ANALISIS PERBANDINGAN METODE *CONVOLUTIONAL NEURAL NETWORK* DAN *SUPPORT VECTOR MACHINE* DALAM PENGENALAN KARAKTER TULISAN KOREA

SKRIPSI

Diajukan untuk Memenuhi Sebagian Syarat Guna Memperoleh
Gelar Sarjana Matematika
dalam Ilmu Matematika

Oleh : **MUHAMMAD FIKRI RIYANTO**
NIM : 2108046041

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI WALISONGO
SEMARANG
2025

PERNYATAAN KEASLIAN

Yang bertanda tangan di bawah ini :

Nama	:	Muhammad Fikri Riyanto
NIM	:	2108046041
Jurusan/Program Studi	:	Matematika/ Matematika

menyatakan bahwa skripsi yang berjudul :

ANALISIS PERBANDINGAN METODE *CONVOLUTIONAL NEURAL NETWORK* DAN *SUPPORT VECTOR MACHINE* DALAM PENGENALAN KARAKTER TULISAN KOREA

secara keseluruhan adalah hasil penelitian/karya saya sendiri, kecuali bagian tertentu yang dirujuk sumbernya.

Semarang, 14 Maret 2025
Pembuat pernyataan,

Muhammad Fikri Riyanto
NIM : 2108046041

KEMENTERIAN AGAMA R.I.
UNIVERSITAS ISLAM NEGERI WALISONGO
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Prof. Dr. Hamka (Kampus II) Ngaliyan Semarang
Telp. 024-7601295 Fax. 7615387

PENGESAHAN

Naskah skripsi berikut ini :

Judul : **ANALISIS PERBANDINGAN METODE *CONVOLUTIONAL NEURAL NETWORK* DAN *SUPPORT VECTOR MACHINE* DALAM PENGENALAN KARAKTER TULISAN KOREA**
Penulis : Muhammad Fikri Riyanto
NIM : 2108046041
Jurusan : Matematika

Telah diujikan dalam sidang *skripsi* oleh Dewan Penguji Fakultas Sains dan Teknologi UIN Walisongo dan dapat diterima sebagai salah satu syarat memperoleh gelar sarjana dalam Ilmu Matematika.

Semarang, 22 April 2025

DEWAN PENGUJI

Penguji I,

Zulaikha, M.Si
NIP : 199204092019032027

Penguji II,

Ariska Kurnia Rachmawati, M.Sc
NIP : 198908112019032019

Penguji III,

Agus Wayan Yulianto, M.Sc
NIP : 198907162019031007

Penguji IV,

Eva Khoirun Nisa, M.Si
NIP : 198701022019032010

Pembimbing,

Ariska Kurnia Rachmawati, M.Sc
NIP : 198908112019032019

NOTA DINAS

Semarang, 14 Maret 2025

Yth. Ketua Program Studi Matematika
Fakultas Sains dan Teknologi
UIN Walisongo Semarang

Assalamu'alaikum warahmatullahi wabarakatuh

Dengan ini diberitahukan bahwa saya telah melakukan bimbingan, arahan dan koreksi naskah skripsi dengan:

Judul	:	ANALISIS PERBANDINGAN METODE *CONVOLUTIONAL NEURAL NETWORK* DAN *SUPPORT VECTOR MACHINE* DALAM PENGENALAN KARAKTER TULISAN KOREA
Nama	:	Muhammad Fikri Riyanto
NIM	:	2108046041
Jurusan	:	Matematika

Saya memandang bahwa naskah skripsi tersebut sudah dapat diajukan kepada Fakultas Sains dan Teknologi UIN Walisongo untuk diujikan dalam Sidang Munaqasyah.

Wassalamu'alaikum warahmatullahi wabarakatuh

Pembimbing,

Ariska Kurnia Rachmawati, M.Sc
NIP : 198908112019032019

ABSTRAK

Pembelajaran bahasa Korea semakin diminati di Indonesia, terutama di kalangan remaja. Namun, pengenalan karakter *Hangeul* masih menghadapi kendala akibat kemiripan bentuk dan pola alfabetnya, yang dapat menyebabkann kesalahan dalam identifikasi karakter. Penelitian ini bertujuan untuk membandingkan kinerja *Convolutional Neural Network* (CNN) dan *Support Vector Machine* (SVM) dalam mengenali karakter tulisan Korea *(Hangeul)*. Dataset yang digunakan terdiri dari 2320 karakter dari *Kaggle*, dengan masing-masing kelas berisi 80 karakter. Data citra diproses melalui *preprocessing*, augmentasi data, pembagian data, ekstraksi fitur HOG dan pelatihan model. Metode CNN menggunakan *optimizer* RMSProp dengan *learning rate* 0,001 dan 15 *epoch*, sedangkan SVM menggunakan kernel RBF dengan *C*= 100 dan *gamma =scale*. Hasil evaluasi menunjukkan bahwa metode CNN lebih unggul dengan akurasi 86%, *precision* 86,57%, *recall* 86,21%, dan *F1-score* 86,08%, dibandingkan SVM yang hanya mencapai akurasi 69%, *precision* 70%, *recall* 69%, dan *F1-score* 69%. Dengan demikian, CNN memiliki selisih akurasi sebesar 17% dari SVM dalam pengenalan karakter tulisan *(Hangeul)*.

Kata kunci : *Hangeul*, CNN *(Convolutional Neural Network)*, SVM *(Support Vector Machine)*, Ekstraksi fitur HOG *(Histogram of Oriented Gradients)*

KATA PENGANTAR

Puji Syukur atas kehadirat Allah SWT yang telah melimpahkan rahmat, taufik dan hidayahNya kepada Penyusun sehingga dapat menyelesaikan tugas akhir dengan judul **"Analisis Perbandingan Metode *Convolutional Neural Network* dan *Support Vector Machine* Dalam Pengenalan Karakter Tulisan Korea"**, yang merupakan kewajiban Mahasiswa program Sarjana Strata 1 (S-1) Fakultas Sains dan Teknologi Universitas Islam Negeri (UIN) Walisongo Semarang, sebagai tugas akhir untuk memperoleh gelar Sarjana Strata 1 (S-1). Shalawat dan salam semoga tercurahkan kepada arwah Nabi Muhammad SAW yang telah menyampaikan ajaran - ajaran yang menjadi pedoman bagi umat manusia untuk menghindari kekhufuran dan jurang kebodohan.

Dalam tugas akhir ini, sudah tentu penyusunan banyak memperoleh bantuan dan bimbingan serta dorongan dari berbagai pihak baik moril maupun materil yang tentunya sangat bermanfaat dalam tugas akhir ini. Pada kesempatan ini, penyusun ingin menyampaikan ucapan terima kasih yang setulus-tulusnya kepada:

1. Budi Cahyono S.Pd., M.Si. selaku ketua Jurusan Matematika di Fakultas Sains dan Teknologi Universitas Islam Negeri Walisongo Semarang.

2. Any Muanalifah M.Si, Ph.D. selaku ketua Program Studi Matematika di Fakultas Sains dan Teknologi Universitas Islam Negeri Walisongo Semarang.

3. Ariska Kurnia Rachmawati M.Sc. selaku Dosen Pembimbing Skripsi yang telah meluangkan waktu, tenaga dan fikirannya

untuk selalu memberikan bimbingan dan arahan, sehingga penyusunan skripsi ini dapat terselesaikan.

4. Seftina Diyah Miasary, S.Si., M.Sc. selaku Dosen Wali yang telah memberikan bimbingan, arahan dan motivasi selama masa studi.
5. Seluruh Dosen Matematika yang telah memberikan ilmu selama dibangku perkuliahan.
6. Cinta pertama dan panutanku, ayahanda Facthur Ridho. Beliau memang tidak sempat merasakan Pendidiksn sampai bangku perkuliahan, namun beliau bekerja keras serta mendidik, memberikan motivasi, memberikan dukungan sehingga penulis mampu menyelesaikan studi sampai sarjana.
7. Pintu Surgaku, Ibunda Srihastutik . Beliau sangat berperan penting dalam proses menyelesaikan program studi saya, beliau juga tidak sempat merasakan pendidikan dibangku perkuliahan, namun beliau tidak henti memberikan semangat serta do'a yang selalu mengiringi langkah penulis sehingga penulis bisa menyelesaikan program studi penulis sampai selesai.
8. Alm. Embah Putri, Alm Embah Kangkung. Terima kasih telah mendidik penulisan sedari kecil, anak kecil yang dulunya tidak bisa apa - apa sudah menjadi sarjana, yang cucumu telah meyandang sarjana ketiga dikeluarga besar. Terima Kasih telah mengajarkan penulis pentingnya pendidikan.
9. Kepada Kakak saya Muhammad Fachrul Hudallah, S.H. terima kasih banyak atas dukungan moril maupun materil,

terima kasih juga atas segala motivasi dan dukungan yang diberikan kepada penulis sehingga penulis mampu merasakan studinya sampai sarjana.

10. Kepada teman- teman Kuliah Kerja Nyata, terima kasih atas kerandoman kalian yang membuat penulis senang, sehingga penulis semangat untuk mengerjakan skripsi ini sampai selesai.

11. Kepada sahabat penulis, Tyas, Aflah,Dita, Khoirin Nisa, Farhan, Jundi, Fifi, Sherly, Linda, Fathimah, Nila, dan Achla, terima kasih atas partisipanya yang selalu mengajak pergi saat penulis merasa kesepian sehingga kini penulis menjadi senang dan semangat untuk mengerjakan skripsi ini sampai selesai.

12. Kepada adik - adik Desa Ngampel Wetan terima kasih atas kelucuan dan kerandoman kalian membuat penulis senang, sehingga penulis semangat untuk mengerjakan skripsi ini sampai selesai.

13. Kepada Komunitas *Realistic Neural Network*, Manusia aku bercerita, My Skill dan Acce English Academy yang sudah membantu dan mendukung penulis untuk menyelesaikan skripsi dengan baik.

14. Serta seluruh pihak yang tidak dapat disebutkan satu - persatu yang telah membantu dan mendukung penulis dalam menyelesaikan skripsi dengan baik.

15. Muhammad Fikri Riyanto, ya! diri saya sendiri. Apresiasi sebesar-besarnya yang telah berjuang untuk menyelesaikan

apa yang telah dimulai. Sulit bisa bertahan sampai dititik ini, terimakasih untuk tetap hidup dan merayakan dirimu sendiri, walaupun sering kali putus asa atas apa yang sedang diusahakan. Tetaplah jadi manunsia yang mau berusaha dan tidak lelah untuk mencoba. AKHIRNYA KELARRR, and there is actually llst of thing i should have to thank to you but you really did well this past 21 years that you have going through lost of thing, Lu keren bangetttt. If the old me read this skripsi again, *i hope you are doing well and living the best of your life*, semoga mimpinya cepat tercapai sehat selalu, jangan *overthinking* mulu. Semangat ya <3

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

Halaman

BAB I

PENDAHULUAN

A. Latar Belakang Masalah

Popularitas budaya K-pop di Indonesia telah mendorong minat masyarakat, terutama kalangan remaja, untuk mempelajari bahasa Korea (Tumiwa dkk., 2024). Sebelum memulai pembelajaran bahasa Korea, penting terlebih dahulu mengenali alfabet- alfabetnya, karena sistem penulisannya berbeda dengan alfabet latin. Bahasa Korea menggunakan sistem alfabet yang unik yang disebut *Hangeul* (Sam, 2023). *Hangeul* sendiri mempunyai 40 alfabet, antara lainnya adalah 19 konsonan dan 21 vokal, yang membentuk dasar dari semua kata dalam bahasa Korea (Hwa dkk., 2013).

Kompleksitas *Hangeul* terletak pada aturan penggabungan alfabet-alfabetnya untuk membentuk suku kata. Aturan ini berbeda dengan sistem pembentukan kata dalam bahasa yang menggunakan alfabet latin. Salah satu tantangan terbesar bagi pemula adalah memahami aturan penggabungan huruf *Hangeul* yang kompleks, terutama karena kemiripan bentuk dan pola antara beberapa huruf dapat menyebabkan kebingungan dalam membedakan makna kata alfabet yang hanya berbeda satu atau dua huruf, misalnya 'ㅗ' (o) dan 'ㅜ' (u). Selain itu, belajar bahasa Korea tidak hanya sekadar menghafal kosakata, tetapi juga memahami sistem penulisan yang kompleks, terutama bagi seseorang yang mencoba mempelajarinya secara mandiri (Hwa dkk., 2013). Salah satu cara dalam pengenalan karakter dapat menggunakan kecerdasan buatan *(Artificial Intelegence)*, terutama

dalam teknik *machine learning*.

Teknik *machine learning* menawarkan solusi yang efisien dan efektif untuk mengidentifikasi dan memahami karakter tersebut dari berbagai konteks, termasuk dalam teks yang ditulis tangan atau dalam bentuk dokumen digital (Sari dkk., 2024). Keunggulan *machine learning* terletak pada kemampuannya mempelajari pola dari data pelatihan dan menerapkan pengetahuan tersebut pada kondisi baru. Proses ini dapat dibandingkan dengan cara manusia belajar mengenali sesuatu, dimana pendekatan *machine learning* terinspirasi dari cara kerja otak manusia dalam mengelompokkan atau mengklasifikasikan informasi.

Machine learning dapat dibagi menjadi dua kategori utama dalam pengenalan karakter tulisan Korea *(Hangeul)*, yaitu *supervised learning* dan *deep learning*. *Supervised learning* adalah model *machine learning* yang dilatih untuk menghasilkan prediksi yang kemudian dibandingkan dengan label. Sementara itu, *deep learning* merupakan metode dalam kecerdasan buatan (*Artificial Intelegence)* yang mengajarkan komputer untuk memproses data dengan cara yang terinspirasi otak manusia (Elwirehardja dkk., 2023). Salah satu metode dalam *supervised learning* adalah *Support Vector Machine (SVM)* yang banyak digunakan untuk klasifikasi dan prediksi data (Chandra S Yoannita, 2023). Metode ini memiliki kerangka matematika yang kuat dan lebih jelas dibandingkan dengan teknik klasifikasi lainnya, serta efektif menangani masalah klasifikasi dengan pendekatan linier maupun non-linier. Kelebihan utama SVM adalah kemampuannya menghasilkan model yang sederhana dan akurat, serta kemampuannya menangani dataset dengan dimensi tinggi, seperti data gambar yang memiliki banyak fitur. Akan tetapi, SVM juga memiliki beberapa kekurangan, antara

lain waktu pelatihan yang cukup lama pada dataset besar dan penurunan performa jika jumlah data pelatihan sangat besar (Sendhilkumar S Geetha, 2023).

Cara kerja *Support Vector Machine* (SVM) dimulai dengan *preprocessing* (Mudhofar S Agustin, 2024; Irfani S Gasim, 2024; Harahap dkk., 2020). Kemudian, data diaugmentasi seperti rotasi dan *flipping* (Fadillah dkk., 2024). Setelah dibagi menjadi data latih dan data uji, fitur diekstrak menggunakan HOG (*Histogram of Oriented Gradients*) (Putra dkk., 2024). SVM membangun *hyperplane* untuk memisahkan kelas dengan margin maksimal atau menggunakan *kernel* untuk data non-linier (Deng dkk., 2013). Evaluasi dilakukan dengan *precision*, *recall*, dan *F1-Score* (Permana dkk., 2022).

Metode *Convolutional Neural Network* (CNN) merupakan salah satu metode dalam *deep learning*, yang subkategori dari *machine learning*. CNN dirancang khusus untuk menangani data dua dimensi, seperti citra gambar, dan sangat efektif dalam pengenalan gambar dan objek (Amri dkk., 2024). Metode ini mempunyai keunggulan utama dibandingkan dengan metode lain, antara lainnya adalah dapat kemampuannya untuk mengidentifikasi pola dalam data citra dengan sangat baik, mengurangi dimensi data melalui teknik konvolusi, dapat menangani data berwarna tanpa memerlukan praproses yang rumit, dan mempunyai tingkat akurasi yang tinggi dalam pengklasifikasian. Namun, selain mempunyai kelebihan metode *Convolutional Neural Network (CNN)* mempunyai kelemahan, antara lainnya adalah kesulitan dalam ekstraksi fitur untuk data non-gambar dan membutuhkan waktu pelatihan yang lama, terutama dengan dataset besar dan jaringan yang dalam, serta

membutuhkan sumber daya komputasi yang tinggi, seperti GPU, untuk pelatihan yang efisien (Goodfellow dkk., 2016).

Cara kerja CNN *(Convolutional Neural Network)* dimulai dengan *preprocessing* (Irfani S Gasim, 2024; Harahap dkk., 2020). Kemudian, data diaugmentasi dengan teknik seperti rotasi dan *flipping* (Fadillah dkk., 2024). Setelah dibagi menjadi data latih dan uji, fitur diekstraksi menggunakan HOG (Azmi dkk., 2023).

CNN memproses *input* melalui lapisan konvolusi untuk mengekstraksi fitur, diikuti ReLU *(Rectified Linear Unit)* untuk non-linearitas dan *max pooling* untuk reduksi dimensi (Rahmadwati dkk., 2024; Heryadi S Wahyono, 2020). Fitur kemudian di-*flatten* dan dikombinasikan dengan hasil HOG sebelum masuk ke lapisan *fully connected* untuk klasifikasi menggunakan *softmax*. Evaluasi dilakukan dengan *accuracy*, *precision*, *recall*, dan *F1-score* (Permana dkk., 2022).

Penelitian-penelitian terkait menunjukkan bahwa baik CNN maupun SVM banyak digunakan dalam berbagai bidang klasifikasi, termasuk pengenalan wajah, analisis sentimen, dan deteksi penyakit. CNN unggul dalam pengenalan citra dan data spasial, seperti pada klasifikasi jenis anggur (akurasi 99%) dan penyakit COVID-19 (97,14%) (Pratiwi, 2023; Wulandari S Fitrianah, 2021), sedangkan metode SVM memberikan hasil akurasi terbaik pada analisis sentimen opini publik (96,3%) dan pengenalan pola tanda tangan (94,5%) (Munawaroh S Alamsyah, 2022; Octariadi S Brianorman, 2019). Kombinasi CNN-SVM juga digunakan untuk klasifikasi kanker kulit dengan hasil terbaik 65,33% (Yohannes S Rivan, 2022). Penelitian lain menunjukkan bahwa KNN unggul dibandingkan SVM dalam klasifikasi kematangan buah mangga dengan akurasi 98,75% (Muchtar S Arjaliyah, 2024).

Metode *Convolutional Neural Network* (CNN) dikenal memiliki akurasi yang tinggi dalam pengolahan citra, karena kemampuannya dalam mengekstrak fitur-fitur penting melalui berbagai lapisan konvolusi dan *pooling*. Keunggulan ini membuat CNN sangat efektif dalam klasifikasi citra, terutama dalam tugas-tugas yang memerlukan pemahaman tentang pola dan struktur data visual. Selain itu, metode *Convolutional Neural Network* (CNN) juga memiliki kelemahan, yaitu proses pelatihan model cukup lama dan sumber daya komputasi yang besar. Namun, seiring perkembangan *hardware* yang semakin pesat, hal tersebut dapat diatasi dengan menggunakan teknologi GPU *(Graphical Processing Unit)*. Selain itu, masalah yang sering muncul adalah kurangnya jumlah data pelatihan yang memadai atau ketidakseimbangan kelas dalam dataset, hal ini dapat disebabkan dari beberapa faktor yaitu sulitnya untuk memperoleh data yang akan dijadikan sebagai dataset dan kurang waktu dalam pengumpulan data, semakin lama waktu yang digunakan untuk pengumpulan data juga menyebabkan masalah lain yaitu biaya yang digunakan akan semakin besar, sehingga semakin banyak data yang tersedia maka hasil akurasinya akan lebih efektif (Goodfellow dkk., 2016).

Di sisi lain, metode *Support Vector Machine* (SVM) lebih efektif dalam menangani data berdimensi tinggi, terutama ketika jumlah fitur yang diolah sangat besar (Sendhilkumar S Geetha, 2023). Data berdimensi tinggi merujuk pada data yang memiliki banyak fitur atau variabel untuk setiap sampel. Misalnya, dalam pengolahan citra, setiap gambar yang terdiri dari ribuan atau bahkan jutaan piksel dapat dianggap memiliki banyak dimensi, dengan setiap piksel sebagai satu fitur. Sebagai

contoh, gambar dengan resolusi 100×100 piksel (10.000 piksel) memiliki 10.000 fitur per piksel. Jika gambar tersebut berwarna (misalnya RGB), maka jumlah fitur citra akan meningkat lebih jauh dan menjadikan data berdimensi tinggi yang sangat kompleks. Sebaliknya, gambar dalam format *grayscale* hanya memiliki satu saluran warna untuk intensitas cahaya, mengurangi dimensi data secara signifikan karena setiap piksel hanya memiliki satu nilai. Hal ini mempermudah pengolahan citra dan mengurangi beban komputasi, tetapi dapat mengurangi informasi yang dapat diperoleh dari gambar jika warna memiliki peran penting dalam klasifikasi.

Support Vector Machine (SVM) juga dapat bekerja dengan baik dalam kasus klasifikasi dengan ruang fitur yang lebih kompleks dari berbagai aspek citra (seperti warna, tekstur, dan bentuk), karena mampu mencari *hyperplane* yang optimal untuk memisahkan kelas-kelas dalam data yang berdimensi tinggi (Brownlee, 2018). Namun, performa SVM sangat dipengaruhi oleh pemilihan kernel yang tepat. Kernel digunakan untuk mengubah data ke ruang dimensi yang lebih tinggi agar data yang sebelumnya tidak terpisah dengan baik dalam ruang asli dapat dipisahkan secara lebih jelas. Pemilihan kernel yang kurang sesuai, seperti kernel linear pada data yang memerlukan kernel non-linear, dapat mengurangi efektivitas model. Selain itu, SVM lebih efisien pada dataset kecil hingga menengah, tetapi performanya dapat menurun jika dataset yang digunakan sangat besar, karena proses pelatihan SVM cenderung lebih lambat dan memerlukan memori yang besar. Hal ini terjadi karena SVM harus mempertimbangkan setiap pasangan data (pasangan titik dalam ruang fitur) untuk membangun modelnya. Proses ini melibatkan perhitungan jarak

antara semua titik data, yang dapat sangat mempengaruhi kinerja algoritma, terutama pada dataset yang sangat besar. Sehingga, SVM dapat mengalami masalah kemampuan menangani data besar, di mana waktu komputasi dan penggunaan memori meningkat secara signifikan seiring dengan bertambahnya jumlah data. Masalah ini muncul karena kompleksitas perhitungan yang meningkat seiring dengan bertambahnya jumlah data, sehingga mempengaruhi efisiensi pelatihan model pada dataset besar (Sendhilkumar S Geetha, 2023).

Berdasarkan hasil pemaparan sebelumnya, metode *Convolutional Neural Network* (CNN) dan *Support Vector Machine* (SVM) telah banyak diterapkan dalam pengenalan karakter, kedua metode tersebut menunjukkan kelemahan secara signifikan. Metode *Convolutional Neural Network* (CNN) memiliki akurasi yang tinggi dalam pengolahan citra, tetapi memerlukan waktu pelatihan yang lama dan sumber daya komputasi yang besar. Sementara itu, SVM lebih efektif dalam menangani data berdimensi tinggi, tetapi performanya sangat dipengaruhi oleh pemilihan kernel yang tepat dan kurang optimal pada dataset besar. Selain itu, sebagian besar penelitian yang ada belum menggunakan teknik ekstraksi fitur seperti *Histogram of Oriented Gradients* (HOG), yang dapat meningkatkan representasi citra dan kinerja model. Oleh karena itu, peneliti tertarik untuk meneliti mengenai pengenalan karakter tulisan Korea *(Hangeul)* menggunakan metode CNN dan SVM, dengan menerapkan ekstraksi fitur *Histogram of Oriented Gradients* (HOG) untuk mengkaji karakter tulisan Korea *(Hangeul)* menggunakan metode CNN dan SVM karena memberikan pemahaman yang lebih mendalam terkait kelebihan dan kekurangan dari masing - masing

metode dalam menangani kompleksitas pada tulisan Korea *(Hangeul)*, melakukan perbandingan dari kedua metode dan memberikan rekomendasi metode terbaik dari kedua metode dalam pengklasifikasian karakter tulisan Korea *(Hangeul)* dengan judul "Analisis Perbandingan Metode *Convolutional Neural Network* dan *Support Vector Machine* dalam Pengenalan Karakter Tulisan Korea"

B. Rumusan Masalah

Berdasarkan hasil dari uraian latar belakang diatas, maka penulis menyimpulkan beberapa rumusan masalah sebagai acuan untuk penyusunan bab- bab selanjutnya. Rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana mengimplementasikan metode *Convolutional Neural Network* untuk pengenalan karakter tulisan Korea *(Hangeul)*?
2. Bagaimana mengimplementasikan metode *Support Vector Machine* untuk pengenalan karakter tulisan Korea *(Hangeul)*?
3. Bagaimana perbandingan hasil pengenalan karakter tulisan Korea *(Hangeul)* menggunakan *Support Vector Machine* dan *Convolutional Neural Network* ?

C. Tujuan Penelitian

Berdasarkan latar belakang dan perumusan masalah diatas, maka tujuan dari penelitian ini adalah:

1. Untuk mengimplementasikan metode *Convolutional Neural Network (CNN)* dalam pengenalan karakter tulisan Korea *(Hangeul)*.
2. Untuk mengimplementasikan metode *Support vector Machine (SVM)* dalam pengenalan karakter tulisan Korea *(Hangeul)*.
3. Untuk membandingkan hasil pengenalan karakter tulisan Korea *(Hangeul)* dengan menggunakan metode *Convolutional Neural Network (CNN)* dan *Support Vector Machine (SVM)*.

D. Manfaat Penelitian

Adapun manfaat penelitian yang dapat diperoleh dari penelitian ini adalah:

1. Bagi Penulis

   Penelitian ini diharapkan dapat meningkatkan pemahaman penulis mengenai metode *Convolutional Neural Network* (CNN) dan *Support Vector Machine* (SVM).

2. Bagi Lembaga

   Penelitian ini diharapkan dapat menjadi sarana bahan pustaka yang berharga bagi civitas akademika khususnya di jurusan Matematika.

3. Bagi Peneliti Selanjutnya

   Penelitian ini diharapkan dapat menjadi referensi untuk penelitian-penelitian selanjutnya, khususnya yang berkaitan dengan metode *Convolutional Neural Network* (CNN) dan *Support Vector Machine* (SVM).

4. Bagi Masyarakat

   Penelitian ini diharapkan dapat mempermudah masyarakat yang berminat belajar bahasa Korea, khususnya dalam mengenali dan memahami huruf *Hangeul* dengan lebih efisien. Selain itu, sistem pengenalan karakter *Hangeul* berbasis kecerdasan buatan ini dapat membantu masyarakat yang berminat belajar bahasa Korea untuk mengenali pola huruf Korea secara akurat dan lebih efisien, sehingga dapat mendukung proses belajar bahasa Korea secara mandiri dan otodidak.

E. Batasan Masalah

Batasan - batasan yang melingkupi fokus penelitian ini adalah sebagai berikut:

1. *Input* dari sistem adalah citra digital dan *output*-nya adalah informasi karakter tulisan Korea *(Hangeul)* yang ada pada citra tersebut.

2. Penelitian ini secara khusus dibatasi pada pengenalan karakter tulisan Korea *(Hangeul)* yang berkisar 29 karakter tulisan Korea *(Hangeul)* dimana setiap karakter terdiri 80 citra karakter tulisan Korea *(Hangeul)*.

3. Aplikasi yang digunakan yaitu *Google Colaboratory* .

4. Pembagian data yang diterapkan pada penelitian ini meliputi rasio 70%:30%, 80%:20%, dan 90%:10%. Beberapa skenario yang dilakukan bertujuan untuk mendapatkan hasil yang terbaik, semakin besar perbandingan data *training* (data latih) dibandingkan data *testing* (data uji) cenderung memberikan hasil akurasi yang lebih tinggi, semakin sedikit data training membuat hasil yang berfluktuasi atau tidak stabil (Swasono dkk., 2023).

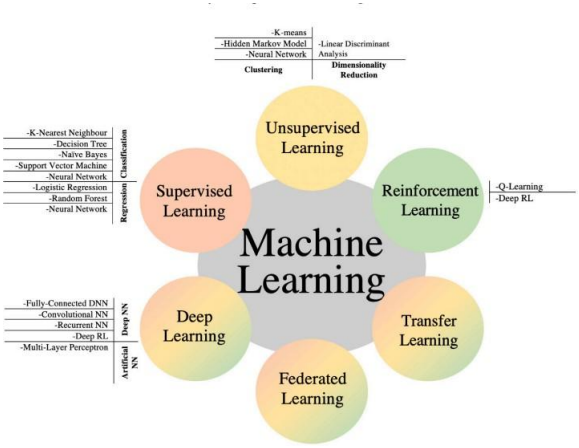
5. Perbandingan dilakukan berdasarkan *accuracy, presisi, recall,* dan *F1-score* dalam mengenali karakter tulisan Korea *(Hangeul)*.

BAB II

LANDASAN TEORI

A. *Machine Learning*

Machine learning merupakan bagian dari ilmu komputer yang difokuskan pada pengembangan algoritma yang dapat mempelajari pola dari data dan membuat prediksi berdasarkan pada pola-pola tersebut. Teknik ini melibatkan proses pemahaman dan penerapan algoritma yang belajar dari contoh-contoh fenomena yang ada(Burkov, 2019).



Gambar 2.1: *Machine Learning*.(Nazar dkk., 2022)

Istilah *machine learning* sendiri pertama kali didefinisikan oleh Arthur Samuel pada tahun 1959. Menurut Arthur Samuel, *machine learning* adalah suatu bidang ilmu komputer yang memberikan kemampuan pembelajaran kepada komputer untuk mengetahui sesuatu tanpa di program secara eksplisit (Heryadi S

Wahyono, 2020).

Dalam pembelajaran *machine learning*, terdapat 6 jenis yang umum digunakan, antara lainnya (Elwirehardja dkk., 2023):

1. *Supervised Learning*

   Supervised learning merupakan metode yang melatih model *machine learning* di mana model akan menganalisis data *input* untuk menghasilkan prediksi, yang kemudian dibandingkan dengan label. Metode *machine learning* ini sangat umum digunakan dalam berbagai bidang *machine learning*, khususnya *regression* dan classification. Contoh kasus dari *regression* adalah prediksi jumlah pembeli suatu produk atau barang, prediksi rating film, perkiraan tingkat keparahan suatu penyakit yang diderita seseorang, dan sebagainya. Sedangkan, contoh kasus dari *classification* adalah klasifikasi jenis penyakit, deteksi email spam, serta *sentiment analysis*, dan lain sebagainya. Selain itu, jenis model *machine learning* yang umum digunakan dengan metode *supervised learning* meliputi *linear regression, logistic regression, Decision Tree (DT), Support Vector Machine (SVM), ANN, eXtreme Gradient Boosting (XGBoost)*, dan sebagainya.

2. *Unsupervised Learning*

   Unsupervised learning merupakan metode yang melatih model *machine learning* untuk menentukan fitur yang mirip ataupun berbeda dari masing-masing data. Teknik *unsupervised learning* digunakan untuk *clustering, association, anomaly detection, dan dimensionality reduction*. Contoh kasus dari *clustering* adalah *market segmentation*

dan *customer churn prediction*. Selain *clustering*, *association* juga mempunyai contoh penerapan, yaitu dalam pembuatan *recommendation engine* pada aplikasi pemutar video atau lagu. Sedangkan, contoh kasus dari *anomaly detection* adalah *fault* diagnosis pada sistem, deteksi transaksi *fraud*, serta deteksi *network security intrusion*. Selain itu, ada juga contoh kasus dari *dimensionality reduction* yaitu mengidentifikasi fitur-fitur kunci dari masing-masing kelompok data.

3. *Reinforcement Learning*

   Reinforcement learning merupakan model *machine learning* yang dilatih untuk mengenali kondisi *environment* dan belajar untuk mengambil *action* yang dapat memberikannya *reward*. Model ini dapat memberikan kondisi *environment* dan belajar untuk mengambil *action* terbaik. Pengambilan *action* ini didasarkan pada konsep yang disebut *exploration* dan *exploitation*. *Exploration* merupakan *action* yang dilakukan untuk mengetahui kondisi atau fitur baru yang tidak diketahui agent pada *environment* dengan mengambil *action* yang tidak optimal. Sedangkan, *exploitation* adalah *action* yang dilakukan untuk mencapai reward maksimal berdasarkan yang sudah diketahui oleh agent tentang *environment*. Contoh dari kasus *reinforcement learning* adalah dalam pembuatan AI untuk game.

4. *Transfer Learning*

   Transfer learning merupakan metode yang menggunakan *network* yang sudah dilatih sebelumnya dan menggunakannya sebagai titik awal untuk mempelajari tugas selanjutnya ataupun baru. Contoh kasus *transfer*

learning adalah menggunakan model *pre-trained* yang dilatih untuk mengenali mobil, lalu model tersebut digunakan untuk mengenali truk. Selain itu, jenis model *machine learning* yang umum digunakan dengan metode *transfer learning* adalah AlexNet, VGG, dan GoogLeNet.

5. *Federated Learning*

   Federated learning merupakan melatih model *machine learning* pada data pengguna tanpa perlu mentransfer data tersebut ke satu lokasi. Contoh dari kasus metode *federated learning* adalah memodelkan dampak pandemi pada penundaan penerbangan, pelengkapan otomatis *Gboard*, prediksi sudut kemudi roda pada kendaraan yang dapat mengemudi sendiri, dan lain sebagainya. Selain itu, jenis model *machine learning* yang umum digunakan dengan metode *federated learning* adalah *Horizontal Federated Learning (HFL)*,*Vertical Federated Learning (VFL)*, dan *Federated Transfer Learning (FTL)*.

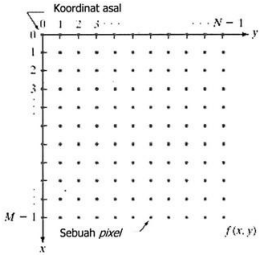
6. *Deep Learning*

   Deep Learning adalah metode dalam kecerdasan buatan (AI) yang mengajarkan komputer untuk memproses data dengan cara yang terinspirasi otak manusia. Contoh dari kasus *deep learning* adalah *sofware* pengenalan wajah, *autonomous* (*self driving) car*, pengenalan suara, penerjemah bahasa, dan sebagainya. Selain itu, arsitektur dalam *Deep Learning* adalah RNN *(Recurent Neural Network)*, LSTM *(Long Short Term Memory)*, CNN *(Convolutional Neural Network)*, dan DBN *(Deep Belief Networks)*.

B. Citra Digital

Citra digital merupakan sebuah *array* yang digunakan untuk pemrosesan gambar 2 dimensi dengan menggunakan bantuan komputer yang berisi nilai - nilai *real* maupun kompleks yang direpresentasikan dengan deretan bit tertentu (Putra, 2010).

Suatu citra dapat didefinisikan sebagai fungsi $f(x, y)$ dengan ukuran $M - 1$ (baris) dan $N - 1$ (kolom), di mana x dan y adalah koordinat spasial. Amplitudo f pada titik koordinat (x, y) disebut sebagai intensitas atau tingkat keabuan dari citra di titik tersebut. Jika nilai x, y, dan amplitudo f secara keseluruhan berhingga *(finite)* dan bernilai diskrit. Gambar 2.2 menunjukkan posisi koordinat dari citra digital(Putra, 2010).

