BAB 2
TINJAUAN PUSTAKA

2.1 Karya Tulis Ilmiah Skripsi

Salah satu jenis karya tulis tulis ilmiah yaitu skripsi. Skripsi adalah istilah yang digunakan di Indonesia untuk mengilustrasikan suatu karya tulis ilmiah berupa paparan tulisan hasil penelitian sarjana S1 yang membahas suatu permasalahan/fenomena dalam bidang ilmu tertentu dengan menggunakan kaidah-kaidah yang berlaku. Skripsi ditulis berdasarkan kajian pada studi pustaka, penyelidikan, observasi, atau penelitian lapangan. Penelitian yang akan dibuat ini menggunakan hasil *scan* beberapa bagian dari karya tulis ilmiah skripsi dalam bentuk gambar/citra untuk dikenali setiap karakter yang terdapat didalamnya.

2.2 *Optical Character Recognition* (OCR)

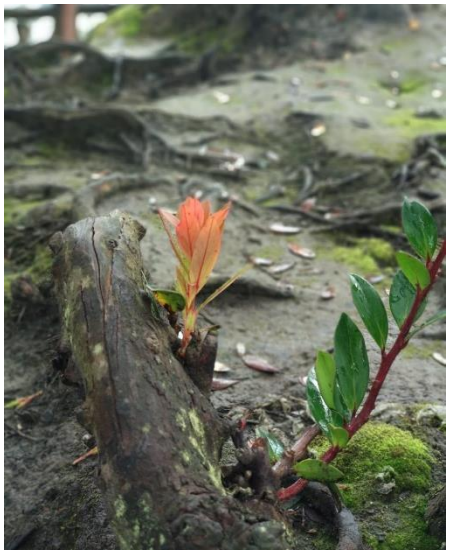
OCR merupakan solusi yang efektif untuk proses konversi dari dokumen cetak ke dalam bentuk dokumen digital. Sistem pengenal huruf ini dapat meningkatkan fleksibilitas atau kemampuan dan kecerdasan komputer. Sistem pengenal huruf yang cerdas sangat membantu usaha digitalisasi informasi dan pengetahuan, misalnya dalam pembuatan koleksi pustaka digital, koleksi sastra kuno, dan lain-lain [13].

Permasalahan yang muncul dalam melakukan proses pengenalan huruf komputer adalah bagaimana sebuah teknik pengenalan dapat mengenali berbagai jenis huruf dengan ukuran, ketebalan, dan bentuk yang berbeda [3]. Oleh karena itu pada penelitian ini bertujuan untuk menguji kemampuan dari metode SVM untuk mengenali karakter cetak pada citra.

2.3 Citra

Citra terbagi menjadi dua jenis, yaitu citra analog dan citra digital, namun dalam studi tentang pengolahan citra, citra yang biasa digunakan adalah citra digital yang merupakan hasil dari alat elektronik yang dapat menangkap gambar. Citra digital dibentuk oleh kumpulan titik yang dinamakan piksel (*pixel* atau "*picture element*"). Setiap piksel digambarkan sebagai satu kotak kecil. Dengan

sistem koordinat yang mengikuti asas pemindaian pada layar TV standar, sebuah piksel mempunyai koordinat berupa (i, j), dimana i menyatakan posisi kolom, dan j menyatakan posisi baris. Piksel pojok kiri-atas mempunyai koordinat (0,0) dan piksel pada pojok kanan mempunyai koordinat (i-1, j-1) [14]. Salah satu contoh citra digital dapat dilihat pada Gambar 2.1.



Gambar 2.1 Contoh Citra Digital

Pada penelitian ini, citra berfungsi sebagai data masukan yang akan berpengaruh terhadap proses dan hasil akhir. Maka dari itu, kualitas citra masukan sangat menentukan tingkat akurasi pada proses pengujian. Oleh karena itu, dilakukan proses pengolahan citra untuk memperoleh kualitas citra yang baik.

2.3.1 Jenis-jenis Citra

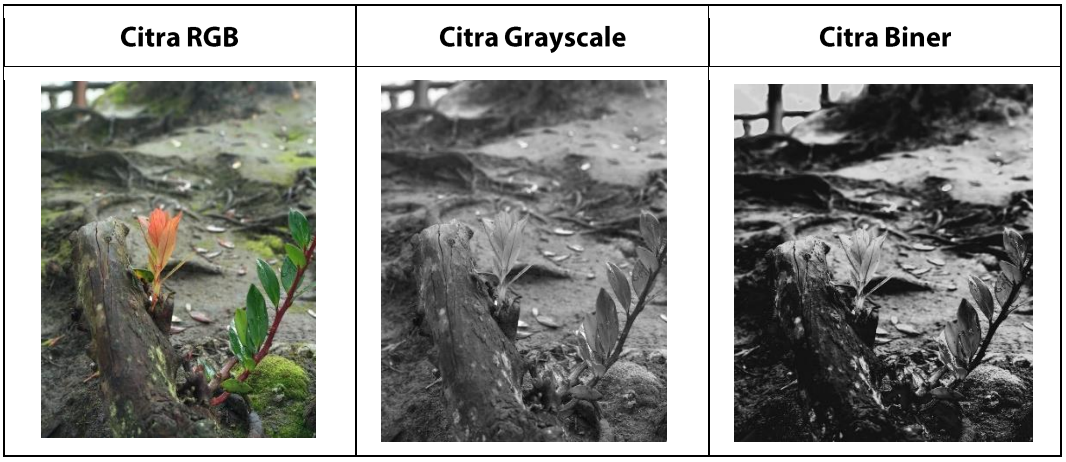
Jenis-jenis citra yang biasa digunakan diantaranya yaitu [15]:

a. Citra Berwarna: Citra yang memiliki 3 buah kanal warna didalamnya, pada umumnya terbentuk dari komponen merah/*red* (R), hijau/*green* (G), dan biru/*blue* (B) yang dimodelkan kedalam ruang warna RGB. RGB adalah standar yang digunakan untuk menampilkan citra berwarna pada layar televisi maupun layar komputer.
b. Citra *Grayscale*: Citra yang hanya memiliki 1 buah kanal sehingga yang ditampilkan hanyalah nilai intensitas atau dikenal juga dengan istilah derajat

keabuan. Jenis citra ini disebut juga sebagai 8-bit *image* karena untuk setiap nilai pikselnya memerlukan penyimpanan sebesar 8-bit.

c. Citra Biner: Citra yang hanya memiliki 2 kemungkinan nilai untuk setiap pikselnya, yaitu 0 atau 1. Nilai 0 akan tampil sebagai warna hitam sedangkan nilai 1 akan tampil sebagai warna putih.

