2.5 Binerisasi Citra

Binerisasi citra merupakan proses menyederhanakan nilai matriks menjadi 0 dan 1, hal ini dilakukan dengan menggunakan metode *simple thresholding* serta setelah mendapatkan nilai hasil *thresholding* kemudian membagi semua piksel dikoordinat (x, y) dengan 1/255 [20]. Berikut ini merupakan contoh blok diagram yang ditunjukan pada Gambar 2.12 dibawah ini.

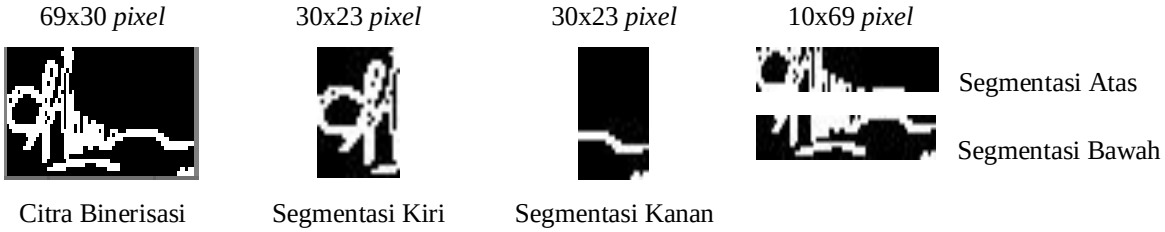


Gambar 0.12 Proses Binerisasi Citra

2.3.2.6 Segmentasi Vertikal & Horizontal

Segmentasi vertikal dan horizontal merupakan proses yang bertujuan untuk mengenali sebuah objek pada citra dan membedakan antar fitur citra [20]. Pada segmentasi vertikal diproses dengan membagi dua bagian wilayah citra tanda tangan secara vertikal untuk membedakan wilayah fitur awal kurva dan coretan akhir, pembagian wilayah tersebut dilakukan dengan memotong ukuran matriks hasil binerisasi citra yaitu 69x30 *pixel* ke dua bagian wilayah tersebut menjadi ukuran matriks 30x23 *pixel*. Sedangkan untuk segmentasi horizontal dilakukan dengan membagi dua wilayah secara horizontal untuk membedakan wilayah fitur

garis tengah dan garis bawah dengan masing-masing dibagi menjadi ukuran 10x69. Berikut ini merupakan contoh citra tanda tangan dari proses segmentasi vertikal dan horizontal.



2.4 Ektraksi Fitur

Ektraksi Fitur (*Feature Extraction*) merupakan suatu pengambilan ciri (*feature*) dari suatu bentuk yang nantinya nilai yang didapatkan akan dianalisis untuk proses selanjutnya. Ektraksi fitur (*Feature Extraction*) bertujuan untuk mencari daerah fitur yang signifikan pada gambar atau citra tergantung pada karakteristik intriksi dan aplikasinya. *Feature Extraction* dilakukan dengan cara menghitung jumlah titik atau *pixel* yang ditemui dalam setiap pengecekan, dimana pengecekan dilakukan dalam berbagai arah *traching* pengecekan pada koordinat katersian dari citra digital yang dianalisis, yaitu vertikal, horizontal, diagonal kanan, dan diagonal kiri. Pada penelitian ini akan menggunakan metode *Chain Code* sebagai *Feature* Extraction nya.

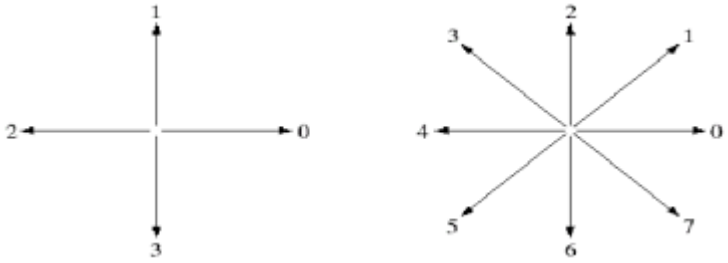
2.4.1 *Chain Code*

Metode *Chain Code* adalah metode yang digunakan untuk mepresentasikan batas objek yang dinyatakan dengan rantai arah. *Chain Code* pertama digunakan untuk mewakili suatu gambar diperkenal oleh *Freeman* pada tahun 1961 sehingga metode ini dinamakan *Freeman Chain Code* (FCC). Sebelum melakukan pengkodean dengan *chain code*, terlebih dahulu batas citra harus divisualisasikan kedalam *Rectangular Cell*. *Rectangular Cell* adalah sel-sel berbentuk segiempat, dimana kontur atau batas objek dapat digambarkan pada sisi-sisi dari sel. *Chain Code* memiliki beberapa keunggulan sebagai berikut [23]:

1. *Chain Code* adalah representasi pada suatu objek biner.