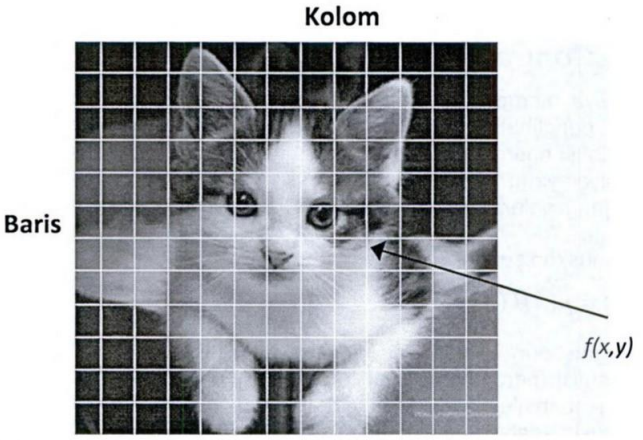


Gambar 2.2: *Koordinat Citra Digital*.(Putra, 2010)

Citra digital dapat ditulis dalam bentuk matriks sebagai berikut (Putra, 2010).

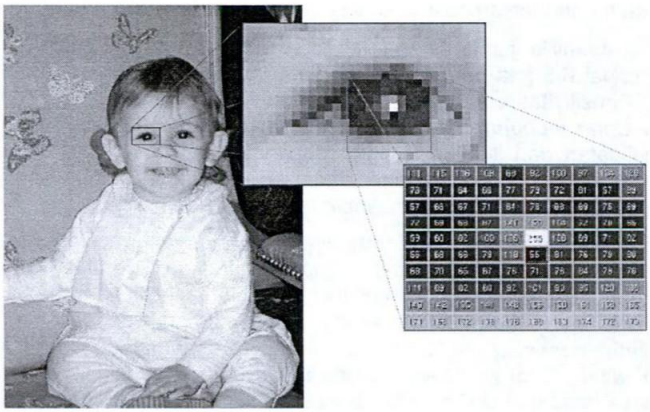
$$f(x, y) = \begin{bmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, N-1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1, 0) & f(M-1, 1) & \cdots & f(M-1, N-1) \end{bmatrix} \quad (2.1)$$

Nilai pada suatu irisan antara baris dan kolom (posisi x , y) disebut dengan *picture elements*, *image elements*, *pels*, atau pixels. Istilah terakhir (pixel) paling sering digunakan pada citra digital. Gambar 2.3 menunjukkan ilustrasi digitalisasi citra denga M = 16 baris dan N = 16 kolom.



Gambar 2.3: *Ilustrasi digitalisasi citra (pixel pada koordinat x=10, y=3 memiliki nilai 110*.(Putra, 2010)

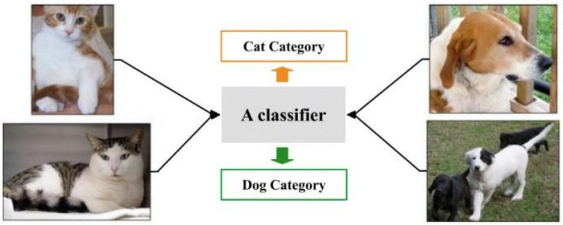
Pada Gambar 2.4 menyajikan contoh lain dari suatu citra digital (citra *grayscale*), dengan intensitas dari citra pada area tertentu.



Gambar 2.4: *Contoh citra grayscale, cuplikan (croping) pada area tertentu beserta nilai intensitasnya*.(Putra, 2010)

C. Klasifikasi Gambar

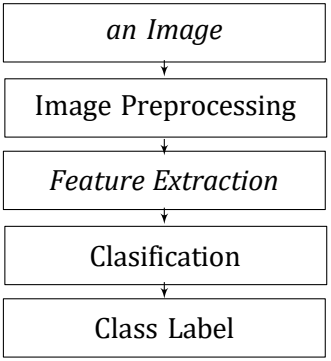
Klasifikasi gambar merupakan salah satu jenis klasifikasi yang digunakan untuk memberikan label kelas yang sudah ditentukan sebelumnya pada gambar berdasarkan konten yang ada didalamnya. Misalnya, sebuah algoritma /sistem klasifikasi gambar dapat mengklasifikasikan gambar kucing ke dalam kategori kucing dan gambar anjing ke dalam kategori anjing, seperti yang ditunjukkan pada Gambar 2.5 (Bi dkk., 2021).



Gambar 2.5: *Contoh Pengklasifikasian gambar kucing dan anjing*(Bi dkk., 2021)

.

Klasifikasi gambar mencakup berbagai tugas sesuai dengan jenis gambar, seperti klasifikasi gambar medis, klasifikasi gambar penginderaan jauh, klasifikasi wajah, klasifikasi tekstur, dan klasifikasi gambar biologis, karena adanya variasi tinggi pada latar belakang, pencahayaan, sudut pandang, skala, deformasi, dan oklusi di antara gambar - gambar, sehingga tugas dalam pengklasifikasian gambar sangat penting(Bi dkk., 2021).



Gambar 2.6: Langkah Gambaran Umum Sistem Klasifikasi Gambar (Bi dkk., 2021)

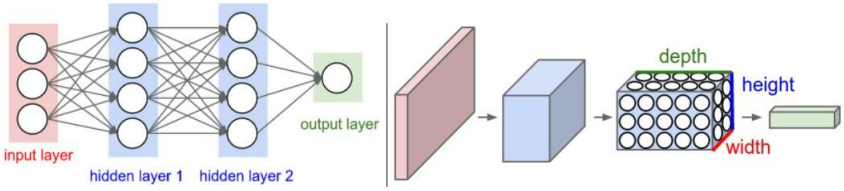
Pada Gambar 2.6 menunjukkan sistem klasifikasi gambar yang umum digunakan. Sistem ini memiliki tiga langkah utama, yaitu prapemrosesan gambar, ekstraksi fitur, dan klasifikasi. Selain tiga langkah ini, operasi lainnya, termasuk segmentasi gambar dan seleksi fitur, juga dapat dimasukkan dalam sistem klasifikasi gambar. *Input* dari sistem ini adalah sebuah gambar dan *output*-nya adalah label kelas yang diprediksi dari gambar tersebut. Langkah pertama yang penting adalah prapemrosesan gambar, di mana noise pada gambar dihilangkan dan kualitas gambar ditingkatkan. Langkah penting kedua adalah tahap ekstraksi fitur, berbagai fitur gambar seperti fitur tepi, fitur bentuk, atau fitur tekstur diambil untuk mendeskripsikan gambar *input*. Setelah fitur-fitur diperoleh, operasi terkait fitur seperti pemilihan fitur, pengelompokan fitur, atau konstruksi fitur yang lebih kecil atau lebih representastif. Selanjutnya, fitur - fitur digunakan dalam proses klasifikasi, di mana sebuah model klasifikasi dilatih untuk memprediksi label kelas untuk gambar *input*. Pada langkah klasifikasi, algoritma klasifikasi dan set pelatihan digunakan untuk melatih pengklasifikasi agar dapat memprediksi label kelas (Bi dkk., 2021).

D. *Convolutional Neural Network*(CNN)

Convolutional Neural Network (CNN) merupakan salah satu bagian dari *deep learning* yang dirancang untuk memproses data yang mempunyai topologi seperti grid, misalnya deret waktu *(grid 1-D)* atau gambar *(grid 2-D)* (Ketkar S Moolayil, 2021). Selain itu, CNN dapat diterapkan dalam pengenalan wajah, analisis dokumen, klasifikasi gambar, klasifikasi video, dan lain -lain. CNN terdiri

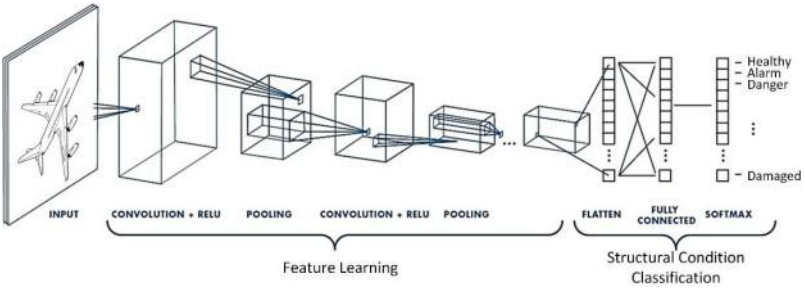
dari beberapa bagian, yaitu lapisan masukan *(input layer)*, lapisan keluaran *(output layer)*, dan sejumlah lapisan tersembunyi *hidden layer*. Lapisan tersembunyi biasanya mencakup *convolutional layers, pooling layers, normalization layers, ReLU layer, fully connected layers, dan loss layer*(Suyanto dkk., 2019). Semua lapisan tersebut tersusun secara bertumpuk - tumpuk secara berlapis - lapis, mirip dengan susunan bahan dalam sepotong burger yang berisi daging, selada, tomat, timun, saus tomat, mayonaise, saus sambal, dan roti bagian atas.

CNN menggunakan arsitektur tiga dimensi, yang terdiri atas lebar *(width)*, tinggi *(height)*, dan dalam *(depth)*, seperti yang di ilustrasikan pada Gambar 2.7 (Suyanto dkk., 2019). Setiap lapisan CNN melakukan transformasi dari volume masukan tiga dimensi (3D *input volume*) ke dalam volume keluaran tiga dimensi aktivasi - aktivasi sel saraf *(3D output volume of neuron activations)*.



Gambar 2.7: *Arsitektur CNN*. (Heryadi S Wahyono, 2021)

Pada Gambar 2.7 menunjukkan bahwa masukkan berupa citra berwarna, di mana lebar dan tinggi menyatakan dimensi citra tersebut sedangkan dalam *depth*nya adalah 3 yang menyatakan kanal *Red, Green, Blue (RGB)*.



Gambar 2.8: *Tahapan Metode CNN*. (Satrio S Agung, 2021)

Pada Gambar 2.8 menunjukkan cara kerja *Convolutional Neural Network* (CNN) dimulai dengan tahap *input* data, dimana data citra gambar akan dimasukkan kedalam sistem. Citra gambar tersebut biasanya dalam format RGB dan diproses melalui beberapa tahap sebelum digunakan dalam model *Convolutional Neural Network* (CNN). Pada tahap pertama, dilakukan *preprocessing*, yang dimulai dengan mengubah gambar dari format RGB menjadi *grayscale* untuk mengurangi kompleksitas data dan memfokuskan pada intensitas cahaya (Mudhofar S Agustin, 2024). Setelah itu, dilakukan *denoising*, yaitu penghapusan noise atau gangguan dalam gambar dengan menggunakan filter seperti *Gaussian Blur* atau *Median Filter* (Wardana dkk., 2024). Proses berikutnya adalah binerisasi, yang mengubah gambar *grayscale* menjadi gambar biner, di mana setiap piksel diubah menjadi dua nilai, antara lainnya adalah 0 (hitam) dan 1 (putih), dengan tujuan memperjelas bentuk objek atau huruf dalam gambar (Irfani S Gasim, 2024). Tahap akhir *preprocessing* adalah *thinning* yang bertujuan mengurangi ketebalan objek atau huruf tanpa mengubah bentuk atau struktur aslinya, memudahkan

ekstraksi fitur dan klasifikasi (Harahap dkk., 2020).

Setelah tahap *preprocessing*, data akan di augmentasikan dengan tujuan untuk memperbanyak variasi data dengan cara memodifikasi gambar asli. Beberapa teknik augmentasi ini dengan memodifikasi citra, seperti membalik, memotong, memutar, dan mengubah skala, sehingga model dapat mengenali variasi data, serta dapat mengurangi *overfitting* (Fadillah dkk., 2024). Selanjutnya, dataset dibagi menjadi data *training*, dan *data uji* dengan rasio tertentu, misalnya 80%:20%, 90%:10%, dan 70%:30%. Tujuan dari pembagian ini adalah untuk melatih dan menguji model agar dapat menggeneralisasi dengan baik pada data baru. Data latih digunakan untuk melatih model CNN, sedangkan data uji digunakan untuk mengevaluasi kinerja model (Azmi dkk., 2023).

Pada tahap ekstraksi fitur, salah satu teknik yang diterapkan dalam penelitian ini adalah *Histogram of Oriented Gradient* (HOG). Fitur HOG diambil dari augmentasi data yang sudah dibagi untuk menghasilkan representasi fitur yang lebih variatif dan kaya. Proses *Histogram of Oriented Gradient* (HOG) dimulai dengan membagi gambar yang telah diproses ke dalam sel-sel kecil dengan ukuran 8×8 piksel atau 16×16 piksel. Selanjutnya, dihitung gradien horizontal dan vertikal pada setiap sel menggunakan operator seperti Sobel, yang mengukur perubahan intensitas warna antar piksel (Putra dkk., 2024). Arah dan nilai gradien digunakan untuk membuat histogram orientasi yang menunjukkan distribusi arah gradien dalam setiap sel. Setelah pembuatan histogram, dilakukan normalisasi untuk mengurangi pengaruh perubahan pencahayaan atau kontras yang dapat memengaruhi akurasi ekstraksi fitur. Vektor fitur akan dibentuk dengan menggabungkan

histogram-histogram menjadi sebuah vektor panjang yang berisi informasi penting tentang karakteristik keseluruhan gambar, seperti tepi, bentuk, dan orientasi objek di dalamnya.

Pembagian data dari augmentasi data yang dihasilkan akan digunakan dalam pelatihan model *Convolutional Neural Network* (CNN). Proses konvolusi dimulai dengan menerapkan filter (kernel) pada gambar *input* (Rahmadwati dkk., 2024). Pada tahap awal, misalnya pada lapisan konvolusi pertama, biasanya digunakan 32 filter untuk mengekstraksi fitur dasar seperti tepi dan tekstur. Filter ini bergerak melintasi gambar, melakukan operasi *dot product* antara nilai piksel di area yang dicakup oleh filter dan bobot filter tersebut. Hasil dari operasi ini menghasilkan peta fitur *(feature map)* yang menyoroti area dengan pola tertentu. Setelah tahap konvolusi pertama, hasil yang diperoleh biasanya diproses melalui fungsi aktivasi seperti ReLU *(Rectified Linear Unit)* untuk menambahkan non-linearitas dalam model. Fungsi aktivasi ini sangat penting karena memungkinkan jaringan untuk mempelajari pola-pola yang lebih kompleks dalam data. Lapisan *pooling* diterapkan untuk mengurangi dimensi peta fitur dan tetap mempertahankan informasi penting. Metode *pooling* yang umum digunakan adalah *max pooling*, yang mengambil nilai maksimum dari setiap area *pooling* (Heryadi S Wahyono, 2020).

Proses *feature extraction* pada CNN dapat diulang dengan menambahkan lapisan konvolusi, misalnya dari 32 filter pada lapisan menjadi 64 filter pada lapisan berikutnya untuk menangkap fitur yang lebih kompleks. Setiap lapisan konvolusi diikuti oleh aktivasi dan *pooling* untuk menyederhanakan data. Setelah *feature extraction*, dilakukan tahap *flatening* yang bertujuan untuk mengubah *output* multidimensional dari lapisan

pooling (misalnya matriks 2D) menjadi vektor satu dimensi. *Flattening* mengubah setiap elemen dalam matriks menjadi satu kolom panjang. Selain itu, vektor dari *flatten* akan dikombinasikan dengan vektor fitur HOG yang berfungsi untuk memperkaya menerima informasi dari citra masukan. Vektor kombinasi ini kemudian menjadi masukan pada lapisan *fully connected* (Dense), yang bertugas melakukan klasifikasi atau regresi berdasarkan fitur yang diekstraksi sebelumnya (Primartha, 2021). *Flattening* memungkinkan jaringan memproses informasi terkompresi dari fitur gambar secara individual sebelum diproses oleh lapisan *fully connected*.

Di akhir arsitektur CNN, terdapat *fully connected layer* yang berperan dalam klasifikasi berdasarkan fitur yang diekstraksi. Lapisan ini terdiri dari beberapa *hidden layer* yang memproses informasi dengan menggunakan fungsi aktivasi ReLU *(Rectified Linear Unit)* bertujuan untuk model dapat mempelajari pola kompleks (Primartha, 2021). Pada lapisan terakhir, digunakan fungsi aktivasi *softmax* untuk menghasilkan *output* probabilitas setiap kelas dalam klasifikasi, memastikan semua probabilitas berada dalam rentang [0, 1] dengan jumlah total 1. Fungsi ini digunakan untuk klasifikasi *multiclass*.

1. *Convolutional Layer*

Convolutional layer (lapisan konvolusi) merupakan elemen inti dari *Convolutional Neural Network* (CNN) (Suyanto dkk., 2019). Pada tahap ini, operasi konvolusi dilakukan terhadap *output* dari *layer* sebelumnya. *Layer* tersebut merupakan bagian inti dari jaringan arsitektur CNN. Konvolusi merupakan istilah matematika yang menggambarkan proses penerapan suatu fungsi

pada *output* fungsi lain yang dilakukan secara berulang (Zhang dkk., 2020). Operasi konvolusi adalah operasi pada dua fungsi argumen bernilai nyata. Operasi ini dilakukan oleh data *input* dan kernel. Kernel dari CNN merupakan *array* multidimensi yang akan menjadi operator untuk melakukan operasi tertentu dengan *array input* dan dioperasikan dengan pola bergeser pada citra *input*, dimana hasil perkalian kedua fungsi pada setiap titik merupakan hasil konvolusi (Rahmadwati dkk., 2024). Nilai pada kernel merupakan nilai yang diperoleh dari hasil pelatihan. Operasi ini juga menerapkan fungsi *output* sebagai *feature map* dari *input* citra (Darmanto, 2019). *Input* dan *output* ini dapat dilihat sebagai dua argumen bernilai riil. Operasi konvolusi dapat dilambangkan dengan asterisk ($*$) atau $\otimes$ yang dituliskan sebagai berikut (Primartha, 2021):

$$\boldsymbol{f(x, y)} * \boldsymbol{g(x, y)} \text{ atau } \boldsymbol{f(x, y)} \otimes \boldsymbol{g(x, y)} \qquad (2.2)$$

Keterangan:

- $f(x, y)$: Fungsi citra *input*
- $g(x, y)$:Kernel konvolusi atau kernel filter.
- x : Indeks kolom menunjukkan posisi horizontal dalam citra atau matriks.
- y : Indeks baris menunjukkan posisi vertikal dalam citra atau matriks.

Pada fungsi kontinu, konvolusi merupakan integral jumlahan dari sebuah fungsi f (x, y) yang digeser atas fungsi g(x, y) dan menghasilkan fungsi h(t). Berikut merupakan fungsi kontinu (Primartha, 2021):

$$\boldsymbol{h(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(a)g(t-a)\,da} \qquad (2.3)$$

Keterangan:

- h(t) : Hasil operasi konvolusi dari fungsi f dan g pada waktu t.
- f(a) dan g($t - a$) : Fungsi-fungsi yang dikonvolusikan.
- g(t) :Kernel konvolusi atau fungsi respon impuls.
- a : Variabel integrasi yang merepresentasikan pergeseran fungsi.
- t : Variabel waktu ketika nilai keluaran h(t) dihitung.

Sedangkan pada fungsi diskrit berdimensi satu, dapat ditulis sebagai berikut (Primartha, 2021):

$$\boldsymbol{h(t) = (f * g)(t) = \sum_{a} f(a)g(t-a)} \qquad (2.4)$$

Keterangan:

- h(t) : Hasil operasi konvolusi dari fungsi f dan g pada waktu t.

- $f(t)$: *Input* citra
- $g(t-a)$: Kernel konvolusi atau kernel filter
- a : Variabel penjumlah yang merepresentasikan pergeseran posisi kernel
- t : Indeks *diskret* tempat keluaran $h(t)$ dihitung.

Fungsi $h(t)$ menghasilkan *output* berupa peta fitur *(feature map)*. Argumen pertama dari fungsi ini adalah $f(t)$ yang merupakan *input* dan argumen kedua adalah $g(t)$ yang merupakan kernel atau filter. Apabila *input x* sebagai citra dua dimensi, maka dapat mengganti t dengan i dan j yang masing - masing mewakili koordinat piksel. Oleh karena itu, operasi konvolusi untuk *input* dengan lebih dari satu dimensi dapat dituliskan sebagai berikut (Zhang dkk., 2020):

$$s(i,j) = (K \otimes I)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(i-m, j-n) K(m,n) \tag{2.5}$$

$$s(i,j) = (K \otimes I)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(i+m, j+n) K(m,n) \tag{2.6}$$

Keterangan:

- i : Indeks baris (koordinat vertikal) pada *output* $s(i, j)$.
- j : Indeks kolom (koordinat horizontal) pada *output* $s(i, j)$.

- $s(i, j)$: Nilai hasil konvolusi pada posisi (i, j) dalam peta fitur atau *output* dari operasi konvolusi.
- m : Indeks baris pada kernel atau filter $K(m, n)$.
- n : Indeks kolom pada kernel atau filter $K(m, n)$.
- $K(i, j)$: Kernel atau filter di posisi (i, j).
- $I(i, j)$: Citra *input* pada koordinat (i, j).

Berdasarkan kedua persamaan diatas merupakan perhitungan dasar dalam operasi konvolusi, dimana ***i*** dan ***j*** adalah sebuah piksel pada citra. Perhitungan ini bersifat kumulatif dan terjadi ketika K berfungsi sebagai kernel, dimana kernel tersebut dapat dibalik relatif terhadap *input*. Sebagai alternatif, operasi konvolusi dapat diinterpretasikan sebagai perkalian matriks antara citra masukan dan kernel, di mana hasilnya dihitung melalui *dot product*. Selain itu, penentuan volume *output* juga dapat ditentukan berdasarkan setiap lapisan *hyperparameters*. Persamaan 2.7 digunakan untuk menentukan dimensi *output* dari suatu proses konvolusi, yang ditunjukkan pada persamaan sebagai berikut (Heryadi S Wahyono, 2021):

$$\frac{W - F + 2P}{S} + 1 \qquad (2.7)$$

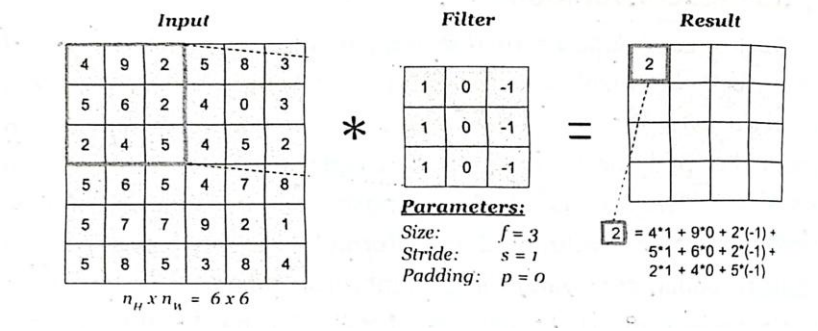
Keterangan:

- *W* : Ukuran volume gambar (dimensi *input*)
- *F* : Ukuran filter (dimensi filter) yang digunakan dalam proses konvolusi.
- *P* :Nilai *padding* yang tambahkan pada gambar *input* untuk menghindari pemangkasan informasi pada batas gambar.
- *S* : Ukuran pergeseran *(stride)*. yang digunakan untuk menentukan sejauh mana filter bergerak pada gambar *input*

Dari persamaan diatas, dapat dihitung ukuran spasial dari volume *output* dimana *hyperparameter* yang dipakai adalah ukuran volume (W), *filter* (F), *stride* yang diterapkan (S) dan jumlah *padding* / *zero padding* digunakan (P) untuk memperluas ukurannya tanpa memengaruhi informasi visual utama pada citra (Yafis S Rijal, 2023) . *Stride* adalah nilai yang digunakan untuk menggeser filter melalui *input* citra dan *zero padding* adalah nilai untuk menempatkan angka nol di sekitar border citra.

Convolutional Layer terdiri dari neuron yang disusun dalam bentuk filter dengan dimensi panjang dan tinggi *(pixels)*. Sebagai contoh, *layer* pertama pada *feature extraction layer* biasanya adalah lapisan konvolusi dengan ukuran $5 \times 5 \times 3$. Filter ini memiliki panjang 5 piksel, tinggi 5 piksel, dan kedalaman 3 yang sesuai

dengan *channel* gambar tersebut. Ketiga filter ini akan digeser ke seluruh bagian gambar. Pada setiap pergeseran, dilakukan operasi "*dot*" antara *input* dan nilai filter tersebut, menghasilkan *output* yang disebut sebagai peta aktivasi atau peta fitur.



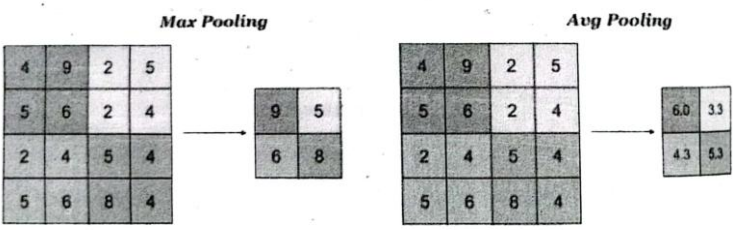
Gambar 2.9: *Convolution Layer* (Heryadi S Wahyono, 2021)

.

2. Operasi *Pooling*

Pooling merupakan proses dalam sebuah *feature map* yang digunakan untuk memilih sebuah nilai piksel yang mewakili sejumlah nilai piksel di sekitarnya dan mengabaikan piksel lainnya. Dua operasi *pooling* yang sering digunakan adalah *max pooling* dan *average pooling* (Heryadi S Wahyono, 2020). Nilai yang diambil pada *average pooling* adalah nilai rata-rata, sedangkan pada *max-pooling* adalah nilai maksimal. Lapisan *pooling* bekerja di setiap tumpukan *feature map* dan melakukan pengurangan pada ukurannya. Bentuk lapisan *pooling* umumnya dengan menggunakan filter dengan ukuran 2×2 yang diaplikasikan dengan langkah sebanyak dua dan beroperasi pada setiap

irisan dari *input*nya. Berikut ini adalah contoh gambar operasi *max-pooling* dan *average pooling*:



Gambar 2.10: Teknik *Max pooling* dan *Average pooling* (Heryadi S Wahyono, 2020)

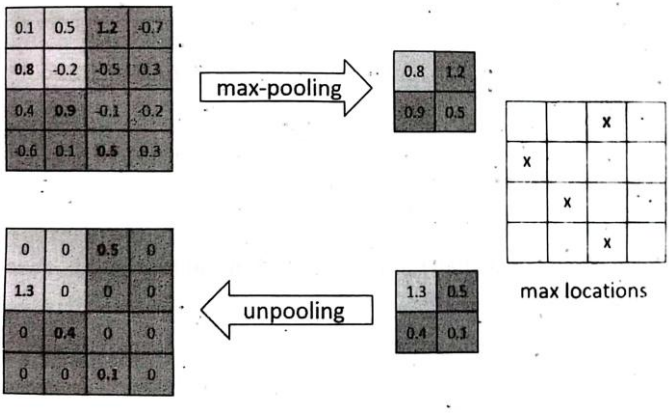
.

Berdasarkan gambar diatas menunjukkan dari proses *max-pooling*. *Output* dari proses *pooling* merupakan sebuah matriks dengan dimensi yang lebih kecil dibandingan dengan citra awal. Lapisan *pooling* ini beroperasi pada setiap irisan kedalaman volume *input* secara bergantian. Pada gambar tersebut, operasi *max-pooling* menggunakan filter berukuran 2×2. *input* dalam proses ini mempunyai ukuran sebesar 4×4. Dari setiap kelompok yang terdiri dari 4 nilai pada *input*, diambil nilai maksimalnya. Hasilnya diperoleh dengan ukuran *output* sebesar 2×2.

3. Operasi *Unpooling*

Unpooling merupakan kebalikan dari *pooling* dimana nilai hasil pada *pooling* (nilai *maxima*) dapat dikembalikan kedalam posisi semula di *feature map* dan mengisi nilai pixel lainnya dengan nilai 0 (Heryadi S Wahyono, 2020). Operasi *unpooling* dapat

diilustrasikan pada Gambar 2.11.

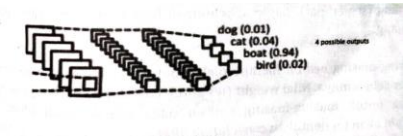


Gambar 2.11: Teknik *Unpooling* (Heryadi S Wahyono, 2020)

.

4. *Fully Connected Layer*

Fully connected layer merupakan *Multi Layer Perceptron* (MLP) yang menggunakan *softmax activation function* dan setiap neuron pada layer sebelumnya terkoneksi dengan setiap *neuron* pada layer berikutnya. Tujuan dari *fully connected layer* adalah untuk klasifikasi *(class) image* (Primartha, 2021), misalnya 4 buah klasifikasi gambar, seperti: anjing, kucing, perahu, dan burung.



Gambar 2.12: *Fully Connected Layer* (Primartha, 2021)

.

Pada proses *fully connected layer* ini, setiap neuron akan melakukan perhitungan secara matematis yang menggabungkan *input* dari lapisan sebelumnya dikalikan dengan bobot berbeda-beda yang akan diteruskan melalui fungsi aktivasi. Berikut ini merupakan persamaan *fully connected layer* (Wardani S Leonardi, 2023).

$$O_{(m)} = \sum_{i=0}^{in} (w_i)^{(m)} \times I_i \qquad (2.8)$$

Dimana:

- $O_{(m)}$:Nilai keluaran (*output*) pada posisi ke-*m*
- w : Bobot ke-*i* untuk posisi keluaran ke-*m*.
- I_i :Nilai *input* ke-*i*
- in : Panjang atau jumlah elemen dari vektor *input*

5. *Droput Regulation*

Droput regulation merupakan salah satu teknik regulasi jaringan syaraf tiruan bertujuan untuk mengurangi terjadinya *overfitting* dengan memilih beberapa neuron secara acak dan tidak

digunakan selama proses pelatihan, yang dimana neuron - neuron tersebut dibuang secara acak (Goodfellow dkk., 2016). Hal ini berarti neuron yang dipilih untuk diabaikan akan diberhentikan sementara dalam jaringan, dan pembobotan baru tidak akan diterapkan pada neuron tersebut saat proses *backpropagation*.

6. *Normalization Layer*

Normalization layer merupakan lapisan yang digunakan untuk mengatasi perbedaan rentang nilai yang signifikan pada citra masukan. Namun, saat ini lapisan normalisasi tidak banyak digunakan secara praktis karena dampaknya yang relatif kecil (Heryadi S Wahyono, 2020).

7. *Rectified Linear Units(ReLU) Layer*

Rectified Linier Units (ReLU) Layer dapat digunakan untuk mengaplikasikan fungsi aktivasi terhadap *input* (x) dari sebuah neuron. Oleh karena itu, operasi *Rectified Linier Units (ReLU) Layer* dapat dituliskan sebagai berikut (Beysolow, 2017):

$$f(x) = \max(0, x) \tag{2.9}$$

Keterangan:

- $f(x)$: *Output* dari fungsi ketika diberikan *input* x.
- $\max(0, x)$: Fungsi yang mengembalikan nilai yang lebih besar di antara 0 dan nilai *input* x.
- x : Nilai *input* sebuah neuron.

Rectified Linier Units (ReLU) Layer ini dapat meningkatkan sifat nonlinieritas fungsi keputusan dan jaringan secara keseluruhan tanpa mempengaruhi bidang - bidang reseptif pada *convolution layer* (Heryadi S Wahyono, 2020).

8. *Loss Layer*

Loss layer merupakan lapisan terakhir dalam CNN yang berfungsi untuk mengukur seberapa jauh hasil prediksi model dari nilai yang sebenarnya (label) dan memberikan penalti atau denda atas penyimpangan tersebut. Fungsi ini sangat penting karena menentukan seberapa baik model belajar dari data latihannya(Heryadi S Wahyono, 2020). Fungsi - fungsi yang diberikan sebagai berikut(Beysolow, 2017):

1. *Sigmoid*

   Sigmoid merupakan salah satu fungsi aktivasi yang digunakan untuk mengubah nilai *input* dalam rentang 0 sampai 1. Selain *sigmoid*, Fungsi aktivasi *softmax* juga menghasilkan *output* dengan nilai rentang 0 sampai 1. Namun, perbedaan utamanya terletak pada jumlah kelas yang dapat ditangani, antara lainnya adalah *softmax function* digunakan untuk klasifikasi dengan banyak kelas *(multi-class)*, sedangkan *sigmoid function* hanya dapat digunakan dengan kasus klasifikasi dengan dua kelas (biner)(Lederer, 2021). Fungsi *sigmoid* memiliki bentuk seperti berikut ini (Beysolow, 2017).

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad (2.10)$$

Keterangan:

- $\sigma(z)$: *Output* dari fungsi *sigmoid*. dari *input* *z*, dengan nilai probabilitas rentang 0 sampai 1.
- *z* :Nilai *input* dengan nilai probabilitas rentang 0 sampai 1.

2. *Softmax loss function*

Softmax loss function digunakan untuk memprediksi satu dari sejumlah kelas yang saling esklusif. *Softmax loss function* dirumuskan sebagai berikut (Beysolow, 2017):

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad (2.11)$$

Notasi $\sigma(z)_j$ menunjukkan bahwa *output* fungsi untuk setiap elemen ke-j dalam vektor keluaran kelas. Argumen *z* adalah hipotesis yang disediakan oleh model pelatihan untuk dapat diklasifikasikan oleh fungsi *softmax*. Selain memberikan hasil yang lebih intuitif, *softmax* juga memiliki interpretasi probabilistik yang lebih baik dibandingkan dengan algoritma klasifikasi lainnya. *Softmax* memungkinkan penghitungan probabilitas untuk semua label yang mungkin. Berdasarkan label yang tersedia, dihasilkan vektor nilai *real* yang kemudian diubah menjadi vektor dengan nilai antara nol dan satu, sehingga total nilai dari vektor tersebut adalah satu saat dijumlahkan.

3. *Euclidean loss function*

Euclidean loss function digunakan untuk mengukur jarak kuadrat rata-rata antara nilai prediksi dan nilai sebenarnya. *Euclidean loss function* dirumuskan sebagai berikut (Beysolow, 2017):

$$E = \frac{1}{2N} \sum_{i=1}^{N} ||\hat{y}_i - y_i||_2^2 \tag{2.12}$$

Keterangan:

- $\hat{y}_i$: Prediksi yang dihasilkan oleh model untuk sampel ke-*i*.
- y_i :Nilai sebenarnya dari sampel ke-*i*.
- N :Jumlah total sampel dalam dataset.
- E :Nilai fungsi kerugian (loss).
- $||\hat{y}_i - y_i||_2^2$:Norma L2, berfungsi untuk menghitung jarak Euclidean antara dua vektor.

4. *Softmax normalization*

Softmax normalization digunakan untuk mengklasterifikasi *multi-class* dalam menetapkan probabilitas desimal kesetiap kelas dalam sebuah sistem *machine learning*. *Softmax normalization* dirumuskan sebagai berikut (Beysolow, 2017):

$$x_i' = \frac{1}{1 + e^{-\frac{x_i - \mu_i}{\sigma_i}}} \tag{2.13}$$

Keterangan:

- x_i :Nilai awal dari fitur ke-i yang akan dinormalisasi.
- σ_i : Simpangan baku *(standard deviation)* dari vektor tersebut yang dihitung dengan rumus:

$$\sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{} (x_j - \mu_i)^2} \quad (2.14)$$

Keterangan:

- x_j :Nilai fitur pada data ke-j.
- μ_i : Rata-rata *(mean)* dari vektor tersebut.
- n :Jumlah total data

- x_i' :*Output* dari *softmax normalization* untuk *input* ke-*i*.
- e :Bilangan *Euler*, basis dari logaritma natural.

9. Kelebihan dan Kekurangan CNN

Dalam mencari solusi untuk menangani suatu permasalahan, penting untuk mempertimbangkan kelebihan serta kelemahan masing - masing metode yang harus di perhatikan. Selanjutnya pemilihan metode yang tepat harus didasarkan karakteristik data yang akan diolah.