Penelitian ini akan menggunakan ketiga jenis citra yang telah disebutkan melalui proses pengolahan citra. Perbedaan ketiga jenis citra tersebut dapat dilihat pada Gambar 2.2.



Gambar 2.2 Perbedaan Citra RGB, Grayscale, dan Biner

2.3.2 Pengolahan Citra Digital

Pengolahan citra adalah istilah umum untuk berbagai teknik yang keberadaanya untuk memanipulasi dan memodifikasi citra dengan berbagai cara. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital (misalnya berasal dari kamera digital) dapat diolah melalui perangkat lunak tertentu [14]. Pada penelitian ini pengolahan citra dilakukan untuk memproses citra masukan sehingga kualitas citra masukan menjadi lebih baik sebelum masuk ke proses berikutnya.

2.4 *Preprocessing* Citra

Preprocessing merupakan suatu proses untuk menghilangkan bagian-bagian yang tidak perlu pada gambar. *Preprocessing* adalah tahap pertama yang harus dilakukan sebelum proses utama dari pengenalan karakter dilakukan. Tahap ini

sangat penting untuk menentukan keberhasilan suatu proses pengenalan pola [3]. Tahap *preprocessing* yang dilakukan dalam penelitian ini yaitu konversi citra RGB ke *grayscale*, *thresholding*, segmentasi citra menggunakan *profile projection*, binerisasi, *resize*, serta ekstraksi fitur zoning.

2.4.1 Konversi Citra ke *Grayscale*

Tahapan awal dalam *preprocessing* pada penelitian ini adalah konversi citra RGB ke citra *grayscale*. *Grayscale* adalah istilah untuk menyebutkan satu citra yang memiliki warna abu-abu, hitam, dan putih. Pada jenis gambar ini, warna dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar antara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih [14]. Karena jenis citra ini hanya memiliki 1 kanal saja, maka citra *grayscale* memiliki tempat penyimpanan yang lebih hemat. Jenis citra ini disebut juga sebagai citra 8-bit [15]. Pada penelitian ini menggunakan metode *grayscale* dengan Persamaan 2.1 sebagai berikut.

$$GS_{(i,j)} = (0{,}299 * R_{(i,j)}) + (0{,}587 * G_{(i,j)}) + (0{,}114 * B_{(i,j)}) \quad (2.1)$$

Keterangan :

$GS_{(i,j)}$ = Citra *grayscale*

$R_{(i,j)}$ = Nilai dari piksel berwarna merah (*Red*)

$G_{(i,j)}$ = Nilai dari piksel berwarna hijau (*Green*)

$B_{(i,j)}$ = Nilai dari piksel berwarna biru (*Blue*)

2.4.2 *Thresholding*

Thresholding yaitu mengubah citra *grayscale* menjadi citra berwarna hitam putih dengan nilai ambang yang sudah ditentukan. Terdapat dua metode *thresholding*, yaitu *global thresholding* dan *local thresholding* [14]. Apabila nilai ambang (*threshold*) bergantung hanya pada satu nilai aras keabuan, pengambangan disebut global, dimana semua piksel dalam citra akan ditentukan oleh satu nilai ambang yang sudah ditentukan. Pengambangan ini biasa digunakan untuk memisahkan tulisan hitam yang berada diatas secarik kertas putih. Namun, jika nilai ambang bergantung pada beberapa nilai sesuai dengan aras keabuan

pada citra maka disebut pengambangan lokal. Dimana nilai ambang untuk setiap piksel ditentukan oleh nilai piksel tetangga, maka nilai ambang untuk masing-masing piksel belum tentu sama. Perbedaan ciri kedua jenis *thresholding* ini yaitu sebagai berikut.

a. Ciri-ciri *global thresholding*

   Ciri-ciri dari *global thresholding* yaitu [14]:

   1) Tidak memperhatikan hubungan spasial antarpiksel.
   2) Sensitif terhadap pencahayaan tidak seragam.
   3) Hanya berlaku untuk keadaan ideal (misalnya, latar belakang berwarna putih dan objek berwarna hitam).
   4) Bergantung kepada pemilihan nilai ambang yang tepat.

b. Ciri-ciri *local thresholding*

   Ciri-ciri dari *local thresholding* yaitu [14]:

   1) Memperhatikan hubungan spasial antarpiksel.
   2) Mampu beradaptasi dengan pencahayaan yang tidak seragam.
   3) Berlaku untuk keadaan ideal atau tidak.

Berdasarkan ciri-ciri yang telah disebutkan, maka dalam penelitian ini akan menggunakan metode *global thresholding*, yang dilakukan dengan cara mengubah nilai citra *grayscale* menjadi nilai 0 atau 255 berdasarkan nilai *threshold* yang telah ditentukan sebelumnya. Metode ini digunakan karena data latih dan data uji yang digunakan merupakan citra yang memiliki latar belakang berwarna putih dan objek berwarna hitam, serta memiliki tingkat pencahayaan yang seragam.

Pemilihan nilai ambang yang tepat sangat berpengaruh pada *global thresholding.* Karena hal tersebut bisa mempengaruhi berhasil atau tidaknya proses *thresholding* pada citra. Berdasarkan penelitian sebelumnya [16], terdapat 5 nilai ambang batas di dalam pengujiannya, yaitu nilai tengah, nilai minimum, nilai maksimum, nilai modulus, dan nilai *minimax between*. Nilai tengah adalah 128, nilai minimum diambil dari nilai tengah dibagi 2 sehingga nilai minimum adalah 64, nilai maksimum adalah 128 ditambah dengan 64 sehingga hasilnya adalah 192, nilai modulus yaitu jika nilai *grayscale* dimodulus dengan 2 = 1 maka akan diubah menjadi 255 sedangkan bila hasilnya sama dengan 0 maka tidak

diubah, sementara nilai *minimax between* hampir sama dengan nilai modulus tetapi jika nilai *grayscale* < 64 maka diubah menjadi 0, dan jika > 192 maka diubah menjadi 255. Penelitian tersebut menyatakan bahwa nilai tengah mempunyai hasil warna yang lebih jelas ketika dibandingkan dengan nilai *threshold* lainnya [16].

Oleh karena itu, nilai ambang yang digunakan pada penelitian ini yaitu sebesar 128. Nilai tersebut didapatkan dari total intensitas derajat keabuan dibagi 2. Dimana nilai total intensitas derajat keabuan yaitu sebesar 255 dibagi 2, hasilnya adalah 127.5 dibulatkan menjadi 128. Apabila nilai citra *grayscale* kurang dari 128, maka nilainya akan diubah menjadi 0 (hitam/objek), dan apabila nilai citra *grayscale* lebih dari 128, maka nilainya akan diubah menjadi 255 (putih/*background*). Seperti pada persamaan berikut.

$$T_{(i,j)} = \begin{cases} 255, GS_{(i,j)} \geq 128 \\ 0, GS_{(i,j)} < 128 \end{cases} \quad (2.2)$$

Keterangan :

$T_{(i,j)}$: Nilai *threshold*

$GS_{(i,j)}$: Nilai citra *grayscale* suatu piksel

2.4.3 Segmentasi Citra Menggunakan *Profile Projection*

Metode *Profile Projection* digunakan untuk memisahkan tulisan perbaris, perkata dan perkarakter. Berikut ini masing-masing penjelasan proses segmentasi.