2. Lebih mudah untuk membandingkan objek menggunakan metode *Chain Code* karena merupakan representasi terjemahan invarian suatu objek biner.
3. *Chain Code* merupakan representasi lengkap dari suatu objek atau kurva, karena itu dapat memudahkan dalam menghitung suatu fitur pada gambar atau citra.
4. *Chain Code* menyediakan kompresi *loseless* dan mempertahankan semua topologi dan morfologi dari suatu informasi yang akan berguna dalam analisis pola garis dalam hal kecepatan dan efektivitas.

Chain Code digunakan untuk mempresentasikan batas suatu objek dengan menggunakan garis berarah yang saling terhubung. Terdapat dua arah dalam *chain code* yaitu arah dalam ketetanggaan empat yang berarti mempunyai empat arah seperti mata angin yaitu utara, timur, selatan dan barat. Lalu, arah dalam ketetanggan delapan yang berarti mempunyai delapan arah seperti pada Gambar 2.12



Gambar 0.13 Arah Ketetanggan Empat dan Delapan

2.5 Klasifikasi

Klasifikasi adalah suatu pengelompokan data dimana data yang digunakan tersebut mempunyai kelas label atau target. Sehingga algoritma-algoritma untuk menyelesaikan masalah klasifikasi dikategorikan ke dalam *Supervised Learning* atau pembelajaran terawasi. Pada tahap ini pola yang didapatkan pada tahap praproses sebelumnya akan diklasifikasikan berdasarkan kesamaan ciri dan fitur nya menggunakan metode *Learning Vector Quantization* 3 (LVQ3).

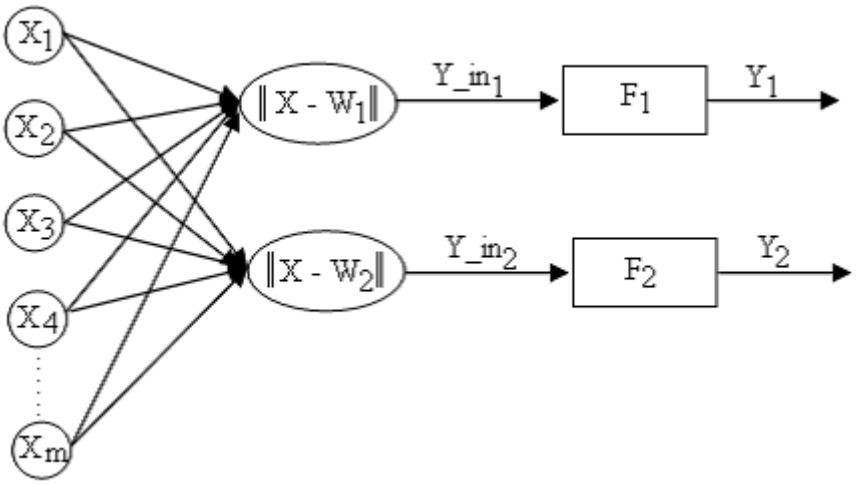
2.5.1 *Learning Vector Quantization*

Learning Vector Quantization (LVQ) adalah suatu metode pelatihan untuk melakukan pembelajaran pada lapisan kompetitif yang terawasi (*Supervised Learning*) yang arsitektur jaringannya berlayer tunggal (*Single Layer*). Kelas-kelas yang didapatkan sebagai hasil dari lapisan kompetitif ini hanya tergantung pada jarak antara vektor-vektor masukan [24]. Jika dua vektor input mendekati sama, maka lapisan kompetitif akan meletakan kedua vektor input tersebut kedalam kelas yang sama. Vector bobot untuk suatu unit keluaran sering dinyatakan sebagai sebuah vector referensi. Setelah pelatihan, jaringan LVQ mengklasifikasi vector masukan dengan menugaskan ke kelas yang sama sebagai unit keluaran, sedangkan yang mempunyai vector referensi diklasifikasikan sebagai vector masukan.

Learning Vector Quantization (LVQ) merupakan jaringan lapisan tunggal (*single-layer net*) di mana lapisan masukan terkoneksi secara langsung dengan setiap neuron pada keluaran. Koneksi antar neuron tersebut dihubungkan dengan bobot/weight. Bobot merupakan nilai hubungan antar lapisan yang mentransfer data dari satu lapisan ke lapisan lainnya, yang berfungsi untuk mengatur jaringan sehingga dapat menghasilkan output yang diinginkan. Bobot pada LVQ sangat penting, karena dengan bobot ini input dapat melakukan pembelajaran dalam mengenali suatu pola. Vektor bobot berfungsi untuk menghubungkan setiap neuron pada lapisan input dengan masing-masing neuron pada lapisan output [25].