Dalam kasus *Convolutional Neural Network* (CNN) ini, adapun berbagai studi yang telah menunjukkan kelebihan metode *Convolutional Neural Network* (CNN) dibandingkan dengan metode

lainnya. Selain itu, CNN mempunyai kelemahan. Berikut kelebihan dari metode *Convolutional Neural Network* (CNN), antara lain (Goodfellow dkk., 2016):

1. Metode CNN dapat memproses *input* dari berbagai tingkat spasial
2. Konvolusinya sangat mudah untuk diterapkan
3. Dirancang khusus untuk merancang gambar 2 dimensi.
4. Metode CNN memiliki tingkat akurasi yang sangat tinggi dalam pengklasifikasian
5. Kemampuannya untuk mengidentifikasi pola dalam data citra dengan sangat baik
6. Mengurangi dimensi data melalui teknik konvolusi
7. Dapat menangani data berwarna tanpa memerlukan praproses yang rumit

Disamping kelebihan, CNN juga mempunyai kelemahan atau keterbatasan, antara lain (Goodfellow dkk., 2016):

1. Kernel yang digunakan hanya dapat diterapkan dalam jumlah yang berbeda tergantung pada ukuran *input* dan *output* dari operasi konvolusinya berskala
2. Waktu pelatihan yang lama terutama dengan dataset besar dan jaringan yang dalam, serta membutuhkan sumber daya komputasi yang tinggi, seperti GPU, untuk pelatihan yang efisien
3. Kesulitan dalam ekstraksi fitur untuk data non-gambar

4. Metode CNN memerlukan sejumlah besar data berlabel untuk pelatihan yang efektif

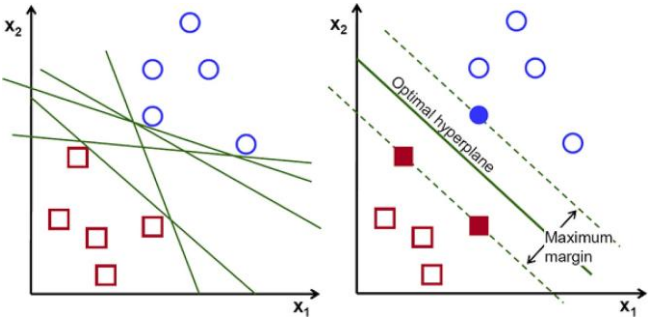
5. Metode CNN cenderung sensitif terhadap perubahan kecil dalam data *input*, seperti adanya noise.

E. *Support Vector Machine*(SVM)

Support Vector Machine (SVM) merupakan salah satu algoritma yang digunakan untuk prediksi, baik dalam kasus pengklasifikasian dengan menggunakan *hyperplane* (Dutt dkk., 2018). *Support Vector Machine* (SVM) berada dalam satu kelas dengan *Artificial Neural Network* (ANN). Metode tersebut masuk kedalam golongan *supervised learning*. *Support Vector Machine* (SVM) pertama kali diperkenalkan oleh Boser, Guyon, dan Vapnik pada tahun 1992 an di *Annual Workshop on Computational Learniing Theory (COLT)* (Boser dkk., 1992).

Konsep dasar dari metode *Support Vector Machine* (SVM) merupakan kombinasi dari berbagai teori komputasi yang ada pada tahun sebelumnya, kernel diperkenalkan oleh Aronszajn pada tahun 1950-an, *Lagrange Multiplier* ditemukan oleh Joseph Louis Lagrange pada tahun 1766, dan demikian juga dengan konsep - konsep pada pendukung yang lainnya. Prinsip dasar dari *Support Vector Machine* adalah sebagai *linier classification*, yang dimana dalam kasus *linier classification* mampu dipisahkan. Namun, *Support Vector Machine* (SVM) dibangun untuk memecahkan permasalahan non- linier ke dalam ruang kerja yang luas yakni melalui memasukan konsepsi kernel (Prasetyo, 2012). *Support Vector Machine* (SVM) bekerja dengan mencari *hyperplane* terbaik dengan memaksimalkan jarak antar kelas. *Hyperplane* merupakan

sebuah fungsi yang digunakan sebagai pemisah antar kelas yang ada (Brownlee, 2018).



Gambar 2.13: SVM berusaha menemukan hyperplane terbaik yang memisahkan kedua class –1 dan +1 (Brownlee, 2018)

.

Cara kerja *Support Vector Machine* (SVM) dimulai dengan tahap *input* data, dimana data citra gambar akan dimasukkan kedalam sistem. Citra gambar tersebut biasanya dalam format RGB dan diproses melalui beberapa tahap sebelum digunakan dalam model *Support Vector Machine* (SVM). Pada tahap pertama, dilakukan *preprocessing*, yang dimulai dengan mengubah gambar dari format RGB menjadi *grayscale* untuk mengurangi kompleksitas data dan memfokuskan pada intensitas cahaya (Mudhofar S Agustin, 2024). Selanjutnya, dilakukan *denoising*, yaitu penghapusan *noise* atau gangguan dalam gambar dengan menggunakan filter seperti *Gaussian Blur* atau *Median Filter* (Wardana dkk., 2024). Proses berikutnya adalah binerisasi, yang mengubah gambar grayscale menjadi gambar biner, di mana setiap piksel diubah menjadi dua nilai: 0 (hitam) dan 1 (putih), dengan tujuan memperjelas bentuk objek atau huruf dalam gambar

(Irfani S Gasim, 2024). Tahap akhir *preprocessing* adalah *thinning* yang bertujuan mengurangi ketebalan objek atau huruf tanpa mengubah bentuk atau struktur aslinya, memudahkan ekstraksi fitur dan klasifikasi (Harahap dkk., 2020).

Setelah tahap *preprocessing*, tahap augmentasi dengan tujuan untuk memperbanyak variasi data dengan cara memodifikasi gambar asli. Beberapa teknik augmentasi ini dengan memodifikasi citra, seperti membalik, memotong, memutar, dan mengubah skala, sehingga model dapat mengenali variasi data , serta dapat mengurangi *overfitting* (Fadillah dkk., 2024). Selanjutnya, data akan dibagi menjadi data *training*, dan *data uji* dengan rasio tertentu, misalnya 80%:20%, 90%:10%, dan 70%:30%. Tujuan dari pembagian ini adalah untuk melatih dan menguji model agar dapat menggeneralisasi dengan baik pada data baru. Data latih digunakan untuk melatih model SVM, sedangkan data uji digunakan untuk mengevaluasi kinerja model.

Pada tahap ekstraksi fitur, salah satu dalam ekstraksi fitur adalah *Histogram of Oriented Gradients* (HOG). Fitur HOG diambil dari augmentasi data yang sudah dibagi untuk menghasilkan representasi fitur yang lebih variatif dan kaya. Proses *Histogram of Oriented Gradient* (HOG) dimulai dengan membagi gambar yang telah diproses ke dalam sel-sel kecil dengan ukuran berukuran 8×8 piksel atau 16×16 piksel. Selanjutnya, dihitung gradien horizontal dan vertikal pada setiap sel menggunakan operator seperti Sobel, yang mengukur perubahan intensitas warna antar piksel (Putra dkk., 2024). Arah dan nilai gradien ini kemudian digunakan untuk membuat histogram orientasi yang menunjukkan distribusi arah gradien dalam setiap sel. Setelah pembuatan histogram, dilakukan normalisasi untuk mengurangi pengaruh perubahan

pencahayaan atau kontras yang dapat memengaruhi akurasi ekstraksi fitur. Vektor fitur akan dibentuk dengan menggabungkan histogram-histogram menjadi sebuah vektor panjang yang berisi informasi penting tentang karakteristik keseluruhan gambar, seperti tepi, bentuk, dan orientasi objek di dalamnya.

Vektor fitur yang dihasilkan digunakan dalam pelatihan model *Support Vector Machine* (SVM). *Support Vector Machine* (SVM) berusaha untuk mengidentifikasi sebuah *hyperplane* yang dapat membagi data ke dalam kelas-kelas yang berbeda. *Support Vector Machine* (SVM) berfokus pada margin, yaitu jarak antara *hyperplane* dan titik data yang paling dekat dari masing-masing kelas. Titik data yang paling dekat dengan *hyperplane* disebut sebagai *support vectors*, yang digunakan untuk menentukan posisi *hyperplane* (Deng dkk., 2013). *Lagrange multiplier* umumnya dilambangkan sebagai α_i untuk setiap titik data, digunakan dalam fungsi objektif yang diminimalkan oleh *Support Vector Machine* (SVM). Fungsi objektif berhubungan dengan margin dan kesalahan klasifikasi, sementara *Lagrange multiplier* berperan dalam memperhitungkan kendala-kendala yang ada dalam model *Support Vector Machine* (SVM). Titik data dengan nilai *Lagrange multiplier* yang lebih tinggi lebih berpengaruh dalam menentukan posisi *hyperplane* yang disebut *support vectors*.

Pada data nonlinier, *Support Vector Machine* (SVM) dapat menggunakan teknik kernel untuk mengubah data menjadi dimensi yang lebih tinggi yang dapat dipisahkan oleh *hyperplane* (Sakhdiah dkk., 2024). Kernel yang umumnya digunakan dalam SVM, antara lainnya adalah linear, *polynomial*, dan *Radial Basis Function* (RBF) (Primartha, 2021). Pemilihan *support vector* dilakukan dalam ruang dimensi yang lebih tinggi. Teknik

kernel ini dapat digunakan dalam menghadapi masalah data yang non linier di ruang fitur asli, yang merupakan tantangan utama dalam pengenalan pola. Setelah model SVM dilatih dan hyperplane ditemukan, model *Support Vector Machine* (SVM) akan digunakan untuk klasifikasi dan memprediksi pada kelompok data cita gambar. Setelah proses pelatihan selesai, evaluasi model SVM dilakukan untuk menilai sejauh mana model dapat mengklasifikasikan citra gambar dengan benar. Tahapan dalam evaluasi yang umum digunakan untuk menilai performa model SVM, antara lainnya adalah *confusion matrix*, *recall*, *precission, accuracy*, dan *F1-Score* (Permana dkk., 2022).

1. *Pattern Recognition* menggunakan *Support Vector Machine*

Konsep *Support Vector Machine* (SVM) adalah mencari *hyperplane* terbaik yang berfungsi sebagai pemisah antara dua kelas data (Brownlee, 2018). Pada parameter *Support Vector Machine* (SVM) mempunyai nilai akurasi lebih baik dibandingkan dengan nilai 0.

Pada Gambar 2.13 menunjukkan bahwa beberapa *pattern* merupakan anggota dari buah kelas, yaitu +1 dan -1. *Pattern* yang tergabung dalam kelas -1 disimbolkan dengan warna merah (kotak), sedangkan *pattern* pada kelas +1 disimbolkan dengan warna biru (lingkaran). Problem klasifikasi dapat diterjemahkan dengan menemukan garis (*hyperplane*) yang memisahkan kedua kelasehingga jarak antara kedua kelas tersebut dapat diukur dengan margin *hyperplane* dan mencari titik maksimalnya (Brownlee, 2018). Margin merupakan jarak antara *hyperplane* dengan *pattern* terdekat dari masing-masing kelas. *Pattern* yang paling dekat ini disebut sebagai *Support Vector* (SV) (Brownlee,

2018).

2. Metode Kernel

Pada teknik data *mining* dan *machine learning* banyak dikembangkan dengan asumsi kelinieran, sehingga algoritma yang dihasilkan terbatas untuk kasus - kasus linier. Oleh karena itu, jika suatu kasus klasifikasi memperlihatkan ketidaklinieran, algoritma seperti *perceptron* yang tidak dapat untuk mengatasinya. Secara umum, kasus pada dunia nyata pada non linier adalah data tersebut tidak dapat dipisahkan secara linier. Namun, tidak semuanya harus diklasifikasi secara nonlinier. Jika ada data yang diinputkan dengan jelas dipisah menjadi dua kategori yang berbeda, maka data tersebut dapat diproses secara linier. Pada data nonlinier, *Support Vector Machine* (SVM) dapat menggunakan metode kernel untuk mengubah data menjadi dimensi yang lebih tinggi yang dapat dipisahkan oleh *hyperplane* (Sakhdiah dkk., 2024). Fungsi kernel yang umum digunakan pada metode *Support Vector Machine* (SVM) adalah sebagai berikut (Bishop dkk., 2002).

1. Kernel Linier

   Kernel Linier digunakan untuk mengukur *dot product* dari dua vektor *input* dalam ruang asli tanpa melakukan transformasi ke ruang fitur yang lebih tinggi. Kernel linier dapat dirumuskan sebagai berikut (Rabbani dkk., 2023):

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i} \cdot \boldsymbol{x_j} \qquad (2.15)$$

   Keterangan:

   - $\boldsymbol{x_i}$: Vektor fitur dari data ke-i (misalnya gambar ke-1)

- x_j : Vektor fitur dari data ke-j (misalnya gambar ke-2)

2. Kernel Polinomial

Kernel Polinomial digunakan untuk mengukur hubungan polinomial antara dua vektor *input* dalam ruang asli (Rabbani dkk., 2023). Kernel polinomial dapat dirumuskan sebagai berikut (Bishop dkk., 2002):

$$K(x_i, x_j) = (x_i \cdot x_j)^d \quad (2.16)$$

Keterangan:

- $K(x_i, x_j)$: Kernel antara dua vektor fitur dari data ke-*i* (misalnya gambar ke-1) dan data ke-*j* (misalnya gambar ke-2).
- x_i :Vektor fitur dari data ke-*i* (misalnya gambar ke-1).
- x_j :Vektor fitur dari data ke-*j* (misalnya gambar ke-2).
- d : Derajat polinomial yang digunakan untuk menentukan tingkat kompleksitas model.

3. *Radial Basis Function* (RBF)

Kernel *Radial Basis Function* (RBF) atau dikenal sebagai kernel Gaussian merupakan salah satu fungsi kernel dalam *Support Vector Machine* (SVM) yang digunakan untuk menangani data yang kompleks dan tidak dapat dipisahkan secara linier. Selain itu, kernel ini digunakan untuk mengukur tingkat kemiripan antar titik data berdasarkan jarak di

antara keduanya, di mana semakin dekat dua titik, semakin besar nilai kemiripannya (Pal dkk., 2024).

Kernel *Radial Basis Function* (RBF) berasal dari fungsi Gaussian yang digunakan untuk menggambarkan distribusi normal. Fungsi Gaussian dalam bentuk distribusi probabilitas untuk satu variabel *x* dapat dituliskan dengan persamaan sebagai berikut (Bishop dkk., 2002).

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \quad -\frac{(x - \mu)^2}{2\sigma^2} \tag{2.17}$$

Di mana:

- μ : Nilai rata-rata atau pusat distribusi
- σ : Simpangan baku yang digunakan untuk menentukan lebar distribusi
- *exp* : Konstanta bilangan euler(2,178...)
- π : Konstanta pi ($\frac{22}{7}$ atau mendekati 3,142.)
- *x* : Nilai dari variabel acak

Pada penerapan kernel *Radial Basis Function* (RBF), yang dimanfaatkan hanya komponen eksponensial dari fungsi Gaussian, karena tujuan utamanya adalah mengukur tingkat kedekatan atau kesamaan antara dua titik data, tidak digunakan untuk merepresentasikan distribusi probabilitas. Oleh sebab itu, bentuk kernel RBF disederhanakan menjadi (Bishop dkk., 2002):

$$K(x_i, x_j) = \exp \quad -\frac{\|x_i - x_j\|^2}{2\sigma^2} \tag{2.18}$$

Keterangan:

- $K(x_i, x_j)$ adalah nilai kernel antara dua vektor data x_i dan x_j.
- Jarak Euclidean antara dua vektor data x_i dan x_j, yang dihitung dengan rumus (Bishop dkk., 2002):

$$\|x_i - x_j\| = \sqrt{\sum_{k=1} (x_{i,k} - x_{j,k})^2} \quad (2.19)$$

  Keterangan :

  - i dan j adalah indeks dari dua vektor data yang dibandingkan, dengan $i, j \in \{1, 2, \ldots, n\}$.
  - k adalah indeks untuk elemen-elemen dalam vektor x_i dan x_j, dengan k berkisar dari 1 hingga n.

- σ adalah parameter yang mengontrol lebar dari fungsi Gaussian yang digunakan untuk mengukur kedekatan antara dua vektor data. Nilai σ mempengaruhi seberapa sensitif model terhadap jarak antar titik. Jika nilai σ besar, maka titik yang relatif lebih jauh dapat dianggap mirip. Sebaliknya, jika σ kecil, maka hanya pasangan titik dengan jarak yang dekat dianggap memiliki kemiripan tinggi.

4. Kernel *Sigmoid*

Kernel *Sigmoid* digunakan untuk menangani data kompleks dan non-linear (Rabbani dkk., 2023). Kernel *sigmoid* dapat dirumuskan sebagai berikut (Bishop dkk.,

2002):

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(\sigma(\boldsymbol{x_i} \cdot \boldsymbol{x_j}) + coef) \qquad (2.20)$$

Keterangan:

- $\boldsymbol{K(x_i, x_j)}$: Kernel antara dua vektor fitur x_i dan x_j
- **tanh** : Fungsi aktivasi hiperbolik tangensial (*hyperbolic tangent*) yang menghasilkan nilai dalam rentang $[-1, 1]$, berfungsi untuk memproyeksikan hasil operasi kernel ke dalam ruang non-linear.
- $\boldsymbol{\sigma}$: Parameter skalar yang berfungsi untuk mengatur bobot atau sensitivitas terhadap hasil *dot product* antara dua vektor.
- $\boldsymbol{x_i} \cdot \boldsymbol{x_j}$: Hasil perkalian dot *(dot product)* antara dua vektor $\boldsymbol{x_i}$ dan $\boldsymbol{x_j}$
- ***coef*** : Konstanta bias, berfungsi untuk mengubah posisi keputusan *(decision boundary)* pada ruang fitur.

Setelah melakukan perhitungan kernel SVM, dilakukan proses pembelajaran SVM. Berikut ini merupakan proses pembelajaran SVM (Samsuri, 2022).

1. Menginisialisasi parameter α, γ, λ, C, ϵ.
   - α : *Alpha*, berfungsi untuk mengidentifikasi *support vector*.

- γ : *Gamma*, berfungsi untuk mengendalikan kecepatan *training*. Nilai *gamma* yang terlalu besar dapat menyebabkan pelatihan tidak stabil, sementara nilai yang terlalu kecil memperlambat konvergensi.
- λ : *Lambda*, berfungsi untuk mengatur regularisasi guna mengurangi kompleksitas model.
- C : *Complexity*, berfungsi untuk memberikan batasan nilai *alpha*.
- ϵ : *Epsilon*, berfungsi untuk menentukan margin toleransi terhadap kesalahan pada prediksi dan membatasi kontribusi kesalahan klasifikasi kecil terhadap fungsi tujuan tanpa memberikan penalti yang berlebihan. Penelitian ini menggunakan *inisialisasi* ϵ= 0,00000001 yang menunjukkan bahwa toleransi kesalahan yang sangat kecil dan memungkinkan model untuk lebih sensitif terhadap deviasi minimal pada data. Berdasarkan penelitian Oktaviana dkk. (2022) membuktikan bahwa penggunaan *inisialisasi* ϵ sebesar 0,00000001 dapat meningkatkan nilai *accuracy* secara signifikan.

2. Melakukan perhitungan matriks Hessian yang berfungsi

untuk mengidentifikasi optimum relatif suatu nilai fungsi. Berikut ini merupakan persamaan matriks Hessian.

$$D_{ij} = y_i y_j \quad K(x_i, x_j) + \lambda^2 \qquad (2.21)$$

Keterangan:

- x_i : Data ke-i
- x_j : Data ke-j
- y_i : Label kelas data ke-i
- y_j : Label kelas data ke-j
- $K(x_i, x_j)$: Kernel antara dua vektor fitur dari data ke-i (misalnya gambar ke-1) dan data ke-j (misalnya gambar ke-2)
- λ : *Lambda*, parameter yang mengontrol kompleksitas model

3. Menghitung nilai E_i, $\delta\alpha_i$, α_i pada data yang dipakai mulai dari data ke-i sampai ke-j dengan persamaan berikut ini.

   - Menghitung nilai E_i

$$E_i = \sum_{j=1}^{n} \alpha_j D_{ij} \qquad (2.22)$$

   Keterangan:

   – α_i : *Alpha* ke-i, berfungsi untuk mengidentifikasi *support vector*.

- D_{ij} : Matriks Hessian, berfungsi untuk menghitung pengaruh *support vector* ke-***j*** terhadap *error* pada data ke-***i*** serta menentukan gradien dan memaksimalkan margin antara kelas-kelas yang ada.
- E_i : *Error rate*, berfungsi untuk menggambarkan tingkat kesalahan atau *error* yang terjadi pada data ke-i dalam model dan menilai seberapa baik model dalam memprediksi atau mengklasifikasikan data.
- n : Jumlah data

- Menghitung nilai $\delta\alpha_i$

$$\delta\alpha_i = \min\{\max[\gamma(1 - E_i), \alpha_i], C - \alpha_i\} \quad (2.23)$$

Keterangan:

- α_i : *Alpha* ke-i, berfungsi untuk mengidentifikasi *support vector*.
- γ : *Gamma*, berfungsi untuk mengendalikan kecepatan *training*. Nilai *gamma* yang terlalu besar dapat menyebabkan pelatihan tidak stabil, sementara nilai yang terlalu kecil memperlambat konvergensi.

- E_i : *Error rate*, berfungsi untuk menggambarkan tingkat kesalahan atau *error* yang terjadi pada data ke-i dalam model dan menilai seberapa baik model dalam memprediksi atau mengklasifikasikan data.

- Menghitung nilai α_i

$$\alpha_i = \alpha_i + \delta\alpha_i \quad (2.24)$$

Keterangan:

- α_i :*Alpha* ke-i, berfungsi untuk mengidentifikasi *support vector*
- $\delta\alpha_i$:*Delta alpha* ke-i, berfungsi sebagai nilai perubahan terhadap α_i pada iterasi tertentu.

4. Proses perhitungan nilai E_i, $\delta\alpha_i$, α_i dilakukan berulang kali sampai mencapai iterasi maksimum.

5. Pada langkah terakhir diperoleh nilai *Support Vector (SV)*, SV bernilai sama dengan α_i yang mempunyai nilai lebih besar dari *ThresholdSV* (misalnya *ThresholdSV=0*).

Setelah dilakukan perhitungan *training* SVM, maka akan diketahui hasil parameter *alpha* yang akan digunakan dalam proses *testing*. Dalam perhitungan proses klasifikasi tahap *testing* dibutuhkan nilai *alpha* dan nilai bias. Berikut ini merupakan langkah-langkah dalam prosesnya (Samsuri, 2022).

- Menentukan nilai bias (*b*) terlebih dahulu.

$$b = -\frac{1}{2}\left[\sum_{i=1}^{m} \alpha_i y_i K(x_i, x^+) + \sum_{i=1}^{m} \alpha_i y_i K(x_i, x^-)\right] \quad (2.25)$$

Keterangan:

- α_i : *Alpha* ke-i, berfungsi untuk mengidentifikasi *support vector*
- y_i : Label kelas dari data ke-i
- $K(x_i, x^+)$: Fungsi kernel antara data ke-*i* dan data dengan kelas label positif
- $K(x_i, x^-)$: Fungsi kernel antara data ke-*i* dan data dengan kelas label negatif
- m : Jumlah total data dari data *training*
- b : Bias yang diperoleh dari hasil perhitungan terhadap *support vector*, digunakan untuk membentuk fungsi keputusan pada SVM

- Selanjutnya, setelah mendapatkan nilai bias (*b*) dari persamaan diatas maka akan dilakukan tahap *testing* dalam klasifikasi dengan memasukkan nilai *alpha* hasil perhitungan *training* SVM.

$$f(x) = \sum_{i=1}^{m} \alpha_i y_i K(x_{testing}, x_i) + b \quad (2.26)$$

Dalam penelitian ini, jenis SVM yang digunakan adalah SVM *multiclass* dengan metode *one against all*. Metode ini

dikembangkan berdasarkan jumlah kelas *k* dalam dataset. *One against all* melibatkan pembentukan *k* model SVM terpisah, di mana setiap model dilatih untuk membedakan satu kelas dari kelas lainnya. Setiap model SVM bertujuan mempelajari fitur dan pola yang paling sesuai untuk kelas yang sedang dipelajari tanpa gangguan dari kelas lain. Pendekatan ini mampu menghasilkan hasil klasifikasi yang lebih akurat.

3. Karakteristik *Support Vector Machine* (SVM)

Karakteristik *Support Vector Machine* (SVM) secara umum sebagai berikut (Putra dkk., 2020):

1. Secara prinsip, SVM adalah *linear classifier*.
2. *Pattern recognition* dilakukan dengan mentransformasikan data pada ruang *input (input space)* ke ruang yang berdimensi lebih tinggi *(feature space)*, dan optimisasi dilakukan pada ruang vector yang baru tersebut. Hal ini membedakan SVM dari solusi *pattern recognition* pada umumnya, yang melakukan optimisasi parameter pada hasil transformasi yang berdimensi lebih rendah daripada dimensi *input space*.
3. Menerapkan strategi *Structural Risk Minimization (SRM)*.
4. Prinsip kerja SVM pada dasarnya hanya mampu menangani klasifikasi dua kelas, meskipun demikian telah dikembangkan untuk klasifikasi lebih dari dua kelas dengan adanya *pattern recognition*.

4. Kelebihan dan Kekurangan *Support Vector Machine (SVM)*

Dalam penentuan solusi untuk menangani suatu permasalahan, penting untuk mempertimbangkan kelebihan serta kelemahan masing - masing metode yang harus diperhatikan. Selanjutnya pemilihan metode yang tepat harus didasarkan yang cermat terhadap karakteristik data yang akan diolah.

Dalam kasus *Support Vector Machine* (SVM) ini, adapun berbagai studi yang telah menunjukkan kelebihan metode SVM dibandingkan dengan metode - metode yang lain. Selain itu, SVM juga mempunyai kelemahan. Berikut kelebihan dari metode *Support Vector Machine* (SVM), antara lain (Sendhilkumar S Geetha, 2023):

1. SVM merupakan metode yang baik secara matematis sehingga menjadikannya salah satu metode dalam *machine learning* yang paling akurat.
2. SVM dapat menangani berbagai masalah, termasuk masalah linier dan nonlinier, serta masalah klasifikasi biner, binomial, dan multi-kelas, baik itu masalah klasifikasi atau regresi.
3. SVM didasarkan pada konsep memaksimalkan margin antar kelas dan karenanya membedakan kelas-kelas dengan baik.
4. Model SVM dirancang untuk mengurangi *overfitting*, sehingga membuat model menjadi sangat stabil.
5. SVM lebih efektif dalam ruang dimensi tinggi.
6. SVM memiliki komputasi yang cepat dan relatif untuk manajemen memori menjadi hemat.

Disamping kelebihan, SVM juga mempunyai kelemahan atau keterbatasan, antara lain (Sendhilkumar S Geetha, 2023):

- SVM sulit dalam memilih fungsi solusi kernel yang sesuai.
- Waktu pelatihan metode SVM lebih lama saat menggunakan dataset yang berukuran besar.
- SVM tidak bekerja dengan baik ketika kumpulan data memiliki lebih banyak noise, yaitu ketika kelas target saling tumpang tindih.
- Sulit untuk memahami dan menginterpretasikan model akhir, bobot variabel
- Performa SVM sensitif terhadap kernel yang dipilih.
- SVM tidak dapat diinterpretasikan dengan baik, khususnya ketika kernel digunakan untuk menangani data yang terpisah-pisah secara nonlinier.
- Biaya komputasi tinggi dalam menyetel *hyperparameter*, terutama ketika berhadapan dengan kumpulan data yang sangat besar.

5. Optimasi Parameter Model

Hyperparameter tuning mempunyai peran penting dalam mengoptimalkan kinerja algoritma dalam *Machine Learning* (ML) (Nugraha S Sasongko, 2022) . *Hyperparameter tuning* digunakan untuk mencari nilai - nilai parameter optimal sebelum model dilatih, karena tidak dapat dioptimalkan untuk menghasilkan hasil yang lebih akurat. Pemilihan *hyperparameter* pada SVM

sangat berpengaruh pada performa dari algoritma SVM. Penelitian ini menerapkan teknik *Grid Search* untuk menentukan nilai *hyperparameter* terbaik pada model. *Grid Search Cross Validation* mengacu pada kombinasi teknik *Grid Search* dan *Cross-Validation*, yang merupakan metode untuk memilih kombinasi model dan *hyperparameter* dengan menguji setiap kemungkinan kombinasi secara sistematis serta melakukan validasi pada masing-masing kombinasi tersebut. Dalam perhitungan parameter gamma, terdapat dua metode penghitungan yang berbeda yaitu *scale* dan *auto* (penghitungan metode terbaik akan dipilih melalui *hyperparameter tuning*). Dalam penelitian ini, dihitung berdasarakan karakteristik data (jumlah fitur S variasi data). Pilihan metode ini akan mempengaruhi cara menghitung nilai sigma (σ). Nilai sigma adalah konsekuensi dari nilai gamma yang diperoleh (Bishop dkk., 2002).

$$\text{Scale} = \frac{1}{\text{jumlah dimensi fitur} \times \text{jumlah variasi fitur}} \tag{2.27}$$

Keterangan:

- Jumlah dimensi fitur : Total jumlah dimensi dari fitur yang dihasilkan.
- Jumlah variasi fitur :Banyaknya variasi dalam fitur.

$$\text{Auto} = \frac{1}{\text{jumlah dimensi fitur}} \tag{2.28}$$

Keterangan:

- Jumlah dimensi fitur : Total jumlah dimensi dari fitur yang dihasilkan.

$$\sigma = \sqrt{\frac{1}{(\text{scale} \times 2)}} \quad (2.29)$$

$$\sigma = \sqrt{\frac{1}{(\text{auto} \times 2)}} \quad (2.30)$$

Keterangan:

- σ : Parameter standar deviasi, dihitung berdasarkan nilai *scale* atau *auto* yang digunakan untuk mengatur skala sebaran data, yang dimana semakin kecil σ, semakin sempit pengaruh satu titik data terhadap yang lain.

F. *Histogram of Oriented Gradients*(HOG)

Histogram of Oriented Gradients (HOG) merupakan salah satu teknik dalam ekstraksi fitur yang bertujuan untuk mengambil informasi penting dari sebuah citra. HOG sering digunakan dalam mengidentifikasi hewan, kendaraan, tulisan, wajah, dan lainnya. Proses ekstraksi fitur HOG melibatkan perhitungan *gradient orientation* dalam area gambar yang terlokalisasi (Putra dkk., 2024).

Fitur *Histogram of Oriented Gradients* (HOG) pada setiap sel diperoleh dengan mengakumulasi nilai *gradient magnitude* berdasarkan orientasi yang searah. Metode HOG dimulai dengan menghitung gradien objek pada arah horizontal dan vertikal (*x, y*) dengan operator Sobel yang bertujuan untuk mendeteksi tepian fitur pada gambar secara horizontal dan vertikal (Supriyadi dkk., 2024). Misalnya, pada citra gambar dengan skala abu -

abu 8 bit (rentang nilai 0-255), perhitungan gradien horizontal (sumbu *x*) dilakukan penggeseran dari sisi kiri latar belakang menuju sisi kanan hingga ditemukan objek yang menunjukkan perbedaan signifikan dalam intensitas piksel, seperti perubahan nilai intensitas yang tajam dari rendah ke tinggi (gradien positif). Selanjutnya, gradien akan melintasi objek secara linier hingga kembali mencapai area latar belakang, yang ditandai dengan penurunan nilai intensitas (gradien negatif). Tahapan awal dalam ekstraksi fitur HOG adalah menghitung nilai gradien yang dapat dilihat pada persamaan berikut (Putra dkk., 2024).

$$G_x = I(x, y) \circledast \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.31}$$

Keterangan:

- $I(x, y)$: *Input* matriks citra
- G_x : Matriks operator Sobel sisi horizontal

$$G_y = I(x, y) \circledast \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.32}$$

Keterangan:

- $I(x, y)$: *Input* matriks citra
- G_y : Matriks operator Sobel sisi vertikal

Selanjutnya menghitung nilai *gradient magnitude* dan *gradient orientation* setiap piksel pada *cell* dengan menggunaan persamaan berikut ini(Tang dkk., 2020).

$$\text{Magnitude}(\mu) = \sqrt{G_x^2 + G_y^2} \quad (2.33)$$

Keterangan:

- μ : *Gradient magnitude*
- G_x : Matriks operator Sobel sisi horizontal
- G_y : Matriks operator Sobel sisi vertikal

Setelah melakukan *gradient magnitude*, gradien akan ditransformasikan ke dalam koordinat sumbu dengan sudut diantara 0° sampai 180° yang disebut *gradient orientation*. *Gradient orientation* dapat dihitung menggunakan persamaan dibawah ini.

$$\text{Angle}(\vartheta) = \tan^{-1} \frac{G_y}{G_x} \quad (2.34)$$

Keterangan:

- ϑ : *Gradient orientation*
- G_x : Matriks operator Sobel sisi horizontal
- G_y : Matriks operator Sobel sisi vertikal

Setelah melakukan perhitungan *gradient orientation*, langkah selanjutnya adalah menghitung histogram orientasi gradien untuk setiap sel. Setiap piksel dalam sebuah sel berkontribusi pada nilai histogram sesuai dengan hasil perhitungan gradiennya. Setelah itu, dilakukan proses normalisasi pada setiap blok untuk meningkatkan ketahanan fitur terhadap variasi pencahayaan. Normalisasi fitur blok bertujuan untuk meminimalkan pengaruh variasi tingkat kecerahan objek di dalam satu blok. Persamaan

berikut menunjukkan proses normalisasi fitur blok (Rivan dkk., 2020).

$$L_1\text{-norm}: \quad v' = \frac{v}{\sqrt{\|v\|_1^2 + \epsilon}} \tag{2.35}$$

$$L_2\text{-norm}: \quad v'' = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon}} \tag{2.36}$$

emiliki hist

v adalah fitur blok yang m ogram, ϵ adalah konstanta dengan nilai 10^{-6} .

G. Evaluasi Model

1. *Confusion Matrix*

Confusion matrix merupakan tabel yang digunakan menggambarkan kinerja model pengklasifikasian, serta menghitung berapa kali pada contoh kelas *p* diklasifikasikan sebagai kelas *x* (Permana dkk., 2022).

		Nilai Prediksi	
		Positive	Negative
Nilai Aktual	Positive	True Positive (TN)	False Negatif (FN)
	Negative	False Positive (FP)	True Negatif (TN)

Gambar 2.14: *Confusion matrix* (Permana dkk., 2022)

.

Dimana:

- *True Positives*(TP) :Nilai sebenarnya yang positif, sedangkan nilai prediksinya

menunjukan positif.

- *False Positive*(FP) :Nilai sebenarnya negatif, sedangkan nilai prediksinya menunjukkan positif.
- *True Negative* (TN): Nilai sebenarnya negatif, sedangkan nilai prediksinya menunjukan positif.
- *False Negative* (FN):Nilai sebenarnya positif, sedangkan nilai prediksinya menunjukan negatif.

2. *Precision*

Precision merupakan proporsi data positif yang dapat diklasifikasikan dengan nilai yang benar dari total jumlah hasil prediksi yang bernilai positif (Permana dkk., 2022). Berikut ini merupakan rumus dalam menghitung nilai *precision*(Permana dkk., 2022).

$$Precision = \frac{TP}{TP + FP} \tag{2.37}$$

Dimana:

- ***True Positive*(TP)** :Nilai sebenarnya yang positif, sehingga nilai prediksinya juga positif.
- ***False Positive* (FP)**: Nilai sebenarnya negatif, sedangkan nilai prediksinya menunjukkan positif..

3. *Recall*

Recall merupakan rasio dengan jumlah data positif (TP) yang diprediksi dengan benar terhadap jumlah data positif aktual

(Permana dkk., 2022). Berikut ini merupakan rumus dalam menghitung nilai *recall* (Permana dkk., 2022).

$$Recall = \frac{TP}{TP + FN} \tag{2.38}$$

Dimana:

- ***True Positive*(TP)** :Nilai sebenarnya yang positif, sehingga nilai prediksinya juga positif.
- ***False Negative* (FN)**: Nilai sebenarnya positif, sedangkan nilai prediksinya menunjukan negatif.

