BAB 2

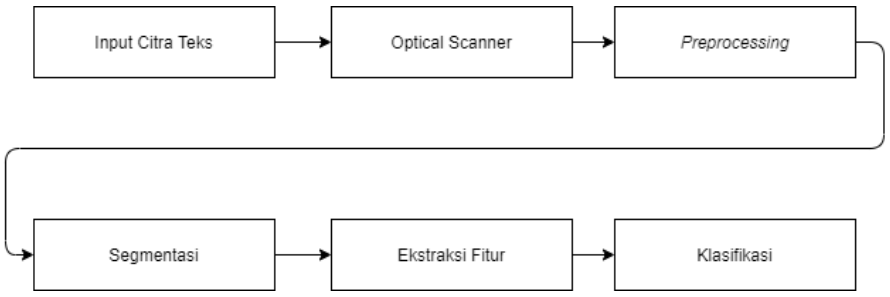
TINJAUAN PUSTAKA

2.1 *Optical Character Recognition* (OCR)

OCR adalah singkatan dari *Optical Character Recognition*. Teknologi ini memungkinkan untuk mengenali karakter secara otomatis melalui mekanisme optik. Dalam kasus manusia, mata manusia adalah mekanisme optik. Otak manusia mendapatkan input yaitu gambar yang dilihat oleh mata, OCR adalah teknologi yang berfungsi seperti kemampuan membaca manusia. Meskipun OCR tidak mampu bersaing dengan kemampuan membaca manusia [13].

OCR merupakan bagian dari perangkat lunak yang mengubah teks dan gambar tercetak menjadi bentuk digital sedemikian rupa sehingga dapat dimanipulasi oleh mesin. Adapun pengertian lain OCR adalah teknologi yang memungkinkan mengonversi berbagai jenis dokumen seperti dokumen kertas yang dipindai, file PDF atau gambar yang diambil oleh kamera digital menjadi data yang dapat diedit dan dicari. Gambar yang diambil oleh kamera digital berbeda dari dokumen atau gambar yang dipindai.

Alur dari sistem OCR biasanya terdapat input, proses, dan output. Input pada OCR biasanya berupa dokumen digital yang didalamnya terdapat unsur-unsur karakter. Selanjutnya kedalam proses, kemudian pada proses biasanya terdapat *preprocessing*, ekstraksi fitur, dan masuk pada tahap klasifikasi. Pada tahap klasifikasi ini untuk menentukan bahwa objek tersebut masuk kedalam salah satu kelas yang telah ditentukan. Hasil keluaran dari klasifikasi adalah huruf yang sesuai dengan ciri dari kelas tersebut. Berikut adalah diagram blok dari system OCR dapat dilihat pada gambar 2.1



Gambar 2.1 Diagram Blok OCR

2.2 *Preprocessing*

Perancangan sistem *preprocessing* menjelaskan alur kerja dalam OCR dalam menyiapkan citra sebagai data input yang siap diolah lebih lanjut oleh sistem. Sistem *preprocessing* yang digunakan pada penelitian ini adalah sebagai berikut :

2.2.1 *Grayscale*

Grayscale atau citra yang hanya memiliki satu buah *channel* sehingga yang ditampilkan hanyalah nilai intensitas atau derajat keabuan [14]. Dalam hal ini,intensitas berkisar antara 0 sampai 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih. Ada beberapa cara untuk mengkonversi citra RGB ke citra *grayscale*. Namun diambil cara mengkonversi yang menghasilkan konversi yang lebih baik dari beberapa cara yang lain yaitu dengan menggunakan persamaan 2.1 sebagai berikut.

$$y = 0{,}2989 * R + 0{,}5870 * G + 0{,}1141 * B \quad (2.1)$$

Berikut contoh citra *grayscale* ditujukan pada gambar 2.2

Gambar 2.2 Contoh Citra *Grayscale*

2.2.2 *Thresholding*

Thresholding merupakan suatu proses dimana hasilnya berupa citra biner dari citra *grayscale* atau citra berwarna dengan mengatur nilai piksel ke nilai 0 atau 1 tergantung dari nilai ambang batasnya apakah nilai piksel tersebut berada dibawah atau diatas ambang batas [15]. *Thresholding* terbagi menjadi dua yaitu *global thresholding* dan *local thresholding*. *Global Thresholding* merupakan

thresholding yang apabila nilai ambang t bergantung hanya pada satu nilai atas keabuan f(x,y). Sedangkan *local thresholding* adalah *thresholding* yang dimana nilai ambang t bergantung pada f(x,y) dan g(y,x) menyatakan properti citra local pada titik (y,x). Proses *thresholding* digunakan untuk mengekstrak *foreground* (tinta) dari background (kertas) dan mengubah menjadi citra biner. Proses thresholding mengubah warna gambar menjadi citra biner (binary image) dimana ditentukan sebuah nilai level threshold kemudian piksel yang memiliki nilai level dibawah level threshold di set menjadi warna putih (0 pada nilai biner) dan nilai diatas nilai threshold di set menjadi warna hitam (1 pada nilai biner). Pada operasi pengambang, nilai intensitas piksel dipetakan ke salah satu dari dua nilai a_1 dan a_2, berdasarkan nilai ambang (threshold) T seperti pada persamaan 2.2:

$$f(x,y) = \begin{cases} a_1 \; f(x,y) < T \\ a_2 \; f(x,y) \geq T \end{cases} \tag{2.2}$$

Jika $a_1 = 0$ dan $a_2 = 1$, maka operasi pengembangan mentransformasikan citra hitam-putih ke citra biner. Dengan kata lain, nilai intensitas piksel semula dipetakan ke dua nilai saja yaitu hitam dan putih. Nilai ambang yang dipakai dapat berlaku untuk keseluruhan piksel atau untuk wilayah tertentu saja (berdasarkan penyebaran nilai intensitas pada wilayah tersebut).