2.5.1.1 Arsitektur Jaringan LVQ

Berikut ini merupakan arsitektur jaringan LVQ yang ditunjukan pada Gambar 2.13 dibawah ini:



Gambar 0.14 Arsitektur Jaringan LVQ

Setiap sub kelas diwakili oleh suatu neuron pada output lapisan kompetitif dan setiap kelas diwakili oleh satu neuron pada output lapisan linear. Neuron pada output lapisan kompetitif biasanya disebut sebagai hidden neuron dan neuron pada output lapisan linear disebut sebagai output neuron. Sub kelas pada lapisan kompetitif adalah hasil kompetisi pada lapisan tersebut sedangkan kelas pada lapisan linear adalah kelas yang di definisikan oleh pengguna (target). Pemrosesan yang terjadi pada setiap neuron adalah mencari jarak terdekat antara suatu vektor masukan ke bobot yang bersangkutan.

Hasil dari lapisan kompetitif ini berupa kelas, yang kemudian akan dihubungkan dengan lapisan output oleh fungsi aktivasi. Fungsi aktivasi (F) yang digunakan pada arsitektur jaringan LVQ adalah fungsi linear. Penggunaan fungsi aktivasi linear bertujuan agar diperoleh output yang sama dengan input, dengan rumus fungsi linear yaitu y = x.

2.5.1.2 Algoritma *Learning Vector Quantization* (LVQ1)

Pada algoritma LVQ1 dibutuhkan beberapa parameter diantaranya adalah [26]:

1. X, vektor-vektor pelatihan (X1,…Xi,…Xn).
2. T, kategori atau kelas yg benar untuk vektor-vektor pelatihan.
3. Wj, vektor bobot pada unit keluaran ke-j (W1j,…Wij,…,Wnj).
4. Cj, kategori atau kelas yang merepresentasikan oleh unit keluaran ke-j

5. learning rate (α), α didefinisikan sebagai tingkat pembelajaran. Jika α terlalu besar, maka algoritma akan menjadi tidak stabil sebaliknya jika α terlalu kecil, maka prosesnya akan terlalu lama. Nilai α adalah 0 < α < 1.
6. Nilai pengurangan learning rate, yaitu penurunan tingkat pembelajaran. Pengurangan nilai α yang digunakan pada penelitian ini adalah sebesar 0.1*α.
7. Nilai minimal learning rate (Mina), yaitu minimal nilai tingkat pembelajaran yang masih diperbolehkan.
8. Pembaharuan bobot dilakukan dengan kondisi:

   a. Jika T = Cj maka: $Wj(t+1) = Wj(t) + \alpha(t)[x(t) - Wj(t)]$ (0.8)

   b. Jika T ≠ Cj maka: $Wj(t+1) = Wj(t) - \alpha(t)[x(t) - Wj(t)]$ (0.9)

Dan berikut ini merupakan tahapan pelatihan pada algoritma LVQ1 [25]:

1. Tetapkan bobot awal variable input ke-j menuju ke kelas ke-i, parameter *learning rate* (α), nilai pengurangan learning rate, nilai minimal learning rate $(Min\ \alpha)$, dan epoch = 0
2. Masukan data input data dan target (T)
3. Kerjakan jika $\alpha > Min\ \alpha$:
   a. Hitung jarak *euclidean* antara vektor W dan vektor X: $\sqrt{(X-W)^2}$
   b. Tentukan jarak terdekat (dmin)
   c. Perbaiki Wj dengan ketentuan pada persamaan (2.8) dan (2.9)
   d. Kurangi nilai α.

Setelah melakukan pelatihan, maka akan diperoleh bobot-bobot akhir (W). Bobot-bobot ini nantinya akan digunakan untuk melakukan simulasi atau pengujian.

.

2.5.1.3 Algoritma *Learning Vector Quantization* 2 (LVQ2)

Algortima LVQ dalam pengembanganya memiliki beberapa variasi, salah satunya adalah LVQ2. Pada algoritma LVQ1 vektor referensi yang paling dekat dengan vektor masukan saja yang diperbaharui. Sedangakan untuk variasi LVQ2, dua vektor (pemenang dan runner-up) diperbaharui jika beberapa kondisi dipenuhi.

Modifikasi pertama adalah LVQ2, kondisi dimana kedua vektor akan diperbaharui jika [27]:

1. Unit pemenang dan runner up (vektor terdekat kedua) merepresentasikan kelas yang berbeda.
2. Vektor masukan mempunyai kelas yang sama dengan runner up.
3. Jarak antara vektor masukan ke pemenang dan jarak antara vektor masukan ke runner up kirakira sama.