2.4.3.1 Segmentasi Baris

Proses segmentasi baris bertujuan untuk memisahkan baris-baris pada citra yang mengandung karakter. Segmentasi baris dilakukan secara horizontal (terhadap sumbu j) dengan cara menjumlahkan nilai piksel hitam pada setiap baris. Jika ditemukan jumlah piksel hitam pada suatu baris <5 atau = 0 maka akan dijadikan acuan untuk pemotongan perbaris [9]. Baris yang sudah ditemukan selanjutnya akan dipotong dan disimpan untuk proses segmentasi kata. Segmentasi baris akan terus dilakukan hingga semua baris yang mengandung karakter pada citra dipotong dan disimpan.

2.4.3.2 Segmentasi Kata

Citra hasil segmentasi baris selanjutnya akan masuk ke tahap segmentasi kata untuk memisahkan setiap kata yang terdapat pada baris. Proses segmentasi ini dilakukan secara vertikal (terhadap sumbu i). Proses yang dilakukan yaitu dengan melihat proyeksi piksel hitam secara vertikal. Namun acuan untuk pemisahannya yang agak berbeda dengan pemisahan baris, karena rentang piksel yang tidak mengandung piksel hitam (berwarna putih) untuk memisahkan kata harus lebih besar agar pemisahan benar-benar memisahkan kata bukan karakter. Acuan yang digunakan yaitu dengan cara menentukan ambang batas spasi untuk memisahkan setiap kata. Ambang batas yang ditentukan yaitu 80% dari tinggi citra baris [17] seperti menggunakan persamaan berikut.

$$S_k = 0{,}3 * t \tag{2.3}$$

Keterangan :

S_k : Nilai ambang batas spasi kata

t : Tinggi citra hasil segmentasi baris

Dimana jika ditemukan jarak kolom dengan jumlah piksel hitam = 0 berjumlah lebih dari nilai ambang spasi kata, maka kolom tersebut akan dianggap sebagai batas antar kata. Lakukan semua tahap tersebut pada setiap baris sehingga kata-kata yang terdapat pada setiap baris dapat terpisah.

2.4.3.3 Segmentasi Karakter

Proses segmentasi karakter hampir sama dengan proses segmentasi kata, hanya berbeda pada penentuan nilai ambang batas. Batas ambang yang digunakan dalam pemisahan karakter yaitu tidak menggunakan batas ambang khusus. Dimana jika ditemukan jarak kolom dengan jumlah piksel hitam = 0, maka kolom tersebut akan dianggap sebagai batas antar huruf. Lakukan semua tahap tersebut pada setiap baris sehingga huruf-huruf yang terdapat pada setiap baris dapat terpisah.

2.4.4 *Resize*

Resize adalah proses yang digunakan untuk mengubah ukuran citra digital dalam piksel, baik menjadi lebih kecil atau lebih besar dari ukuran sebenarnya. Proses resize pada penelitian ini tidak memerlukan metode khusus, caranya hanya dengan dilakukan perbandingan ukuran antara citra hasil *thresholding* (pada tahap latih) dan hasil segmentasi (pada tahap uji) dengan menggunakan persamaan 2.4 untuk mendapatkan posisi koordinat dari titik x yang baru dan persamaan 2.5 untuk mendapatkan posisi koordinat dari titik y yang baru. Dimana kedua persamaan tersebut adalah sebagai berikut.

$$i_r = \frac{pb * pp}{pa} \tag{2.4}$$

Keterangan:

i_r = Posisi koordinat i baru

pb = Ukuran panjang dari matriks baru

pp = Posisi koordinat i lama

pa = Ukuran panjang dari matriks lama

Untuk mencari koordinat j yang baru, digunakan Persamaan 2.5 sebagai berikut.

$$j_r = \frac{lb * pp}{la} \tag{2.5}$$

Keterangan:

j_r = Posisi koordinat j baru

lb = Ukuran lebar dari matriks baru

pp = Posisi koordinat j lama

la = Ukuran lebar dari matriks lama

Pada penelitian ini *resize* digunakan untuk mengubah ukuran *pixel* sesuai dengan kebutuhan dalam melakukan proses ektraksi ciri. Hal ini dilakukan agar semua citra karakter dari hasil segmentasi mempunyai ukuran yang sama (normalisasi ukuran citra) sebelum masuk ke tahap ekstraksi fitur.

2.4.5 Binerisasi

Setelah melakukan proses *resize*, selanjutnya yaitu melakukan binerisasi. Proses binerisasi untuk mengubah citra *hasil resize* menjadi citra biner, artinya mengubah warna tiap-tiap piksel pada citra bernilai 0 dan 255 ke dalam piksel bernilai 0 dan 1. Sehingga citra hanya berwarna hitam dan putih. Namun dalam penelitian ini nilai biner diubah dengan cara membalik nilai, jika nilai piksel citra hasil *resize* $= 255$ maka akan diubah menjadi 0 (hitam/*background*), dan apabila nilai citra *grayscale* $= 1$ akan diubah menjadi 1 (putih/objek). Hal ini dilakukan karena nilai objek akan dilakukan untuk proses ekstraksi fitur. Persamaan 2.6 digunakan untuk proses binerisasi, jika citra yang akan diubah berasal dari proses *thresholding*.

$$B_{(i,j)} = \begin{cases} 1, T_{(i,j)} = 0 \\ 0, T_{(i,j)} = 255 \end{cases} \quad (2.6)$$

Keterangan :

$B_{(i,j)}$: Nilai biner

$T_{(i,j)}$: Nilai citra suatu piksel hasil proses *thresholding*

Jika citra yang akan diubah berasal dari proses *resize* maka pengubahan menjadi citra biner akan dilakukan dengan mernggunakan Persamaan 2.7.

$$B_{(i,j)} = \begin{cases} 1, R_{(i,j)} = 0 \\ 0, R_{(i,j)} = 255 \end{cases} \quad (2.7)$$

Keterangan :

$B_{(i,j)}$: Nilai biner

$R_{(i,j)}$: Nilai citra suatu piksel hasil proses *resize*

2.4.6 Ektraksi Fitur Zoning

Ekstraksi fitur zoning merupakan salah satu metode ekstaksi ciri dimana inti dari metode ini adalah citra dibagi pada sejumlah zona yang sama untuk dikenali ciri dari setiap karakter huruf yang selanjutnya menghasilkan sebuah nilai untuk diproses pada tahapan klasifikasi [18].