2.2.3 Segmentasi (*Maximally Stable Extremal Regions*)

Segmentasi merupakan suatu proses generik dimana gambar menjadi beberapa wilayah/daerah (region) [15]. Pada pengolahan citra, peran segmentasi sangatlah penting sebelum memasuki proses-proses seperti klasifikasi, ekstraksi fitur, dan lain-lain. Proses segmentasi bertujuan untuk membagi citra ke dalam basis elemen sesuai dengan kriteria yang ditentukan. Sehingga citra yang diambil atau yang digunakan hanya yang penting saja. Metode yang digunakan dalam proses segmentasi yaitu metode *Maximally Stable Extremal regions*.

Maximally stable extremal regions (MSER) secara dasar adalah *regions detector*. Metode ini adalah salah satu metode yang populer digunakan dengan pendekatan *Connected Component-Based* [16]. Metode ini menerapkan prinsip *watershed segmentation* untuk mendeteksi calon objek yang akan dikenali [17]. Keuntungan dari metode ini adalah tahan terhadap penskalaan dan rotasi.

Keuntungan tersebut cocok untuk *text scene* dan multi orientasi. Metode ini dalam melakukan segmentasi yaitu dengan menentukan area yang memiliki keterkaitan komponen atau komponen terhubung yang stabil dari beberapa kumpulan gambar tingkat abu-abu atau *grayscale* [18]. Untuk menentukan komponen terhubung apa yang stabil yaitu mudahnya digambarkan dengan cara membuat pohon komponen dimana pada pembentukan pohon komponen terdapat *parent* dan *child*, *parent* merupakan posisi paling atas dan *child* adalah anak dari *parent* artinya berada di bawahnya dan tiap tingkatan pohon komponen merupakan level batas ambangnya [19]. Dasar perhitungan MSER dimulai dengan melakukan pemilihan atau sortir urutan piksel-piksel dari intensitas rendah ke intensitas tinggi atau sebaliknya (misal pada citra *grayscale* yang mempunyai intensitas {0,…,255}). Intensitas ini yang akan dinamakan *threshold*. Iterasi dimulai dari *threshold* rendah (0) ke *threshold* tinggi (255) dan pada masing-masing *threshold* diubah-ubah dinamakan MSER regions. MSER banyak digunakan pada aplikasi *text localization* dan *recognition*. Dalam setiap *threshold* citra, area *extremal* ditandai sebagai komponen yang berhubungan, sehingga terbentuk sebuah urutan dari masing-masing komponen.Di beberapa *threshold* dua atau lebih komponen akan bergabung menjadi satu. Pada *threshold* ini terjadi perubahan bentuk komponen secara signifikan, hal ini menjadikan komponen tidak stabil dan letak lokasi menjadi menjadi tidak presisi, terutama dalam hal perubahan intensitas dan *noise*. *Maximally Stable Extremal Regions* akan diekstrak dari urutan komponen pada *threshold* yang dipresentasikan oleh perbedaan minimum komponen lokal dalam area di dalam ruang *threshold*. Untuk melakukan proses segmentasi *Maximally Stable Extremal Regions* yaitu harus dilakukan inisialisasi parameter. Berikut inisialisasi parameter dan penjelasannya pada MSER :

1. Delta

   Delta merupakan parameter yang digunakan untuk menentukan berapa banyak iterasi yang diproses selama thresholding.
2. Max Area

Max Area merupakan parameter yang digunakan untuk menentukan ukuran maksimum dari setiap karakter yang telah dipisahkan dengan pendekatan Connected Component Labeling.

3. Min Area

   Min Area merupakan parameter yang digunakan untuk menentukan ukuran minimum dari setiap karakter yang telah dipisahkan dengan pendekatan Connected Component Labeling.

4. Max Variation

   Max Variation merupakan parameter yang digunakan untuk menentukan perbandingan nilai variasi pada setiap karakter yang sudah dipisahkan dengan pendekatan Connected Component Labeling dan dianggap stabil ketika nilai variasi lebih kecil dari Max Variation.

Algoritma *Maximally Stable Extremal regions* (MSER) terdiri dari beberapa tahapan utama yang dapat dijelaskan sebagai berikut[12] :

1. Thresholding citra grayscale dengan cara iterasi/*looping* dari intensitas 0-255 atau nilai intensitas yang sudah diinisialisasi. Pada setiap iterasi dilakukan pemisahan objek dengan pendekatan Connected Component Labeling dan buat parent-child structure atau component tree sehingga tiap objek atau karakter akan terpisah menjadi extremal regions (kumpulan titik yang saling terkait antara titik satu dengan titik yang lainnya).