4. *Accuracy*

Accuracy merupakan parameter pengukuran *fundamental* yang paling mendasar, dimana dalam kasus klasifikasi biner (Permana dkk., 2022). Berikut ini merupakan rumus untuk menghitung nilai *Accuracy*(Permana dkk., 2022).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.39}$$

Dimana:

- ***True Positive* (TP)** :Nilai sebenarnya yang positif, sehingga nilai prediksinya juga positif.
- ***False Positive* (FP)** : Nilai sebenarnya negatif, sedangkan nilai prediksinya menunjukkan positif.

- ***True Negative* (TN)**: Nilai sebenarnya negatif, sedangkan nilai prediksinya menunjukan positif.
- ***False Negative*(FN)**: Nilai sebenarnya positif, sedangkan nilai prediksinya menunjukan negatif.

5. *F1- Score*

F1 - Score digunakan untuk menghitung hubungan antara nilai *Precision* dan nilai *Recall*(Permana dkk., 2022). Berikut merupakan rumus untuk menghitung nilai *F1- Score* (Permana dkk., 2022).

$$F1\text{- }Score = \frac{Precision \times Recall}{Precission + Recall} \tag{2.40}$$

H. *Optical Character Recognition (OCR)*

Optical Character Recognition (OCR) merupakan metode dalam bidang pengolahan citra dan *Computer Vision* (CV) yang bertujuan untuk mengenali dan mengekstraksi huruf serta gambar untuk dikonversi menjadi bentuk teks digital (Hanif dkk., 2023). Aplikasi OCR sering digunakan dalam pendokumentasian citra, seperti pembuatan koleksi pustaka digital, koleksi sastra kuno digital, dan lain-lain (Hanif dkk., 2023). Dalam proses *Optical Character Recognition* (OCR) dapat dilihat sebagai berikut (Utami, 2021):

1. *Data Capture*

   Data capture merupakan proses dimana suatu dokumen *(hardcopy)* menjadi suatu file gambar berbentuk digital.

2. *Preprocessing*

   Preprocessing merupakan suatu proses yang digunakan untuk menghilangkan bagian - bagian yang tidak digunakan kembali pada gambar *input* untuk proses selanjutnya.

3. Segmentasi

   Segmentasi merupakan proses untuk memisahkan area pengamatan *region* pada tiap karakter yang akan dideteksi.

4. Normalisasi

   Normalisasi merupakan proses untuk mengubah dimensi *region* pada tiap karakter.

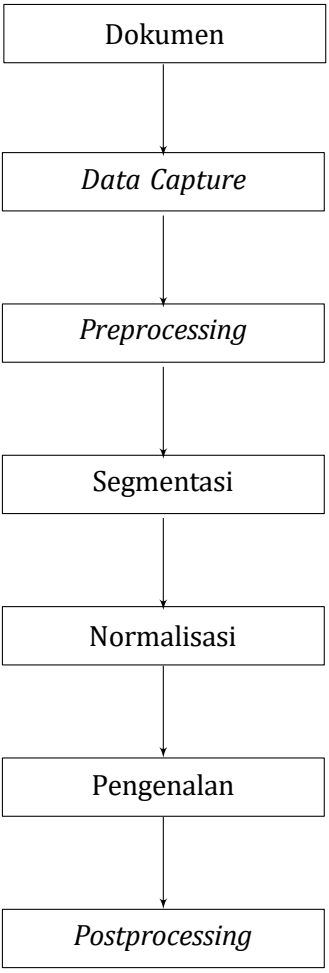
5. Pengenalan

   Pada tahap ini digunakan untuk mengenali karakter yang diamati dengan cara membandingkan ciri - ciri karakter yang diperoleh dengan ciri - ciri karakter yang ada pada database. Karakter huruf, angka yang disimpan pada *terseract master*.

6. *Postprocessing*

   Pada tahap ini digunakan untuk mengoreksi ejaan sesuai dengan bahasa yang digunakan.

Sistem *Optical Character Recognition* (OCR) berjalan sesuai dengan Flowchart yang dibangun. Flowchart atau alur sistem *Optical Character Recognition* (OCR) dapat dilihat pada gambar 2.15.



Gambar 2.15: Struktur *Optical Character Recognition (OCR)*

I. *Google Colaboratory*

Google Colaboratory merupakan platform berbasis *could* yang memanfaatkan *Notebook Jupyter* untuk memfasilitasi penelitian dan pendidikan *machine learning* (Rizky S Andarsyah, 2023).

Google Colaboratory dilengkapi dengan *runtime* yang secara khusus dikonfigurasi untuk *deep learning task*. Keuntungan dalam menggunakan *Colab* adalah menawarkan akses gratis ke sumber daya komputasi yang handal, seperti GPU dan TPU yang berguna untuk melatih model *machine learning* yang besar (Rizky S Andarsyah, 2023).

J. Tulisan Korea (Huruf *Hangeul*)

Pada tahun 1443, Raja Sejong dan beberapa ilmuwan meenciptakan huruf *Hangeul* guna untuk meningkatkan literasi di Korea (Rachmawati dkk., 2023). *Hangeul* merupakan sistem alphabet dalam penulisan Korea. Sebelum *Hangeul* digunakan, masyarakat Korea menggunakan aksara Cina yang kompleks, rumit dan sulit dipelajari sehingga menyulitkan masyarakat Korea untuk menggunakannya. Raja Sejong, yang menunjukkan belas kasihan kepada rakyatnya dengan menciptakan sistem alfabet Korea yang mudah dipelajari yang melambangkan bunyi bahasa Korea. Sistem alphabet ini disebut dengan *"Hangeul"*. Pada awalnya, *Hangeul* sering disebut '훈 민 정 음 *(Hunminjeongeum)*' yang berarti suara yang benar untuk diajarkan kepada rakyat atau 'tulisan untuk rakyat" (Sam, 2023).

Huruf *Hangeul* berdasarkan kategorinya dapat dibagi menjadi dua jenis, yaitu vokal dan konsonan. Ketika pertama kali diperkenalkan, *Hangeul* terdiri dari sekitar 28 karakter, termasuk 17 konsonan dan 11 vokal. Akan tetapi, dalam perkembangan modern saat ini, *Hangeul* mempunyai 40 karakter, yang terdiri dari 14 konsonan dan 10 vokal. Selain karakter dasar, ada juga 5 konsonan ganda dan 11 vokal ganda yang merupakan

gabungan dari dua huruf vokal(Rachmawati dkk., 2023). Huruf vokal dalam huruf Korea dapat terbentuk dalam dua arah, baik secara horizontal maupun vertikal, seperti ㅏ, ㅑ, ㅓ, ㅕ, dan ㅣ merupakan contoh vokal yang terbentuk secara vertikal, sementara ㅗ, ㅛ, ㅜ, ㅠ, dan ㅡ merupakan contoh vokal yang terbentuk secara horizontal (Hwa dkk., 2013). Pada konsonan dapat dituliskan di sebelah kiri vokal jika digabungkan dengan vokal vertikal, sedangkan konsonan dituliskan di atas vokal jika digabungkan dengan vokal horizontal (Hwa dkk., 2013). Tabel 2.2 dan 2.1 merupakan gambaran huruf *Hangeul* baik vokal maupun konsonan (Hwa dkk., 2013).

Dalam Tabel 2.2 dan 2.1 merupakan *alphabet* atau huruf konsonan dari bahasa Korea, yang terdiri dari 14 huruf. Namun, secara keseluruhan tulisan Korea *(Hangeul)* memiliki 19 huruf konsonan, diantaranya adalah konsonan dasar dan 5 lainnya merupakan konsonan ganda (Hwa dkk., 2013). Terdiri dari bunyi konsonan seperti k/g, n, t/d, dan seterusnya, karena dalam pelafalan huruf tersebut terdapat kesamaan sehingga satu huruf dapat menjadi dua pelafalan yang berbeda (Hwa dkk., 2013).

Tabel 2.1: Daftar Huruf Konsonan dalam Karakter Tulisan Korea (*Hangeul*)

Huruf	**Nama**	**Lafal**
ㄱ	기역	[k], [g]
ㄴ	니은	[n]
ㄷ	디귿	[t], [d]
ㄹ	리을	[r], [l]
ㅁ	미음	[m]
ㅂ	비읍	[p], [b]
ㅅ	시옷	[s], [sh]

Tabel 2.2: Lanjutan Daftar Huruf Konsonan dalam Karakter Tulisan Korea (*Hangeul*)

Huruf	Nama	Lafal
ㅇ	이응	[ŋ]
ㅈ	지읒	[c]
ㅊ	치읓	[c^h]
ㅋ	키읔	[k^h]
ㅌ	티읕	[t^h]
ㅍ	피읖	[p^h]
ㅎ	히읗	[h]

Tabel 2.3: Daftar Huruf Konsonan Ganda dalam Karakter Tulisan Korea (*Hangeul*)

Huruf	Nama	Lafal
ㄲ	쌍기역	[kk]
ㄸ	쌍디귿	[tt]
ㅃ	쌍비읍	[pp]
ㅆ	쌍시옷	[ss]
ㅉ	쌍지읒	[jj]

Tabel 2.3. merupakan huruf ganda pada karakter huruf *Hangeul* yang terdiri dari lima konsonan, yang merupakan dari bentuk huruf konsonan ganda (Hwa dkk., 2013).

Tabel 2.4: Tabel Huruf Vokal Dasar dalam Karakter Tulisan Korea (*Hangeul)*

Huruf	Nama	Lafal
ㅏ	아	[a]
ㅑ	야	[ja]
ㅓ	어	[@]
ㅕ	여	[j]
ㅗ	오	[o]
ㅛ	요	[jo]
ㅜ	우	[u]
ㅠ	유	[ju]
ㅡ	으	[1]
ㅣ	이	[i]

Tabel 2.4 menunjukkan huruf vokal *Hangeul* dasar yang memiliki huruf berjumlah sepuluh huruf vokal dasar. Dalam penulisannya, huruf vokal ditulis dari arah atas kebawah, dan dari arah kiri ke kanan (Rostineu, 2014).

Tabel 2.5: Tabel Huruf Vokal Perluasan dalam Karakter Tulisan Korea (*Hangeul*)

Huruf	Nama	Lafal
ㅐ	애	[ɛ]
ㅒ	얘	[jɛ]
ㅔ	에	[e]
ㅖ	예	[je]
ㅘ	와	[wa]
ㅙ	왜	[wɛ]
ㅚ	외	[we]
ㅝ	워	[wo]
ㅞ	웨	[we]
ㅟ	위	[wi]
ㅢ	의	[1i]

Tabel 2.5 merupakan huruf *Hangeul* vokal perluasan dengan mempunyai huruf sebesar sebelas vokal perluasan dengan gabungan vokal dasar (Hwa dkk., 2013).

K. Penelitian Terdahulu

Berdasarkan penelitian yang dilakukan oleh Khafifah Munawaroh, S Alamsyah dengan judul *"Performance Comparison of SVM, Naive Bayes, and KNN Algoorithms for Analysis of Publice Opinion Sentiment Against Covid-19 Vaccination on Twitter"* membahas tentang penggunaan akurasi sebagai tolak ukur untuk megidentifikasi model terbaik dalam melakukan klasifikasi. Dalam pengujiannya, algoritma *Support Vector Machine (SVM)* mencapai akurasi tertinggi, yaitu 96,3%. Sementara itu, algoritma *Naive Bayes* dan *K-Nearest Neighbors* (KNN) menghasilkan nilai akurasi masing masing sebesar 94% dan 91%. Akurasi SVM yang unggul ini ditingkatkan dengan penerapan ekstraksi fitur TF-IDF, memanfaatkan pustaka TextBlob (Munawaroh S Alamsyah, 2022).

Penelitian yang dilakukan oleh Annisa Ollivia Cahya Pratiwi dengan judul "Klasifikasi jenis Anggur berdasarkan bentuk Daun menggunakan *Convolutional Neural Network* dan *K-Nearest neighbor"* membahas tentang kinerja metode CNN dan KNN dievaluasi dengan menggunakan *confusion matrix*. Hasil penelitian ini menunjukkan bahwa akurasi sebesar 99% untuk CNN, sedangkan metode KNN hanya mencapai akurasi 53% (Pratiwi, 2023).

Penelitian yang dilakukan oleh Kenan Donuk, Ali Ari, mehmet Fatih Ozdemir, S Davut Hanbay dengan judul " *Deep Feature Selection for Facial Emotion Recognition Based on BPSO and SVM*"

membahas tentang Sistem 3-tahap diusulkan untuk deteksi emosi dari gambar wajah. Performa Kinerja sistem yang diusulkan telah diuji dengan dataset Fer+. Sebagai hasil dari pengujian, akurasi 85,74 % diukur. Hasil penelitian menunjukkan bahwa kombinasi BPSO dan SVM berkontribusi terhadap akurasi klasifikasi dan kecepatan dataset FER+ (Donuk dkk., 2023).

Penelitian yang dilakukan oleh Ahmad Alaik Maulani, Sri Winarno, Junta Zeniarja, Rusyda Tsaniya Eka Ptri, S Ailsa Nurina Cahyani dengan judul *"Comparison of Hyperparameter Optimization Techniques in Hybrid CNN-LSTM Model for Heart Disease Classification"* menunjukkan bahwa model *hybrid* CNN dan LSTM dengan grid search mencapai akurasi 91,67%, *recall (sensitivitas)* 89,66 %, spesifisitas 93,55 %, presisi 92,86%, 91,23% *f1-score*, dan nilai *AUC* 0,9310. Hasil ini mengkonfirmasi bahwa menggunakan model *hybrid* CNN dan LSTM dengan pendekatan *grid search* lebih cocok untuk mengklasifikasikan penyakit jantung (Maulani dkk., 2024).

Penelitian yang dilakukan oleh Novelinda Permata Wulandari, S Devi Fitrianah dengan judul "Analisa Perbandingan Algoritma CNN dan MLP Dalam Mendeteksi Penyakit COVID-19 Pada Citra X-Ray paru" yang menunjukkan bahwa algoritma CNN mengungguli algoritma MLP dengan tingkat akurasi 97,14 %, sedangkan algoritma MLP hanya mencapai 91,39%. Performa yang lebih unggul disebabkan karena algoritma CNN mempunyai lebih banyak lapisan daripada algoritma MLP, sehingga memungkinkannya untuk proses data spasial secara efektif (Wulandari S Fitrianah, 2021).

Perbedaan penelitian ini dengan penelitian sebelumnya ada pada perbandingan penggunaan metode, penambahan

tahap ekstraksi fitur *Histogram of Oriented Gradient* (HOG) dan data augmentasi masing - masing metode, dan dataset yang digunakan. Pada jurnal tersebut satu diantaranya berkaitan dengan perbandingan antara metode CNN dan KNN dalam pengklasifikasian jenis anggur berdasarkan bentuk daunnya, lalu satu jurnal lainnya berkaitan dengan perbandingan antara metode SVM, *Naive bayes*, dan KNN dalam menganalisa tentang sentimen opini publik mengenai vaksinasi Covid-19 di twitter.

Dalam hal ini dikarenakan metode yang didapat mempunyai kenggulan masing-masing dari berbagai kasus, meskipun ada beberapa penelitian menunjukkan kelemahan pada kedua metode tersebut. Maka, peneliti mencoba melakukan terobosan dengan cara membandingkan metode mana yang lebih unggul dari masing - masing kedua jurnal tersebut yang diantaranya jurnal pertama adalah metode CNN dan jurnal kedua adalah SVM dalam mengklasifikasikan citra gambar karakter tulisan Korea *(Hangeul)*. Selain itu, penelitian ini menggunakan alat bantu *Google Collaboratorium* untuk merancang model SVM dan CNN, serta melakukan pengklasifikasian citra gambar karakter tulisan Korea *(Hangeul)*.

BAB III

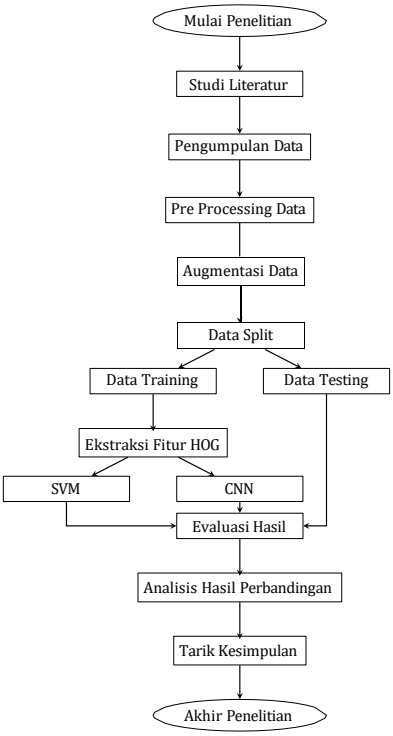
Metode Penelitian

A. Metode Penelitian

Metode yang digunakan dalam penelitian ini adalah studi literatur. Studi literatur adalah kegiatan yang meliputi pengumpulan data pustaka, membaca dan mencatat, serta melakukan pengolahan terhadap bahan penelitian(Adelia dkk., 2024). Penelitian ini dengan melakukan pengumpulan informasi berdasarkan referensi dari jurnal, maupun buku yang berhubungan dengan metode *Support Vector Machine (SVM)* dan *Convolutional Neural Network (CNN)*, antara lainnya adalah buku *Deep Learning With Python* (Ketkar S Moolayil, 2021),*Deep Learning* (Goodfellow dkk., 2016), *Introduction to Deep Learning Using R: A Step-by-Step Guide to Learning and Implementing Deep Learning Models Using R* (Beysolow, 2017), *Multimedia Database Management Systems (Artech House Computing Library)* (Lu, 1999), *Support Vector Machines: Optimization Based Theory Algorithms, and Extensions*(Deng dkk., 2013).

B. Alur Penelitian

Langkah atau tahapan yang dilakukan pada penelitian ini digambarkan melalui Gambar 3.1:



Gambar 3.1: Diagram Alir Penelitian

Tahapan proses penelitian ini dapat di ilustrasikan pada Gambar 3.1. Berikut ini merupakan tahapan dalam penelitian pengkasifikasian karakter tulisan Korea *(Hangeul)*, antara lainnya :

1. Langkah awal dimulai dengan *studi literature* yang bertujuan untuk mencari informasi yang relevan terhadap metode yang

digunakan.

2. Lakukan pengumpulan data karakter tulisan Korea*(Hangeul)* yang bersumber dari website *Kaggle* (JAMESCASIA, 2023) dengan bertujuan untuk mendapatkan kumpulan data yang relevan dalam penelitian ini.

3. Lakukan tahap *pre processing* data yang bertujuan untuk membersihkan, menyelaraskan ukuran dari setiap citra agar seragam, dan menyiapkan data agar dapat di proses ke tahap selanjutnya (Nurafiya S Chandra, 2024).

4. Lakukan augmentasi data yang bertujuan untuk mengurangi *overfitting* dengan cara memperbesar ukuran dataset dengan upaya seminimal mungkin (Fadillah dkk., 2024). Teknik augmentasi ini dengan memodifikasi citra, seperti membalik, memotong, memutar, dan mengubah skala, sehingga model dapat lebih umum dalam mengenali variasi data.

5. Lakukan pembagian data menjadi dua bagian, antara lainnya adalah data *training* dan data *testing*. Pembagian ini dilakukan untuk kedua netode yang akan dilakukan dalam penelitian ini, antara lainnya adalah *Support Vector Machine* (SVM) dan *Convolutional Neural Network* (CNN).

6. Data latih digunakan untuk pengklasifikasian data dengan kedua metode tersebut. Metode SVM dan CNN dilatih dengan menggunakan data *training* yang digunakan untuk mempelajari pola dan fitur yang relevan.

7. Sebelum dilakukan tahap pelatihan dengan menggunakan data *training* pada kedua metode tersebut akan dilakukan

tahap ekstraksi fitur HOG *(Histogram of Oriented Gradients)* yang bertujuan untuk mendeteksi penampilan objek dan bentuknya dalam citra dengan menghitung kemunculan orientasi gradien pada bagian-bagian lokal citra karakter tulisan Korea *(Hangeul)*.

8. Setelah dilakukan proses ekstraksi fitur, akan dilakukan pelatihan model dan pengevaluasian hasil model dari kedua metode tersebut yang bertujuan untuk menilai kinerja masing-masing model. Evaluasi ini mencakup dari analisis akurasi dan *confusion matrix* untuk menentukan efektivitas dari kedua metode tersebut dalam pengenalan karakter tulisan Korea *(Hangeul)*

Berikut ini merupakan tahapan dalam penelitian ini, antara lainnya adalah:

1. Pengumpulan Data

Dataset yang digunakan dalam penelitian ini merupakan kumpulan gambar karakter tulisan Korea *(Hangeul)* yang diperoleh dari website *Kaggle* (`https://www.kaggle.com/datasets/wayperwayp/hangulkorean-characters`) (JAMESCASIA, 2023). Dataset ini berjudul *Handwritten Hangul Characters* dan diunggah oleh JAMESCASIA pada tahun 2023. Dataset ini dipilih untuk penelitian ini karena berisi kumpulan gambar yang mewakili masing-masing karakter tulisan Korea *(Hangeul)*. Dataset ini terdiri atas 29 kelas, masing-masing kelas berisi 80 karakter *Hangeul*. Berikut merupakan karakter tulisan Korea *(Hangeul)*.

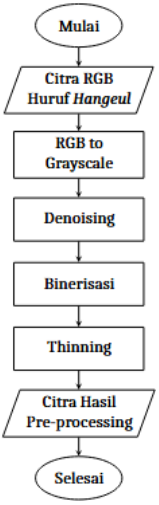
Tabel 3.1: Tabel Karakter Tulisan Korea (*Hangeul*)

Huruf	**Lafal**
ㅐ	[ae]
ㅂ	[b]
ㅃ	[bb]
ㅊ	[ch]
ㅔ	[e]
ㅓ	[eo]
ㅡ	[eu]
ㄱ	[g]
ㄲ	[gg]
ㅎ	[h]
ㅣ	[i]
ㅈ	[j]
ㅋ	[k]
ㅁ	[m]
ㄴ	[n]
ㅇ	[ng]
ㄷ	[d]
ㅗ	[o]
ㅍ	[p]
ㄹ	[r/l]
ㅅ	[s]
ㅆ	[ss]
ㅌ	[t]
ㅜ	[u]
ㅑ	[ya]
ㅒ	[yae]
ㅖ	[ye]
ㅛ	[yo]
ㅠ	[yu]

Terdapat 29 kelas dengan total 2.320 citra, di mana masing-masing kelas memiliki 80 citra gambar karakter tulisan Korea *(Hangeul)* yang berbeda. Setiap gambar memiliki resolusi 28×28 piksel dengan format `JPG`.

2. Pra-pemrosesan Data

Data yang didapat kemudian akan melewati tahap pra-pemrosesan data agar data siap untuk diproses oleh algoritma dan hasil pemrosesan menjadi efektif. Pra -pemrosesan merupakan tahapan dalam menstandarkan ukuran citra, yang bertujuan untuk menyelaraskan ukuran dari setiap citra agar seragam (Nurafiya S Chandra, 2024). Meskipun demikian, dalam penelitian ini ukuran citra gambarnya sudah seragam sehingga tidak memerlukan proses *resize*. Berikut ini merupakan tahapan - tahapan dalam pra-pemrosesan dapat dilihat pada Gambar 3.2



Gambar 3.2: Diagram alir tahap Pra-Pemrosesan

Adapun tahapan yang dilakukam dalam *pra-pemrosesan*, antara lain:

1. Mengubah citra gambar dari format citra RGB aslinya menjadi citra *grayscale*. Proses perhitungan manual konversi RGB ke *grayscale* dapat dilihat sebagai berikut (Zhi dkk., 2024):

   $$Grayscale = (0,2989 \times Red) + (0,5870 \times Green) + (0,1140 \times Blue) \quad (3.1)$$

   Tahapan konversi citra RGB menjadi citra *grayscale* ditunjukkan pada Gambar3.3.

Gambar 3.3: *RGB to Grayscale*.(Mudhofar S Agustin, 2024)

2. Melakukan *denoising* menggunakan metode *median filter*. Tujuan dari *denoising* adalah untuk menghilangkan noise yang dihasilkan oleh distribusi normal pada citra (Wardana dkk., 2024).Berikut ini merupakan tahapan *denoising* pada Gambar3.4.

Gambar 3.4: Tahap *Denoising*.(Umam dkk., 2024)

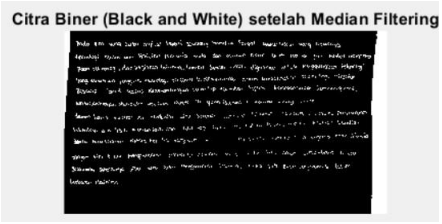
3. Berikutnya, citra *grayscale* mengalami konversi biner *(binerisasi)* dengan menggunakan metode otsu. Selama proses ini, background putih diubah menjadi hitam (Irfani S Gasim, 2024). Sehingga, karakter tulisan Korea *(Hangeul)* yang awalnya bewarna hitam diubah menjadi putih. Proses perhitungan manual binerisasi dapat dilihat sebagai berikut (Mellyssa dkk., 2020) .

$$\text{Citra Biner}(x, y) = \begin{cases} 1, & \text{jika } I(x, y) > T \\ 0, & \text{jika } I(x, y) \leq T \end{cases} \quad (3.2)$$

Keterangan:

- T : Ambang batas
- $I(x, y)$: Intensitas piksel

Tahapan binerisasi ditunjukkan pada Gambar 3.5 sebagai berikut.



Gambar 3.5: Tahap *Binner*.(Irfani S Gasim, 2024)

Selain menjadikan citra *denoising* menjadi citra biner, di dalam proses binarisasi juga dilakukan proses invers. Proses ini dilakukan dengan mengubah *background* yang sebelumnya berwarna putih menjadi warna hitam, serta

karakter tulisan Korea *(Hangeul)* yang sebelumnya berwarna hitam menjadi warna putih. Tujuan dari proses invers ini adalah untuk mempermudah tahap selanjutnya dalam analisis atau pengolahan citra, terutama jika algoritma yang digunakan lebih sesuai dengan latar belakang hitam dan objek berwarna putih. Berikut ini merupakan rumus dalam perhitungan invers matriks (Gonzales dkk., 2018):

$$I'(x, y) = 1 - I(x, y) \quad (3.3)$$

Keterangan:

- $I'(x, y)$: Hasil Invers
- $I(x, y)$: Intensitas piksel

4. Tahap terakhir yaitu *thinning*, yang berarti mengurangi ketebalan karakter tulisan Korea (*Hangeul*) dalam citra (Harahap dkk., 2020). Berikut ini merupakan tahapan *Thinning* pada Gambar3.6.

Gambar 3.6: Tahap *Thinning*.(Harahap dkk., 2020)

5. Citra yang telah diperbaiki kualitasnya dari hasil *pre-processing*, akan dilakukan proses selanjutnya yaitu data augmentasi dan pembagian data.

3. Data Augmentasi

Data augmentasi merupakan salah satu teknik yang umumnya digunakan untuk mengurangi *overfitting* dengan cara memperbesar ukuran dataset dengan upaya seminimal mungkin (Fadillah dkk., 2024). Proses augmentasi biasanya dilakukan melalui transformasi pada data atau menduplikasi data asli tanpa mengubah label pada bagian tertentu dari data tersebut. Data biasanya diaugmentasikan dengan melakukan transformasi pada data atau dapat membuat salinan dari sumber data tanpa mengubah label yang tertera pada bagian dari data tersebut. Dalam penelitian ini dengan menggunakan 2320 data, jumlah ini mungkin terlihat kecil dibandingkan dengan dataset yang disebutkan sebelumnya. Namun, ada cara yang efektif untuk memperkaya variasi fitur, yaitu menggunakan teknik augmentasi data. Berikut merupakan teknik augmentasi data yang diterapkan dalam penelitian ini, antara lainnya:

1. Rotasi *(rotation)*: Memutar gambar pada sudut tertentu untuk menghasikan orientasi yang berbeda (Fajarendra dkk., 2024).

2. Pergeseran lebar *(width shift)*: Menggeser gambar secara horizontal untul mengubah posisi objek didalamnya (Fajarendra dkk., 2024).

3. Pergeseran tinggi *(height shift)*: Menggeser gambar secara vertikal untuk memodifikasi lokasi objek dalam gambar (Fajarendra dkk., 2024).

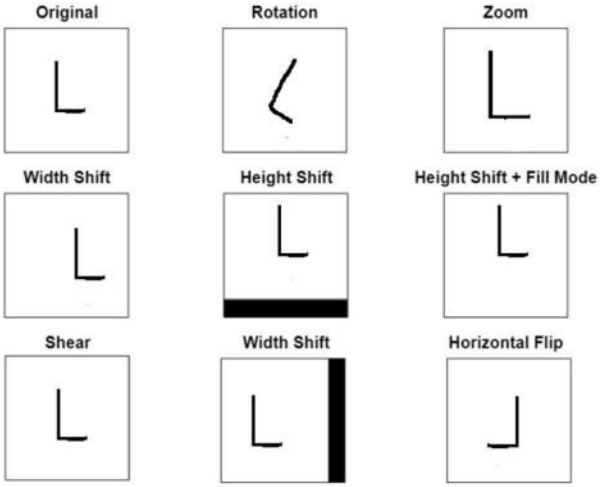
4. Mencukur *(shear)*: Mengubah bentuk gambar dengan menggeser sebagian dari gambar secara horizontal atau

vertikal (Fajarendra dkk., 2024).

5. Memperbesar *(zoom)*: Memperbesar bagian tertentu dari gambar untuk menciptakan variasi dalam skala (Fajarendra dkk., 2024).

6. Membalik secara horizontal *(horizontal flip)*: Membalik gambar dari kiri ke kanan untuk menghasilkan citra yang merupakan pantulan dari gambar asli (Fajarendra dkk., 2024).

Dalam mengisi area kosong yang muncul akibat perubahan tinggi atau lebar, dilakukan pengisian piksel yang baru. Hasil dari augmentasi ini diilustrasikan pada Gambar 3.7.



Gambar 3.7: Augmentasi Data (Toyib dkk., 2024)

4. Pembagian Dataset

Hasil dari data augmentasi selanjutnya akan dibagi menjadi 2 jenis pembagian dataset, yaitu data *training* dan data *testing*. Pembagian data *training* dan data *testing* dalam citra gambar karakter tulisan Korea *(Hangeul)* ini digunakan untuk menentukan nilai akurasi. Kesalahan dalam menentukan komposisi kedua jenis data ini dapat mempengaruhi nilai dan presisi yang dihasilkan (Azmi dkk., 2023). Data *training* dan data *testing* merupakan konsep umum yang dapat digunakan didalam *machine learning*. Model pembelajaran mesin yang dilatih menggunakan sekumpulan data untuk dipelajari disebut data *training*. Hasil pembelajaran dari data ini kemudian diterapkan untuk mengolah dataset baru yang disebut data *testing* (Azmi dkk., 2023). Penelitian ini melakukan percobaan klasifikasi dengan dua presentase data *training* dan data *testing* yang berbeda yaitu dengan skala pembagian proporsi 70% *training* (8120 data): 30% *testing* (3480 data), 80% *training* (9280 data) :20% *testing* (2320 data), 90% *training* (10440 data):10% *testing* (1160 data) dari masing - masing kelas karakter tulisan Korea *(Hangeul)*. Beberapa skenario yang dilakukan bertujuan untuk mendapatkan hasil yang terbaik, semakin besar perbandingan data *training* (data latih) dibandingkan data *testing* (data uji) cenderung memberikan hasil akurasi yang lebih tinggi, semakin sedikit data training membuat hasil yang berfluktuasi atau tidak stabil (Swasono dkk., 2023). Pembagian jumlah data *training* dan data *testing* pada kedua presentase yang berbeda dapat disajikan pada Tabel 3.2.

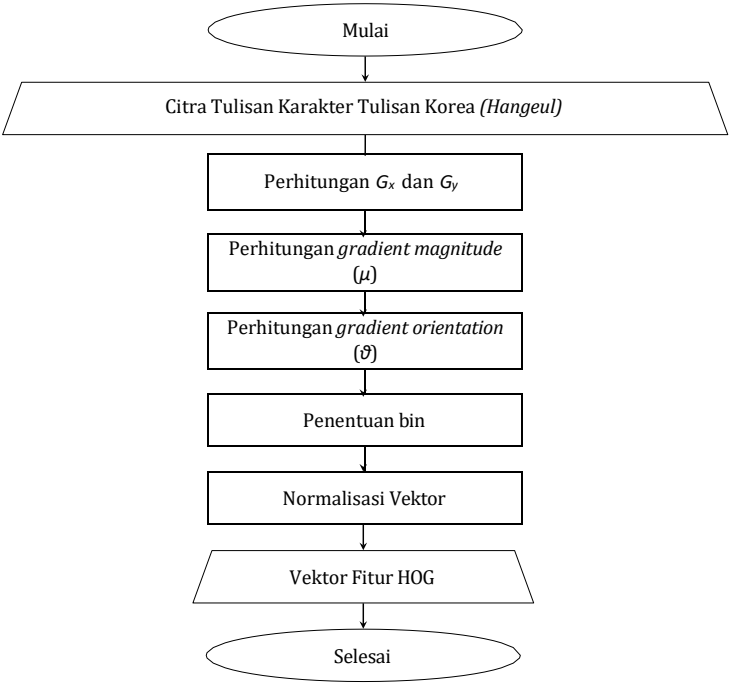
Tabel 3.2: Skenario Pembagian Data

	Skenario 70%:30%		Skenario 80%:20%		Skenario 90%:10%	
	Data Training	*Data testing*	*Data Training*	*Data Testing*	*Data Training*	*Data Testing*
	8120	3480	9280	2320	10440	1160

5. Ekstraksi Fitur

Ekstraksi fitur merupakan proses untuk mengubah citra karakter tulisan Korea (*Hangeul*) menjadi representasi informasi numerik. Hasil dari proses ini selanjutnya dapat digunakan untuk mengidentifikasi kemiripan antar karakter tulisan Korea (*Hangeul*). Metode ekstraksi fitur yang digunakan dalam penelitian ini adalah HOG (*Histogram of Oriented Gradients*). HOG merupakan metode ekstraksi fitur yang umum digunakan dalam pemrosesan citra dan *Computer Vision*. Metode ini sering diterapkan dalam proses identifikasi objek, seperti hewan, kendaraan, teks, wajah, dan lain sebagainya (Putra dkk., 2024).

Pada metode HOG, citra dibagi menjadi beberapa blok, dan setiap blok dibagi menjadi beberapa *cell*. Dalam penelitian ini, digunakan *cell* berukuran 8×8 piksel.Berdasarkan penelitian yang dilakukan oleh Devita dkk. (2019) membuktikan bahwa membagi citra ke dalam *cell* berukuran lebih kecil, seperti (8,8) piksel, lebih efektif dalam mengekstraksi ciri dibandingkan dengan ukuran *cell* yang lebih besar. Berikut ini merupakan diagram alir proses *Histogram of Oriented Gradients* (HOG).