Kondisi ini diperlihatkan di dalam notasi berikut:

a. X vektor masukan saat ini
b. Yc vektor referensi terdekat dengan X
c. Yr vektor referensi terdekat berikutnya dengan X (runner up)
d. Dc jarak dari X ke Yc
e. Dr jarak dari X ke Yr

Vektor referensi dapat diperbaharui jika masuk ke dalam daerah yang disebut window (ε). Window yang digunakan untuk memperbaharui vektor referensi didefinisikan sebagai berikut:

Vektor masukan X akan masuk ke dalam window bila

$$\frac{d_c}{d_r} > 1 - \varepsilon, \frac{d_c}{d_r} < 1 + \varepsilon \qquad (0.10)$$

Dengan nilai ε tergantung dari jumlah data pelatihan. Berdasarkan Kohonen (1990an) dalam Fausett (1994) nilai ε = 0.3 adalah nilai yang disarankan. Vektor Yc dan Yr akan diperbaharui bila kondisi 1,2 dan 3 terpenuhi. Vektor Yc dan Yr diperbaharui dengan menggunakan persamaan:

$$Y_c(t+1) = Y_c(t) - \alpha(t)[X(t) - Y_c(t)] \qquad (0.11)$$

$$Y_r(t+1) = Y_r(t) + \alpha(t)[X(t) - Y_r(t)] \qquad (0.12)$$

Berikut ini merupakan tahapan pelatihan pada algoritma LVQ2 [27]:

1. Lakukan inisialisasi bobot w dan j
2. Input α (learning rate) atau derajat pembelajaran dan ε (window)

3. Untuk setiap pelatihan vektor pelatihan W temukan j sehingga $|X_i - W_j|$ bernilai minimum
4. Perbaiki dengan ketentuan :
   a. Jika $T = C_j$ maka $W_j = W_j + \alpha(X_i - W_j)$
   b. Jika $T \neq C_j$ maka $D_1 > (1-\varepsilon) * D_2 \; AND \; D_2 < (1+\varepsilon) * D_1$

      Jika ***True*** maka W yang tidak termasuk vektor X diperbahurui dengan menggunakan persamaan (2.11), Sedangkan W yang termasuk vektor X diperbaharui dengan menggunakan persamaan (2.12)
   c. Maka diperoleh W_j baru, jika ***False*** maka $W_j = W_j - \alpha(X - W_j)$
   d. Lakukan pengurangan α

2.5.1.4 Algoritma *Learning Vector Quantization* 2.1 (LVQ2.1)

Modifikasi LVQ2.1 mempertimbangkan dua vektor referensi tederkat, yaitu $Y_c1 \; dan \; Y_c2$. Kondisi dalam memperbaharui kedua vektor tersebut adalah apabila salah satu dari vektor tersebut (misal, Y_c1) masuk ke dalam kelas yang sama dengan vektor masukan x, sementara vektor lainnya (misal, Y_c2) tidak masuk ke dalam kelas yang sama dengan vektor masukan x, maka dalam LVQ2 vektor x harus masuk ke dalam *window* agar bisa terjadi pembaharuan. Pendefinisian *window* sebagai berikut [27]:

$$min \left[\frac{dc1}{dc2}, \frac{dc2}{dc1}\right] > 1-\in$$
$$max \left[\frac{dc1}{dc2}, \frac{dc2}{dc1}\right] < 1+\in \quad (0.13)$$

Jika kondisi-kondisi tersebut terpenuhi, maka vektor referensi yang masuk ke dalam kelas yang sama dengan vektor x akan diperbaharui menggunakan persamaan:

$$Y_C1(t+1) = Y_C1(t) + \alpha(t)[X(t) - Y_C1(t)] \quad (0.14)$$

Sedangkan untuk vektor referensi yang tidak masuk ke dalam kelas yang sama dengan vektor x akan diperbaharui menggunakan persamaan:

$$Y_C2(t+1) = Y_C2(t) - \alpha(t)[X(t) - Y_C2(t)] \quad (0.15)$$

2.5.1.5 Algoritma *Learning Vector Quantization* 3 (LVQ3)

LVQ 3 merupakan sebuah algoritma hasil pengembangan dari LVQ sebelumnya yaitu LVQ1, LVQ2, dan LVQ2.1. Pada LVQ3 vector referensi dapat diperbaharui jika masuk kedalam daerah yang disebut *window* (ε). *Window* yang digunakan untuk memperbaharui vektor referensi dapat didefiniskan sebagai berikut [27]:

$$min\left(\frac{dc}{dr}, \frac{dr}{dc}\right) > (1-\varepsilon)(1+\varepsilon) \tag{0.16}$$