   Berikut citra *grayscale* sebagai citra inputan dari proses MSER :

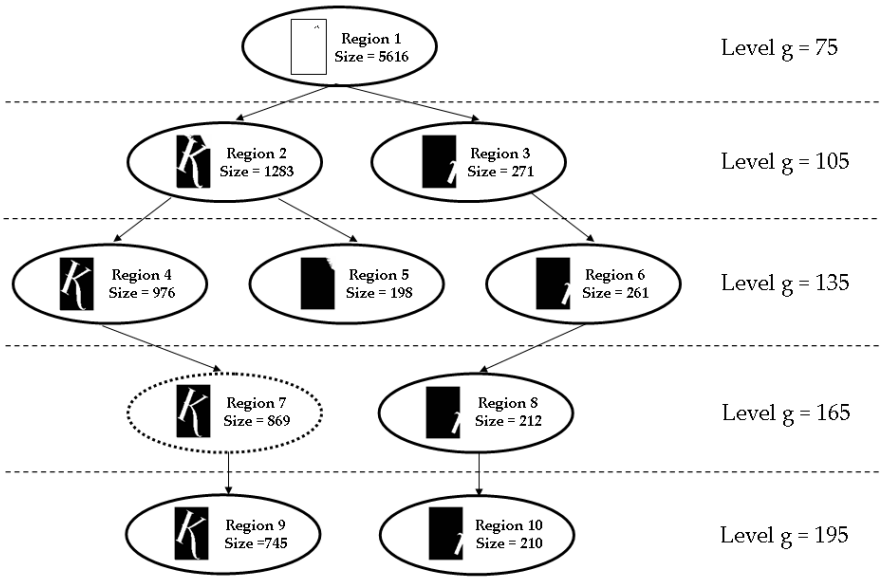
Gambar 2.3 Citra Grayscale Sebagai Input

Berikut ini adalah contoh looping pada proses MSER dengan delta = 30 dan threshold awal = 75 :

Tabel 2.1 Tabel Iterasi Thresholding

No	Level Threshold	Citra Iterasi/*looping*
1	75	
2	105	
3	135	
4	165	
5	195	
6	225	
7	255	

2. Hitung size dan nilai variasi pada setiap karakter atau region dengan persamaan 2.3 yaitu $var_i = \frac{|Area_i| - |Area_{i-\Delta}|}{|Area_i|}$. (2.3)



Gambar 2.4 Gambar Pohon Komponen MSER

Dari gambar 2.4 menunjukan pembuatan pohon komponen untuk menggambarkan pada tabel 2.1.

3. Kemudian objek tersebut diseleksi apakah teks atau bukan teks dengan 3 aturan, yaitu aturan MaxArea, MinArea dan MaxVariation dengan parameter yang sudah diinisialisasi.
4. Setelah diseleksi maka akan diambil nilai variasi yang lebih rendah pada tiap karakter untuk mendapatkan hasil segmentasi.
5. Hasil segmentasi didapatkan dari nilai variasi yang paling rendah. Region 7 yang teridentifikasi sebagai MSER, maka region 7 diambil sebagai hasil segmentasi

2.2.4 *Resize*

Resize dilakukan untuk menyamakan setiap ukuran citra karakter hasil segmentasi karena tidak semuanya citra mempunyai tinggi dan lebar yang sama.

Metode yang digunakan adalah bukan metode khusus melainkan menggunakan perbandingan ukuran dari setiap citra hasil segmentasi dengan ukuran citra yang diinginkan.

Berikut ini adalah rumus yang digunakan dalam proses *resize*:

Menghitung nilai koordinat baris :

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

Keterangan :

x = Nilai piksel baris baru

pb = Ukuran panjang matriks baru

pp = Posisi piksel baris

pa = Ukuran panjang matriks lama

Menghitung nilai koordinat kolom :

$$y = \frac{lb * lp}{la} \tag{2.5}$$

Keterangan :

y = Nilai piksel kolom baru

lb = Ukuran lebar matriks baru

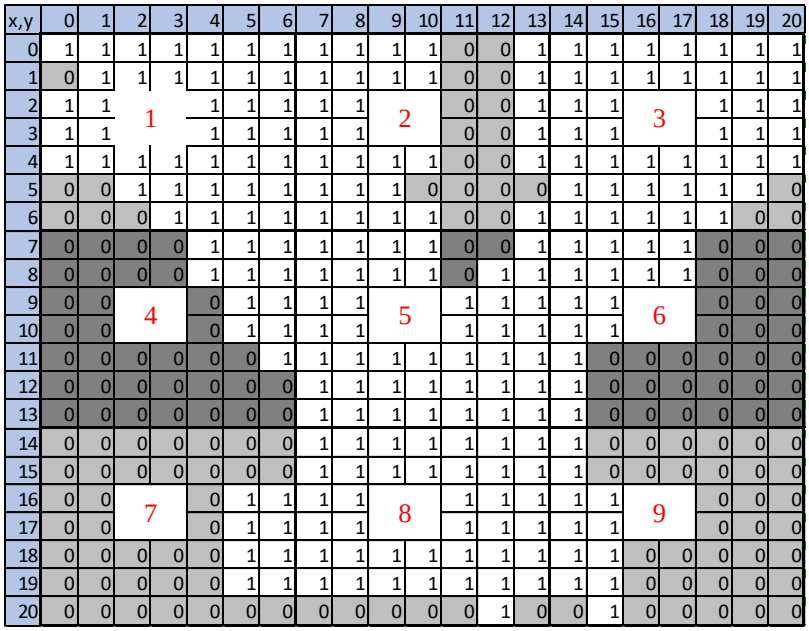
lp = Posisi piksel kolom

la = Ukuran lebar matriks lama

2.3 Ekstraksi Fitur

Ekstraksi fitur merupakan bagian fundamental dari analisis citra digital yaitu suatu proses pengukuran terhadap data yang telah dinormalisasi untuk membuat nilai fitur. Fitur adalah karakteristik unik dari suatu objek pada citra digital [20]. Kemudian nilai fitur ini digunakan oleh pengklasifikasian untuk mengenali unit masukan dengan unit target keluaran dan memudahkan pengklasifikasian karena nilai ini mudah untuk dibedakan. Pada penelitian ini, metode ekstraksi fitur yang digunakan adalah metode *zoning*.