Gambar 3.8: Diagram Alir *Histogram of Oriented Gradients*

- Perhitungan nilai gradien pada citra gambar

  Lakukan perhitungan nilai gradien sumbu *x* dan sumbu *y* pada setiap piksel citra dengan perkalian konvolusi menggunakan kernel operator Sobel seperti berikut (Tang dkk., 2020).

$$SobelX = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (3.4)$$

$$SobelY = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (3.5)$$

Selanjutnya menghitung nilai *gradient magnitude* dan *gradient orientation* setiap piksel pada *cell* dengan menggunaan persamaan berikut ini(Tang dkk., 2020).

$$Magnitude(\mu) = \sqrt{G_x^2 + G_y^2} \quad (3.6)$$

Keterangan:

- μ : *Gradient magnitude*
- G_x : Matriks operator Sobel sisi horizontal
- G_y : Matriks operator Sobel sisi vertikal

$$Angle(\vartheta) = \tan^{-1}\frac{G_y}{G_x} \quad (3.7)$$

Keterangan:

- ϑ : *Gradient orientation*
- G_x : Matriks operator Sobel sisi horizontal
- G_y : Matriks operator Sobel sisi vertikal

- Proses penentuan bin

Proses penentuan bin digunakan untuk membangun histogram fitur HOG. Histogram ini memiliki 9 bin dengan rentang nilai 0° hingga 180°, sehingga setiap bin memiliki selisih sebesar 20°. Penentuan bin dilakukan menggunakan persamaan berikut (Akbar S Utaminingrum, 2019).

$$\text{Vote} = \frac{\Delta Ri}{H} \times G_i \tag{3.8}$$

Keterangan:

- H : Rentang nilai sudut.
- G_i : Nilai gradien.
- ΔR_i : Batas atas gradien - Nilai orientasi gradien.

- Normalisasi

  Proses normalisasi bertujuan untuk mengurangi pengaruh variasi pencahayaan pada gambar. Pada tahap ini, histogram dari seluruh sel dalam satu blok digabungkan. Hasil penggabungan tersebut kemudian dinormalisasi menggunakan suatu persamaan sebagai berikut (Rivan dkk., 2020).

$$L_1\text{norm}: \quad v' = \frac{v}{\sqrt{\|v\|_1^2 + \epsilon}} \tag{3.9}$$

$$L_2\text{norm}: \quad v'' = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon}} \tag{3.10}$$

iliki histogr

v adalah fitur blok yang mem am, ϵ adalah konstanta dengan nilai 10^{-6}.

C. Metode *Machine Learning*

Penelitian ini menggunakan teknik *machine learning* yaitu *Convolutional Neural Network* (CNN) dan *Support Vector Machine* (SVM), untuk mengklasifikasikan karakter tulisan Korea *(Hangeul)*.

Tujuannya untuk mengevaluasi seberapa baik kinerja masing - masing metode dalam mengklasifikasikan gambar karakter

tulisan Korea *(Hangeul)*, serta memberikan wawasan yang lebih mendalam tentang penerapan teknik *machine learning* dalam klasifikasi citra. Berikut ini merupakan penjelasan dan tahap - tahapan yang digunakan dalam penelitian ini.

1. *Support Vector Machine(SVM)*

Proses pembentukan model SVM merupakan langkah awal sebelum melakukan proses identifikasi. Hal ini melibatkan perancangan model berdasarkan proses *training* dan *testing* yang meliputi pemilihan kernel untuk mencapai model optimal menggunakan metode SVM. Model yang optimal ini kemudian digunakan untuk mengenali karakter tulisan Korea *(Hangeul)*. Tahapan proses identifikasi ini diilustrasikan pada Gambar 3.9, yang menggunakan metode klasifikasi SVM.

Dari Gambar 3.9 menunjukkan bahwa citra karakter tulisan Korea *(Hangeul)* yang telah melewati tahap *pre-processing* akan melalui tahapan augmentasi data terlebih dahulu. Berikut ini merupakan beberapa tahapan dalam pemodelan SVM, antara lainnya:

1. Langkah awal dimulai dengan inisiasi untuk mempersiapkan proses dalam klasifikasi menggunakan metode *Support Vector Machine (SVM)*

2. Lakukan augmentasi data yang bertujuan untuk mengurangi *overfitting* dengan cara memperbesar ukuran dataset dengan upaya seminimal mungkin (Fadillah dkk., 2024). Teknik augmentasi ini dengan memodifikasi citra, seperti membalik, memotong, memutar, dan mengubah skala, sehingga model dapat lebih umum dalam mengenali variasi data.

3. Bagi dataset menjadi data latih dan data uji. Gunakan pembagian data rasio, antara lain 80% : 20%, 70%:30%, dan 90%:10%. Data latih digunakan untuk melatih model SVM, sedangkan data uji digunakan untuk mengevaluasi kinerja model.

4. Lakukan Ekstraksi fitur HOG pada data *training* dan *testing*.

5. Lakukan proses pelatihan model, di mana model SVM akan mempelajari pola dari data pelatihan yang berupa fitur hasil ekstraksi HOG. Tujuannya adalah agar model dapat membedakan atau mengklasifikasikan data sesuai dengan label yang diberikan.

6. Setelah proses pelatihan model, lakukan pemilihan kernel. Gunakan data latih yang sudah diekstraksi fitur untuk melatih model SVM dengan memilih kernel yang bertujuan mengubah data *non-linier* menjadi *linier* dalam ruang fitur yang lebih tinggi. Pada tahap ini, dipilih kernel yang paling sesuai dengan jenis data yang digunakan, seperti kernel linier, kernel polinomial, *Radial Basis Function* (RBF), dan kernel *Sigmoid*. Pemilihan kernel yang tepat sangat penting untuk meningkatkan akurasi model.

7. Setelah pemilihan kernel,lakukan inisialisasi parameter yang diperlukan dalam SVM, seperti nilai alfa, *gamma, C, ϵ,* dan *lambda*.

8. Lakukan perhitungan matriks Hessian untuk setiap data berdasarkan kernel yang telah dipilih.

9. Lakukan perhitungan *error* pada setiap data dalam proses

pelatihan untuk menentukan sejauh mana prediksi model berbeda dari nilai sebenarnya.

10. Lakukan penyesuaian nilai alfa (α) berdasarkan nilai *error* yang diperoleh dalam tahap sebelumnya.

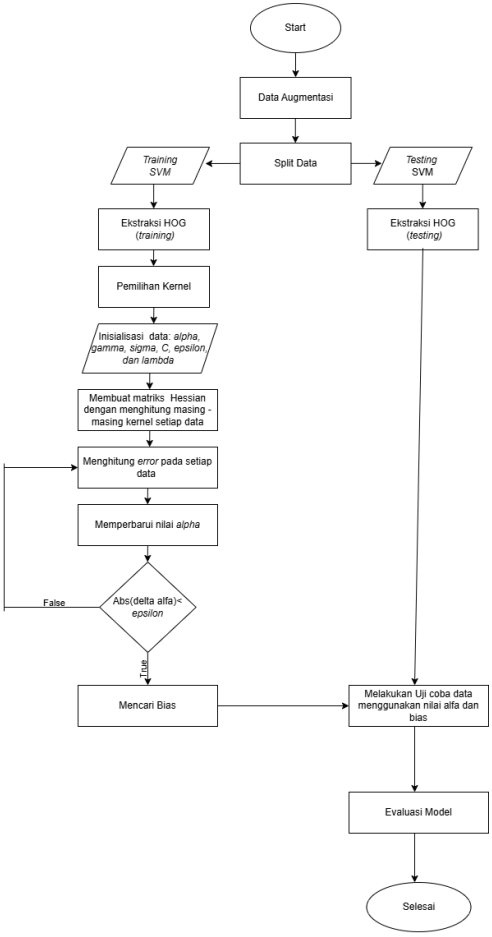
11. Lakukan pengecekan konvergensi dengan membandingkan perubahan nilai alfa ($\delta\alpha$) dengan epsilon (ϵ). Jika $|\delta\alpha| < \epsilon$, lanjutkan ke tahap pencarian bias. Jika tidak, ulangi proses penyesuaian nilai alfa.

12. Lakukan pencarian bias untuk meningkatkan performa model dalam memisahkan kelas pada SVM.

13. Setelah model dilatih, tahap pengujian dilakukan menggunakan data pengujian. Tujuannya adalah untuk mengevaluasi kinerja model yang telah dilatih. Model akan memprediksi hasil berdasarkan data pengujian, kemudian hasilnya dibandingkan dengan label asli untuk mengukur akurasi.

14. Lakukan evaluasi model untuk mengukur kinerja sistem dalam mengenali karakter tulisan Korea *(Hangeul)* yang teridentifikasi dengan benar dan yang tidak teridentifikasi dengan benar.

15. Setelah evaluasi, model dengan kinerja terbaik dipilih dengan pengujian beberapa parameter dan kernel yang berbeda selama proses pelatihan. Model terbaik akan digunakan untuk identifikasi kemiripan karakter tulisan Korea *(Hangeul)* dengan benar.



Gambar 3.9: Flowchart SVM

2. *Convolutional Neural Network* (CNN)

Proses pembentukan model CNN merupakan langkah juga sebelum melakukan proses identifikasi. Perancangan model diperoleh dari proses *training* dan *testing*, dimana pada prosesnya terdapat proses *convolution, pooling* dan aktivasi ReLU, sehingga

akan diperoleh model terbaik pada metode CNN. Model terbaik akan digunakan untuk mengidentifikasi dalam pengenalan karakter tulisan Korea *(Hangeul)*. Tahapan proses identifikasi ini dapat ditunjukkan pada Gambar 3.10 dengan menggunakan metode klasifikasi yaitu CNN.

Gambar 3.10 menunjukkan bahwa citra karakter tulisan Korea *(Hangeul)* yang telah melewati tahap *pre-processing* dan augmentasi data akan dilakukan pembagian dataset terlebih dahulu. Berikut ini merupakan beberapa tahapan dalam pemodelan CNN, antara lainnya:

1. Lakukan augmentasi untuk meningkatkan variasi dan ukuran dataset. Metode augmentasi yang dapat diterapkan antara lain:

   (a) Rotasi citra dengan derajat acak.
   (b) Pergeseran lebar *(width shift)*
   (c) Pergeseran tinggi *(height shift)*
   (d) Mencukur *(shear)*
   (e) Memperbesar *(zoom)*
   (f) Membalik secara horizntal *(horizontal flip)*

2. Setelah dilakukan augmentasi data, data akan dibagi menjadi data latih dan data uji menggunakan rasio pembagian data, yaitu 80% : 20%, 70% : 30%, dan 90% : 10%. Data latih digunakan untuk melatih model CNN dan ekstraksi fitur, sedangkan data uji digunakan untuk mengevaluasi kinerja model.

3. Lakukan Ekstraksi fitur HOG pada data latih dan data uji secara terpisah.

4. Proses CNN *Pipeline*

   (a) Konvolusi

   Terapkan filter atau kernel ukuran 3×3 pada citra *input*. Proses konvolusi dilakukan dengan menggeser filter diatas citra *input*, menghitung perkalian elemen-per-elemen antara nilai piksel citra dan nilai filter, lalu menjumlahkan hasilnya untuk menghasilkan *feature map*. *Padding* (menambahkan nilai nol di sekitar citra) diterapkan agar ukuran *output* tetap sama dengan ukuran *input*. Konvolusi yang diterapkan adalah 32 dan 64.

   (b) Fungsi Aktivasi ReLU

   Fungsi aktivasi *Rectified Linear Unit* (ReLU) diterapkan untuk menghilangkan nilai negatif dalam *feature map*, sehingga hanya nilai positif yang diteruskan ke lapisan berikutnya.

   (c) *Max Pooling*

   Gunakan *max pooling* dengan ukuran area 2×2 untuk mereduksi dimensi *feature map*. Metode ini memilih nilai maksimum dalam area yang ditentukan, membantu mereduksi jumlah parameter, meningkatkan efisiensi komputasi, dan memberikan ketahanan terhadap noise. *Stride* sebesar 1 digunakan untuk memastikan filter tidak melompati piksel setelah setiap langkah.

   (d) *Flatten*

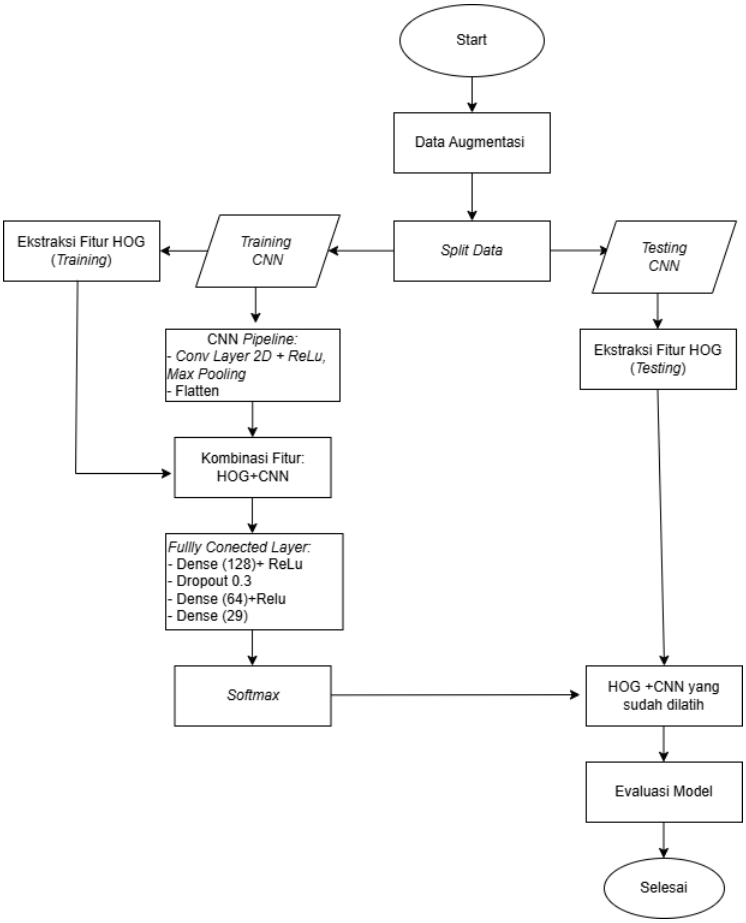
   Ubah *feature map* yang dihasilkan menjadi vektor satu dimensi.

5. Kombinasikan hasil dari HOG *(Histogram Of Gradient)* dengan CNN *(Convolutional Neural Network) pipeline*

6. Hasil kombinasi fitur HOG dengan CNN diteruskan ke *hidden layer* yang terdiri dari Dense 128 neuron dengan fungsi aktivasi ReLu, Dropout 0,3, dan Dense 64 neuron dengan fungsi aktivasi ReLu. Terapkan *dropout* untuk mengurangi risiko *overfitting*.

7. Lapisan akhir menggunakan 29 neuron yang sesuai dengan jumlah kelas (karakter *Hangeul*) dan fungsi aktivasi *softmax* untuk menghasilkan distribusi probabilitas prediksi.

8. Model yang telah dilatih dengan kombinasi fitur HOG dan CNN, akan dilanjutkan ke evaluasi model yang berfungsi untuk melihat nilai akurasi prediksi dengan citra asli.



Gambar 3.10: Flowchart CNN

D. Evaluasi Model

Tahap terakhir adalah tahap evaluasi model. Tahap ini terbagi menjadi 4 bagian yang digunakan sebagai acuan dalam menentukan baik buruknya dari suatu model yang digunakan, antara lainnya adalah *Precision*, *Recall*, *accuracy*, dan *f1-score*.

BAB IV

HASIL DAN PEMBAHASAN

Tahap - tahapan yang dilakukan penelitian ini dimulai dari pengumpulan data, *pre processing*, augmentasi data, pembagian data, ekstraksi fitur, SVM, CNN, evaluasi hasil, analisis hasil perbandingan, identifikasi kemiripan citra *input* dan *output*, dan kesimpulan.

A. Pengumpulan Data

Tahapan awal pada penelitian ini adalah pengumpulan data citra. Dataset yang digunakan bersumber dari *Kaggle* (`https://www.kaggle.com/datasets/wayperwayp/hangulkorean-characters`) (JAMESCASIA, 2023) dengan judul *Handwritten Hangul Characters* dan diunggah oleh JAMESCASIA pada tahun 2023. Data citra yang digunakan merupakan citra karakter tulisan Korea *(Hangeul)* yang terdiri dari 29 kelas dengan total 2320 citra yang dimana masing - masing kelas mempunyai 80 citra gambar karakter tulisan huruf Korea *(Hangeul)* yang berbeda dengan ukuran 28 $\times$ 28 piksel dengan format *JPG.

Tabel 4.1: Tabel Karakter Tulisan Korea (*Hangeul*)

Huruf	**Lafal**
ㅐ	[ae]
ㅂ	[b]
ㅃ	[bb]
ㅊ	[ch]
ㅔ	[e]
ㅓ	[eo]
ㅡ	[eu]
ㄱ	[g]
ㄲ	[gg]
ㅎ	[h]
ㅣ	[i]
ㅈ	[j]
ㅋ	[k]
ㅁ	[m]
ㄴ	[n]
ㅇ	[ng]
ㄷ	[d]
ㅗ	[o]
ㅍ	[p]
ㄹ	[r/l]
ㅅ	[s]
ㅆ	[ss]
ㅌ	[t]
ㅜ	[u]
ㅑ	[ya]
ㅒ	[yae]
ㅖ	[ye]
ㅛ	[yo]
ㅠ	[yu]

B. *Pre-processing Data*

Tahap kedua yang dilakukan pada penelitian ini adalah tahapan *pre-processing*. Tujuan dari tahapan ini adalah untuk memperbaiki kualitas citra. Pada tahap ini, beberapa *library* yang digunakan untuk meningkatkan kualitas citra agar siap untuk dianalisis lebih lanjut. Berikut adalah beberapa library yang digunakan dalam penelitian ini:

```
#import data
from google.colab import drive
#bar porgress
from tqdm import tqdm
# Membaca dan memproses citra
import cv2
import pandas as pd #membaca data
from PIL import Image
import numpy as np #operasi array
import os
#Skeletonisasicitra
from skimage.morphology import skeletonize
# Ekstraksi fitur
from skimage.feature import hog
# Grayscale
from skimage.color import rgb2gray
#SVM
import joblib #penyimpanan svm
from sklearn.svm import SVC
from sklearn.preprocessing
#Pelabelan
import LabelEncoder, StandardScaler
from sklearn.preprocessing
import label_binarize
from sklearn.preprocessing import LabelEncoder
```

Gambar 4.1: *Library* yang digunakan

```
#visualisasi data
import matplotlib.pyplot as plt
import seaborn as sns

#Evaluasi model
from sklearn.metrics
import confusion_matrix,
classification_report, roc_curve, auc
from sklearn.metrics
import classification_report,
confusion_matrix, precision_recall_fscore_support
#Hyperparameter
from sklearn.model_selection import GridSearchCV

#CNN
from tensorflow.keras.utils
import to_categorical
from tensorflow.keras.models
import Model
from tensorflow.keras.layers
import Input, Conv2D, MaxPooling2D,
Flatten, Dense, Concatenate
from tensorflow.keras.optimizers
import Adam, RMSprop
from tensorflow.keras.callbacks
import EarlyStopping

# Pembagian data dan pencarian hyperparameter
from sklearn.model_selection import
import train_test_split

#Augmentasi data
from tensorflow.keras.preprocessing.image
import ImageDataGenerator, load_img, img_to_array
```

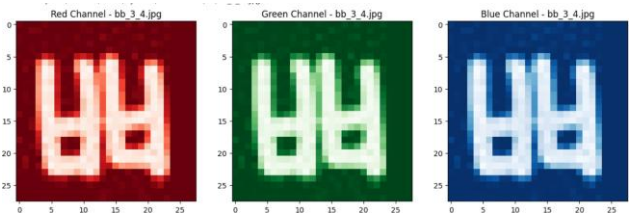
Gambar 4.2: *Library* yang digunakan(Lanjutan)

1. Citra RGB

Jenis citra yang digunakan pada penelitian ini ialah citra RGB *(Red, Green, and Blue)* karakter tulisan Korea *(Hangeul)*. Selanjutnya citra karakter tulisan Korea *(Hangeul)* yang bersumber dari website *Kaggle* (`https://www.kaggle.com/datasets/`

wayperwayp/hangulkorean-characters) (JAMESCASIA, 2023) dalam bentuk proses pemindaian *(scanning)*. Ukuran citra karakter tulisan Korea *(Hangeul)* yang seragam yaitu berukuran 28×28 dan karakter tulisan Korea *(Hangeul)* memiliki rotasi yang sama. Data citra karakter tulisan Korea *(Hangeul)* ini berformat *.JPG.

Gambar 4.3: Salah Satu Sampel Citra Asli Karakter Huruf *(Hangeul)* "BB"



Gambar 4.4: Salah Satu Sampel Citra RGB *(Red, Green, Blue)* Karakter Huruf *(Hangeul)* "BB"

Berikut ini merupakan salah satu sampel matriks yang merepresentasikan komponen RGB *(Red, Green, and Blue)* dari huruf *Hangeul*, yang menunjukkan distribusi tingkat kecerahan piksel pada kanal warna *Red*, *Green*, dan *Blue*, yang ditampilkan

sebagai berikut.

247	255	255	255	248	255	253	255	255	255	248	255	255	248	251	255	255	244	255	244	255	254	255	246	255	255	255	255
255	255	244	246	254	255	253	252	255	254	250	247	255	255	255	241	249	252	255	255	255	247	255	255	255	255	255	255
250	255	255	255	255	255	250	253	255	248	255	254	255	239	250	255	255	255	248	244	255	255	255	248	255	255	255	255
251	255	251	253	247	253	252	255	255	251	253	245	253	255	253	255	249	255	255	253	255	242	248	255	255	255	255	255
255	255	249	255	250	255	255	250	248	255	255	255	255	255	239	249	251	255	249	255	255	254	255	255	255	255	255	255
250	255	255	243	184	216	248	249	254	206	187	216	253	199	116	153	255	254	234	255	245	241	255	250	255	255	255	255
255	249	248	136	34	74	221	255	255	101	22	75	221	125	0	20	214	254	255	245	127	104	201	255	255	255	255	255
255	243	255	102	0	18	207	248	247	90	11	0	172	89	3	29	198	248	255	201	21	0	138	242	255	255	255	255
254	255	254	92	1	24	194	255	255	171	3	13	110	117	21	10	185	255	251	192	17	21	85	249	255	255	255	255
255	252	252	92	17	36	201	253	251	195	28	24	101	119	18	17	154	251	255	225	33	9	70	249	255	255	255	255
255	254	255	85	6	14	191	250	255	216	26	7	75	125	10	10	155	255	248	239	50	8	53	233	255	255	255	255
253	248	255	88	16	23	203	255	253	231	37	9	64	143	16	15	133	252	241	241	61	12	45	224	255	255	255	255
255	250	252	93	20	27	192	245	248	237	56	20	54	144	14	13	130	255	255	244	66	8	30	215	255	255	255	255
255	255	255	103	2	15	186	255	255	235	53	14	45	145	18	8	107	201	184	147	35	8	20	203	255	255	255	255
244	255	255	128	11	22	160	226	205	170	38	18	45	140	29	21	28	55	32	16	6	20	19	196	255	255	255	255
255	255	245	139	22	2	48	45	19	45	11	23	26	96	7	23	13	15	8	4	28	33	5	179	255	255	255	255
255	255	241	149	27	14	19	20	9	26	11	16	41	135	18	23	28	53	93	115	81	28	16	149	255	255	255	255
255	255	255	187	23	17	25	44	67	86	40	22	32	177	53	8	70	221	244	255	157	4	14	137	255	255	255	255
236	253	255	198	7	18	85	242	252	247	89	15	15	191	50	6	49	255	255	248	138	14	24	118	255	255	255	255
255	255	246	218	48	23	43	243	254	251	58	12	43	212	46	24	44	185	204	153	51	8	4	128	255	255	255	255
250	251	255	237	38	1	44	225	231	203	40	18	29	198	66	22	21	15	25	29	17	40	10	99	255	255	255	255
255	235	255	255	75	21	25	40	43	37	5	33	22	195	117	0	12	16	21	19	0	5	16	70	255	255	255	255
255	240	255	238	94	0	19	14	14	22	17	15	24	206	211	128	83	93	89	77	69	25	27	68	255	255	255	255
250	255	253	251	205	50	22	36	41	73	42	2	60	220	255	255	255	243	255	251	204	42	0	120	255	255	255	255
255	255	255	253	244	232	221	214	242	243	214	178	193	245	255	248	254	255	248	252	255	202	176	231	255	255	255	255
253	254	255	255	254	252	249	247	255	255	255	245	251	255	255	254	248	255	249	251	255	243	228	255	255	255	255	255
255	255	254	254	255	255	255	255	255	255	255	255	255	249	249	254	251	255	253	247	255	255	255	255	255	255	255	255
255	255	255	253	252	252	253	254	255	254	252	255	251	245	251	255	255	255	255	249	248	255	255	246	255	255	255	255

Gambar 4.5: Salah Satu Sampel Nilai Matriks *Red* Huruf *Hangeul* "BB"

247	255	255	255	248	255	253	255	255	255	248	255	255	249	252	255	255	245	255	244	255	254	255	246	255	255	255	255
255	255	244	246	254	255	253	252	255	254	250	247	255	255	255	242	250	253	255	255	255	247	255	255	255	255	255	255
250	255	255	255	255	255	250	253	255	248	255	254	255	239	250	255	255	255	248	244	255	255	255	248	255	255	255	255
251	255	251	253	247	253	252	255	255	251	253	245	253	255	253	255	249	255	255	253	255	242	248	255	255	255	255	255
255	255	249	255	250	255	255	250	248	255	255	255	255	255	239	249	251	255	249	255	255	254	255	255	255	255	255	255
250	255	255	243	184	216	248	249	254	206	187	216	253	199	115	153	255	254	234	255	245	241	255	250	255	255	255	255
255	249	248	136	34	74	221	255	255	101	21	74	220	124	0	19	213	254	254	245	127	104	201	255	255	255	255	255
255	243	255	102	0	18	207	248	246	89	10	0	171	88	2	28	197	247	254	200	20	0	138	242	255	255	255	255
254	255	254	92	0	23	193	254	254	170	2	12	109	116	19	9	184	254	250	191	16	20	84	248	255	255	255	255
255	252	252	92	16	35	200	252	250	194	27	22	99	117	16	15	152	250	254	224	32	8	69	248	255	255	255	255
255	254	255	85	5	13	190	249	254	214	24	5	73	123	8	8	153	253	246	238	49	7	52	232	255	255	255	255
253	248	255	88	15	22	202	254	251	229	35	7	62	141	14	13	131	250	239	239	59	11	44	223	254	255	255	255
255	250	251	92	19	26	191	244	246	235	54	18	51	141	11	10	128	253	253	242	64	7	29	214	254	255	255	255
255	255	254	102	1	14	185	253	253	233	51	12	42	142	15	5	105	199	182	145	33	6	19	202	254	255	255	255
244	255	254	127	10	21	159	224	203	168	36	15	42	137	26	18	25	53	30	14	4	18	18	195	254	255	255	255
255	255	244	138	21	1	46	43	17	43	8	20	23	93	4	20	10	13	6	2	26	31	4	178	254	255	255	255
255	255	240	148	26	13	17	18	7	24	8	13	38	132	15	20	25	51	91	113	79	26	15	148	254	255	255	255
255	255	254	186	22	16	24	42	65	84	38	19	29	174	50	5	67	219	242	253	155	2	13	136	254	255	255	255
236	253	254	197	6	17	84	240	250	245	87	13	12	188	47	3	47	253	253	246	136	12	23	117	254	255	255	255
255	255	245	217	47	22	42	242	252	249	56	10	40	209	43	21	42	183	202	151	49	7	3	127	254	255	255	255
250	251	255	237	37	0	43	224	229	201	38	16	27	196	64	20	19	13	23	27	15	39	9	98	254	255	255	255
255	235	255	255	74	20	24	39	42	35	3	31	20	193	115	0	10	14	19	18	0	4	15	69	255	255	255	255
255	240	255	238	93	0	18	13	13	21	16	13	22	204	209	126	81	92	88	76	68	24	26	67	255	255	255	255
250	255	253	251	204	49	21	35	40	72	41	1	59	219	253	254	254	242	254	250	203	41	0	119	255	255	255	255
255	255	255	253	244	232	221	214	241	242	213	177	192	244	254	247	253	254	247	251	254	202	176	231	255	255	255	255
253	254	255	255	254	252	249	247	255	255	254	244	250	254	254	253	247	255	248	251	255	243	228	255	255	255	255	255
255	255	254	254	255	255	255	255	255	255	255	255	255	249	248	254	251	255	253	247	255	255	255	255	255	255	255	255
255	255	255	253	252	252	253	254	255	254	252	255	251	245	251	255	255	255	255	249	248	255	255	246	255	255	255	255

Gambar 4.6: Salah Satu Sampel Nilai Matriks *Green* Huruf *Hangeul* "BB"

247	255	255	255	248	255	251	253	253	253	246	253	251	244	247	251	251	240	253	242	253	252	253	244	255	255	255	255
255	255	244	246	254	255	251	250	253	252	248	245	251	251	251	237	245	248	253	253	253	245	253	253	255	255	255	255
250	255	255	255	255	255	250	253	253	246	253	252	253	237	248	253	253	253	246	242	253	253	255	248	255	255	255	255
251	255	251	253	247	253	252	255	253	249	251	243	251	253	253	253	247	253	253	251	253	240	248	255	255	255	255	255
255	255	249	255	250	255	255	250	248	255	255	255	255	255	241	249	251	255	249	255	255	254	255	255	255	255	255	255
250	255	255	243	184	216	248	249	254	206	189	218	255	201	120	155	255	254	236	255	245	241	255	250	255	255	255	255
255	251	250	138	36	76	223	255	255	103	26	79	225	129	5	24	218	255	255	247	129	106	203	255	255	255	255	255
255	245	255	104	2	20	209	250	251	94	15	5	177	94	10	34	203	252	255	205	25	2	140	244	255	255	255	255
255	255	255	94	5	28	198	255	255	176	8	20	117	124	30	17	192	255	255	197	22	25	89	253	255	255	255	255
255	254	254	94	21	40	205	255	255	202	35	33	110	128	29	26	163	255	255	230	38	14	75	253	255	255	255	255
255	255	255	87	10	18	196	255	255	225	35	18	86	136	22	21	164	255	255	246	57	13	58	237	255	255	255	255
255	250	255	90	20	27	208	255	255	240	48	20	76	155	28	27	144	255	250	250	70	19	50	229	255	255	255	255
255	252	255	97	25	32	199	252	255	248	68	32	68	158	28	27	141	255	255	253	75	15	35	220	255	255	255	255
255	255	255	107	7	20	193	255	255	246	65	26	59	159	34	22	119	212	195	158	44	17	27	208	255	255	255	255
246	255	255	132	16	27	167	235	216	181	50	32	61	156	47	37	42	67	43	27	15	29	26	201	255	255	255	255
255	255	249	143	27	7	57	54	30	56	25	37	42	112	25	39	27	27	19	15	37	42	12	184	255	255	255	255
255	255	245	153	32	19	28	29	20	37	25	30	57	151	36	39	42	65	104	126	90	37	23	154	255	255	255	255
255	255	255	191	28	22	32	53	78	97	52	36	48	193	71	24	84	233	255	255	166	13	21	142	255	255	255	255
238	255	255	202	12	23	92	251	255	255	101	27	29	205	66	20	61	255	255	255	147	23	31	123	255	255	255	255
255	255	250	222	53	28	50	250	255	255	70	24	57	226	60	38	55	196	215	162	60	15	9	133	255	255	255	255
252	253	255	239	42	5	49	232	240	212	51	29	41	210	78	34	32	24	34	38	26	47	15	104	255	255	255	255
255	237	255	255	79	25	30	45	50	46	14	44	33	206	129	11	21	25	30	26	7	10	21	74	255	255	255	255
255	242	255	240	98	4	23	19	19	29	24	24	33	215	222	137	92	100	96	82	74	30	32	72	255	255	255	255
252	255	255	253	209	54	26	40	46	78	47	9	67	227	255	255	255	248	255	255	209	46	4	124	255	255	255	255
255	255	255	255	246	234	223	216	246	247	218	183	198	250	255	253	255	255	252	255	255	204	178	233	255	255	255	255
255	255	255	255	255	254	251	249	255	255	255	249	255	255	255	255	252	255	253	253	255	245	230	255	255	255	255	255
255	255	255	254	255	255	255	255	255	255	255	255	255	251	253	255	253	255	255	247	255	255	255	255	255	255	255	255
255	255	255	253	252	252	253	254	255	254	252	255	253	247	253	255	255	255	255	249	248	255	255	246	255	255	255	255

Gambar 4.7: Salah Satu Sampel Nilai Matriks *Blue* Huruf *Hangeul* "BB"

2. *RGB to Grayscale*

Citra RGB karakter tulisan Korea *(Hangeul)* diubah kedalam bentuk *grayscale*, yang berarti setiap piksel akan memiliki nilai antara 0-255 berdasarkan tingkat keabuannya. Berikut ini merupakan perhitungan manual untuk beberapa piksel berdasarkan Persamaan 3.1:

- **Pixel (0,0):** *Red*=247, *Green*=247, *Blue*=247

  $Grayscale = 0,2989 \times Red + 0,5870 \times Green + 0,1140 \times Blue$

  $Grayscale = 0,2989 \times 247 + 0,5870 \times 247 + 0,1140 \times 247$

  $Grayscale = 246,98$

- **Pixel (1,1):** R=255, G=255, B=255

  $Grayscale = 0,2989 \times Red + 0,5870 \times Green + 0,1140 \times Blue$

  $Grayscale = 0,2989 \times 255 + 0,5870 \times 255 + 0,1140 \times 255$

  $Grayscale = 254,97$

- **Pixel (2,2):** R=255, G=255, B=255

  $Grayscale = 0,2989 \times Red + 0,5870 \times Green + 0,1140 \times Blue$

  $Grayscale = 0,2989 \times 255 + 0,5870 \times 255 + 0,1140 \times 255$

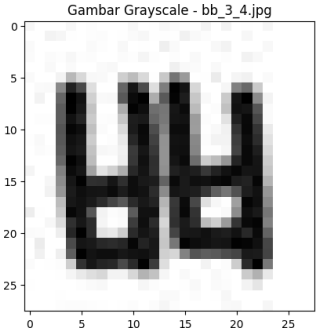
  $Grayscale = 254,97$

Selain perhitungan manual, pengubahan RGB ke *grayscale* dapat dilakukan dengan menggunakan program bahasa Python dengan menggunakan beberapa *library* antara lain adalah modul *OpenCV (Open Source Computer Vision Library)* yang berfungsi untuk memproses citra dan video, modul *numpy* yang digunakan untuk membuat dan mengelola array (matriks) yang menyimpan nilai-nilai pixel dari gambar, serta modul *matplotlib*, pustaka plotting yang digunakan untuk membuat grafik dan visualisasi data dalam Python. Implementasi pengubahan dari RGB ke *grayscale* dengan menggunakan Python dapat dilihat di Lampiran 1, yang berisi kode lengkap untuk memproses gambar karakter tulisan Korea *(Hangeul)* dari format RGB menjadi *grayscale*.

Pada Gambar 4.8 menunjukkan citra RGB yang sudah diubah menjadi citra *grayscale*. Salah satu citra karakter tulisan Korea *(Hangeul)* "BB" nilai keabuan citra ditunjukkan pada Gambar 4.9 yang diambil dari nilai keabuan dari citra karakter tulisan Korea *(Hangeul)*. Gambar 4.9 menunjukkan nilai pada setiap piksel

bervariasi berdasarkan tingkat keabuannya.

Berikut merupakan gambar citra hasil *grayscale* dan nilai matriks yang menunjukkan tingkat keabuan dari setiap piksel dalam citra tersebut.



Gambar 4.8: Salah Satu Sampel Citra RGB Huruf *Hangeul* "BB" diubah menjadi *Grayscale*.