Vector Yc akan diperbaharui bila kondisi persamaan window terpenuhi atau bernilai ***TRUE*** dan salah satu dari 2 vector referensi terdekat berada dalam kelas yang sama dengan vector masukan. Maka vektor referensi yang masuk ke dalam kelas yang sama dengan vektor masukan akan diperbaharui menggunakan persamaan (2.14) dan vektor referensi yang tidak masuk ke dalam kelas yang sama dengan vektor x akan diperbaharui menggunakan persamaan (2.15).
Apabila persamaan *window* tidak terpenuhi atau bernilai ***FALSE*** maka vector Yc dan Yr diperbaharui dengan menggunakan persamaan sebagai berikut:

$$Y(t+1) = Y(t) + \varepsilon\, \alpha(t)[X(t) - Y(t)] \tag{0.17}$$

Parameter yang dibutuhkan dalam pembelajaran pada LVQ3 diantaranya adalah [28]:

1. X, vektor-vektor pelatihan (X1,…Xi,…Xn).
2. T, kategori atau kelas yg benar untuk vektor-vektor pelatihan.
3. Wj, vektor bobot pada unit keluaran ke-j (W1j,…Wij,…,Wnj).
4. Cj, kategori atau kelas yang merepresentasikan oleh unit keluaran ke-j
5. learning rate (α), α didefinisikan sebagai tingkat pembelajaran. Jika α terlalu besar, maka algoritma akan menjadi tidak stabil sebaliknya jika α terlalu kecil, maka prosesnya akan terlalu lama. Nilai α adalah $0 < \alpha < 1$.
6. Nilai pengurangan learning rate, yaitu penurunan tingkat pembelajaran.

7. Nilai minimal learning rate (Mina), yaitu minimal nilai tingkat pembelajaran yang masih diperbolehkan. Pengurangan nilai α yang digunakan pada penelitian ini adalah sebesar 0.1*α
8. Nilai window (ε), yaitu nilai yang digunakan sebagai daerah yang harus dipenuhi untuk memperbaharui vektor referensi pemenang (Yc1) dan runner-up (Yc2) jika berada dikelas yang berbeda. Dengan menggunakan persamaan (2.16).
9. Jika memenuhi kondisi window (ε) , maka vektor referensi yang masuk ke dalam kelas yang sama dengan vektor x akan diperbaharui menggunakan persamaan 2.14.
10. Sedangkan vektor refrensi yang tidak masuk ke dalam kelas yang sama dengan vektor x akan diperbaharui menggunakan persamaan 2.15.

2.6 Pengujian *Confusion Matrix*

Confusion matrix adalah suatu metode yang biasanya digunakan untuk melakukan perhitungan akurasi pada konsep data mining atau Sistem Pendukung Keputusan. 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). nilai *True Negative* (TN) merupakan jumlah data negatif yang terdeteksi dengan benar, sedangkan *False Positive* (FP) merupakan data negatif namun terdeteksi sebagai data positif. Sementara itu, *True Positive* (TP) merupakan data positif yang terdeteksi benar. *False Negative* (FN) merupakan kebalikan dari True Positive, sehingga data positif, namun terdeteksi sebagai data negatif. Berikut ini merupakan contoh dari *confusion matrix* [29] yang ditunjukan pada Tabel 2.3.

Tabel 0.3 Confusion Matrix

		True Values	
		True	**False**
Prediction	**True**	TP *Correct result*	FP *Unexpected result*
	False	FN	TN

		Missing result	*Correct absence*

Perhitungan untuk mendapatkan akurasi atau tingkat pengenalan, dapat menggunakan persamaan (2.18).

$$\text{Akurasi} = \frac{TP+TN}{TP+TN+FP+FN} \quad (0.18)$$

Perhitungan untuk mengetahui *Error rate* atau tingkat kesalahan atau keliruan klasifikasi, dapat menggunakan persamaan (2.19).

$$\text{F1-Score} = 2\ x\ \frac{Percision\ x\ Recall}{Precision + Recall} \quad (0.19)$$

Perhitungan untuk mendapatkan *Recall* atau sensitivitas atau *true positive rate*, dapat menggunakan persamaan (2.20).

$$Recall = \frac{TP}{TP+FN} \quad (0.20)$$

Perhitungan untuk mendapatkan *Precission*, dapat menggunakan persamaan (2.21).