Zoning merupakan salah satu ekstraksi fitur yang populer dan sederhana untuk diimplementasikan [21]. Setiap citra dibagi menjadi N x M zona dan dari setiap zona tersebut dihitung nilai fitur sehingga didapatkan fitur dengan Panjang N x M. salah satu cara untuk menghitung nilai fitur setiap zona adalah dengan menghitung jumlah piksel hitam setiap zona dan membaginya dengan jumlah piksel hitam terbanyak yang terdapat pada salah satu zona. Contoh pembagian 9 zona pada citra biner dapat dilihat pada gambar sebagai berikut :



Gambar 2.5 Pembagian Zona Pada Citra Biner

Ada beberapa algoritma untuk metode ekstraksi ciri *zoning*, diantaranya metode ekstraksi ciri jarak metrik ICZ (*image centroid and 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. Dalam penelitian ini konsep metode ekstraksi fitur yang digunakan adalah metode *image centroid zone* (ICZ)[22]. Berikut merupakan tahapan dalam proses ekstraksi ciri ICZ :

1. Hitung *centroid* dari citra dengan persamaan berikut.

$$x_c = \frac{(x_1.p_1 + x_2.p_2 + \cdots + x_n.p_n)}{(p_1 + p_2 + \cdots + p_n)} \quad (2.6)$$

$$y_c = \frac{(y_1.p_1+y_2.p_2+\cdots+y_n.p_n)}{(p_1+p_2+\cdots+p_n)} \quad (2.7)$$

Keterangan:

x_c = *centroid* koordinat x

y_c = *centroid* koordinat y

x_n = koordinat x dari piksel ke-n

y_n = koordinat y dari piksel ke-n

p_n = nilai piksel ke-n

2. Bagi kedalam *n* buah zona yang sama besar sesuai dengan ukuran citra.
3. Hitung jarak antara titik *centroid* dengan koordinat piksel yang memiliki nilai 1 dengan persamaan berikut.

$$d(p,c)\sqrt{(x_p - x_c)^2 + (y_p - y_c)^2} \quad (2.7)$$

Keterangan :

d = jarak antara dua titik

P = koordinat titik berat

C = koordinat piksel

x_p = koordinat x titik berat

y_p = koordinat y titik berat

x_c = koordinat x piksel

y_c = koordinat y piksel

4. Ulangi langkah nomor 3 untuk piksel yang ada di semua zona.
5. Hitung rata-rata dari jarak yang telah didapatkan pada langkah 3 dengan persamaan berikut.

$$\text{Rataan jarak} = \frac{\sum_{i=0,y=0}^{n}(jarak(0,0)+jarak(0,1)+\cdots+jarak(n,n))}{Banyak\ titik} \quad (2.8)$$

6. Ulangi langkah nomor 5 sampai mendapatkan nilai dari masing-masing rata-rata jarak pada setiap zona.

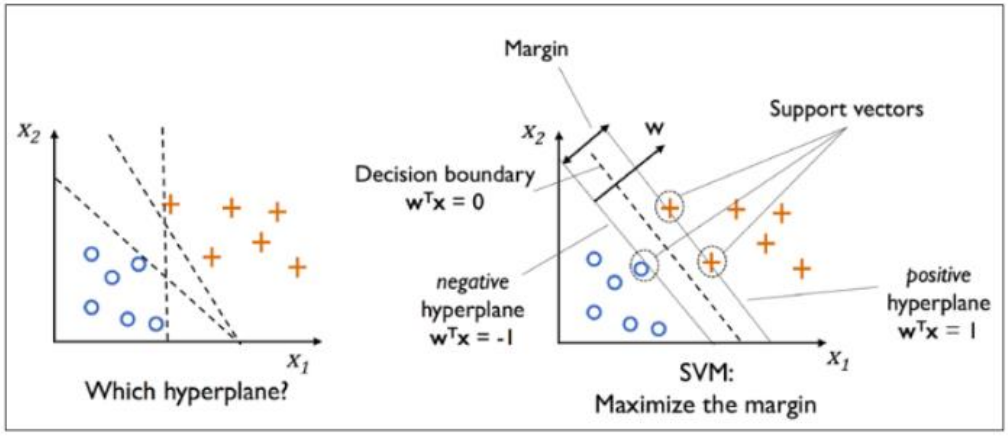
7. Terakhir akan mendapatkan *n* buah nilai fitur akan didapat untuk melakukan klasifikasi dan pengenalan

2.4 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 local. SVM memiliki konsep yang jauh lebih matang, lebih jelas secara matematis, dibanding teknik-teknik klasifikasi sebelum era 1990-an[23]. SVM menemukan *hyperplane* dengan memaksimalkan jarak antar kelas (margin) seperti diilustrasikan pada Gambar 2.6. *Hyperplane* adalah garis batas pemisah data antar kelas, sedangkan margin adalah jarak antara *hyperplane* dengan data terdekat masing-masing kelas [24].