246	254	254	255	245	255	253	254	253	255	250	252	255	247	246	255	251	245	254	243	255	252	252	245	254	254	254	254
255	255	243	243	254	255	246	251	255	248	248	244	250	255	254	239	249	252	253	248	254	248	255	252	254	254	254	254
248	254	254	253	255	254	250	253	252	250	254	255	254	238	250	253	254	253	247	248	254	250	253	247	254	254	254	254
250	254	251	252	244	251	253	253	255	248	252	243	255	252	249	255	249	254	255	251	251	245	245	253	254	254	254	254
255	253	247	253	250	255	255	248	244	255	255	252	255	253	243	247	248	252	248	253	255	250	252	255	254	254	254	254
246	255	253	246	181	213	244	248	255	204	186	215	253	198	118	150	255	255	234	254	245	238	255	248	254	254	254	254
255	247	248	134	37	71	222	255	252	102	21	76	220	125	1	21	212	253	255	241	128	106	199	255	254	254	254	254
253	244	253	103	0	21	206	248	247	89	11	1	173	89	4	29	199	248	253	204	19	0	139	241	254	254	254	254
255	255	253	93	1	25	193	255	255	170	5	14	111	117	22	9	187	255	248	195	18	22	85	248	254	254	254	254
253	250	252	94	16	35	202	253	253	194	25	24	103	122	21	19	155	250	255	225	30	9	71	250	254	254	254	254
255	253	255	84	6	14	193	247	253	217	30	13	77	125	10	13	156	254	246	243	50	7	53	235	254	254	254	254
252	249	255	89	16	21	205	255	252	233	36	9	64	146	18	18	135	254	241	243	61	16	46	220	254	254	254	254
254	249	250	95	20	25	195	242	247	243	60	23	56	149	15	13	130	254	250	244	70	6	27	219	254	254	254	254
253	255	253	102	2	16	188	255	255	231	55	16	45	147	20	12	111	206	187	149	36	11	22	200	254	254	254	254
245	255	253	126	10	24	159	223	207	176	41	19	47	143	31	25	26	57	35	18	7	22	21	196	254	254	254	254
253	254	244	143	21	2	47	49	19	47	11	25	31	99	8	27	17	16	9	5	31	33	5	179	254	254	254	254
253	255	245	146	28	12	18	22	12	26	14	18	42	138	23	25	31	56	94	116	83	28	17	149	254	254	254	254
255	253	253	186	22	19	29	44	70	88	44	25	34	180	56	8	72	221	246	255	155	5	17	134	254	254	254	254
237	251	255	204	5	17	88	240	251	246	91	16	17	194	54	9	52	253	255	244	142	16	24	120	254	254	254	254
255	254	243	219	47	23	43	246	253	250	60	14	46	214	47	26	45	186	206	156	51	11	2	128	254	254	254	254
248	250	255	235	38	2	43	225	232	204	41	19	30	199	67	24	23	16	26	32	15	42	9	100	254	254	254	254
255	235	253	255	74	23	26	39	44	39	7	35	24	199	119	3	15	13	20	20	2	4	17	72	254	254	254	254
252	243	252	238	93	0	20	14	14	23	16	16	24	209	211	131	84	97	90	76	71	25	29	62	254	254	254	254
250	255	251	251	208	49	24	36	41	74	41	5	60	221	254	254	254	241	255	251	203	41	3	121	254	254	254	254
254	255	255	252	244	233	221	214	243	243	214	176	195	244	254	246	252	255	248	251	252	202	177	230	254	254	254	254
252	253	255	255	254	252	248	246	252	255	254	245	249	255	255	254	248	252	249	250	255	241	227	255	254	254	254	254
255	254	254	253	253	254	254	255	253	254	254	255	253	246	248	253	252	253	253	247	251	255	253	253	254	254	254	254
255	254	255	255	253	255	255	255	255	255	255	255	255	253	255	254	254	254	253	252	253	252	250	255	255	255	255	255

Gambar 4.9: Salah Satu Sampel Nilai Matriks Citra RGB Huruf *Hangeul* "BB" diubah menjadi *Grayscale*.

3. *Denoising*

Proses selanjutnya pada tahap *pre-processing* adalah *denoising*. Proses ini digunakan untuk menghilangkan noise yang terdapat dalam gambar. Metode yang digunakan pada proses *denoising* adalah metode *median filter*. Proses *denoising* dilakukan dengan menggunakan program bahasa Python dengan menggunakan *Google Laboratory* yang ditunjukkan pada lampiran 2, di mana *library* yang digunakan adalah `OpenCV` dan `NumPy`. Kode Python untuk melakukan proses ini menggunakan fungsi `cv2.medianBlur()` dari pustaka `OpenCV` dan *source code* lengkapnya dapat dilihat pada Lampiran 2.

Proses *median filter* dilakukan dengan memilih nilai median dari piksel-piksel yang ada di sekitar piksel target. Berikut adalah contoh perhitungan manual pada beberapa piksel citra:

- **Proses *median filter* untuk Piksel (1, 1)**
  Nilai piksel sekitar:

243	246	248
254	254	254
254	255	255

  Nilai median = 254

- **Proses *median filter* untuk Piksel (1, 2)**
  Nilai piksel sekitar:

243	243	253
254	254	254
254	255	255

Nilai median = 254

- **Proses *median filter* untuk Piksel (1, 3)**
  Nilai piksel sekitar:

243	243	245
253	254	254
	255	255

254

Nilai median = 254

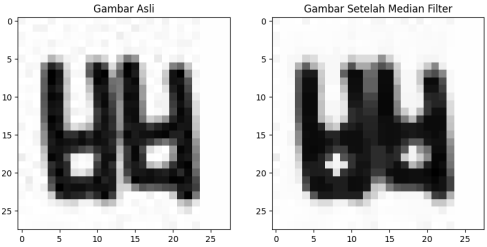
- **Proses *median filter* untuk Piksel (1, 4)**
  Nilai piksel sekitar:

243	245	253
254	254	255
	255	255

255

Nilai median = 254

Pada Gambar 4.10 menunjukkan citra karakter tulisan Korea *(Hangeul)* yang mengandung *noise* dan hasil setelah proses *denoising*.



Gambar 4.10: Salah Satu Sampel Citra yang mengandung Noise Huruf *Hangeul* "BB"

Berikut ini merupakan salah satu sampel matriks yang merepresentasikan hasil *denoising* dari huruf *Hangeul* yang ditampilkan sebagai berikut.

246	254	254	255	245	255	253	254	253	255	250	252	255	247	246	255	251	245	254	243	255	252	252	245	254	254	254	254
255	254	254	254	254	254	253	253	253	252	250	252	252	250	250	251	252	252	248	253	250	253	252	253	254	254	254	254
248	254	253	252	253	253	253	253	252	252	248	252	252	252	252	250	253	253	252	251	250	251	250	253	254	254	254	254
250	253	253	252	253	253	253	253	252	252	252	254	253	252	250	249	253	252	252	251	251	251	250	253	254	254	254	254
255	253	253	250	250	250	251	253	248	252	248	252	252	252	247	248	252	252	253	251	251	250	250	254	254	254	254	254
246	253	248	247	213	222	248	248	248	244	204	220	220	220	150	212	248	252	253	248	245	245	250	254	254	254	254	254
255	253	247	181	103	181	222	248	248	186	89	173	173	125	89	118	212	253	253	241	204	139	238	254	254	254	254	254
253	253	247	103	37	37	206	248	248	102	21	21	111	111	22	22	199	248	248	204	106	85	139	248	254	254	254	254
255	253	250	94	25	25	202	248	248	170	24	24	103	103	22	22	187	248	248	204	22	22	85	248	254	254	254	254
253	253	252	93	25	25	193	253	253	194	25	25	103	103	21	21	156	248	248	225	30	30	71	248	254	254	254	254
255	253	250	89	21	21	202	252	252	217	30	30	77	77	19	19	155	246	246	241	50	46	53	235	254	254	254	254
252	253	249	89	21	21	195	247	247	233	36	36	64	64	18	18	135	246	246	243	61	46	46	220	254	254	254	254
254	253	249	95	21	21	195	247	247	233	55	45	56	56	18	18	130	206	243	187	61	27	27	219	254	254	254	254
253	253	250	102	24	24	188	223	242	207	55	45	47	47	25	25	57	130	187	70	22	22	22	200	254	254	254	254
245	253	253	126	21	21	49	188	207	55	41	31	45	45	27	25	26	35	35	31	22	22	22	196	254	254	254	254
253	253	245	143	24	21	24	47	47	26	25	25	42	42	27	25	26	31	35	31	28	22	28	179	254	254	254	254
253	253	245	146	22	21	22	29	44	26	25	25	34	42	27	25	27	56	94	94	33	28	28	149	254	254	254	254
255	253	251	186	22	19	22	44	70	70	26	25	34	54	54	31	52	94	244	155	116	24	24	134	254	254	254	254
237	253	251	204	23	23	43	88	246	91	60	34	34	54	54	47	52	206	244	206	142	17	17	128	254	254	254	254
255	251	250	219	38	38	43	232	246	232	60	30	30	54	54	45	26	52	186	142	42	16	24	120	254	254	254	254
248	253	250	235	47	38	39	44	225	60	39	30	35	67	67	26	23	23	26	26	20	11	17	100	254	254	254	254
255	252	250	238	74	26	23	39	39	39	23	24	30	119	131	67	23	23	26	26	25	17	29	72	254	254	254	254
252	252	251	251	93	26	24	26	39	39	23	24	35	199	209	131	97	90	90	76	41	25	29	72	254	254	254	254
250	252	252	251	233	93	36	36	41	41	41	41	176	211	244	252	246	248	248	248	202	71	62	177	254	254	254	254
254	254	255	252	251	233	221	221	243	243	214	195	221	249	254	254	252	252	251	251	250	203	202	230	254	254	254	254
252	254	254	254	253	252	248	248	252	253	254	249	246	249	253	252	252	252	251	251	251	251	241	253	254	254	254	254
255	254	254	253	253	252	252	253	254	254	254	253	250	250	254	254	253	253	252	250	250	253	253	254	254	254	254	254
255	254	253	252	251	251	252	253	255	254	250	253	250	245	254	255	255	254	255	248	245	255	254	245	254	254	254	254

Gambar 4.11: Salah Satu Sampel Nilai Matriks Citra yang mengandung Noise Huruf *Hangeul* "BB".

4. Binerisasi

Pada tahap ini, metode yang digunakan adalah metode otsu, dengan menggunakan pembagian level sebesar 0,572 pada sampel Gambar 4.12. Berikut adalah perhitungan untuk menentukan nilai

ambang batas *(Threshold)*:

$$T = Pembagianlevel \times 255$$
$$T = 0,572 \times 255$$
$$T = 146$$

Sampel citra *denoising* sudah di konversi ke skala keabuan *(grayscale)*, seperti ditunjukkan pada Gambar 4.12 merupakan pembagian antar tingkat level yang akan diubah menjadi bentuk biner yaitu 0 (piksel *background*) dan 1 (piksel karakter tulisan Korea *(Hangeul))*.

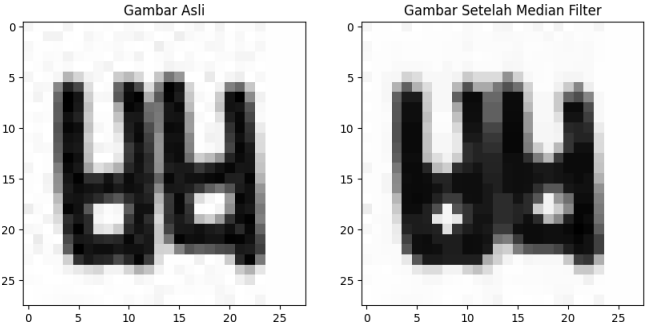
Gambar 4.12 merupakan sampel citra yang sudah di *denoising*. Gambar 4.13 yang telah diproses menggunakan metode otsu. Berikut perhitungan manual berdasarkan Persamaan 3.2. dalam proses *denoising* dengan nilai ambang batasnya T adalah 146.

$$\text{Citra Biner}(x, y) = \begin{cases} 1, & \text{jika } I(x, y) > T(146) \\ 0, & \text{jika } I(x, y) \leq T(146) \end{cases}$$

- Nilai Intensitas Piksel di (1,1) pada gambar hasil *denoising* adalah 254, dikarenakan $I(x, y)(254) > T(146)$ maka biner piksel di (1,1) pada gambar biner adalah 1
- Nilai Intensitas Piksel di (2,2) pada gambar hasil *denoising* adalah 254, dikarenakan $I(x, y)(254) > T(146)$ maka biner Piksel di (2,2) pada gambar biner adalah 1

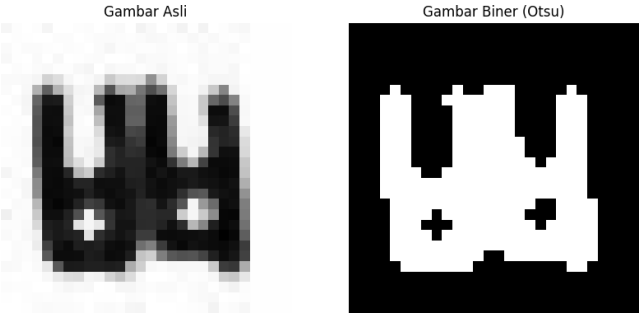
Selain perhitungan manual, proses ini juga dilakukan menggunakan program Python dengan *library* seperti `OpenCV` dan `NumPy`, serta fungsi `threshold` untuk melakukan binerisasi gambar yang ditunjukkan pada lampiran 3. Pada proses ini, sampel

citra bernilai 1 berarti piksel tersebut merupakan *background*, dan nilai 0 menunjukkan piksel yang mewakili karakter tulisan tangan Korea *(Hangeul)*. Berikut merupakan salah satu gambar citra hasil *denoising*.

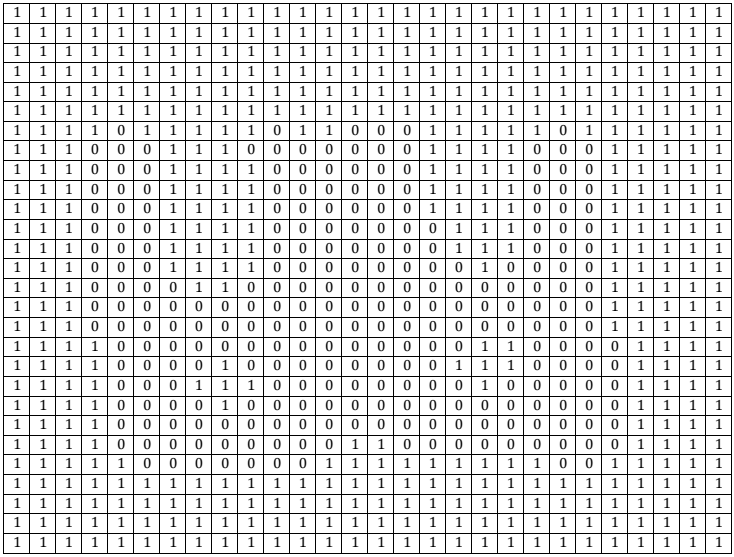


Gambar 4.12: Salah Satu Sampel Citra Mengandung Noise Huruf *Hangeul* "BB"

Berikut merupakan salah satu gambar citra dan nilai matriks hasil dari binerisasi.



Gambar 4.13: Salah Satu Sampel Citra dan Nilai yang Mengandung Binerisasi Huruf *Hangeul* "BB"



Gambar 4.14: Salah Satu Sampel Nilai Matriks Citra Mengandung Binerisasi Huruf *Hangeul* "BB"

Selain mengubah citra hasil *denoising* menjadi citra biner, proses binarisasi juga melibatkan tahapan invers. Tahap ini dilakukan untuk mempermudah tahapan analisis atau pengolahan citra selanjutnya, terutama ketika algoritma yang digunakan lebih efektif jika objek utama berwarna putih dan latar belakang berwarna hitam (Gonzales dkk., 2018).

Dalam implementasinya, proses invers dapat dilakukan menggunakan program bahasa Python *library* OpenCV dengan fungsi `cv2.bitwise_not()` yang membalikkan setiap nilai piksel dalam citra biner yang program kompleksnya ditunjukkan pada lampiran 4. Sebagai alternatif, proses invers biner dapat dilakukan secara manual dengan menggunakan fungsi `invert_binary_manual()` yang mengubah matriks biner.

Berikut ini merupakan perhitungan manual untuk beberapa piksel berdasarkan pada Persamaan 3.3.:

- **Pixel (0,0)**

$$I'(0, 0) = 1 - 1$$
$$I'(0, 0) = 0$$

- **Pixel (1,1)**

$$I'(0, 0) = 1 - 1$$
$$I'(0, 0) = 0$$

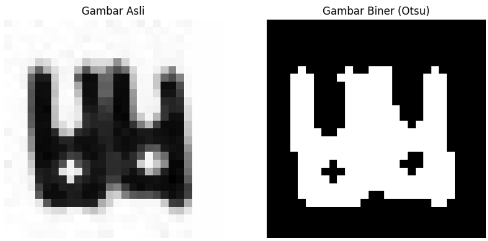
- **Pixel (2,2)**

$$I'(0, 0) = 1 - 1$$
$$I'(0, 0) = 0$$

Dimana matriks biner dengan nilai 1 diubah menjadi 0, dan sebaliknya. Hasil dari proses invers tersebut dapat dilihat pada Gambar 4.15.

Keterangan:

- 1 : warna putih (background)
- 0 : warna hitam (karakter tulisan Korea (*Hangeul)*)



Gambar 4.15: Salah Satu Citra Hasil Invers Huruf *Hangeul* "BB"

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Gambar 4.16: Salah Satu Nilai Invers Huruf *Hangeul* "BB"

Sampel citra hasil binerisasi pada Gambar 4.14 setelah dilakukan invers maka yang awalnya *background* bernilai 1 menjadi bernilai 0 dan citra karakter tulisan Korea *(Hangeul)* yang awalnya bernilai 0 menjadi bernilai 1. Hasil invers dari Gambar 4.15 ditunjukkan oleh Gambar 4.16.

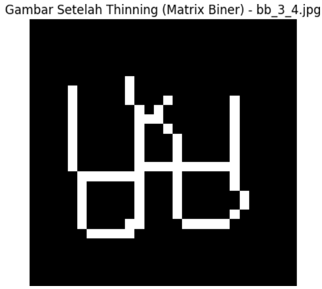
5. *Thinning*

Proses *thinning* merupakan teknik yang digunakan untuk mengurangi ketebalan karakter tulisan Korea *(Hangeul)* yang sebelumnya setebal 2 piksel atau lebih menjadi ketebalan 1 piksel saja (Chaki S Dey, 2019). Proses ini penting dalam pengenalan karakter, karena mengurangi noise dan memastikan bahwa hanya kontur utama dari karakter yang dipertahankan. Hal ini sangat bermanfaat dalam pengolahan citra tulisan tangan, terutama untuk memastikan bahwa bentuk tulisan yang dikenali lebih ramping dan tidak terpengaruh oleh ketebalan garis yang tidak relevan.

Implementasi *thinning* dilakukan menggunakan bahasa pemrograman Python dengan memanfaatkan beberapa *library* penting, seperti *OpenCV, NumPy, Matplotlib*, dan *scikit-image*. Fungsi *thinning* diimplementasikan dengan menggunakan fungsi `skeletonize` dari `skimage.morphology` yang berfungsi untuk mengurangi gambar biner menjadi kerangka dengan ketebalan satu piksel. Sebelumnya, gambar dikonversi menjadi biner dengan menggunakan thresholding Otsu dari *OpenCV*. Proses *thinning* ini dilakukan dengan cara menghapus piksel-piksel yang tidak memenuhi kondisi tertentu, mempertahankan hanya garis utama dari karakter. *Source code* lengkapnya dapat dilihat di Lampiran 5.

Hasil dari tahap *thinning* dapat dilihat pada Gambar 4.17, yang menunjukkan bahwa goresan karakter tulisan Korea *(Hangeul)* yang sebelumnya memiliki ketebalan lebih dari 1 piksel menjadi lebih tipis, dengan ketebalan hanya satu piksel. Pada Gambar 4.16, setelah melalui proses *thinning*, karakter-karakter yang memiliki ketebalan lebih dari 1 piksel, diubah menjadi satu piksel seperti pada Gambar 4.17. Proses ini memastikan

bahwa karakter-karakter tersebut tetap terjaga bentuknya, tetapi dengan ketebalan yang lebih ramping, yang dapat dilihat dengan nilai matriks pada Gambar 4.18. Nilai pada piksel-piksel yang sebelumnya memiliki ketebalan 2 atau lebih menjadi berkurang 1 piksel, seperti yang ditunjukkan pada gambar berikut.



Gambar 4.17: Salah Satu Sampel Citra Hasil *Thinning* Karakter Tulisan Korea *(Hangeul)*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Gambar 4.18: Salah Satu Nilai Sampel Citra Hasil *Thinning* Huruf *Hangeul* "BB"

C. Data Augmentasi

Setelah dilakukan tahap *preprocessing*, selanjutnya adalah proses augmentasi data yang digunakan untuk meningkatkan performa dan akurasi dari model serta mencegah terjadinya *overfitting* dan *underfitting*. Berdasarkan penelitian yang dilakukan oleh Ayuni dkk. (2023) menunjukkan bahwa teknik augmentasi data terbukti dapat meningkatkan jumlah dan variasi data, yang berdampak pada peningkatan akurasi model. Pada dataset citra penyakit daun padi, akurasi model sebelum augmentasi mencapai 85,71%, sedangkan setelah augmentasi meningkat menjadi 98,91%, menunjukkan bahwa augmentasi data berperan signifikan dalam meningkatkan performa model.

Tahapan ini memanfaatkan beberapa *library* Python, antara lainnya adalah `tensorflow` dan `keras` untuk melakukan augmentasi data. Fungsi `Image Data Generator from tensorflow.keras.preprocessing.image` digunakan untuk melakukan augmentasi data pada setiap dataset. Augmentasi data dilakukan sebanyak 6 jenis dengan parameter yang berbeda-beda, yaitu `rotation_range`, `width_shift_range`,`height_shift_range`, `horizontal_flip`, `shear_range`, dan `zoom_range`. Gambar dimuat dengan fungsi `load_img` dan diubah menjadi array menggunakan `img_to_array`, kemudian diproses oleh generator. Setelah augmentasi, gambar disimpan dalam format BGR menggunakan `cv2.imwrite`, karena OpenCV menggunakan urutan warna *Blue-Green-Red. Source code* lengkap untuk augmentasi dapat dilihat di Lampiran 7. Proses augmentasi data sebelum dilakukan augmentasi dan sesudah dilakukan

augmentasi data ini dapat dilihat pada Gambar 4.19 dan 4.20 sebagai berikut.

Gambar 4.19: Salah Satu Sampel Citra Sebelum Augmentasi Data Huruf *Hangeul*

Gambar 4.20: Salah Satu Sampel Citra Setelah Augmentasi Data Huruf *Hangeul*

D. Pembagian Data

Setelah tahap augmentasi data, data akan dibagi menjadi data *training* dan data *testing* untuk mengukur kinerja model secara objektif dan menghindari adanya *overfitting*. Pembagian data ini dilakukan menggunakan program Python dengan memanfaatkan beberapa *library* utama seperti `NumPy`, `scikit-learn`, dan `OpenCV`. Fungsi `train_test_split` digunakan untuk memecah dataset sesuai dengan rasio tertentu, sedangkan fungsi-fungsi tambahan membantu dalam memuat citra, melakukan *preprocessing*, normalisasi, serta menyimpan hasil split ke dalam struktur folder yang telah ditentukan. Proses ini dilakukan dengan rasio pembagian data yang berbeda, yaitu 80% (*training*):20% (*testing*), 70% (*training*):30% (*testing*), dan 90% (*training*):10% (*testing*). Program Python yang digunakan untuk melakukan tahap ini dapat dilihat secara lengkap pada Lampiran

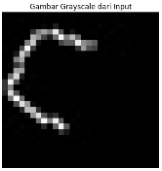
6. Pembagian dataset ini dapat dilihat pada Gambar 4.21 di bawah ini.

```
Membagi dan menyimpan dataset 80:20...
80:20 Split: 9280 sampel pelatihan, 2320 sampel pengujian
Membagi dan menyimpan dataset 70:30...
70:30 Split: 8120 sampel pelatihan, 3480 sampel pengujian
Membagi dan menyimpan dataset 90:10...
90:10 Split: 10440 sampel pelatihan, 1160 sampel pengujian
```

Gambar 4.21: Pembagian Data Karakter Tulisan Korea *(Hangeul)*

E. Ekstraksi Fitur

Tahap selanjutnya setelah pembagian data adalah ekstraksi fitur, dimana setiap citra melalui proses ekstraksi sebelum mengidentifikasi kemiripan di antara karakter tulisan Korea *(Hangeul)*. Gambar yang digunakan dalam tahap ini adalah gambar karakter tulisan Korea *(Hangeul)* yang telah diproses sebelumnya untuk memastikan data berkualitas tinggi. Gambar berkualitas tinggi akan meningkatkan kejelasan informasi di dalamnya, meminimalkan *noise* dan memfasilitasi proses identifikasi yang lebih akurat. Sebelum dilakukan ekstraksi fitur HOG citra karakter tulisan Korea *(Hangeul)* dilakukan proses *thinning* diubah kedalam *grayscale* jika data sebelumnya belum diubah, seperti pada Gambar 4.22.



Gambar 4.22: Sampel Citra Hasil *Thinning*

Dari citra karakter tulisan Korea *(Hangeul)* seperti pada Gambar 4.22 didapatkan cuplikan matriks seperti pada Gambar 4.23.

0	0	0	0	0	0	0,003921569	0,003921569
0,007843137	0	0,007843137	0,019607843	0	0,003921569	0	0,003921569
0	0	0	0	0,003921569	0	0,003921569	0,031372549
0	0,007843137	0	0,007843137	0	0,188235294	0,490196078	0,62745098
0	0	0,007843137	0	0,003921569	0,690196078	0,560784314	0,270588235
0	0,003921569	0	0,007843137	0,494117647	0,77254902	0,070588235	0
0,011764706	0	0	0,309803922	0,97254902	0,258823529	0	0,003921569
0	0,003921569	0,090196078	0,788235294	0,470588235	0,007843137	0	0

Gambar 4.23: Salah satu cuplikan matriks sampel citra hasil *grayscale* alfabet *Hangeul* "D" dengan dimensi 8×8

Pada tahap ini, dilakukan perhitungan *gradient magnitude* setiap piksel dari masing - masing *cell* pada arah sumbu *horizontal* dan vertikal dengan menerapkan operator Sobel pada matriks citra karakter tulisan Korea *(Hangeul)*. Proses ini melibatkan operasi konvolusi dengan matriks operator Sobel untuk sumbu *horizontal* (G_x) dan vertikal (G_y), seperti ditunjukkan pada Persamaan 2.31 dan 2.32.

Berikut merupakan contoh perhitungan konvolusi pada sampel matriks citra karakter tulisan Korea *(Hangeul)* berukuran 3×3 dengan matriks operator Sobel sumbu *horizontal* (G_x).

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \circledast \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sampel perhitungan untuk setiap elemen adalah sebagai berikut:

$$
\begin{aligned}
(1,1) &= 0(-1) + 0(0) + 0(1) + 0,007843137(-2) + 0(0) \\
&\quad +0,007843137(2) + 0(-1) + 0(0) + 0(1) \\
&= 0 \\
(1,2) &= 0(-1) + 0(0) + 0(1) + 0(-2) + 0,007843137(0) \\
&\quad +0,019607843(2) + 0(-1) + 0(0) + 0(1) \\
&= 0,039215686 \\
(2,1) &= 0,007843137(-1) + 0(0) + 0,007843137(1) + 0(-2) + 0(0) \\
&\quad +0(2) + 0(-1) + 0,007843137(0) + 0(1) \\
&= 0 \\
(2,2) &= 0(-1) + 0,007843137(0) + 0,019607843(1) + 0(-2) \\
&\quad +0(0) + 0(2) + 0,007843137(-1) + 0(0) \\
&\quad +0,\ 007843137(1) \\
&= 0,019607843
\end{aligned}
$$

Maka diperoleh salah satu hasil konvolusi matriks dengan sampel (1,1),(1,2),(2,1),(2,2) sumbu *horizontal* (G_x) yaitu:

$$G_x = \begin{bmatrix} 0 & 0,039215686 \\ 0 & 0,019607843 \end{bmatrix}.$$

Selanjutnya, melakukan perhitungan manual konvolusi cuplikan matriks salah satu sampel citra karakter Tulisan Korea *(Hangeul)* berukuran 3×3 dengan matriks operator Sobel sumbu vertikal (G_y):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0,007843137 & 0 & 0,007843137 \\ 0 & 0 & 0 \end{bmatrix} \circledast \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Maka diperoleh salah satu hasil konvolusi matriks dengan sampel (1,1),(1,2),(2,1),(2,2) sumbu vertikal (G_y) yaitu:

$$
\begin{aligned}
(1,1) &= 0(-1) + 0(-2) + 0(-1) + 0,007843137(0) \\
&\quad +0(0) + 0,007843137(0) + 0(1) + 0(2) + 0(1) \\
&= 0 \\
(1,2) &= 0(-1) + 0(-2) + 0(-1) + 0(0) + 0,007843137(0) \\
&\quad +0,019607843(0) + 0(1) + 0(2) + 0(1) \\
&= 0 \\
(2,1) &= 0,007843137(-1) + 0(-2) + 0,007843137(-1) + 0(0) + 0(0) \\
&\quad +0(0) + 0(1) + 0,007843137(2) + 0(1) \\
&= 0 \\
(2,2) &= 0(-1) + 0,007843137(-2) + 0,019607843(-1) + 0(0) \\
&\quad +0(0) + 0(0) + 0,007843137(1) + 0(2) \\
&\quad +0,\ 007843137(1) \\
&= -0,019607843
\end{aligned}
$$

Maka didapatkan sampel perhitungan matriks sumbu vertikal (G_y) yaitu:

$$
G_y = \begin{bmatrix} 0 & 0 \\ 0 & -0,019607843 \end{bmatrix}
$$

Setelah mendapatkan nilai masing-masing dari sumbu *x* dan *y* pada tiap piksel, dilakukan perhitungan untuk mencari *gradient magnitude* menggunakan Persamaan 2.33. Berikut sampel perhitungan untuk mencari *gradient magnitude*:

$$\mu(1,1) = \sqrt{(0)^2 + (0)^2}$$
$$= 0$$
$$\mu(1,2) = \sqrt{(0,039215686)^2 + (0)^2}$$
$$= 0,039215686$$
$$\mu(2,1) = \sqrt{(0)^2 + (0)^2}$$
$$= 0$$
$$\mu(2,2) = \sqrt{(0,019607843)^2 + (-0,019607843)^2}$$
$$= 0,02772967769$$

Maka didapatkan sampel matriks *gradient magnitude* (μ) yaitu:

$$\mu = \begin{bmatrix} 0 & 0,039215686 \\ 0 & 0,02772967769 \end{bmatrix}$$

Selanjutnya dilakukan juga perhitungan untuk mencari *gradient orientation* (ϑ) dengan menggunakan Persamaan 2.34. Berikut sampel perhitungan untuk menentukan *gradient orientation*:

$$\vartheta(1,1) = \arctan \frac{0}{0} = 90^\circ$$
$$\vartheta(1,2) = \arctan \frac{0}{0,039215686} = 0^\circ$$
$$\vartheta(2,1) = \arctan \frac{0}{0} = 90^\circ$$
$$\vartheta(2,2) = \arctan \frac{-0,019607843}{0,019607843} = 135^\circ$$

Maka didapatkan sampel matriks gradient orientation (ϑ), yaitu:

$$\vartheta = \begin{matrix} 90^\circ & 0^\circ \\ 90^\circ & 135^\circ \end{matrix}$$

Langkah selanjutnya adalah melakukan proses *orientation binning* pada setiap *pixel* di dalam sel. Pada perhitungan histogram gradien, nilai *magnitude* digunakan untuk menentukan bobot untuk memperbarui nilai dalam setiap bin. Sedangkan, nilai orientasi digunakan untuk menentukan rentang derajat dalam perhitungan (misalnya yaitu nilai orientasi piksel adalah 35°, maka mempunyai bin yang mewakili rentang 20°-40°). Dalam pembagiannya, terdapat 9 bin (bagian), dimana setiap bin mewakili 20°, mulai dari 0 hingga 180°.

Berdasarkan penelitian yang dilakukan oleh Devita dkk. (2019) membuktikan bahwa membagi citra ke dalam sel berukuran lebih kecil, seperti (8,8) piksel, lebih efektif dalam mengekstraksi ciri dibandingkan dengan ukuran sel yang lebih besar. Pemilihan jumlah bin yang lebih banyak juga memungkinkan histogram *gradient magnitude* dapat menangkap variasi bentuk dengan lebih baik, sehingga meningkatkan kualitas fitur yang diekstraksi. Selain itu, penelitian yang dilakukan oleh Akbar S Utaminingrum (2019) menunjukkan bahwa penggunaan 9 bin dengan rentang 20° per bin dapat meningkatkan ketahanan terhadap variasi pencahayaan dan distribusi gradien dalam klasifikasi berbasis citra digital.

Histogram yang dihasilkan merupakan vektor atau array dari 9 bin, di mana setiap bin bertanggung jawab terhadap sudut-sudut tertentu, yaitu: Bin 0 untuk sudut 0° hingga 20°, Bin 1 untuk 20° hingga 40°, Bin 2 untuk 40° hingga 60°, Bin 3 untuk 60° hingga 80°, Bin 4 untuk 80° hingga 100°, Bin 5 untuk 100° hingga 120°,

Bin 6 untuk 120° hingga 140°, Bin 7 untuk 140° hingga 160°, dan Bin 8 untuk 160° hingga 180°. Hasil dari tahap *orientation binning* ini akan ditampilkan berdasarkan matriks *gradient magnitude* (μ) dan matriks *gradient orientation* (ϑ). Berikut merupakan contoh perhitungan yang ditulis menggunakan Persamaan 3.8.

Tabel 4.2: Hasil *orientation binning* terhadap matriks *gradient magnitude* [g] dan matriks *gradient orientation* [ϑ]

				0	0			
0				0	0	0, 0207973	0, 00693242	
0°	20°	40°	60°	80°	100°	120°	140°	160°

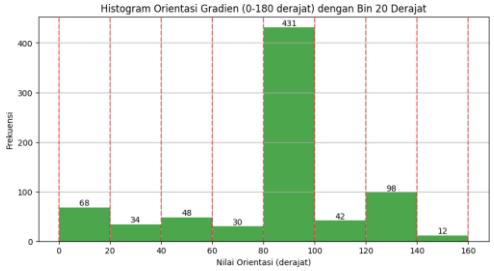
$$Bin(1,\ 1) = \frac{((90^{\circ}) \_\ 80^{\circ})}{20^{\circ}} \times 0 = 0,$$
$$Bin(1,\ 1) = \frac{(100^{\circ} \_\ (90^{\circ}))}{20^{\circ}} \times 0 = 0,$$

$$Bin(1,\ 2) = 0^{\circ} \times 0,039215686 = 0,$$
$$Bin(2,\ 1) = \frac{((90^{\circ}) - 80^{\circ})}{20^{\circ}} \times 0 = 0,$$
$$Bin(2,\ 1) = \frac{(100^{\circ} \_\ (90^{\circ}))}{20^{\circ}} \times 0 = 0,$$
$$Bin(2,\ 2) = \frac{(135^{\circ} \_\ 120^{\circ})}{20^{\circ}} \times 0,02772967769 = 0,0207973,$$
$$Bin(2,\ 2) = \frac{(140^{\circ} \_\ 135^{\circ})}{20^{\circ}} \times 0,02772967769 = 0,00693242$$

Perhitungan diatas dilakukan secara berulang dengan cara yang sama , sehingga dihasilkan dalam matriks 1×9 sebagai berikut

Tabel 4.3: Hasil *orientation binning* matriks 1×9 terhadap matriks *gradient magnitude* [g] dan matriks *gradient orientation* [ϑ].

68	34	48	30	431	42	98	12	21
0° - 20°	20° - 40°	40° - 60°	60° - 80°	80° -100°	100° -120°	120° -140°	140° -160°	160° - 180°



Gambar 4.24: Histogram Awal

Selanjutnya, dilakukan normalisasi blok untuk memastikan fitur tetap konsisten. Proses ini menggunakan langkah perhitungan yang sama seperti sebelumnya. Normalisasi ini akan membagi citra yang awalnya berdimensi 8×8 piksel per blok menjadi sub blok 4×4 dan setiap bagian berukuran 2×2 piksel yang ditunjukkan sebagai berikut.