Ada beberapa algoritma untuk metode ekstraksi ciri *zoning*, diantaranya metode ekstraksi ciri jarak metrik ICZ *(image centroid zone)*, metode ekstraksi ciri jarak metrik ZCZ *(Zone centroid and zone)*. Kedua algoritma tersebut menggunakan citra digital sebagai input dan menghasilkan fitur untuk klasifikasi dan pengenalan sebagai *output*-nya. Pada penelitian ini digunakan metode ekstraksi ciri jarak metrik ICZ. Berikut merupakan tahapan dalam proses ekstraksi ciri ICZ [19].

1. Hitung centroid dari citra masukan menggunakan persamaan berikut.

$$c_i = \frac{(P_{i1}.p_1 + P_{i2}.p_2 + \cdots + P_{in}.p_n)}{(p_1 + p_2 + \cdots + p_n)} \quad (2.8)$$

$$c_j = \frac{(P_{j1}.p_1 + P_{j2}.p_2 + \cdots + P_{jn}.p_n)}{(p_1 + p_2 + \cdots + p_n)} \quad (2.9)$$

Keterangan:

c_i = *centroid* koordinat i

c_i = *centroid* koordinat j

P_{in} = koordinat i dari piksel ke-n

P_{jn} = koordinat j dari piksel ke-n

p_n = nilai piksel ke-n

2. Bagi citra masukan ke dalam n zona yang sama. Contoh pembagian 3 Zona pada citra biner dapat dilihat pada Tabel 2.1.

Tabel 2.1 Pembagian Zona Pada Citra Biner

0	0	1	0	0	**Zona 1 (atas)**
0	1	0	1	0	
1	0	0	0	1	
0	1	0	0	0	**Zona 2 (Tengah)**
0	0	1	0	0	
0	0	0	1	0	
1	0	0	0	1	**Zona 3 (Bawah)**
0	1	0	1	0	
0	0	1	0	0	

3. Hitung jarak antara centroid citra dengan masing-masing piksel yang ada dalam zona.

$$d(P_{i,j}, c_{i,j}) = \sqrt{(P_i - c_i)^2 + (P_j - c_j)^2} \quad (2.10)$$

Keterangan:

d = jarak antara koordinat *centroid* (i,j) dengan koordinat objek

P_i = koordinat i objek

P_j = koordinat j objek

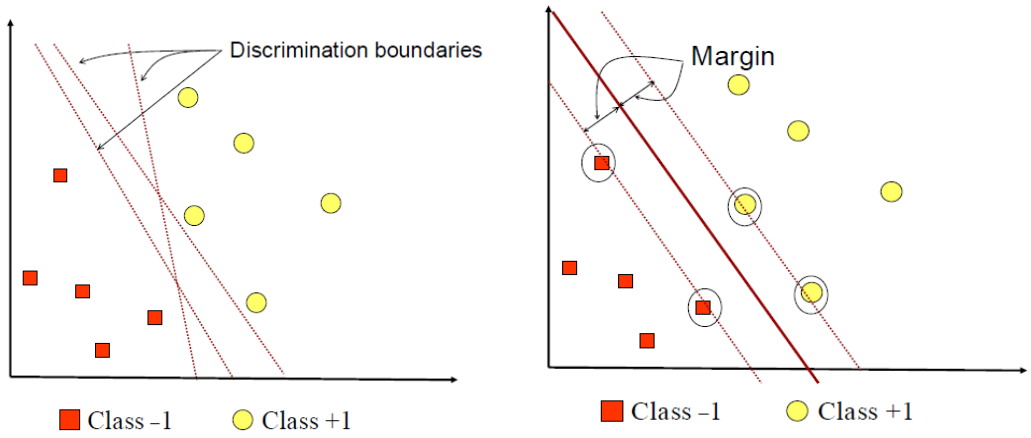
c_i = centroid koordinat i

c_j = centroid koordinat j

4. Ulangi langkah ke 3 untuk setiap piksel yang ada di zona
5. Hitung rata-rata jarak antara titik-titik tersebut.
6. Ulangi langkah-langkah tersebut untuk keseluruhan zona,
7. Hasilnya adalah n fitur yang akan digunakan dalam klasifikasi dan pengenalan.

2.5 *Support Vector Machine* (SVM)

Support Vector Machine (SVM) merupakan salah satu algoritma machine learning yang paling popular untuk klasifikasi dan regresi. Konsep klasifikasi dengan SVM dapat dijelaskan secara sederhana sebagai usaha untuk mencari hyperplane terbaik yang berfungsi sebagai pemisah dua buah kelas data pada urang input. Hyperplane (batas keputusan) pemisah terbaik antara kedua kelas dapat ditemukan dengan mengukur margin hyperplane tersebut dan mencari titik maksimalnya. Margin adalah jarak antara hyperlane tersebut dengan data terdekat dari masing-masing kelas. Data yang paling dekat ini disebut *support vector* [20].



a. Batas keputusan yang mungkin b. Batas keputusan dengan margin maksimal

Gambar 2.3 *Hyperplane* SVM

Citra hasil ekstraksi fitur kemudian akan diubah menjadi bentuk vektor yang nantinya akan digunakan pada pelatihan ataupun pengujian SVM. Format yang digunakan adalah sebagai berikut.

[nilai_piksel_1, nilai_piksel_2,..., nilai_piksel_n]

Gambar 2.4 Format Vektor Masukan SVM

SVM adalah salah satu algoritma yang termasuk dalam *supervised learning*, oleh karena itu dibutuhkan satua buah vektor lagi yang berisikan nama kelas dari vektor yang akan dilatih. Jumlah dari vektor kelas (y) adalah = N data latih yang digunakan. Pelatihan SVM membutuhkan minimal 2 data latih dengan kelas yang berbeda. Langkah pertama dalam pelatihan adalah kernelisasi vektor masukan. Kernel yang digunakan bisa berbagai macam, beberapa kernel yang terdapat pada SVM meliputi :

a. Kernel linier

$$K(x_v, x) = x_v^T x \tag{2.11}$$

b. Polynomial

$$K(x_v, x) = (\Upsilon\, x_v^T x + r)^p, \Upsilon > 0 \tag{2.12}$$

c. *Radial Basis Function* (RBF)

$$K(x_v, x) = \exp(-\Upsilon |x_v - x|^2), \Upsilon > 0 \tag{2.13}$$

d. Sigmoid Kernel

$$K(x_v, x) = \tanh(\Upsilon\, x_v^T x + r) \tag{2.14}$$

Dalam penelitian ini, kernel yang digunakan yaitu kernel linear karena berdasarkan penelitian sebelumnya, kernel linear menghasilkan tingkat akurasi yang tinggi dalam kasus pengenalan karakter [10]. Karena data yang digunakan tidak bisa dipisahkan secara linier dan kelas yang digunakan sebanyak 80 kelas, maka pada penelitian ini menggunakan SVM nonlinier dengan menggunakan pendekatan *one vs all.*