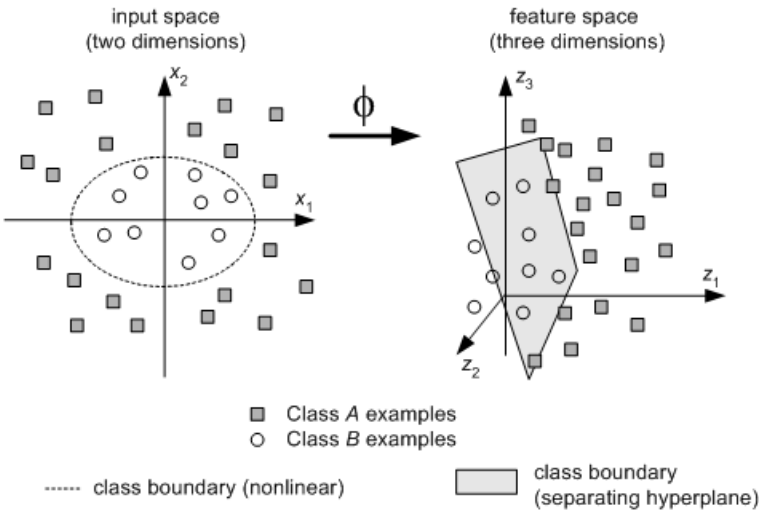
SVM hanya menemukan satu *hyperplane* yang posisinya tepat di tengah-tengah antara dua kelas. Jadi, *hyperplane* tersebut membelah himpunan data menjadi dua kelas secara sama. SVM memaksimalkan batas *hyperplane*, dalam Teknik ini SVM berusaha menemukan pemisah (*classifier*) yang optimal sehingga untuk pola yang tidak terlihat diminimalkan dan memisahkan dua set data dari dua kelas yang berbeda. Adapun data terdekat dengan *hyperplane* pada masing-masing kelas inilah yang disebut *support vector* [25]. Objek-objek data data yang disebut *support vector* ini paling sulit diklasifikasikan karena posisinya hampir tumpang tindih dengan kelas lain. Mengingat sifatnya yang kritis, hanya *support vector* ni yang diperhitungkan oleh SVM untuk menemukan *hyperplane* paling optimal. Berikut ilustrasi dari SVM yang dapat dilihat pada gambar di bawah ini :

.



Gambar 2.6 SVM berusaha untuk menemukan hyperplane terbaik untuk memisahkan dua kelas -1 dan +1

Pada gambar 2.6 memperlihatkan beberapa *pattern* yang merupakan anggota dari dua buah kelas yang berbeda, yaitu -1 dan +1. *Pattern* yang tergabung dalam kelas -1 disimbolkan dengan warna merah (kotak), sedangkan *pattern* tergabung dalam kelas +1 disimbolkan dengan warna kuning (bulat). Pada dasarnya SVM adalah sebuah *linear classifier* (Kumar 2009). Namun, SVM dapat menggunakan konsep *kernel trick* pada ruang berdimensi tinggi. Ide dasarnya adalah ketika terdapat himpunan data satu dimensi yang terpisah secara nonlinier sehingga memerlukan *hyperplane* H berbentuk lingkaran. Dengan sebuah fungsi $\varphi : R^2 \rightarrow R^3$ yang berupa $(x_{1,}x_2) \rightarrow (z_1, z_2, z_3) \coloneqq (x_1^2, \sqrt{(2)}x_1x_2, x_1^2)$, himpunan data dua dimensi tersebut dapat dikonversikan menjadi himpunan data tiga dimensi yang dapat dipisahkan secara linier dengan sebuah *hyperplane* berbentuk bidang datar [23]. Berikut contoh himpunan data tiga dimensi pada gambar 2. :



Gambar 2.7 Pada gambar sebelah kiri himpunan data dua dimensi yang terpisah secara non linier, gambar sebelah kanan konversi dari dua dimensi menjadi himpunan data tiga dimensi yang linier

Jadi, secara umum fungsi untuk mengkonversi himpunan data pada ruang masukan (*input space*) ke dalam ruang fitur (*feature space*) yang berdimensi lebih tinggi dapat diformulasikan sebagai

$$\varphi : R^p \rightarrow R^q, \text{ dimana } p < q \tag{2.9}$$

Lalu konsep diatas dapat diformulasikan, misalkan data yang terdapat pada himpunan data latih dinotasikan sebagai $x_i \in R^d$ sedangkan label kelas dinyatakan sebagai $y_i \in \{-1, +1\}$ untuk i – 1, 2, …, l, dimana l adalah jumlah data. Langkah pertama dalam pelatihan adalah kernelisasi vektor masukan. Kernel yang digunakan bisa berbagai macam, pada penelitian ini kernel yang digunakan adalah kernel *Radial Basis Function* (RBF) :

$$K(x_i, x_j) = exp\ exp\ (-\gamma||x_i - x_2||^2)\ , \gamma > 0 \tag{2.10}$$

Kernel RBF digunakan karena kernel RBF menghasilkan tingkat akurasi yang lebih tinggi dalam kasus pengenalan karakter. Karena data yang digunakan tidak bisa dipisahkan secara linier dan kelas yang digunakan sebanyak 62 kelas. Maka pada penelitian ini menggunakan SVM non linear dengan pendekatan *one vs all*.

2.4.1 SVM Non linear

SVM sebenarnya adalah *hyperplane* 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:

$$\Phi: D_q \rightarrow D_r \, , x \rightarrow \Phi(x) \tag{2.11}$$

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, … xn ∈ 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.

$$(\Phi(x_1), y_1, \Phi(x_2), y_2, \ldots \Phi(x_n), y_n) \in D\ r \tag{2.12}$$

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_i) dan (x_j) dinotasikan sebagai $\Phi(x_i).\Phi(x_j)$. 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.

$$K(x_i, x_j) = \Phi(x_i).\Phi(x_j) \tag{2.13}$$

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

$$f(x) = \sum_{i=1}^{n} a_i, y_i K(x_i, x_j) + b \tag{2.14}$$

N adalah jumlah data yang menjadi support vektor, x_i adalah support vector, x_y adalah data uji yang akan diprediksi kelasnya.