$$Precission = \frac{TP}{TP+FP} \quad (0.21)$$

2.7 *Unified Modeling Language*

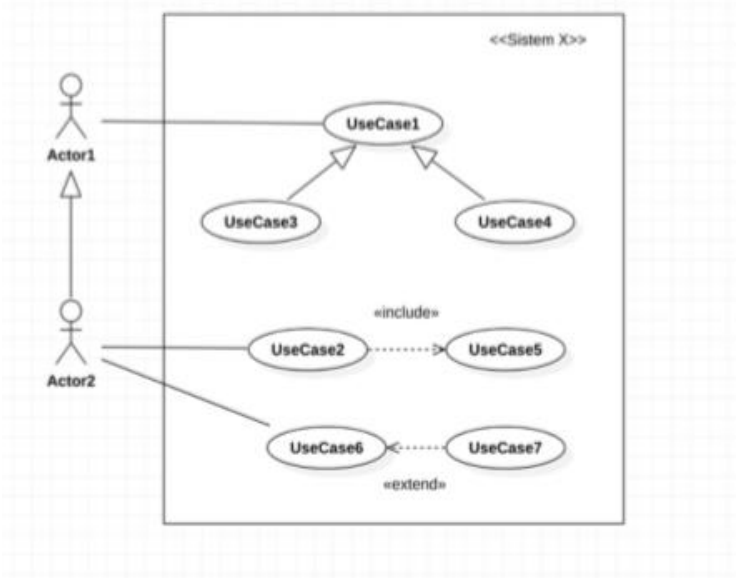
Unified Modeling Language (UML) merupakan kumpulan dari diagram-diagram yang digunakan untuk memodelkan metode analisis berorientasi object dan design beriorientasi object (*OOAD&D/Object Oriented Analysis and Design*). **UML** adalah bahasa untuk menspesifikasi, memvisualisasi, membangun dan mendokumentasikan *artifacts* (bagian dari informasi yang digunakan untuk dihasilkan oleh proses pembuatan perangkat lunak, *artifact* tersebut dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya. Selain itu UML adalah bahasa pemodelan yang menggunakan konsep orientasi *object*.

UML versi 2.0 mencakup 13 macam diagram dan perangkat yang berfungsi untuk menggambarkan sistem informasi berorientasi objek dengan sangat lengkap dan rinci. Meski demikan, tidak selalu 13 diagram tersebut digunakan saat para pengembang sedang mengembangkan perangkat lunak yang berorientasi objek.

Beberapa diagram UML yang digunakan dalam penelitian ini, antara lain ialah Use *Case Diagram*, *Activity Diagram*, *Sequance Diagram*, dan *Class Diagram*.

2.7.1 *Use Case Diagram*

Use case diagram merupakan model diagram UML yang digunakan untuk menggambarkan requirement fungsional yang diharapkan dari sebuah sistem. *Use case* diagram adalah diagram *usecase* yang digunakan untuk menggambarkan secara ringkas siapa yang menggunakan sistem dan apa saja yang bisa dilakukannya. *Use case* class digunakan untuk memodelkan dan menyatakan unit fungsi/layanan yang disediakan oleh sistem ke pemakai. Diagram *usecase* tidak menjelaskan secara detail tentang penggunaan *usecase*, namun hanya memberi gambaran singkat hubungan antara *usecase*, aktor, dan sistem. Tujuan *Use Case* adalah memetakkan kebutuhan sistem, merepresentasikan interaksi pengguna terhadap sistem, dan Untuk mengetahui kebutuhan diluar sistem. Berikut ini adalah contoh *Use Case* yang ditunjukan pada Gambar 2.14.



Gambar 0.15 *Use Case Diagram*

1. *Actor*

   Actor mempresentasikan seseorang atau sesuatu (seperti perangkat,sistem lain) yang berinteraksi dengan sistem. *Actor* hanya berinteraksi dengan *use case* tetapi tidak memiliki kontrol atas *use case*.

2. *Use Case*

   Use Case menjelaskan apa yang sistem kita harus kerjakan (fungsional). Satu *use case* dapat mewakili satu kebutuhan *user*.

3. Relasi

   Ada beberapa relasi yang terdapat pada *use case* diagram:

   a. Relasi *Include*, digunakan untuk konsep reusable *use case*. *Use case* yang di-*include* wajib dieksekusi secara eksplisit oleh pengguna sistem.
   b. Relasi *Extend*, mirip dengan kasus relasi include akan tetapi use case yang meng-*extend* dieksekusi secara opsional.
   c. Relasi *Generalization*, relasi ini digunakan apabila butuh pemodelan aktor atau *use case* yang lebih spesifik (Specialization).