2.5.1 SVM Nonlinier

SVM sebenarnya adalah *hyperlane* linier yang hanya bekerja pada data yang dapat dipisahkan secara linier. Untuk data yang distribusi kelasnya tidak linier biasanya digunakan pendekatan kernel pada fitur data awal set data. Kernel dapat didefinisikan sebagai suatu fungsi yang memetakan fitur data dari dimensi awal ke fitur lain yang berdimensi lebih tinggi. Pendekatan ini berbeda dengan metode klasifikasi pada umumnya yang justru mengurangi dimensi awal untuk menyederhanakan proses komputasi dan memberikan akurasi prediksi yang lebih baik. Algoritma pemetaan kernel ditunjukan pada persamaan berikut.

$$\Phi: D^q \rightarrow D^r, x \rightarrow \Phi(x) \tag{2.15}$$

Φ merupakan fungsi kernel yang dapat digunakan untuk pemetaan, D merupakan data latih, q merupakan set fitur dalam satu data lama, dan r merupakan set fitur yang baru sebagai hasil pemetaan untuk data latih. Sementara x merupakan data latih, dimana $x1, x2, \dots x_n \in D^q$ merupakan fitur-fitur yang akan dipetakan ke fitur berdimensi tinggi r, jadi untuk set data yang digunakan sebagai pelatihan dengan algoritma yang ada dari dimensi fitur yang lama D ke dimensi baru r. Misalnya, untuk n sampel data.

$$(\Phi(x_1), y_1, \Phi(x_2), y_2, \dots \Phi(x_1), y_n) \in D^r \tag{2.16}$$

Selanjutnya dilakukan proses pelatihan yang sama sebagaimana pada SVM linier. Proses pemetaan pada fase ini memerlukan perhitungan dot-product dua buah data pada ruang fitur baru. Dot-product kedua buah vektor (x_v) dan (x_w) dinotasikan sebagai $\Phi(x_v).\Phi(x_w)$. nilai dot-product kedua buah vektor ini dapat dihitung secara tidak langsung, yaitu tanpa mengetahui fungsi transformasi Φ. Teknik komputasi seperti ini kemudian disebut trik kernel, yaitu menghitung dot-product dua buah vektor di ruang dimensi baru dengan memakai komponen kedua buah vektor tersebut di ruang asal, seperti berikut.

$$K(x_v, x_w) = \Phi(x_v).\Phi(x_w) \tag{2.17}$$

Dan prediksi pada set data dengan dimensi fitur baru yang diformulasikan dengan

$$f(z) = \sum_{v=1}^{N} \alpha_v y_w K(x_v, x_w) + b \tag{2.18}$$

N adalah jumlah data yang menjadi support vektor, x_v adalah *support vector*, x_w adalah data uji yang akan diprediksi kelasnya.

2.5.2 Metode *One Vs All*

Pada awal dikembangkannya, SVM digunakan untuk klasifikasi dua kelas saja (*binary class*), namun penelitian dikembangkan lebih lanjut untuk mengklasifikasikan data kedalam tiga kelas, empat kelas, lima kelas, dan lebih banyak lagi. Pada prinsipnya, SVM multi kelas dapat diimplementasikan dengan cara menggabungkan beberapa SVM biner [21].

Metode *One vs All* dilakukan dengan cara membandingkan satu kelas dengan semua kelas lainnya, sehingga dikenal sebagai *One Against theRest*. Untuk mengklasifikasikan data kedalam N *hyperplane*, sejumlah N model SVM biner haruslah dibangun. Setiap model SVM ke-m dilatih dahulu menggunakan keseluruhan data latih, kemudian untuk menjawab apakah sebuah data dapat diklasifikasikan sebagai kelas ke-m atau tidak.

Contoh dalam penggunaan metode ini, untuk mengklasifikasikan data kedalam 4 kelas, maka yang pertama harus dilakukan adalah membangun 4 buah SVM biner. SVM biner pertama dilatih menggunakan keseluruhan data latih untuk mengklasifikasikan data masuk kedalam kelas ke 1 atau bukan. SVM biner kedua dilatih menggunakan keseluruhan data latih untuk mengklasifikasikan data masuk kedalam kelas ke 2 atau bukan. Tahap ini dilakukan terus menerus sampai SVM ke empat dilatih menggunakan keseluruhan data untuk mengklasifikasikan data masuk kedalam kelas ke 4 atau bukan. Tahap pelatihan dengan 4 buah SVM biner dapat dilihat pada Tabel 2.2.

Tabel 2.2 Contoh SVM Biner dengan Metode *One vs All*

$y_i = 1$	$y_i = -1$	**Hipotesis *Hyperplane***
Kelas 1	Bukan kelas 1	$f(z_1) = \sum_{v=1}^{N} \alpha_v y_w K(x_v, x_1) + b$
Kelas 2	Bukan kelas 2	$f(z_2) = \sum_{v=1}^{N} \alpha_v y_w K(x_v, x_2) + b$
Kelas 3	Bukan kelas 3	$f(z_3) = \sum_{v=1}^{N} \alpha_v y_w K(x_v, x_3) + b$
Kelas 4	Bukan kelas 4	$f(z_4) = \sum_{v=1}^{N} \alpha_v y_w K(x_v, x_4) + b$

Dalam klasifikasi kasus multikelas, *hyperplane* yang terbentuk adalah lebih dari satu. Metode *One vs All* untuk kasus klasifikasi n-kelas, menemukan n *hyperplane* dimana n adalah banyak kelas dan $f(z)$ adalah *hyperplane*. Dalam metode ini $f(z_n)$ diujikan dengan semua data dari kelas n dengan label +1, dan semua data dari kelas lain dengan label -1.