2.4.2 Metode *One Vs All*

Pada awalnya dikembangkannya, SVM digunakan untuk klasifikasi dua kelas saja, namun penelitian dikembangkan lebih lanjut untuk mengklasifikasi 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.

Metode *One Against All* membandingkan satu kelas dengan semua kelas lainnya. Untuk mengklasifikasikan data ke dalam *k* kelas, kemudian harus dibangun sejumlah *k* model SVM biner. Setiap model SVM biner ke-*i* dilatih menggunakan keseluruhan data, untuk mencari jawaban apakah semua data diklasifikasikan sebagai kelas ke-*i* atau tidak. Misalkan masalah klasifikasi pada 4 kelas, maka dibangun 4 SVM biner, di mana SVM biner pertama dilatih menggunakan semua data latih untuk mengklasifikasikan data ke dalam kelas C_1 atau bukan, SVM biner kedua dilatih menggunakan semua data latih untuk mengklasifikasikan data ke dalam kelas C_2 atau bukan, dan seterusnya[23]. Berikut ilustrasi 4 buah SVM biner seperti pada tabel 2.2.

Tabel 2.2 Contoh 4 SVM biner dengan Metode *One Against All*

$y_i = 1$	$y_i = -1$	Hipotesis
Kelas 1	Bukan kelas 1	$f(x_1) = \sum_{i=0}^{n} a_i y_i \ K(x_i, x_1) + b$
Kelas 2	Bukan kelas 2	$f(x_2) = \sum_{i=0}^{n} a_i y_i \ K(x_i, x_2) + b$
Kelas 3	Bukan kelas 3	$f(x_3) = \sum_{i=0}^{n} a_i y_i \ K(x_i, x_3) + b$

Kelas 4	Bukan kelas 4	$f(x_4) = \sum_{i=0}^{n} a_i y_i \ K(x_i, x_4) + b$

Konsep pada OAA yaitu dimisalkan pada kasus empat kelas, kelas 1, 2, 3, dan 4. Bila akan diujikan $\rho^{(1)}$, semua data dalam kelas 1 diberi label +1 dan data dari kelas lainnya diberi label -1. Pada $\rho^{(2)}$, semua data dalam kelas 2 diberi label +1 dan data dari kelas lainnya diberi label -1 dst hingga data terakhir. Kemudian dicari hyperplane dengan algoritma SVM dua kelas. Maka akan didapat hyperplane untuk masing-masing kelas diatas. Kemudian kelas dari suatu data barus x ditentukan berdasarkan nilai terbesar dari hyperplane:

$$kelas\ x = \arg \max_{i=1,\dots N} \left(\sum_{i=1}^{N} \alpha_i \, y_i K(x_i, x_j) + b \right) \tag{2.15}$$

2.4.3 *Sequential Minimal Optimization Training*

Sequential Minimal Optimization (SMO) adalah proses algoritma untuk proses pelatihan SVM yang dapat memberikan solusi pada masalah optimasi. Pada dasarnya penggunaan SVM hanya terbatas pada masalah yang kecil karena algoritma pelatihan SVM cenderung lambat, kompleks, dan sulit untuk diimplementasikan. Berdasarkan hasil dari penelitian sebelumnya, algoritma SMO lebih sederhana, lebih mudah diimplementasikan, dan lebih cepat waktu komputasinya daripada algoritma *Chunking* (Platt 1998)[27].

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

$$\begin{aligned} \alpha_i = 0 \quad &\Leftrightarrow \ y_i f(x) \geq 1 \\ 0 < \alpha_i < c \ &\Leftrightarrow \ y_i f(x) = 1 \\ \alpha_i = c \quad &\Leftrightarrow \ y_i f(x) \leq 1 \end{aligned} \tag{2.16}$$

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

1. Inisialisasi nilai $\alpha_i = 0$, $\forall_i \in [1,2,3]$ sebanyak data latih yang akan digunakan, nilai b=0, nilai c = 1, dan toleransi (tol) = 0.00001, iterasi = 0, max_iterasi = 3, dan $\varepsilon = 10^{-5}$.
2. Melakukan perhitungan nilai error (E_i) 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.
$$E_i = f(z_i) - y_i \tag{2.17}$$
3. Lakukan pengecekan apakah nilai α_v sudah teroptimasi atau belum dengan nilai toleransi (tol) = 0.00001 dan c =1 berdasarkan persamaan berikut.
$$(y_1 * E_1 < -\text{tol } \textbf{and } \alpha_i < c) \textbf{ or } (y_1 * E_1 > \text{tol } \textbf{and } \alpha_i > 0) \tag{2.18}$$
4. Jika kondisi terpenuhi, maka selanjutnya memilih nilai alpha kedua (α_j) yang dilakukan secara acak. Kemudian selanjutnya akan dihitung kembali nilai error untuk nilai alpha yang terpilih (E_j) 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_j dengan menggunakan persamaan berikut.
$$E_j = f(z_j) - y_j \tag{2.19}$$
5. Setelah kedua nilai error didapatkan, nilai α data latih pertama dan kedua yang sebelumnya di inisialisasi di awal akan disimpan, yang selanjutnya ditulis menjadi α_i^{old} dan α_j^{old}, dimana $\alpha_i^{old} = \alpha_i$ dan $\alpha_j^{old} = \alpha_j$.
6. Selanjutnya dicari nilai L dan H untuk menentukan *boundary* dari *alpha*. Nilai L dan H bisa didapatkan dengan persamaan berikut jika $y_i \neq y_j$.
$$L = \max(0, \alpha_j - \alpha_i) \tag{2.20}$$
$$H = \min(c, c + \alpha_j - \alpha_i) \tag{2.21}$$
Dan jika $y_i = y_j$, maka persamaan menjadi :

$$L = \max(0, \alpha_j + \alpha_i - c) \quad (2.22)$$

$$H = \min(c, \alpha_j - \alpha_i) \quad (2.23)$$

7. 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_j^{new}) dengan menggunakan persamaan berikut.