2.7.2 *Activity Diagram*

Activity Diagram merupakan rancangan aliran aktivitas atau aliran kerja dalam sebuah sistem yang akan dijalankan. *Activity Diagram* memberikan gambaran yang lebih baik tentang langkah-langkah di *use case* pada *use case description*. Selain itu *activity diagram* memiliki komponen dengan bentuk tertentu yang dihubungkan dengan tanda panah. Panah tersebut mengarah keurutan aktivitas yang terjadi dari awal hingga akhir. Fungsi *activity diagram* ialah memperlihatkan urutan aktifitas proses pada sistem, Membantu memahami proses secara keseluruhan, *activity diagram* dibuat berdasarkan sebuah atau berapa *use case*, dan menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Berikut ini adalah contoh activity diagram yang di tunjukan pada Gambar 2.15 berserta komponennya:

a. *Initial State*, adalah awal dimulainya suatu aliran kerja pada activity diagram dan hanya terdapat satu *initial state*.
b. *Final State*, adalah bagian akhir dari suatu aliran kerja pada sebuah activity diagram dan bisa terdapat lebih dari suatu *final state*.
c. *Activity*, adalah aktivitas atau pekerjaan yang dilakukan dalam aliran kerja.
d. *Decision*, berfungsi untuk mengambarkan pilihan kondisi dimana ada kemungkinan perbedaan transisi.

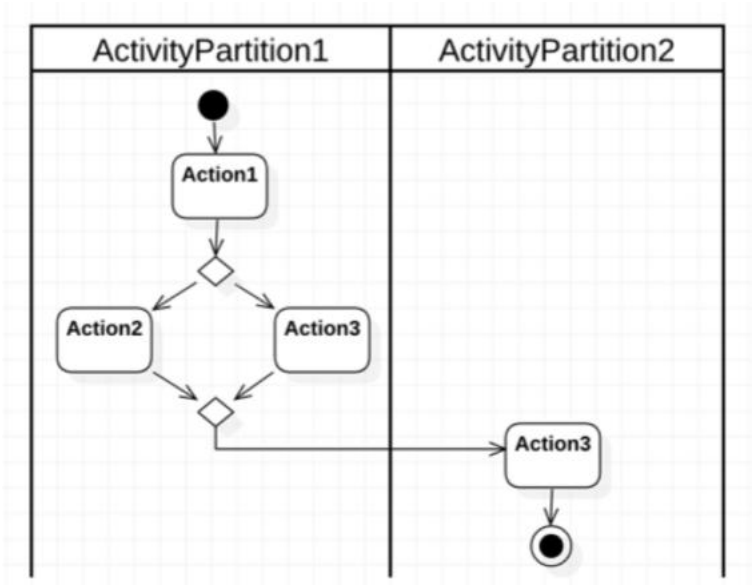
e. *Merge*, berfungsi untuk menggabungkan kembali aliran kerja yang sebelumnya telah dipecah oleh *decision*.

f. Transition / Association

   Transition untuk menghubungkan aktivitas selanjutnya setelah aktivitas sebelumnya.

g. Synchronization

   1. *Synchronization Fork*, digunakan untuk memecah *behavior* menjadi aktivitas yang pararel.
   2. *Synchronization Join*, digunakan untuk menggabungkan kembali aktivitas yang pararel.