Konsep pada *One vs All* dimisalkan pada kasus empat kelas diatas, yaitu kelas 1, 2, 3, dan 4. Bila akan diujikan $f(z_1)$, semua data dalam kelas 1 diberi label +1 dan data dari kelas 2, 3 dan 4 diberi label -1. Pada $f(z_2)$, semua data dalam kelas 2 diberi label +1 dan data dari kelas 1,3 dan 4 diberi label -1. Sementara untuk $f(z_3)$, semua data dalam kelas 3 diberi label +1 dan data dari kelas 1, 2 dan 4 diberi label -1. Begitu juga untuk $f(z_4)$, semua data dalam kelas 4 diberi label +1 dan data dari kelas 1, 2 dan 3 diberi label -1. Kemudian dicari *hyperplane* dengan algoritma SVM dua kelas. Maka akan didapat *hyperplane* untuk masing-masing kelas diatas. Kemudian kelas dari suatu data baru (z) ditentukan berdasarkan nilai terbesar dari *hyperplane* [22] menggunakan Persamaan 2.19.

$$\text{kelas} = \arg \max_{v=1,\dots N} \left(\sum_{v=1}^{N} \alpha_v y_v K(x_v, x_w) + b \right) \tag{2.19}$$

Pada penelitian ini akan menggunakan metode *One vs All* untuk klasifikasi *multiclass* SVM. Metode ini digunakan karena pada penelitian sebelumnya menghasilkan tingkat akurasi yang tinggi yaitu sebesar 99,75% [10]. Dengan menggunakan metode ini akan dibangun sebanyak n model SVM biner, dimana n merupakan banyaknya kelas. Banyak kelas dalam penelitian ini berjumlah sebanyak 80 kelas, diambil dari data latih yaitu sebanyak 80 jenis karakter.

2.5.3 *Sequential Minimal* Optimization (SMO)

Dalam penelitian ini, menggunakan SMO untuk menyelesaikan permasalahan dalam mencari nilai *lagrange multiplier* dengan kondisi KKT berikut.

$$\begin{aligned} \alpha_v = 0 \quad &\Leftrightarrow \quad y_v f(z) \geq 1 \\ 0 < \alpha_v < 0 \quad &\Leftrightarrow \quad y_v f(z) = 1 \\ \alpha_v = c \quad &\Leftrightarrow \quad y_v f(z) \leq 1 \end{aligned} \tag{2.20}$$

Parameter c berguna untuk mengontrol pertukaran antara magrin dan error klasifikasi. Semakin besar nilai c, semakin besar pula pelanggaran yang dikenakan untuk tiap klasifikasi. Langkah-langkah SVM menggunakan SMO yaitu :

a. inisialisasi nilai $\alpha_v = 0$, $\forall_v \in [1,2,3]$ sebanyak data latih yang akan digunakan, nilai b=0, nilai c = 1, dan toleransi (tol) = 0.00001, iterasi = 0, max_iterasi = 3, dan $\varepsilon = 10^{-5}$.
b. Melakukan perhitungan nilai eror (E_v) untuk data latih pertama (selanjutnya ditulis E_1). Nilai E_1 didapatkan dari fungsi prediksi Persamaan 2.18 yang dikurangkan dengan nilai y data latihnya.

$$E_v = f(z_v) - y_v \tag{2.21}$$

b. Lakukan pengecekan apakah nilai α_v sudah teroptimasi atau belum dengan nilai toleransi (tol) = 0.00001 dan c =1 berdasarkan persamaan berikut.

$$(y_1 * E_1 < -\text{tol} \textbf{ and } \alpha_v < c) \textbf{ or } (y_1 * E_1 > \text{tol} \textbf{ and } \alpha_v > 0) \tag{2.22}$$

c. Jika kondisi terpenuhi, maka selanjutnya memilih nilai alpha kedua (α_w) yang dilakukan secara acak. Kemudian selanjutnya akan dihitung kembali nilai eror untuk nilai alpha yang terpilih (E_w) yang dilakukan dengan cara seperti menghitung nilai E_1 sebelumnya, dengan cara mengurangkan fungsi prediksi Persamaan 2.18 dengan nilai y data latihnya. Baru kemudian mencari nilai E_w dengan menggunakan persamaan berikut.

$$E_w = f(z_w) - y_w \tag{2.23}$$

d. Setelah kedua nilai eror didapatkan, nilai α data latih pertama dan kedua yang sebelumnya di inisialisasi diawal akan disimpan, yang selanjutnya ditulis menjadi α_v^{old} dan α_w^{old}, dimana $\alpha_v^{old} = \alpha_v$ dan $\alpha_w^{old} = \alpha_w$.
e. Selanjutnya dicari nilai L dan H untuk menentukan *boundary* dari *alpha*. Nilai L dan H bisa didapatkan dengan persamaan berikut jika $y_v \neq y_w$.

$$L = \max(0, \alpha_w - \alpha_v) \tag{2.24}$$

$$H = \min(c, c + \alpha_w - \alpha_v) \tag{2.25}$$

Dan jika $y_v = y_w$, maka persamaan menjadi :

$$L = \max(0, \alpha_w + \alpha_v - c) \tag{2.26}$$

$$H = \min(c, \alpha_w - \alpha_v) \tag{2.27}$$

f. Jika nilai L = H, maka pelatihan diteruskan ke kelas selanjutnya. Namun, jika nilai nilai $L \neq H$. Maka selanjutnya mencari nilai η untuk mencari nilai alpha kedua yang baru (α_w^{new}) dengan menggunakan persamaan berikut.

$$\eta = 2K(x_v, x_w) - K(x_v, x_w) - K(x_v, x_w) \tag{2.28}$$

g. Jika nilai $\eta \geq 0$, teruskan pelatihan ke kelas selanjutnya. Namun jika nilai $\eta < 0$ lanjutkan perhitungan untuk mencari nilai α_w^{new} menggunakan persamaan berikut.

$$\alpha_w^{new} = \alpha_w - \frac{y_w(E_v - E_w)}{\eta} \tag{2.29}$$

h. Setelah nilai α_w^{new} dan *boundary* ditemukan, maka langkah selanjutnya adalah mengupdate α_w^{new} dengan aturan persamaan berikut.

$$\alpha_w^{new} = \begin{cases} H, \alpha_w^{new} > H \\ \alpha_w^{new}, L \leq \alpha_w^{new} \leq H \\ L, \alpha_w^{new} < H \end{cases} \tag{2.30}$$

i. Kemudian cek apakah hasil pengurangan nilai α_w^{new} dan α_w^{old} kurang dari nilai ε, dimana nilai $\varepsilon = 10^{-5}$. Jika ya lanjutkan pelatihan ke data latih berikutnya, jika tidak lanjutkan mencari nilai α_v^{new} dengan menggunakan persamaan berikut.

$$\left|\alpha_w^{new} - \alpha_w^{old}\right| < \varepsilon \tag{2.31}$$

Jika hasilnya *false*, maka perhitungan dilanjutkan dengan mencari nilai α_v^{new} (α_1^{new}) dengan menggunakan persamaan berikut.

$$\alpha_v^{new} = \alpha_v + y_v * y_w(\alpha_w^{old} - \alpha_w^{new}) \quad (2.32)$$

j. Setelah nilai α_v^{new} dan α_w^{new} didapat langkah selanjutnya adalah meng*update* nilai b, langkah pertama yang dilakukan yaitu mencari nilai $b\alpha_v$, $b\alpha_v$ merupakan nilai b terhadap α_v sehingga nilai disimpan dalam variabel yang disebut dengan $b\alpha_v$.