$$\eta = 2K(x_i, x_j) - K(x_i, x_j) - K(x_i, x_j) \quad (2.24)$$

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

$$\alpha_j^{new} = \alpha_j - \frac{y_j(E_i - E_j)}{\eta} \quad (2.25)$$

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

$$\alpha_j^{new} = \begin{cases} H, \alpha_j^{new} > H \\ \alpha_j^{new}, L \leq \alpha_j^{new} \leq H \\ L, \alpha_j^{new} < H \end{cases} \quad (2.26)$$

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

$$\left|\alpha_j^{new} - \alpha_j^{old}\right| < \varepsilon \quad (2.27)$$

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

$$\alpha_i^{new} = \alpha_i + y_i * y_j(\alpha_j^{old} - \alpha_j^{new}) \quad (2.28)$$

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

$$b_1 = b - E_i - y_i(\alpha_i^{new} - \alpha_i^{old})K(x_i, x_i) - y_j(\alpha_j^{new} - \alpha_j^{old})K(x_i, x_j) \quad (2.29)$$

12. Setelah nilai $b\alpha_i$ didapat, kemudian cari nilai b terhadap α_j (disimpan dengan variabel yang disebut $b\alpha_j$) menggunakan Persamaan 2.30 sebagai berikut.

$$b_2 = b - E_j - y_i(\alpha_i^{new} - \alpha_i^{old})K(x_i, x_j) - y_j(\alpha_j^{new} - \alpha_j^{old})K(x_j, x_j) \quad (2.30)$$

13. Setelah nilai $b\alpha_i$ dan $b\alpha_j$ didapatkan, langkah selanjutnya yaitu mencari nilai b berdasarkan persamaan berikut.

$$b = \begin{cases} b_1, 0 < \alpha_i^{new} < c \\ b_2, 0 < \alpha_j^{new} < c \\ \frac{b_1 + b_2}{2}, \text{otherwise} \end{cases} \quad (2.31)$$

14. Selanjutnya hitung num changed alphas dengan persamaan berikut.

$$\text{Num\_changed\_alphas} = \text{num\_change\_alpha} + 1 \quad (2.32)$$

15. 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.

2.5 Sertifikat

Menurut Kamus Besar Bahasa Indonesia (KBBI) arti dari kata sertifikat merupakan tanda atau surat keterangan (pertanyaan) tertulis atau tercetak dari orang yang berwenang yang dapat digunakan sebagai bukti kepemilikan atau suatu kejadian [28].

Sertifikat memiliki fungsi untuk membuktikan bahwa seseorang telah mengikuti suatu kegiatan pelatihan, kegiatan *workshop*, kegiatan seminar, kegiatan kemah pramuka dan kegiatan lainnya yang dilaksanakan dengan maksud memberikan edukasi kepada pesertanya. Bagian-bagian sertifikat tersebut adalah sebagai berikut:

1. Kop sertifikat atau piagam dan logo organisasi
2. Tulisan sertifikat atau piagam
3. Nomor surat
4. Nama peserta

5. Waktu dan tempat

6. Pengesahan

2.6 Penelitian Terkait

Dalam penelitian skripsi ini, penulis sedikit banyak terinspirasi dan mereferensi dari penelitian-penelitian sebelumnya yang berkaitan dengan latar belakang pada skripsi ini. Berikut ini penelitian terdahulu yang berhubungan dengan skripsi ini antara lain :

Penelitian yang dilakukan oleh Indra Riyanto "Deteksi Teks Menggunakan *Text Flow* Pada Sertifikat" diperoleh akurasi terbaik masing-masing untuk *Recall* sebesar 67% *Precision* sebesar 58% dan *F-Score* sebesar 62%. Ada tahap yang harus dibenahi yaitu pertama proses thresholding, karena pada penelitian tersebut karakter yang memiliki ukuran lebih kecil dibandingkan dengan karakter yang lainnya pada suatu sertifikat akan lebih sulit dikenali karena akan menjadi tidak sesuai dengan karakter yang seharusnya yang mengakibatkan kurang efektifnya metode Text Flow.

Penelitian yang dilakukan oleh Reza Yogi Andriana "*Optical Character Recognition* (OCR) Menggunakan *Support Vector Machine* dan *Zoning* Pada Sertifikat" mendapatkan akurasi sebesar 83,27952%. Ada metode khusus yang digunakan yaitu metode ekstrakasi ciri dengan *zoning* dan untuk pengenalannya menggunakan *Support Vector Machine*.

Penelitian yang dilakukan oleh Angga Maulana Purba "*Text Detection In Indonesian Card Based On Maximally Stable Extremal Regions*" dengan akurasi 84,5%. Pada penelitian tersebut berfokus pada pendeteksian objek pada KTP yang menggabungkan antara Metode MSER dan metode lainnya seperti *Grayscale, Canny Edge Detection, Horizontal* RLSA dan Progressive Probabilistic Hough Transform.