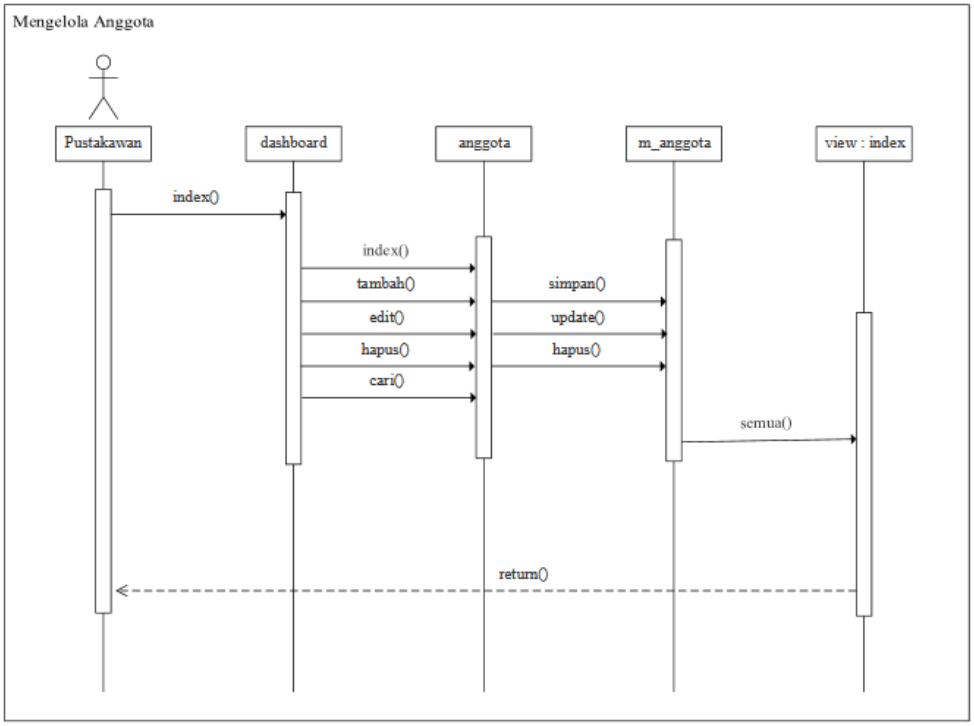


Gambar 0.16 ***Activity Diagram***

2.7.3 *Sequence Diagram*

Sequence Diagram adalah suatu diagram yang menggambarkan interaksi objek dan mengindikasikan (memberi petunjuk atau tanda) komunikasi diantara objek-objek tersebut. *Sequence* diagram digunakan untuk menggambarkan perilaku pada sebuah skenario dan mendeskripsikan bagaimana entitas dan sistem berinteraksi, termasuk pesan yang digunakan saat interaksi. Semua pesan dideskripsikan dalam urutan pada eksekusi. *Sequence* diagram berhubungan erat

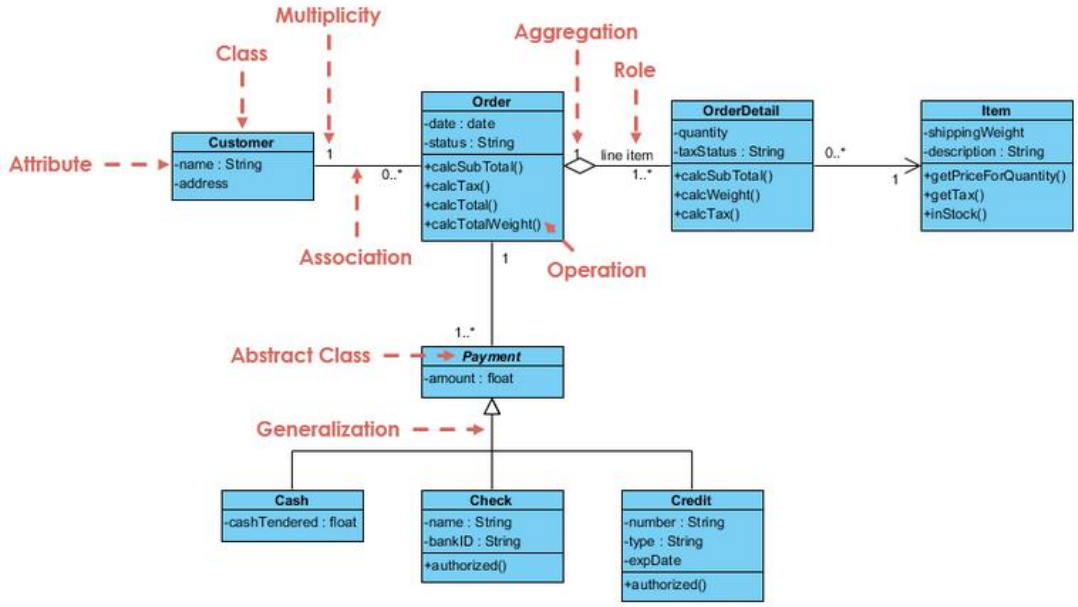
dengan *Use Case Diagram*, dimana 1 *Use Case* akan menjadi 1 *Sequence Diagram*. Contoh *sequence diagram* dapat dilihat pada Gambar 2.16



Gambar 0.17 *Sequence Diagram*

2.7.4 *Class Diagram*

Class Diagram adalah salah satu jenis diagram yang paling berguna di UML, hal ini karena dapat dengan jelas memetakan struktur sistem tertentu dengan memodelkan kelas, atribut, operasi serta hubungan antar objek. *Class Diagram* mampu memberikan pandangan yang lebih luas mengenai suatu sistem dengan cara menunjukkan kelas serta hubungan-hubungannya. Diagram class dapat dikatakan bersifat statis, itu karena diagram kelas tidak menggambarkan apa yang terjadi jika mereka berhubungan melainkan menggambar hubungan apa yang terjadi.



Gambar 0.18 *Class Diagram*

2.8 Python

Python adalah bahasa pemrograman interpretatif multiguna. Tidak seperti bahasa lain yang susah untuk dibaca dan dipahami, python lebih menekankan pada keterbacaan kode agar lebih mudah untuk memahami sintaks. Hal ini membuat Python sangat mudah dipelajari baik untuk pemula maupun untuk yang sudah menguasai bahasa pemrograman lain. Bahasa ini muncul pertama kali pada tahun 1991, dirancang oleh seorang bernama Guido van Rossum. Sampai saat ini Python masih dikembangkan oleh *Python Software Foundation*. Bahasa Python mendukung hampir semua sistem operasi, bahkan untuk sistem operasi Linux, hampir semua distronya sudah menyertakan Python di dalamnya [30].