$$b_1 = b - E_v - y_v(\alpha_v^{new} - \alpha_v^{old})K(x_v, x_v) - y_w(\alpha_w^{new} - \alpha_w^{old})K(x_v, x_w) \quad (2.33)$$

k. Setelah nilai $b\alpha_v$ didapat, kemudian cari nilai b terhadap α_w (disimpan dengan variabel yang disebut $b\alpha_w$) menggunakan Persamaan 2.33 sebagai berikut.

$$b_2 = b - E_w - y_v(\alpha_v^{new} - \alpha_v^{old})K(x_v, x_w) - y_w(\alpha_w^{new} - \alpha_w^{old})K(x_w, x_w) \quad (2.34)$$

l. Setelah nilai $b\alpha_v$ dan $b\alpha_w$ didapatkan, langkah selanjutnya yaitu mencari nilai b berdasarkan persamaan berikut.

$$b = \begin{cases} b_1, 0 < \alpha_v^{new} < c \\ b_2, 0 < \alpha_w^{new} < c \\ \frac{b_1 + b_2}{2}, \text{otherwise} \end{cases} \quad (2.35)$$

m. Selanjutnya hitung num changed alphas dengan persamaan berikut.

$$\text{Num\_changed\_alphas} = \text{num\_change\_alpha} + 1 \quad (2.36)$$

n. Maka diperoleh hasil pelatihan pada iterasi pertama, lakukan pelatihan terhadap semua data latih. Nilai α akan terus berubah selama iterasi. Langkah tersebut akan diulang hingga semua nilai alpha teroptimasi atau sudah mencapai maksimal iterasi.

Tabel 2.3 Algoritma SMO

Sumber : The Simplified SMO Algorithm [23]

Algoritma :Simplified SMO
Input : **c** : parameter regularisasi
tol : toleransi numerik
max_pass : Maksimum iterasi
$[x_v, y_v], \ldots, [x_w, y_w]$: data latih
Output : α, b

```
1   α_v = 0, ∀_v
2   b = 0
3   pass = 0
4   while (pass < max_pass) do
5     num_changed_alphas = 0
6     for i = 1 to v begin
7        Hitung E_m := f(x_v) − y_v  (2.32)
8        if ((y_v × E_v < −tol and α_v < c) or (y_v × E_v > tol and α_v > 0)) then
9            pilih w ≠ v secara acak
10           Hitung E_w := f(x_w) − y_w
11           Simpan α_v^old := α_v, α_w^old := α_w
12           Hitung L dan H
13           if (L==H) then
14                continue
15           endif
16           Hitung η  (2.41)
19           if (η ≥ 0) then
20                continue
21           end if
22           Hitung dan simpan nilai α_v baru (2.40) dan (2.42)
23           if (|α_w − α_w^old| < 10^−5) then
24                continue
25           end if
26           hitung dan simpan nilai α_v baru (2.43)
27           hitung b1 dan b2 (2.44) dan (2.45)
28           hitung nilai b (2.46)
20           num_changed_alphas = num_changed_alphas +1
30       end if
31    end for
32    if (num_changed_alphas == 0) then
33        pass := pass +1
34    else
35        pass := 0
36    end if
37  end while
```

2.6 Pengujian Sistem

Pada penelitian ini, proses pengujian sistem terbagi menjadi dua, yaitu pengujian fungsionalitas sistem atau pengujian *black box* dan pengujian metode untuk mengukur tingkat akurasi yang dihasilkan metode dalam mengenali

karakter pada citra. Berikut ini masing-masing penjelasan dari kedua jenis pengujian.

2.7.1 Pengujian *Black Box*

Pengujian *black box* juga dikenal sebagai pengujian perilaku, berfokus pada persyaratan fungsional perangkat lunak. Artinya, teknik pengujian ini bertujuan untuk menemukan kesalahan fungsionalitas, kesalahan antarmuka, kesalahan dalam struktur data, kesalahan perilaku atau kinerja, dan kesalahan inisialisasi dan penghentian [12].

2.7.2 Pengujian Akurasi

Akurasi merupakan seberapa dekat suatu angka hasil pengukuran terhadap angka sebenarnya (*true value* dan *reference value*). Tingkat akurasi diperoleh dengan persamaan sebagai berikut [5].

$$\text{Akurasi} = \frac{\text{jumlah karakter sama}}{\text{jumlah seluruh karakter}} \times 100\% \tag{2.37}$$

2.8 Bahasa Pemrograman *Python*

Python adalah Bahasa pemrograman bersifat umum. Python diciptakan pada tahun 1990 oleh Guido van Rossum. Bahasa level tinggi, stabil, dinamis, orientasi objek dan cross *platform* adalah karakteristik yang membuat Bahasa python disukai oleh banyak pengembang. Tidak seperti bahasa lain yang susah untuk dibaca dan dipahami, python lebih menekankan pada keterbacaan kode agar lebih mudah untuk memahami sintaks. Bahasa pemrograman Python saat ini dikembangkan dan dikelola oleh suatu tim relawan dengan nama *Python Software Foundation* [24]. Pada penelitian ini menggunakan Bahasa pemrograman python untuk membangun perangkat lunak.

2.8.1 Jupyter Notebook

Jupyter Notebook (file yang berekstensi *ipynb*) adalah dokumen yang dihasilkan oleh Jupyter Notebook App yang berisikan kode komputer dan *rich text element* seperti paragraf, persamaan matematik, gambar dan tautan (*links*).

Jupyter Notebook dikenal sebelumnya sebagai *IPython Notebook* dan dalam waktu dekat akan berevolusi menjadi *Jupyter Lab* [25]. Pada penelitian ini jupyter notebook digunakan untuk membangun rancangan code program berbentuk *console*.

2.8.2 Jetbrains Pycharm

IDE lain yang cukup populer dikalangan developer Python adalah PyCharm. PyCharm sendiri memiliki dua versi yaitu Professional Edition dan Community Edition. PyCharm Professional Edition merupakan versi berbayar dari PyCharm dan Community Edition merupakan versi gratis yang tersedia bagi komunitas python dengan lisensi Apache 2. *Pycharm* digunakan sebagai *tool* dalam membangun aplikasi dalam penelitian ini..

2.8.3 Flask

Flask adalah *microframework* untuk Python yang dibuat dengan *toolkit* wsgi dan Jinja2. Flask dibuat dan dimaintain oleh Armin Ronacher. Pertama kali dirilis 4 tahun yang lalu (April 2010), Flask saat ini berada di versi 0.10.1 dan dilisensikan dengan BSD license. Aplikasi web yang dibuat dengan Flask disimpan dalam satu berkas .py. Flask ingin menjadi *web framework* yang sederhana namun dapat diperluas dengan beragam pustaka tambahan yang sesuai dengan kebutuhan penggunanya. Pada penelitian ini flask digunakan untuk membangun aplikasi web dan menyatukannya dengan source code program yang sudah dibuat di *pycharm*.