0	0	0	0
0,007843137	0	0,007843137	0,019607843
0	0	0	0
0	0,007843137	0	0,007843137

Gambar 4.25: Salah Satu cuplikan matriks Sampel Citra Hasil *Grayscale* alfabet *Hangeul* "D" dengan Dimensi 4×4

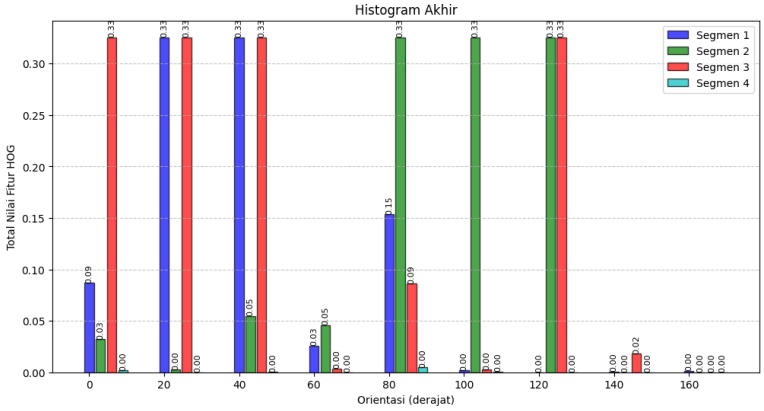
Perhitungan menggunakan perhitungan yang sama seperti sebelumnya, sehigga terbentuk menjadi 4 histogram, dimana masing - masing histogram berukuran 1×9. Pada Tabel 4.4 menunjukkan bahwa setiap nilai histogram mengalami konstruksi

vektor fitur, yaitu proses penggabungan histogram yang telah di normalisasi hingg membentuk ukurann 1×36. Nilai yang terdapat dalam Tabel 4.4 digunakan untuk menentukan nilai vektor fitur.

Tabel 4.4: Nilai Matriks 1×36

780,0	523,0	523,0	620,0	351,0	200,0	0	0	100,0	230,0	200,0	440,0	620,0	523,0	523,0	523,0	0	0	523,0	523,0	523,0	300,0	680,0	0,0	223,0	510,0	800,0	100,0	0	0	0	400,0	0	0	0	0

Hasil dari matriks 1×36 digambarkan dalam bentuk histogram matriks akhir seperti yang ditunjukkan pada Gambar 4.26



Gambar 4.26: Histogram Akhir

Nilai yang terdapat dalam matriks akhir akan di proses degan menggunakan Persamaan 2.35. dan 2.36.

$$v = [0{,}087132852;\ 0{,}325614727;\ 0{,}325614727;\ \cdots]$$

$$\|v\|_2^2 = |0{,}087132852|^2 + |0{,}325614727|^2 + |0{,}325614727|^2 + \cdots$$

$$\|v\|_2^2 = (0{,}087132852)^2 + (0{,}325614727)^2 + (0{,}325614727)^2 + \cdots$$

$$\|v\|_2^2 = 4$$

Kemudian, dilakukan pembagian tiap elemen pada vektor dengan $\sqrt{\|v\|_2^2 + \epsilon}$, sehingga diperoleh hasil sebagai berikut:

$$L_2\text{norm}: \quad v'' = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon}}$$

$$= \frac{[0{,}087132852;\ 0{,}325614727;\ 0{,}325614727;\ \cdots]}{\sqrt{\|v\|_2^2 + \epsilon}}$$

$$= \frac{[0{,}087132852;\ 0{,}325614727;\ 0{,}325614727;\ \cdots]}{\sqrt{4 + 10^{-6}}}$$

$$= \frac{[0{,}087132852;\ 0{,}325614727;\ 0{,}325614727;\ \cdots]}{2}$$

Elemen - elemen v" didapatkan sebagai berikut:

$$v_1'' = \frac{[0{,}087132852]}{(2)}$$

$$v_1'' = 0{,}043566421$$

$$v_2'' = \frac{[0{,}325614727]}{(2)}$$

$$v_2'' = 0{,}162807343$$

$$v_3'' = \frac{[0{,}325614727]}{(2)}$$

$$v_3'' = 0{,}162807343$$

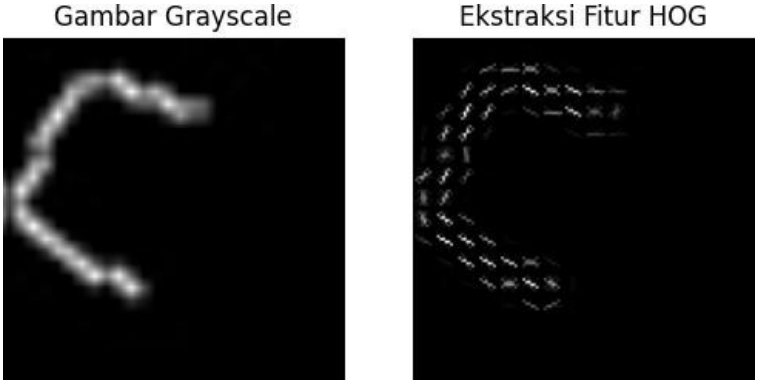
Sehingga, vektor fitur normalisasi diperoleh:

$$\frac{0{,}08713}{2}; \frac{0{,}32561}{2}; \frac{0{,}32561}{2}; \cdots = [0, 04357; 0, 16281; 0, 16281; \cdots]$$

Selanjutnya, untuk memperoleh fitur vektor secara keseluruhan gambar, vektor 1×36 per blok digabungkan menjadi keseluruhan blok, terdapat 2 posisi horizontal, dan 2 posisi vertikal dari blok, tiap blok terdiri 1×36 vektor sehingga menghasilkan vektor akhir berjumlah:

$$\text{Total vektor} = 2 \times 2 \times 36 = 144 \text{ vektor}$$

Source code untuk ekstraksi fitur HOG dapat dilihat pada Lampiran 8. Hasil akhir dari visualisasi fitur HOG karakter tulisan Korea *(Hangeul)* dapat dilihat pada Gambar 4.27



Gambar 4.27: Visualisasi fitur HOG Karakter Tulisan Korea *(Hangeul)*

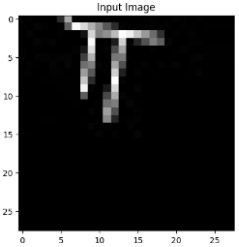
F. Pelatihan *Convolutional Neural Network* (CNN)

1. Perhitungan Manual

Setelah melakukan ekstraksi fitur HOG, langkah selanjutnya adalah melakukan perhitungan dengan menggunakan metode CNN (*Convolutional Neural Network)* dengan menggunakan gabungan hasil dari nilai fitur *Histogram Of Oriented Gradients* (HOG) dengan *flatten*. Pada tahap ini, data *training* akan dilakukan pelatihan dengan menggunakan CNN yang sudah di *grayscale* bertujuan untuk sistem dapat mengenali tulisan karakter Korea (*Hangeul*) dalam pengujian sistem yang nanti akan dilakukan. Berikut ini merupakan tahapan yang dilakukan dalam proses pelatihan *Convolutional Neural Network* (CNN) dengan menggunakan sampel kelas "Yu".

- ***Convolutional Layer***

  Convolutional layer merupakan langkah awal dalam CNN yang bertujuan untuk menghasilkan *feature map*. Berikut ini merupakan pembahasan untuk lapisan konvolusi.



Gambar 4.28: Sampel *Input Image*

0,00392157	0,01176471	0	0,00392157	0
0,00392157	0	0,01568627	0,00784314	0
0	0,01568627	0	0	0
0	0	0,01568627	0,00392157	0,01176471
0,00784314	0	0	0	0

Gambar 4.29: Salah Satu Nilai *Input* Sampel Citra dengan Ukuran 5 $\times$ 5

Gambar 4.28 dan 4.29 menunjukkan *input* yang digunakan dalam proses *convolution layer* direpresentasikan oleh nilai - nilai piksel yang berukuran 28 $\times$ 28. Oleh karena itu, penulis hanya memilih sebagian kecil piksel sebagai sampel,dengan ukuran 5$\times$5. Penelitian ini juga menggunakan kernel berukuran 3$\times$3. Pemilihan matriks kernel 3$\times$3 didasarkan pada adanya pusat kernel yang akan diterapkan pada setiap indeks matriks gambar *input* untuk proses perhitungan *convolution layer*. Proses konvolusi dengan kernel dapat dilihat pada Gambar 4.30.

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

$\times$

0, 0039	0, 0118	0
0, 0039	0	0, 0157
0	0, 0157	0

Gambar 4.30: Proses Konvolusi Dengan Kernel 3 $\times$ 3

Proses pergerakan matriks kernel 3 $\times$ 3 dimulai dari sudut kiri atas pada gambar *input*, dengan *stride* sebesar 1. Kernel kemudian bergerak secara horizontal melintas seluruh piksel pada baris pertama hingga selesai diproses

dan menjadi matriks baru. Pergeseran matriks ini dapat dilihat pada Gambar 4.31.

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

0, 004	0, 012	0	0, 004	0
0, 004	0	0, 016	0, 008	0
0	0, 016	0	0	0
0	0	0, 016	0, 004	0, 012
0, 008	0	0	0	0

Gambar 4.31: Proses Pergeseran Matriks

Pada setiap pergeseran piksel, akan dilakukan perhitungan antara piksel dari *input* gambar yang dikalikan dengan matriks kernel. Hasil dari perkalian ini akan membentuk *feature map*. Proses perhitungan menentukan ukuran matriks pada *feature map* yang sesuai pada Persamaan 2.7. dapat ditunjukkan sebagai berikut.

$$\frac{W - F + 2P}{S} + 1 = \frac{5 - 3 + 2(0)}{1} + 1$$

$$= \frac{2}{1} + 1$$

$$\frac{W - F + 2P}{S} + 1 = 3$$

Perhitungan dari pergeseran matriks Gambar 4.31 adalah sebagai berikut.

$$L_{(1,1)} = (0,004 \times 0,0039) + (0,012 \times 0,0118) + (0 \times 0)$$
$$+ (0,004 \times 0,0039) + (0 \times 0) + (0,016 \times 0,0157)$$
$$+ (0 \times 0) + (0,016 \times 0,0157) + (0 \times 0)$$

$$L_{(1,1)} = 0,000661284$$

$$L_{(1,2)} = (0,012 \times 0,0039) + (0 \times 0,0118) + (0,004 \times 0)$$
$$+ (0 \times 0,0039) + (0,016 \times 0) + (0,008 \times 0,0157)$$
$$+ (0,016 \times 0) + (0 \times 0,0157) + (0 \times 0)$$

$$L_{(1,2)} = 0,000171626$$

$$L_{(1,3)} = (0 \times 0,0039) + (0,004 \times 0,0118) + (0 \times 0)$$
$$+ (0,016 \times 0,0039) + (0,008 \times 0) + (0 \times 0,0157)$$
$$+ (0 \times 0) + (0 \times 0,0157) + (0 \times 0)$$

$$L_{(1,3)} = 0,000108574$$

$$L_{(2,1)} = (0,004 \times 0,0039) + (0 \times 0,0118) + (0,016 \times 0)$$
$$+ (0 \times 0,0039) + (0,016 \times 0) + (0 \times 0,0157)$$
$$+ (0 \times 0) + (0 \times 0,0157) + (0,016 \times 0)$$

$$L_{(2,1)} = 0,0000156$$

$L_{(2,2)}$ = (0 × 0, 0039) + (0, 016 × 0, 0118) + (0, 008 × 0)

+ (0, 016 × 0, 0039) + (0 × 0) + (0 × 0, 0157)

+ (0 × 0) + (0, 016 × 0, 0157) + (0, 004 × 0)

$L_{(2,2)}$ = 0, 0005024

$L_{(2,3)}$ = (0, 016 × 0, 0039) + (0, 008 × 0, 0118) + (0 × 0)

+ (0 × 0, 0039) + (0 × 0) + (0 × 0, 0157)

+ (0, 016 × 0) + (0, 004 × 0, 0157) + (0, 012 × 0)

$L_{(2,3)}$ = 0, 0002196

$L_{(3,1)}$ = (0 × 0, 0039) + (0, 016 × 0, 0118) + (0 × 0)

+ (0 × 0, 0039) + (0 × 0) + (0, 016 × 0, 0157)

+ (0, 008 × 0) + (0 × 0, 0157) + (0 × 0)

$L_{(3,1)}$ = 0, 00044

$L_{(3,2)}$ = (0, 016 × 0, 0039) + (0 × 0, 0118) + (0 × 0)

+ (0 × 0, 0039) + (0, 016 × 0) + (0, 004 × 0, 0157)

+ (0 × 0) + (0 × 0, 0157) + (0 × 0)

$L_{(3,2)}$ = 0, 0001252

$$L_{(3,3)} = (0 \times 0,0039) + (0 \times 0,0118) + (0 \times 0)$$
$$+ (0,016 \times 0,0039) + (0,004 \times 0) + (0,012 \times 0,0157)$$
$$+ (0 \times 0) + (0 \times 0,0157) + (0 \times 0)$$
$$L_{(3,3)} = 0,0002508$$

Hasil dari perhitungan *input* gambar pada *convolution layer* ditunjukkan pada Gambar 4.32.

0,000661284	0,000171626	0,000108574
0,0000156	0,0005024	0,0002196
0,00044	0,0001252	0,0002508

Gambar 4.32: Hasil Perhitungan *Convolution Layer*

- **Fungsi Aktivasi *Rectified Linear Unit* (ReLU)**

Setelah melewati proses *convolution layer*, selanjutnya akan dilakukan proses yang kedua yaitu fungsi aktivasi *Rectified Linear Unit* (ReLU). Fungsi aktivasi *Rectified Linear Unit* (ReLU) bertujuan untuk memperkenalkan non-linearitas ke dalam jaringan dengan menerapkan pada Persamaan 2.9. Hasil keluaran dari *convolution layer* ditunjukkan pada Gambar 4.33 .

0,000661284	0,000171626	0,000108574
0,0000156	0,0005024	0,0002196
0,00044	0,0001252	0,0002508

Gambar 4.33: Hasil Keluaran dari *Convolution Layer*

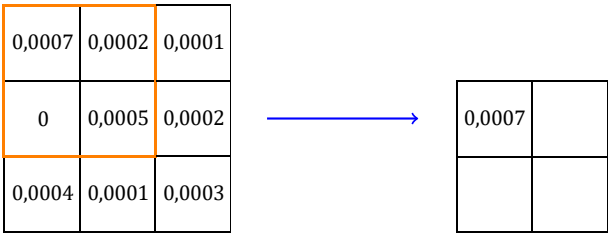
Persamaan fungsi aktivasi ReLU diterapkan sehingga setiap elemen negatif dalam matriks diubah menjadi nol, sementara elemen positif tetap dipertahankan. Hasil transformasi ReLU dapat dituliskan sebagai berikut:

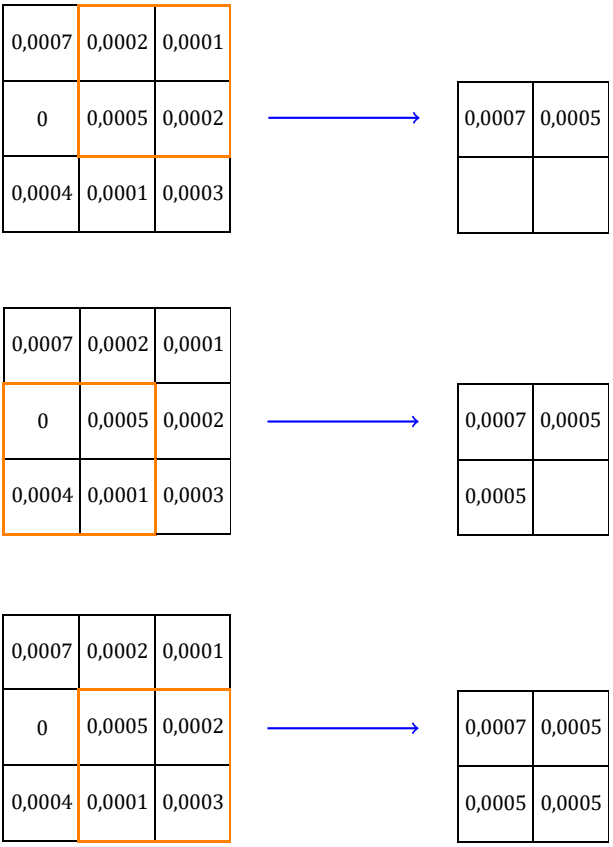
0,000661284	0,000171626	0,000108574
0,0000156	0,0005024	0,0002196
0,00044	0,0001252	0,0002508

Gambar 4.34: Hasil Keluaran dari ReLU

- **Operasi *Pooling***

Setelah melakukan proses fungsi aktivasi ReLU, langkah selanjutnya adalah operasi *pooling*. Pada tahap ini, peneliti menggunakan *max pooling*, yang bertujuan untuk mempertahankan fitur paling menonjol dalam gambar, seperti tepi dan tekstur. Selain itu, peneliti menggunakan *stride* 1 dengan matriks kernel berukuran 2×2. Proses perhitungan *max pooling* dapat dilihat pada Gambar 4.35.





Gambar 4.35: Proses *Max Pooling*

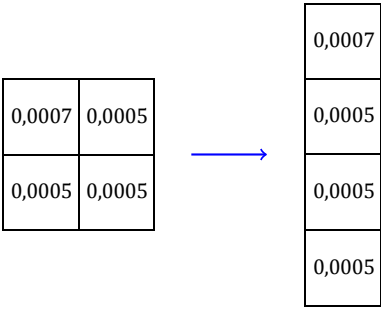
Proses *max pooling* berfungsi untuk mengambil nilai maksimal dari kernel 2×2, setelah memperoleh nilai maksimal, maka akan didapatkan berupa matriks baru. Hasil akhir dapat ditunjukkan pada Gambar 4.36.

0,0007	0,0005
0,0005	0,0005

Gambar 4.36: Hasil *Max Pooling*

- ***Flatten Layer* Nilai Fitur HOG *(Histogram of Oriented Gradients)***

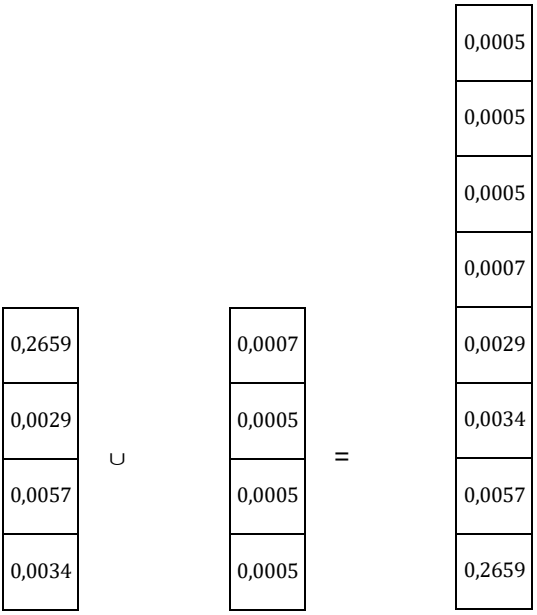
Hasil akhir dari operasi *max pooling* akan diubah menjadi 1 vektor, dimana hasil dari *flatten layer* digunakan sebagai *input* pada proses selanjutnya yaitu *fully connected layer*. Hasil dari *flatten layer* dapat ditunjukkan pada Gambar 4.37.



Gambar 4.37: Proses *Flatten Layer*

Setelah proses *flatten*, hasil dari *flatten* akan digabungkan dengan hasil dari nilai fitur HOG *(Histogram of Oriented Gradients)* yang bertujuan untuk model dapat mengenali pola yang lebih kompleks dan meningkatkan akurasi klasifikasi gambar. Selanjutnya, *fully connected layer* akan memproses hasil dari gabungan *flatten* dan hasil nilai fitur HOG *(Histogram of Oriented Gradients)* ini untuk membuat keputusan akhir dalam klasifikasi. Hasil gabungan

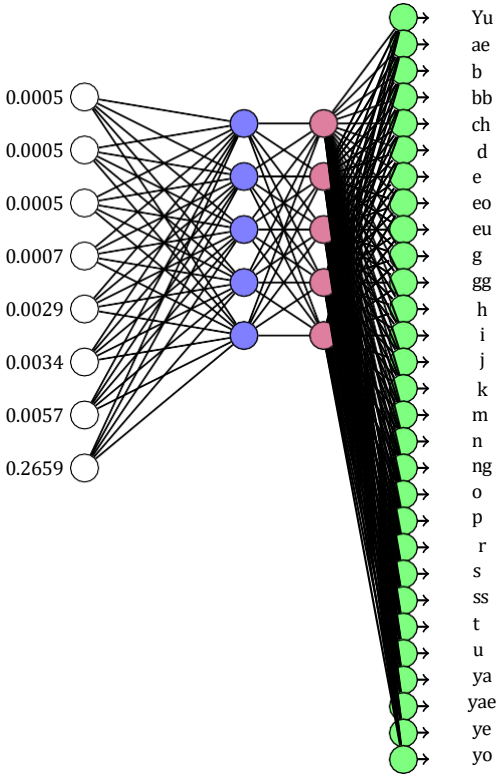
dari *flatten layer* dengan hasil nilai fitur HOG *(Histogram of Oriented Gradients)* dapat ditunjukkan pada Gambar 4.38.



Gambar 4.38: Proses Penggabungan HOG dan *Flatten* CNN

- ***Fully Connected Layer***

  Setelah memperoleh hasil dari gabungan *flatten layer* dengan nilai fitur HOG, langkah selanjutnya adalah menerapkan ke proses *fully connected layer*. Dalam hal ini, *fully connected layer* berfungsi sebagai *hidden layer*, yang berperan dalam menentukan ke *output* yang dimana suatu citra akan diklasifikasikan. Representasi dari *fully connected layer* ditunjukkan pada Gambar 4.39



Gambar 4.39: *Fully Connected Layer with Two Hidden Layers*

Berikut ini merupakan perhitungan manual dari *fully connected layer* dari Persamaan 2.8 dengan Dense 128 dan bobot yang berbeda-beda.

$$J_{(m)} = \sum_{i=0}^{in} I_i \times (w_i)^{(m)}$$

$$J_{(1)} = (0,0005 \times 0,1) + (0,0005 \times 0,1) + (0,0005 \times 0,1) + (0,0007 \times 0,1) + (0,0029 \times 0,1) + (0,0034 \times 0,1) + (0,0057 \times 0,1) + (0,2659 \times 0,1)$$

$$J_{(1)} = 0,02801$$

$$J_{(2)} = (0,0005 \times 0,2) + (0,0005 \times 0,2) + (0,0005 \times 0,2) + (0,0007 \times 0,2) + (0,0029 \times 0,2) + (0,0034 \times 0,2) + (0,0057 \times 0,2) + (0,2659 \times 0,2)$$

$$J_{(2)} = 0,05602$$

$$J_{(3)} = (0,0005 \times 0,3) + (0,0005 \times 0,3) + (0,0005 \times 0,3) + (0,0007 \times 0,3) + (0,0029 \times 0,3) + (0,0034 \times 0,3) + (0,0057 \times 0,3) + (0,2659 \times 0,3)$$

$$J_{(3)} = 0,08403$$

$$J_{(4)} = (0,0005 \times 0,4) + (0,0005 \times 0,4) + (0,0005 \times 0,4)$$

$$+ (0,0007 \times 0,4) + (0,0029 \times 0,4) + (0,0034 \times 0,4)$$

$$+ (0,0057 \times 0,4) + (0,2659 \times 0,4)$$

$$J_{(4)} = 0,11204$$

Hidden layer pertama di ilustrasikan dengan J_1,J_2,J_3, dan J_4 dari *input* I_1 sampai I_8 dikalikan dengan niai bobot yang berbeda - beda sehingga diperoleh nilai J_1=0,02801, J_2=0,05602, J_3=0,08403, dan J_4=0,11204.

Setelah melakukan perhitungan Dense 128 (*hidden layer) pertama*, selanjutnya akan dilakukan dengan *dropout layer* yang berfungsi untuk mengurangi kompleksitas model dengan menonaktifkan sejumlah *neuron* secara acak, sehingga mencegah *overfitting* dan meningkatkan generalisasi.

Setelah *dropout layer*, model akan melanjutkan ke lapisan Dense 64 *(hidden layer* kedua), yang berfungsi untuk mengekstraksi fitur lebih lanjut dan memperkuat representasi data sebelum akhirnya diteruskan ke lapisan *output* untuk klasifikasi, dimana pada nilai J_1,J_2,J_3, dan J_4 akan dikalikan dengan nilai bobot yang berbeda agar menghasilkan nilai h_1, h_2, h_3, dan h_4. Berikut ini merupakan perhitungan menentukan nilai h_1, h_2, h_3, dan h_4.

$h_{(1)}$ = (0, 02801 × 0, 5) + (0, 05602 × 0, 5) + (0, 08403 × 0, 5)

+ (0, 11204 × 0, 5)

$h_{(1)}$ = 0, 14005

$h_{(2)}$ = (0, 02801 × 0, 6) + (0, 05602 × 0, 6) + (0, 08403 × 0, 6)

+ (0, 11204 × 0, 6)

$h_{(2)}$ = 0, 16806

$h_{(3)}$ = (0, 02801 × 0, 7) + (0, 05602 × 0, 7) + (0, 08403 × 0, 7)

+ (0, 11204 × 0, 7)

$h_{(3)}$ = 0, 19607

$h_{(4)}$ = (0, 02801 × 0, 8) + (0, 05602 × 0, 8) + (0, 08403 × 0, 8)

+ (0, 11204 × 0, 8)

$h_{(4)}$ = 0, 22408

Setiap nilai h_1, h_2, h_3, dan h_4 yang diperoleh akan dikalikan dengan bobot yang berbeda untuk menghasilkan nilai O_1 sampai O_{29} sebagai *hidden layer* selanjutnya. Berikut ini merupakan perhitungan manual menentukan nilai O_1 sampai O_{29}, akan tetapi dalam contoh perhitungan manual dibawah ini hanya menggunakan 5 sampel kelas saja yang akan digunakan untuk prediksi kelas, yaitu kelas Yu, ae, b, bb,

dan ch.

$$O_{(1)} = (0, 14005 \times 0, 1) + (0, 16806 \times 0, 1) + (0, 19607 \times 0, 1) + (0, 22408 \times 0, 1)$$

$$O_{(1)} = 0, 072826$$

$$O_{(2)} = (0, 14005 \times 0, 2) + (0, 16806 \times 0, 2) + (0, 19607 \times 0, 2) + (0, 22408 \times 0, 2)$$

$$O_{(2)} = 0, 145652$$

$$O_{(3)} = (0, 14005 \times 0, 3) + (0, 16806 \times 0, 3) + (0, 19607 \times 0, 3) + (0, 22408 \times 0, 3)$$

$$O_{(3)} = 0, 218478$$

$$O_{(4)} = (0, 14005 \times 0, 4) + (0, 16806 \times 0, 4) + (0, 19607 \times 0, 4) + (0, 22408 \times 0, 4)$$

$$O_{(4)} = 0, 291304$$

$$O_{(5)} = (0, 14005 \times 0, 5) + (0, 16806 \times 0, 5) + (0, 19607 \times 0, 5) + (0, 22408 \times 0, 5)$$

$$O_{(5)} = 0, 36413$$

- **Loss Layer**

  Pada tahap akhir dari *Convolutional Neural Network* (CNN) adalah tahap *loss layer* yang digunakan untuk menghitung perbedaan antara *output* jaringan dan nilai prediksi yang sebenarnya. Pada perhitungan manual ini, peneliti menggunakan fungsi aktivasi *softmax* yang berfungsi untuk menghitung nilai probabilitas dalam pengklasifikasian kelas. Berikut ini merupakan perhitungan dengan menggunakan Persamaan 2.11.

$$\sigma(z)_1 = \frac{e^{0,072826}}{e^{0,072826} + e^{0,145652} + e^{0,218478} + e^{0,291304} + e^{0,36413}}$$

$$\sigma(z)_1 = 0,171978217$$

$$\sigma(z)_2 = \frac{e^{0,145652}}{e^{0,072826} + e^{0,145652} + e^{0,218478} + e^{0,291304} + e^{0,36413}}$$

$$\sigma(z)_2 = 0,184970032$$

$$\sigma(z)_3 = \frac{e^{0,218478}}{e^{0,072826} + e^{0,145652} + e^{0,218478} + e^{0,291304} + e^{0,36413}}$$

$$\sigma(z)_3 = 0,198943293$$

$$\sigma(z)_4 = \frac{e^{0,291304}}{e^{0,072826} + e^{0,145652} + e^{0,218478} + e^{0,291304} + e^{0,36413}}$$

$$\sigma(z)_4 = 0,21397214$$

$$\sigma(z)_5 = \frac{e^{0,36413}}{e^{0,072826} + e^{0,145652} + e^{0,218478} + e^{0,291304} + e^{0,36413}}$$

$$\sigma(z)_5 = 0,230136318$$

$$\sigma(z)_1 + \sigma(z)_2 + \sigma(z)_3 + \sigma(z)_4 + \sigma(z)_5 = 1$$

$$0,172 + 0,185 + 0,199 + 0,21 + 0,23 = 1$$

Berdasarkan perhitungan nilai $\sigma(y)$ diatas, didapatkan bahwa nilai probabilitas yang lebih besar adalah 0,23, sehingga *input* citra yang dimasukan di prediksi adalah kelas "ch" bukan kelas "yu" (*input*).

2. Implementasi

– Perancangan Model

Pada tahap ini, penulis membuat arsitektur model CNN dengan cara melatih model *deep lerning* dengan semua data yang tersedia dan melakukan *hyperparameter turning* untuk mendapatkan model dengan performa yang optimal. Berikut ini merupakan *source codenya*:

```
# Membangun model CNN + HOG
def create_model(hog_feature_size):
input_image = Input(shape=(28, 28, 3))
x = Conv2D(32, (3, 3), activation='relu')(input_image)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)

input_hog = Input(shape=(hog_feature_size,))
combined_features = Concatenate()([x, input_hog])
z = Dense(128, activation='relu')(combined_features)
z = Dropout(0.3)(z)  # Mengurangi overfitting
z = Dense(64, activation='relu')(z)
output = Dense(num_classes, activation='softmax')(z)

model = Model(inputs=[input_image, input_hog], outputs=output)
return model
```

Gambar 4.40: *Source Code* Modelling CNN

Gambar 4.41 menunjukkan bahwa struktur layer model yang akan digunakan adalah *convolution*, *pooling*, *flatten*, dan Dense. Pembangunan model CNN dengan menggabungkan fitur dari gambar dan *Histogram of Oriented Gradients* (HOG) yang digunakan untuk pengklasifikasian. Model ini memanfaatkan dua inputan, antara lainnya adalah gambar berukuran 28 $\times$ 28 $\times$ 3 dan fitur HOG, yang digabungkan untuk membentuk representasi fitur yang lebih informatif.

Setelah proses *convolution layer* dan *pooling* pada gambar, fitur yang diperoleh digabungkan dengan *input* HOG dan diproses lebih lanjut menggunakan lapisan Dense untuk menghasilkan *output* klasifikasi dengan fungsi aktivasi *softmax*. Berikut ini merupakan hasil *output* dari arsitektur CNN.

Layer (type)	Output Shape	Param #	Connected to
input_layer_32 (InputLayer)	(None, 28, 28, 3)	0	-
conv2d_32 (Conv2D)	(None, 26, 26, 32)	896	input_layer_32[0][0]
max_pooling2d_32 (MaxPooling2D)	(None, 13, 13, 32)	0	conv2d_32[0][0]
conv2d_33 (Conv2D)	(None, 11, 11, 64)	18,496	max_pooling2d_32[0][0]
max_pooling2d_33 (MaxPooling2D)	(None, 5, 5, 64)	0	conv2d_33[0][0]
flatten_16 (Flatten)	(None, 1600)	0	max_pooling2d_33[0][0]
input_layer_33 (InputLayer)	(None, 144)	0	-
concatenate_16 (Concatenate)	(None, 1744)	0	flatten_16[0][0], input_layer_33[0][0]
dense_48 (Dense)	(None, 128)	223,360	concatenate_16[0][0]
dropout_16 (Dropout)	(None, 128)	0	dense_48[0][0]
dense_49 (Dense)	(None, 64)	8,256	dropout_16[0][0]
dense_50 (Dense)	(None, 29)	1,885	dense_49[0][0]

Total params: 252,893 (987.86 KB)
Trainable params: 252,893 (987.86 KB)
Non-trainable params: 0 (0.00 B)

Gambar 4.41: *Output Modelling CNN*

Berdasarkan Gambar 4.41 menunjukkann bahwa model yang dirancang untuk klasifikasi gambar dengan tambahan fitur yang non-visual. Model ini menerima *input* gambar berukuran $28 \times 28 \times 3$ melalui dengan lapisan *input layer*. Proses ekstraksi fitur dimulai dengan *Conv2D* pertama yang menggunakan 32 filter berukuran 3×3, menghasilkan keluaran $26 \times 26 \times 32$, yang kemudian diproses oleh *MaxPoolinng2D* untuk mengurangi dimensi menjadi $13 \times 13 \times 32$. Lapisan konvolusi kedua *(Conv2D)* memiliki 64 filter dengan ukuran 3×3, menghasilkan *output* $11 \times 11 \times 64$, yang kembali diperkecil oleh *MaxPooling2D* menjadi $5 \times 5 \times 64$. Selanjutnya, fitur - fitur yang diperoleh diubah menjadi 1 vektor dengan menggunakan *flatten*, dengan

mengubah tensor menjadi vektor berdimensi 1600.

– ***Hyperparameter Tuning***

Proses ini melibatkan pengujian berbagai kombinasi *hyperparameter* bertujuan untuk menentukan performa yang optimal dalam pelatihan model. Beberapa parameter yang diuji meliputi jumlah *epoch* (15 dan 25) digunakan untuk menentukan jumlah iterasi pelatihan, *Batch Size* (32 dan 64) digunakan untuk mengontrol jumlah sampel yang diproses sebelum pembaruan model. Selain itu, dan Optimizer Adam dan RMSprop digunakan untuk menyesuaikan bobot jaringan guna meminimalkan loss, dan *Learning Rate* (0,001 dan 0,0001) yang digunakan untuk mengatur kecepatan pembelajaran model. Pemilihan kedua Optimizer ini didasarkan pada penelitian yang dilakukan oleh (Julianto dkk., 2022) mendapatkan hasil yang sangat baik dengan nilai akurasi sebesar 97,56% untuk RMSprop dan 96,87% untuk Adam.