2.7 Confusion Matrix

Akurasi klasifikasi merupakan suatu ukuran ketepatan dalam melakukan klasifikasi yang menunjukkan performansi teknik klasifikasi secara keseluruhan. Semakin tinggi akurasi klasifikasi, maka artinya semakin baik pula performansi teknik klasifikasi [29]. Dalam melakukan pengukuran akurasi, dapat digunakan *confusion matrix*. *Confusion Matrix* adalah sebuah tabel yang memuat hasil klasifikasi, dengan 2 kelas, yaitu kelas aktual dan kelas prediksi yang masing-masing memiliki nilai positif dan negatif (dalam prediksi 2 kelas). Untuk penjelasan lebih lanjut, terdapat pada Tabel 2.1.

Tabel 2. 1 *Confusion Matrix*

Aktual	**Prediksi**	
	positif	**negatif**
positif	TP	FN
negatif	FP	TN

Adapun untuk perhitungan tingkat akurasi menggunakan persamaan 2.30 sebagai berikut [30]:

$$Akurasi = \frac{\sum_{i=1}^{L} \frac{TP_i+TN_i}{TP_i+FN_i+FP_i+TN_i}}{L} x\ 100\% \quad (2.30)$$

$$Precision = \frac{\sum_{i=1}^{L} \frac{TP_i}{TP_i+FP_i}}{L} x\ 100\% \quad (2.31)$$

$$Recall = \frac{\sum_{i=1}^{L} \frac{TP_i}{TP_i+FN_i}}{L} x\ 100\% \quad (2.32)$$

$$F1 - Score = \frac{Precision\ x\ Recall}{Precision+Recall} x\ 2 \quad (2.33)$$

Keterangan:

- TP (*True Positive*) adalah jumlah prediksi benar pada kelas positif.
- FP : (*False Positive*) adalah jumlah prediksi salah pada kelas positif.
- FN : (*False Negative*) adalah jumlah prediksi salah pada kelas negatif.

- TN : (True Negative) adalah jumlah prediksi benar pada kelas negatif.
- L : adalah Jumlah Keseluruhan Data yang Diuji atau Jumlah Kelas.

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 [31].

Bahasa ini muncul pertama kali pada tahun 1991, dirancang oleh seorang bernama Guido van Rossum. Sampai saat ini *Python* mendukung hamper semua sistem operasi, bahkan untuk sistem operasi Linux, hampir semua distronya sudah menyertakan *Python* di dalamnya.

Dengan kode yang simpel dan mudah diimpilementasikan, seorang programmer dapat lebih mengutamakan pengembangan aplikasi yang dibuat, bukan malah sibuk mencari *syntax error*.

2.9 Open CV

OpenCV merupakan salah satu *library open-source* yang digunakan untuk berbagai macam operasi pemrosesan gambar dan video. Beberapa fitur seperti pengenalan wajah atau pelacakan objek dapat dibuat dengan *OpenCV*. *Library* ini tersedia untuk beberapa bahasa pemrograman yaitu bahasa C, C++, dan Python serta kompatibel dengan sistem operasi Windows, Linux, dan MacOS [32]. Adapun contoh *syntax* dari OpenCV yang digunakan pada penelitian ini adalah berikut.

Tabel 2.3 Contoh Syntax dari OpenCV

Contoh Code	Fungsi	Keterangan
cv.imread(img,flags)	Membaca citra	Img = citra yang akan dibaca

		Flags = mode warna citra yang akan dibaca
cv.resize(img,size, interpolation)	Mengubah ukuran citra	Img = citra yang akan dirubah ukurannya size = ukuran citra Interpolation = metode interpolasi
cv.imshow('Image', img)	Menampilkan citra	Image = judul citra yang akan ditampilkan Img = citra yang akan ditampilkan

2.10 Numpy

Numpy adalah sebuah library untuk komputasi ilmiah yang menggunakan bahasa pemrograman *Python*. *NumPy* menyediakan berbagai macam operasi seperti matriks, *sorting*, aljabar linier dasar, statistik dasar dan lain-lain [33]. Adapun contoh *syntax* dari *NumPy* yang digunakan pada penelitian ini adalah sebagai berikut :

Tabel 2.4 Contoh Syntax dari Numpy

Contoh Code	Fungsi	Keterangan
Numpy.array()	Membuat Array	-
Numpy.shape	Menghitung ukuran suatu array	-
Numpy.random.seed(seed)	Membangkitkan nilai acak	Seed = diisi dengan bilangan bulat

2.11 Anaconda

Anaconda adalah *open data science platform* yang dibangun menggunakan Bahasa pemrograman python [34]. Lebih dari 4,5 juta pengguna, Anaconda

menjadi data science ecosystem paling terkenal dan terpercaya di dunia. Anaconda menyediakan lebih dari 1000 paket data science untuk mengolah data menjadi informasi yang diperlukan [34]. Versi Anaconda open-source menyediakan distribusi dengan performa tingkat tinggi dari bahasa pemrograman Python dan R yang berisikan lebih dari 100 paket paling populer dari Python, R dan Scala untuk data science. Dengan Anaconda pengguna dapat mengakses lebih dari 720 paket yang dapat dengan mudah dipasang dengan conda. Anaconda dapat digunakan di berbagai platform seperti Windows, Linux dan Mac.


UNIVERSITAS KOMPUTER INDONESIA
UNIKOM