2.9 *Unified Modeling Language (UML)*

Unified Modeling Language (UML) adalah bahasa pemodelan untuk menspesifikasikan, memvisualisasikan, membangun dan mendokumentasikan artifak-artifak dari sistem. UML digunakan dalam pengembangan sistem perangkat lunak yang menggunakan pendekatan berorientasi objek [26]. Adapun yang menjadi pegangan dalam pembuatan UML ini berdasarkan *Ebook* oleh Rules Miles [27].

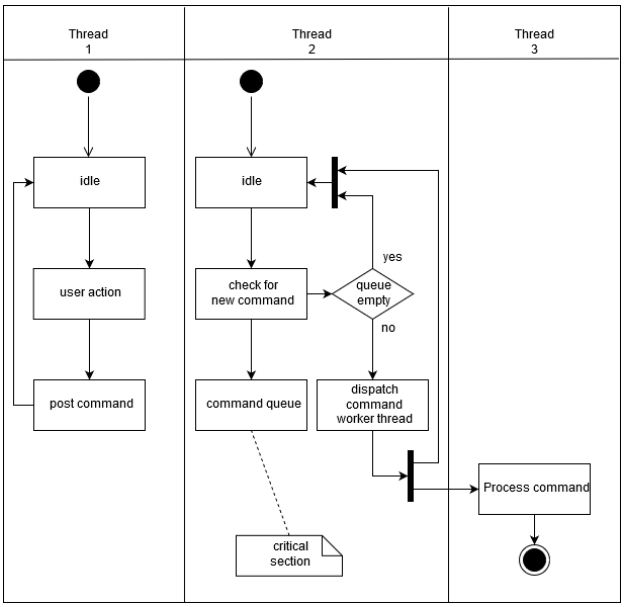
2.9.1 *Use Case Scenario*

Use-case scenario dijelaskan secara tekstual dalam beberapa format tergantung kebutuhannya, yaitu singkat (*brief*), *informal* (*casual*), atau lengkap (*fully dressed*), yang dapat dijelaskan dalam satu atau dua kolom [26]. Pada penelitian ini digunakan format yang diperkenalkan oleh [27] dengan bagian yang terlibat sebagai berikut.

1. Nama *Use Case* (*Use case name*)
2. Persyaratan Terkait (*Related Requirements*), kondisi spesifik yang harus terpenuhi sebelum *use-case* dieksekusi oleh aktor.
3. Tujuan (*Goal in Context*), penggunaan *use case* dan menjelaskan mengapa *use case* ini menjadi penting digunakan.
4. Kondisi Awal (*Preconditions*), kondisi awal sebelum *use case* dieksekusi.
5. Kondisi Akhir Berhasil (*Successful End Condition*), kondisi ketika *use case* berhasil dieksekusi.
6. Kondisi Akhir Gagal (*Failed End Condition*), kondisi ketika *use case* gagal dieksekusi.
7. Aktor Utama (*Primary Actors*), Aktor utama yang menggunakan *use case*.
8. Aktor Sekunder (*Secondary Actors*), Aktor sekunder/lainnya yang menggunakan *use case*.
9. Pemicu (*Trigger*), pemicu oleh aktor sehingga *use case* dieksekusi.
10. Alur Utama (*Main Flow*), penjelasan terkait alur utama dari *use case* apabila berjalan secara normal.
11. Ekstensi (*Ekstensions*), yaitu jalur alternatif dari interaksi yang terjadi antara aktor dan sistem yang mencakup percabangan (pilihan) maupun skenario yang gagal sehingga tujuan aktor tidak terpenuhi.

2.9.2 *Activity Diagram*

Activity diagram adalah diagram *flowchart* yang diperluas yang menunjukkan aliran kendali satu aktivitas ke aktivitas lain berdasarkan use case diagram [26]. Berikut adalah contoh dari *activity diagram* pada Gambar 2.5.



Gambar 2.5 Contoh *Activity Diagram*

2.9.3 *Class Diagram*

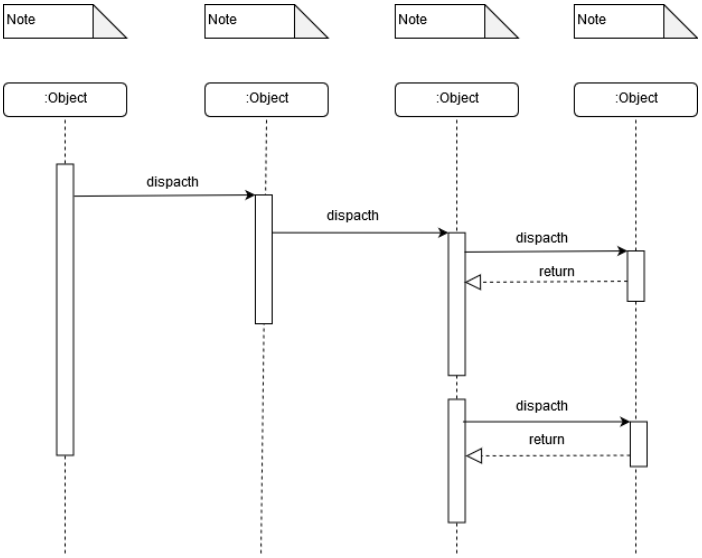
Class diagram merupakan inti dari setiap sistem berorientasi objek, oleh karena itu kaitannya sangat erat dengan *use-case diagram*. *Class diagram* bertujuan untuk menjelaskan berbagai jenis objek dan hubungannya dengan kelas lainnya yang dimiliki oleh sistem. Hal ini lah yang menjadikan *class diagram* memiliki dua macam informasi, yaitu informasi tentang kondisi awal objek dan bagaimana berperilaku dalam lingkungannya [26].

2.9.4 *Sequence Diagram*

Sequence diagram atau dikenal juga sebagai diagram interaksi bertujuan untuk memodelkan interaksi secara *runtime* antara bagian – bagian tertentu pada sistem yang membentuk alur perpindahan sebuah objek [27].

Diagram ini akan meyampaikan urutan dari setiap interaksi pada suatu proses atau bagian – bagian sistem. Dengan menampilkan apa yang menjadi pemicu dari sebuah proses dan menunjukkan informasi lain berupa peristiwa

dalam suatu interaksi [27]. Berikut adalah contoh dari sequence diagram pada gambar 2.6.



Gambar 2.5 Contoh *Sequence Diagram*