Pengujian ini dengan menggunakan pembagian data sebesar 70:30, 80:20, dan 90:10 yang bertujuan untuk mengevaluasi performa model secara optimal. Berikut ini merupakan *source code* dari *parameter tuning* dengan rasio masing - masing.

```
# Hyperparameter tuning
optimizers = [(Adam, "Adam"), (RMSprop, "RMSprop")]
learning_rates = [0.001, 0.0001]  # Dikurangi untuk efisiensi
epochs_list = [15, 25]
batch_sizes = [32, 64]

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss',
patience=3, restore_best_weights=True)

# Menyimpan hasil evaluasi
results = []

for optimizer_class, optimizer_name in optimizers:
    for lr in learning_rates:
        for epochs in epochs_list:
            for batch_size in batch_sizes:
                model = create_model()
                optimizer = optimizer_class(learning_rate=lr)
                model.compile(optimizer=optimizer,
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Gambar 4.42: *Source Code Parameer Tuning*

Pada Gambar 4.42 menunjukkan model di uji coba dengan menggunakan *optimizer*, *epoch*, dan *learning rate* yang berbeda per kombinasi nya bertujuan untuk memilih kombinasi parameter yaang mempunyai performa terbaik. Berikut ini merupakan hasil *parameter tuning* terbaik yang diperoleh dari masing - masing pembagian data, yaitu 70:30, 80:20, dan 90:10.

* **Rasio 70:30**

Tabel 4.5: Hasil Parameter Tuning Rasio 70:30

Optimizer	Learning Rate	Epochs	Batch Size	Test Accuracy
Adam	0,001	15	32	0,8089080453
Adam	0,001	15	64	0,7948275805

Tabel 4.6: Lanjutan Hasil Parameter Tuning Rasio 70:30

Optimizer	Learning Rate	Epochs	Batch Size	Test Accuracy
Adam	0,001	25	32	0,8051724434
Adam	0,001	25	64	0,8149425387
Adam	0,0001	15	32	0,6534482837
Adam	0,0001	15	64	0,5910919309
Adam	0,0001	25	32	0,711206913
Adam	0,0001	25	64	0,6775861979
RMSprop	0,001	15	32	0,8169540167
RMSprop	0,001	15	64	0,8025861979
RMSprop	0,001	25	32	0,8431034684
RMSprop	0,001	25	64	0,8221264482
RMSprop	0,0001	15	32	0,5936781764
RMSprop	0,0001	15	64	0,5152298808
RMSprop	0,0001	25	32	0,6925287247
RMSprop	0,0001	25	64	0,6324712634

Pada Tabel 4.5 dan 4.6 menunjukkan bahwa hasil akurasi yang diperoleh dari proses parameter tuning yang dilakukan oleh penulis berdasarkan kombinasi parameter yang sudah ditentukan.

```
Best Hyperparameters: Optimizer         RMSprop
Learning Rate       0.001
Epochs                 25
Batch Size             32
Test Accuracy    0.843103
Name: 10, dtype: object
```

Gambar 4.43: Hasil Parameter Tuning terbaik dengan Rasio 70:30

Pada Gambar 4.43 menunjukkan bahwa hasil parameter yang adalah menggunakan *optimizer*: RMSprop, *learning rate*: 0,001, *Batch Size*: 32,dan mendapatkan akurasi sebesar 84,3103% dalam epoch ke-25.

* **Rasio 80:20**

Tabel 4.7: Hasil Parameter Tuning Rasio 80:20

Optimizer	Learning Rate	Epochs	Batch Size	Test Accuracy
Adam	0,001	15	32	0,8254310489
Adam	0,001	15	64	0,8159482479
Adam	0,001	25	32	0,8241379261
Adam	0,001	25	64	0,8189654946
Adam	0,0001	15	32	0,6836206913
Adam	0,0001	15	64	0,6353448033
Adam	0,0001	25	32	0,7318965793
Adam	0,0001	25	64	0,6969827414
RMSprop	0,001	15	32	0,7961207032
RMSprop	0,001	15	64	0,8064655066
RMSprop	0,001	25	32	0,8245689869
RMSprop	0,001	25	64	0,7775862217
RMSprop	0,0001	15	32	0,6327586174
RMSprop	0,0001	15	64	0.5599138141
RMSprop	0,0001	25	32	0,6900861859
RMSprop	0,0001	25	64	0,6603448391

Pada Tabel 4.7 menunjukkan bahwa hasil akurasi yang diperoleh dari proses parameter tuning yang dilakukan oleh penulis berdasarkan kombinasi parameter yang sudah ditentukan.

```
Best Hyperparameters: Optimizer          Adam
Learning Rate       0.001
Epochs                 15
Batch Size             32
Test Accuracy    0.825431
Name: 0, dtype: object
```

Gambar 4.44: Hasil Parameter Tuning terbaik dengan Rasio 80:20

Pada Gambar 4.44 menunjukkan bahwa hasil parameter yang adalah menggunakan *optimizer*: Adam, *learning rate*: 0,001, *Batch Size*: 32,dan mendapatkan akurasi sebesar 82,5431% dalam

epoch ke-15.

* **Rasio 90:10**

Tabel 4.8: Hasil Parameter Tuning Rasio 90:10

Optimizer	Learning Rate	Epochs	Batch Size	Test Accuracy
Adam	0,001	15	32	0,8474137783
Adam	0,001	15	64	0,8396551609
Adam	0,001	25	32	0,8327586055
Adam	0,001	25	64	0,836206913
Adam	0,0001	15	32	0,6844827533
Adam	0,0001	15	64	0,6370689869
Adam	0,0001	25	32	0,7465517521
Adam	0,0001	25	64	0,711206913
RMSprop	0,001	15	32	0,8620689511
RMSprop	0,001	15	64	0,7801724076
RMSprop	0,001	25	32	0,8301724195
RMSprop	0,001	25	64	0,8181034327
RMSprop	0,0001	15	32	0,6370689869
RMSprop	0,0001	15	64	0,5965517163
RMSprop	0,0001	25	32	0,7232758403
RMSprop	0,0001	25	64	0,6991379261

Pada Tabel 4.8 menunjukkan bahwa hasil akurasi yang diperoleh dari proses parameter tuning yang dilakukan oleh penulis berdasarkan kombinasi parameter yang sudah ditentukan. Berikut ini merupakan hasil parameter tuning terbaik dengan rasio 90:10.

```
Best Hyperparameters: Optimizer        RMSprop
Learning Rate       0.001
Epochs                 15
Batch Size             32
Test Accuracy    0.862069
Name: 8, dtype: object
```

Gambar 4.45: Hasil Parameter Tuning terbaik dengan Rasio 90:10

Pada Gambar 4.45 menunjukkan bahwa hasil parameter terbaik yang digunakan adalah *optimizer*: RMSprop, *learning rate*: 0,001, *Batch Size*: 32,dan mendapatkan akurasi sebesar 86,2069% dalam epoch ke-15.

- ***Training Model***

Pada tahap ini, penulis akan membuat *training* model dengan memanfaatkan parameter terbaik yang sudah diuji pada tahap sebelumnya. Selama proses pelatihan, *verbose* yang digunakan adalah 1, karena dalam perkembangan model dapat ditampilkan secara ringkas dan informatif pada setiap *epoch*. Selain itu, *validation data* digunakan untuk membantu mengukur kinerja model secara berkala, dan *callback* diterapkan. *Early stopping* digunakan untuk menghentikan pelatihan secara otomatis jika tidak ada peningkatan yang signifikan, sehingga dapat mencegah terjadinya *overfitting* dan meningkatkan efisiensi pelatihan. Berikut ini merupakan *source code* dalam pelatihan model.

```
print(f"Training with {optimizer_name}, LR: {lr},
Epochs: {epochs}, Batch: {batch_size}")
                history = model.fit(
                    [x_train, x_train_hog], y_train,
                    epochs=epochs, batch_size=batch_size,
                    validation_data=([x_test, x_test_hog],
                    y_test),
                    callbacks=[early_stopping], verbose=1
                )
```

Gambar 4.46: *Source Code Training Model*

Pada Gambar *Source Code* 4.46 menunjukkan bahwa model akan di *training* dari rasio 70:30, 80:20, 90:10, serta parameter yang digunakan adalah RMSprop dan Adam. Berikut ini merupakan Hasil dari *training* model dari masing - masing rasio 70:30, 80:20, 90:10 dengan parameter yang terbaik.

* **Rasio 70:30**

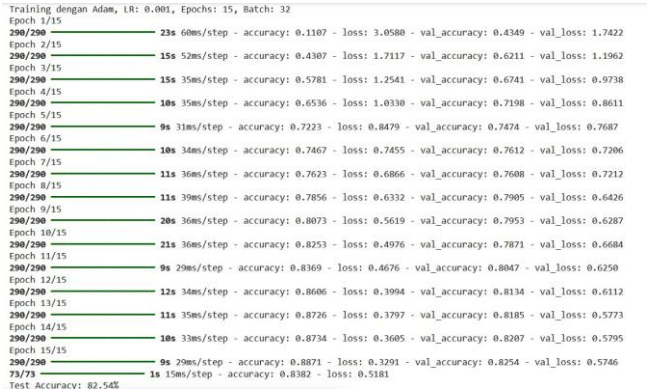
```
Epoch 2/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 20s 40ms/step - accuracy: 0.3750 - loss: 1.9453 - val_accuracy: 0.5132 - val_loss: 1.5041
Epoch 3/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 21s 42ms/step - accuracy: 0.5079 - loss: 1.4801 - val_accuracy: 0.6348 - val_loss: 1.1358
Epoch 4/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 20s 41ms/step - accuracy: 0.5877 - loss: 1.2211 - val_accuracy: 0.6411 - val_loss: 1.0608
Epoch 5/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - accuracy: 0.6482 - loss: 1.0407 - val_accuracy: 0.6917 - val_loss: 0.9560
Epoch 6/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 12s 41ms/step - accuracy: 0.6929 - loss: 0.9154 - val_accuracy: 0.7362 - val_loss: 0.8006
Epoch 7/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 21s 43ms/step - accuracy: 0.7322 - loss: 0.7976 - val_accuracy: 0.7658 - val_loss: 0.7375
Epoch 8/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 20s 41ms/step - accuracy: 0.7536 - loss: 0.7151 - val_accuracy: 0.7810 - val_loss: 0.7076
Epoch 9/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 20s 38ms/step - accuracy: 0.7860 - loss: 0.6491 - val_accuracy: 0.7796 - val_loss: 0.6752
Epoch 10/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - accuracy: 0.8137 - loss: 0.5478 - val_accuracy: 0.7954 - val_loss: 0.6529
Epoch 11/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 12s 41ms/step - accuracy: 0.8205 - loss: 0.5239 - val_accuracy: 0.7353 - val_loss: 0.8150
Epoch 12/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 10s 35ms/step - accuracy: 0.8492 - loss: 0.4614 - val_accuracy: 0.7931 - val_loss: 0.6383
Epoch 13/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 11s 37ms/step - accuracy: 0.8571 - loss: 0.4086 - val_accuracy: 0.8201 - val_loss: 0.6032
Epoch 14/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 11s 40ms/step - accuracy: 0.8662 - loss: 0.3889 - val_accuracy: 0.8181 - val_loss: 0.6145
Epoch 15/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - accuracy: 0.8813 - loss: 0.3474 - val_accuracy: 0.8132 - val_loss: 0.5849
Epoch 16/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - accuracy: 0.8815 - loss: 0.3405 - val_accuracy: 0.8213 - val_loss: 0.6139
Epoch 17/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - accuracy: 0.8941 - loss: 0.3108 - val_accuracy: 0.8431 - val_loss: 0.5641
Epoch 18/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 21s 42ms/step - accuracy: 0.9044 - loss: 0.2814 - val_accuracy: 0.8250 - val_loss: 0.6525
Epoch 19/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - accuracy: 0.9121 - loss: 0.2580 - val_accuracy: 0.8170 - val_loss: 0.6784
Epoch 20/25
254/254 ━━━━━━━━━━━━━━━━━━━━ 11s 41ms/step - accuracy: 0.9146 - loss: 0.2404 - val_accuracy: 0.8460 - val_loss: 0.5924
109/109 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8612 - loss: 0.4743
Test Accuracy: 84.31%
109/109 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
```

Gambar 4.47: Hasil Training Model dari Parameter Tuning Terbaik dengan Rasio 70:30

Pada Gambar 4.47, hasil *training model* dengan skema pembagian data 70:30 menunjukkan bahwa adanya peningkatan akurasi dari 9,94% menjadi 86,12% dalam 25 epoch, disertai adanya penurunan nilai loss dari 3,1467 menjadi 0,2404. Selain itu, akurasi validasi

meningkat dari 37,90% menjadi 84,60%. Pada tahap pengujian, model yang dihasilkan dengan pembagian data 70:30 mencapai akurasi sebesar 84,31%.

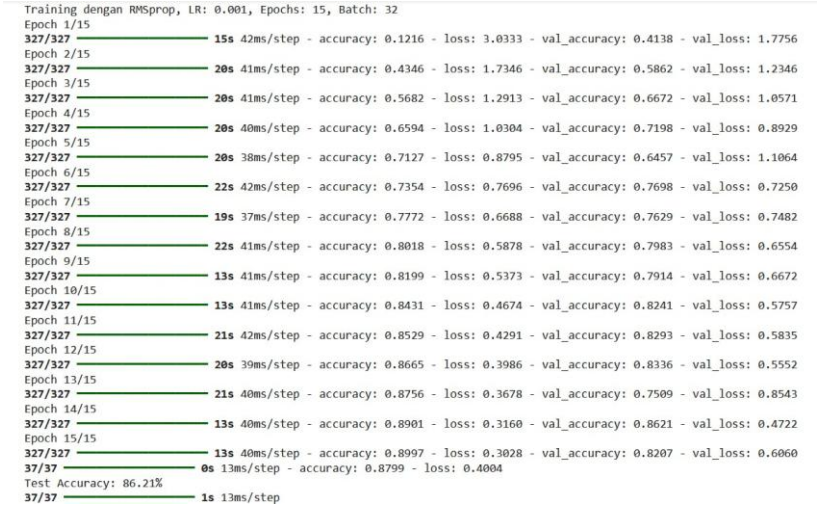
* **Rasio 80:20**



Gambar 4.48: Hasil Training Model dari Parameter Tuning Terbaik dengan Rasio 80:20

Pada Gambar 4.48, hasil *training model* dengan skema pembagian data 80:20 menunjukkan bahwa terjadi peningkatan akurasi dari 11,07% menjadi 88,71% dalam 15 epoch, disertai dengan penurunan nilai loss dari 3,0580 menjadi 0,3291. Selain itu, akurasi validasi meningkat dari 43,49% menjadi 82,54%. Pada tahap pengujian model yang dihasilkan dengann pembagian data 80:20.

* **Rasio 90:10**



Gambar 4.49: Hasil Training Model dari Parameter Tuning Terbaik dengan Rasio 90:10

Pada Gambar 4.49, hasil *training model* dengan skema pembagian data 90:10 menunjukkan bahwa terjadi peningkatan akurasi dari 12,16% menjadi 89,97% dalam 15 epoch, disertai dengan penurunan nilai loss dari 3,0333 menjadi 0,3028. Selain itu, akurasi validasi meningkat dari 41,38% menjadi 82,07%. Pada tahap pengujian, model yang dihasilkan dengan pembagian data 90:10 mencapai akurasi sebesar 86,21%.

* **Evaluasi Model**

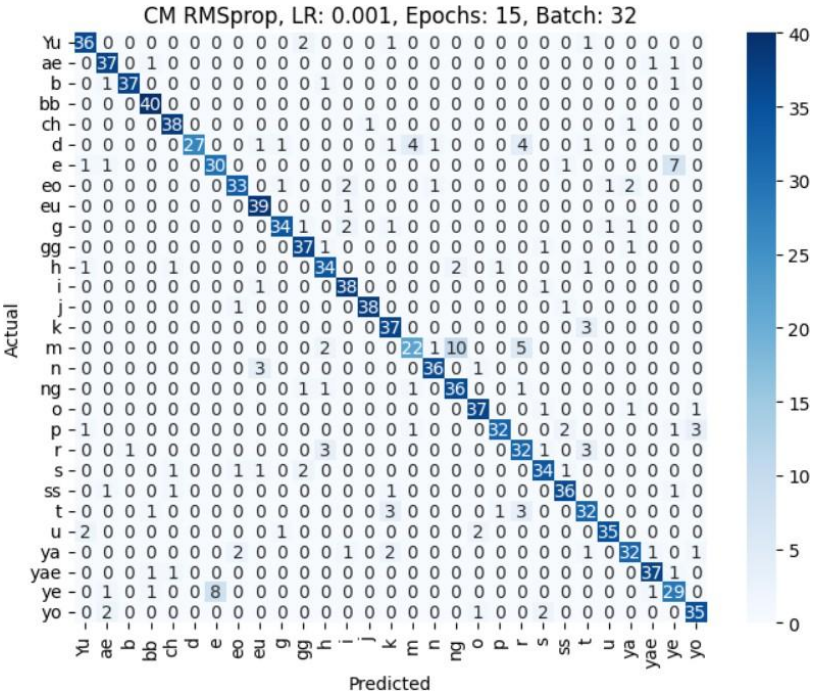
Pada tahap terakhir, peneliti akan melakukan pengujian terhadap tingkat *accuracy*, *recall*,

precission, dan *F1-score*. Pengujian ini dilakukan dengan menggunakan *confussion matrix* berdasarkan Persamaan 2.18, 2.19, 2.20, dan 2.21. Berikut ini *Source code* evaluasi model CNN yang ditunjukkan pada Gambar 4.50.

```
 test_loss, test_accuracy = model.evaluate
([x_test, x_test_hog], y_test)
print(f"Test Accuracy: {test_accuracy
* 100:.2f}%")

results.append([optimizer_name, lr, epochs,
batch_size, test_accuracy])

y_pred = model.predict([x_test, x_test_hog])
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)
 print("Classification Report:")
print(classification_report(y_true_classes,
y_pred_classes, target_names=
label_encoder.classes_))

precision, recall, f1, _ =
precision_recall_fscore_support(y_true_classes,
y_pred_classes, average='weighted')
print(f"Precision: {precision:.4f}, Recall:
{recall:.4f}, F1-score: {f1:.4f}")
conf_matrix = confusion_matrix(y_true_classes,
y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues', xticklabels=label_encoder.
classes_, yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'CM {optimizer_name}, LR: {lr},
Epochs: {epochs}, Batch: {batch_size}')
plt.show()
```

Gambar 4.50: *Source Code* Evaluasi Model

Dalam skenario pada pengujian ini, sampel parameter yang ditampilkan adalah rasio 90% :10%, yang sudah diaugmentasi sebesar 10440 data latih dan 1160 data uji. Pengujian ini dilakukan dengan menggunakan parameter terbaik yaitu dengan *batch size* sebesar 32, *learning rate* sebesar 0,001, dan jumlah *epoch* sebesar 15 dengan optimizer RMSprop. Berikut ini merupakan hasil pengujian model dalam bentuk *confusion matrix*.



Gambar 4.51: *Confusion Matrix dengan Rasio 90:10*

Pada Gambar 4.51 menampilkan hasil *confusion matrix* yang menunjukkan bahwa adanya hubungan antara *true label* dan *predicted label.* Berdasarkan *confusion matrix* tersebut, perhitungan manual nilai *accuracy*, *recall*, *precision*, dan *F1-Score* dengan menggunakan sampel "YU" sebagai berikut:

- *Accuracy score*

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$= \frac{36 + 1115}{36 + 1115 + 5 + 4} \times 100\%$$

$$Accuracy = 99,2241379\%$$

- *Recall*

$$Recall = \frac{TP}{TP + FN} \times 100\%$$

$$= \frac{36}{36 + 4} \times 100\%$$

$$= \frac{36}{40} \times 100\%$$

$$Recall = 90\%$$

· *Precision*

$$Precision = \frac{TP}{TP + FP} \times 100\%$$

$$= \frac{36}{36 + 5} \times 100\%$$

$$= \frac{36}{41} \times 100\%$$

$$Precision = 87,804878\%$$

· *F1-Score* .

$$F1\text{-}Score = 2 \times (\frac{Recall \times Precision}{Recall + Precision} \times 100\%$$

$$= 2 \times (\frac{90\% \times 87,804878\%}{90\% + 87,804878\%})$$

$$F1\text{-}Score = 88,7210\%$$

Setelah melakukan perhitungan manual dengan sampel satu kelas "Yu", berikut ini merupakan tabel hasil dari evaluasi model dengan pembagian 90%:10% dengan semua karakter.

Tabel 4.9: Hasil Evaluasi Model dengan Pembagian 90:10

Class	Precision	Recall	F1-Score
Yu	0,88	0,90	0,89
ae	0,86	0,93	0,89
b	0,97	0,93	0,95
bb	0,91	1,00	0,95
ch	0,90	0,95	0,93
d	1,00	0,68	0,81
e	0,79	0,75	0,77
eo	0,89	0,82	0,86
eu	0,87	0,97	0,92
g	0,92	0,85	0,88
gg	0,86	0,93	0,89
h	0,81	0,85	0,83
i	0,86	0,95	0,90
j	0,97	0,95	0,96
k	0,80	0,93	0,86
m	0,79	0,55	0,65
n	0,92	0,90	0,91
ng	0,75	0,90	0,82
o	0,90	0,93	0,91
p	0,94	0,80	0,86
r	0,71	0,80	0,75
s	0,85	0,85	0,85
ss	0,88	0,90	0,89
t	0,76	0,80	0,78
u	0,95	0,88	0,91
ya	0,84	0,80	0,82
yae	0,93	0,93	0,93
ye	0,71	0,72	0,72
yo	0,88	0,88	0,88
Accuracy		0,86	
Macro Avg	0,87	0,86	0,86
Weighted Avg	0,87	0,86	0,86

Berdasarkan hasil evaluasi model pada Gambar 4.51 dan Tabel 4.9 menunjukkan bahwa nilai akurasi keseluruhan sebesar 86%, *precission* sebesar 87% , *recall* sebesar 86% dan *f1-score*

sebesar 86% yang berarti pembagian data dengan rasio 90%:10% dapat mengidentifikasi sebagian besar karakter tulisan Korea *(Hangeul)* dengan baik. Akan tetapi, terdapat beberapa kelas degan nilai *recall* dan *precision* yang rendah dapat mengakibatkan dalam mengidentifikasi kelas yang sering tertukar.

Salah satu kelas dengan *recall* dan *precision* terendah adalah kelas "m" dengan *precision* 79%, *recall* sebesar 55% dan *f1-score* sebesar 65%. Nilai ini menunjukkan bahwa banyak sampel dari kelas "m" salah dikasifikasikan sebagai kelas lain. Selain itu, kelas "d" mempunyai nilai *precision* tertinggi sebesar 100%, tetapi nilai *recall* yang rendah 68%, yang berarti model selalu benar dalam memprediksi kelas "d", tetapi terdapat banyak sampel dari kelas tersebut yang tidak terdeteksi dengan baik.

Sementara itu, kelas dengan nilai *recall* tertinggi adalah kelas "bb", dengan nilai *recall* mencapai 100%, berarti model dapat mengenali semua sampel dalam kelas ini dengan benar tanpa adanya kesalahan. Selain itu, kelas "yae" mempunyai nilai *precision* tertinggi sebesar 93% yang menunjukkan bahwa model sangat jarang salah memprediksi kelas lain sebagai kelas "yae" dan mempunyai tingkat akurasi tinggi dalam mengenali kelas tersebut. Hal ini mengindikasikan bahwa model lebih baik dalam mengidentifikasi

beberapa kelas tertentu dibandingkan kelas yang lainnya.

Berikut ini merupakan hasil evaluasi model secara keseluruhan dengan rasio 70:30, 80:20, dan 90:10.

Tabel 4.10: Hasil Seluruh Pengujian

Split Data		Batch Size	Optimizer	Learning Rate	Epoch	Akurasi (%)	Precision (%)	Recall (%)	F1 Score (%)
Training	Testing								
70	30	32	Adam	0,001	15	81	8,05	80,89	80,80
					25	81	81,14	80,52	80,41
				0,0001	15	65	65,71	65,34	64,86
					25	71	71,10	71,12	70,83
			RMSProp	0,001	15	82	82,52	81,70	81,50
					25	84	84,84	84,31	84,23
				0,0001	15	59	59,57	59,37	58,13
					25	69	69,54	69,25	69,12
		64	Adam	0,001	15	79	80,11	79,48	79,32
					25	81	82,18	81,49	81,28
				0,0001	15	59	59,52	59,11	58,54
					25	68	67,94	67,76	67,31
			RMSProp	0,001	15	80	80,82	80,26	80,11
					25	82	82,56	82,21	82,05
				0,0001	15	52	53,15	51,52	50,30
					25	63	63,48	63,25	62,86
80	20	32	Adam	0,001	15	83	82,95	82,54	82,40
					25	82	82,78	82,78	82,41
				0,0001	15	68	68,65	68,36	67,88
					25	70	70,26	69,78	69,49
			RMSProp	0,001	15	80	80,29	79,61	79,55
					25	82	83,01	82,46	82,39
				0,0001	15	63	63,90	63,20	62,54
					25	69	69,22	69,01	68,58
		64	Adam	0,001	15	82	82,47	81,59	81,33
					25	82	82,05	81,90	81,70
				0,0001	15	64	63,48	63,53	62,94
					25	70	69,62	69,70	69,35
			RMSProp	0,001	15	81	81,43	80,65	80,31
					25	78	78,04	77,76	77,53
				0,0001	'15	56	56,54	55,99	54,82
					25	66	66,16	66,03	65,64
90	10	32	Adam	0,001	15	85	85,40	84,74	84,62
					25	83	83,66	83,28	83,18
				0,0001	15	68	68,94	68,45	68,19
					25	75	75,10	74,66	74,49
			RMSProp	0,001	15	86	86,57	86,21	86,08
					25	83	83,85	83,02	83,05
				0,0001	15	64	64,10	63,71	63,02
					25	72	72,81	72,33	72,19
		64	Adam	0,001	15	84	84,66	83,97	83,91
					25	84	84,14	83,62	83,46
				0,0001	15	64	64,32	63,71	63,29
					25	71	71,55	71,12	70,83
			RMSProp	0,001	15	78	79,82	78,02	77,69
					25	82	82,56	81,81	81,65
				0,0001	15	60	60,94	59,66	58,74
					25	70	70,39	69,91	69,73

Berdasarkan hasil pengujian pada Tabel 4.10 dengan melakukan 48 eksperimen menunjukkan bahwa pada hasil pengujian ke 37 mendapatkan akurasi tertinggi diperoleh dengan pembagian data 90% *training* dan 10% *testing* akurasi sebesar 86%, *precision* sebesar 86,57%, *recall* sebesar 86,21%, dan *F1-Score* sebesar 86,08% dengan menggunakan *optimizer* RMSProp, 0,001 *learning rate*, 15 *epoch* dan 32 *batch size*.

G. Pelatihan *Support Vector Machine* (SVM)

Proses pelatihan ini, data yang telah dilakukan ekstraksi akan dibagi menjadi 2 yaitu untuk proses *training* (pelatihan) dan *testing* (pengujian). Dari keseluruhan data yang berjumlah 2320 citra karakter tulisan Korea *(Hangeul)* diaugmentasikan menjadi 11600 data, yang digunakan sebesar 70% *training* (8120 data): 30% *testing* (3480 data), 80% *training* (9280 data) :20% *testing* (2320 data), 90% *training* (10440 data):10% *testing* (1160 data). Data tersebut yang sudah dilakukan ekstraksi fitur HOG akan menghasilkan citra ekstraksi fitur HOG yang mempunyai representasi numerik berjumlah 144 fitur pada setiap gambar, dimana digunakan sebagai inputan fitur dalam klasifikasi SVM. Selain itu, penelitian ini juga menggunakan empat kernel berbeda yaitu linear, *polynomial, Radial Basis Function* (RBF), dan *sigmoid* untuk menentukan kernel terbaik dalam proses klasifikasi sesuai dengan Persamaan 2.15 sampai 2.20.

1. Perhitungan Manual

Pada pelatihan model SVM dalam penelitian ini, digunakan 3 sampel [’ae’, ’ae’, ’g’] dengan nilai label kelas [1, 1, -1], yang menjadi nilai *y* pada langkah perhitungan matriks Hessian. Berikut ini adalah contoh representasi numerik dari dataset gambar *training* menggunakan [’ae’, ’ae’, ’g’] sebagai nilai *x* dalam perhitungan kernel linier, *polynomial*, *Radial Basis Function* (RBF), dan *sigmoid*. Dataset ini mempunyai ekstraksi fitur HOG sebesar 144 fitur, yang ditunjukkan pada Tabel 4.11, karena data bersifat *non-linearly separable*, maka data harus ditransformasikan ke ruang yang lebih besar *(feature space)* dengan menggunakan *kernel trick* agar model dapat memisahkan dengan baik dalam penentuan *hyperplane*.

Tabel 4.11: Contoh Nilai Fitur

Sampel	Fitur 1	Fitur 2	…	Fitur 56	Fitur 57	Fitur 144
ae_aug0	0,30785	0,09936	…	0,22476	0,04281	0
ae_aug1	0,00222	0	…	0,34091	0,34091	0,01649
g_aug0	0	0	…	0,300894364	0,300894364	0

- **Kernel Linier**

  Pada tahap kernel linier, akan dilakukan perhitungan berdasarkan Persamaan 2.15 dengan menggunakan beberapa sampel data yang terdapat pada Tabel 4.11, dengan penetapan nilai parameter yang diperoleh melalui optimasi model dengan menggunakan salah satu sampel perhitungan pembagian rasio 90:10 yang telah di jelaskan pada tahap optimasi model, dengan rincian C = 0,1; ϵ = 0,00000001; λ = 0,01724 dan γ = 0,003328. Tabel 4.12 menyajikan contoh hasil dari perhitungan kernel linier untuk beberapa sampel. Hasil perhitungan kernel linier untuk tiga

sampel sesuai yang tercantum dalam Tabel 4.3 menunjukkan perhitungan nilai sampel 1 ae terhadap sampel 1 ae pada pembagian 90:10, sesuai dengan Persamaan 2.14.

$$K(x_1, x_1) = x_1 \cdot x_1$$

$$K(x_1, x_1) = ((0,307855623 \times 0,307855623) +$$

$$(0,099364826 \times 0,099364826)$$

$$+ (0,088561086 \times 0,088561086) + \cdots)$$

$$K(x_1, x_1) = 3,999999997.$$

Tabel 4.12: Sampel Hasil Perhitungan Kernel Linier

y / **D**	**1**	**1**	**-1**
1	3,999999997	1,670888351	2,213579222
1	1,670888351	3,999999999	2,261912177
-1	2,213579222	2,261912177	3,999999997

Setelah hasil perhitungan kernel linier dihasilkan, maka dilanjutkan dengan melakukan perhitungan nilai matriks Hessian dari seluruh data latih dengan λ= 0.01724 menggunakan Persamaan 2.21 sehingga diperoleh contoh perhitungannya adalah sebagai berikut.

$$D_{11} = (y_1)(y_1)\ \ K(x_i, x_j) + \lambda^2$$

$$D_{11} = (1)(1)\ \ 3,999999997 + (0,01724)^2$$

$$D_{11} = (1)(1)\ (4,000297215)$$

$$D_{11} = 4,000297215$$

Perhitungan ini dilakukan secara berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti pada Tabel 4.13 berikut ini.

Tabel 4.13: Sample Hasil Perhitungan Matriks Hessian

D	1	2	3
1	4,000297215	1,67E+00	-2,21387644
2	1,671185569	4,000297216	-2,262209395
3	-2,21387644	-2,262209395	4,000297214

Setelah hasil perhitungan matriks Hessian diperoleh, maka akan dilanjutkan dengan tahap selanjutnya yaitu menghitung nilai *error* terlebih dahulu dengan nilai awalnya adalah 0, menggunakan Persamaan 2.22 seperti berikut ini:

$$E_{11} = \sum_{j=1}^{3} \alpha_1 D_{1j}$$

$$E_{11} = (0 \times 4,000297215) + (0 \times (1,67E+00)) + (0 \times (-2,21387644))$$

$$E_{11} = 0$$

Lakukan perhitungan secara berulang hingga data terakhir, sehingga menghasilkan data seperti Tabel 4.14 sebagai berikut.

Tabel 4.14: Sampel Hasil Perhitungan *Error*

D	E						
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	1,15E-02	2,30E-02	0,046011873	0,092023747	0,184047493	3,46E-01
2	0	0,011346062	0,022692124	0,045384247	0,090768495	0,181536989	0,340927339
3	0	-0,001583425	-0,003169356	-0,006338713	-0,012677425	-0,02535485	-0,047578862

Selanjutnya, setelah nilai *error* (E_i) diperoleh, maka nilai selanjutrnya akan menghitung nilai *delta alpha* yang bertujuan untuk memperbarui nilai *alpha* sebagai parameter bobot setiap data, perhitungan dilakukan dengan nilai α mula - mulanya adalah 0 dan C=0,1 dengan Persamaan 2.23. sehinngga perhitungan manualnya diperoleh:

$$\delta\alpha_1 = \min\{\max[\gamma(1 - E_1), \alpha_1], C - \alpha_1\}$$

$$\delta\alpha_1 = \min\{\max[0,003328(1 - 0), 0], (0,1) - 0\}$$

$$\delta\alpha_1 = \min\{\max[0,003328, 0], (0,1)\}$$

$$\delta\alpha_1 = 0,003328$$

Lakukan perhitungan secara berulang hingga data terakhir, sehingga menghasilkan data seperti Tabel 4.15 sebagai berikut.

Tabel 4.15: Sampel Hasil Perhitungan *Delta Alpha*

D	$\delta\alpha$						
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0,003328	0,003328	0,006656	0,013312	0,026624	0,046752	0
2	0,003328	0,003328	0,006656	0,013312	0,026624	0,046752	0
3	0,003328	0,00333327	0,00666127	0,013322539	0,026645079	0,046709843	0

Perhitungan nilai *alpha* pada tahap ini bertujuan untuk melakukan pembaruan terhadap nilai *alpha*. Perhitungan ini menggunakan Persamaan 2.24. Perhitungan nilai *alpha* untuk data dengan kelas sampel sama dengan huruf ae dapat dilihat pada perhitungan berikut ini :

$$\alpha_{12} = \alpha_{11} + \delta\alpha_{11}$$

$$\alpha_{12} = 0 + 0,003328$$

$$\alpha_{12} = 0,003328$$

Proses perhitungan dilakukan secara berulang hingga diperoleh nilai $\|\delta\alpha\| < \epsilon$, dengan $\epsilon = 0,00000001$. Jika kondisi ini belum tercapai, iterasi akan terus berlanjut dengan menghitung *error* untuk setiap data, menentukan nilai $\delta\alpha$ berikutnya, dan memperbarui nilai *alpha* secara terus-menerus. Pada iterasi ke-7, kondisi telah terpenuhi dengan menunjukkan bahwa $\delta\alpha = 0$, sebagaimana ditampilkan pada Tabel 4.15. Dengan demikian, perhitungan dihentikan, dan diperoleh nilai $\alpha = 0, 1$, seperti yang terlihat pada Tabel 4.16.

Tabel 4.16: Sampel Hasil Perhitungan Pembaharuan *Alpha*

D	α						
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0,003328	0,006656	0,013312	0,026624	0,053248	0,1
2	0	0,003328	0,006656	0,013312	0,026624	0,053248	0,1
3	0	0,003328	0,00666127	0,013322539	0,026645079	0,053290157	0,1

Setelah diperoleh nilai akhir *alpha* yang memenuhi kriteria, langkah selanjutnya adalah menghitung nilai bias menggunakan Persamaan 2.25. Nilai bias, yang dilambangkan dengan *b*, menggambarkan bidang relatif terhadap koordinat pusat. Nilai $K(x_i, x^+)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label positif, sementara $K(x_i, x^-)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label negatif. Nilai $K(x_i, x^+)$ diambil

dari data pertama, sementara $K(x_i, x^-)$ diambil dari data ketiga. Nilai-nilai data ini diperoleh melalui perhitungan matriks Hessian seperti yang ditunjukkan dalam Tabel 4.13. Oleh karena itu, perhitungannya adalah sebagai berikut.

$$K(x_1, x^+) = \sum_{i=1}^{n} \alpha_7 y_1 K(x_i, x^+)$$

$$K(x_1, x^+) = (0, 1 \times 1 \times 4, 000297215) + (0, 1 \times 1 \times (1, 67E + 00))$$

$$+ (0, 1 \times (-1) \times (-2, 21387644))$$

$$K(x_1, x^+) = 0, 788417366$$

$$K(x_1, x^-) = \sum_{i=1}^{n} \alpha_7 y_1 K(x_i, x^-)$$

$$K(x_1, x^-) = (0, 1 \times 1 \times (-2, 21387644)) + (0, 1 \times 1 \times$$

$$(-2, 262209395)) + (0, 1 \times (-1) \times (4, 000297214))$$

$$K(x_1, x^-) = -0, 847638305$$

Selanjutnya menghitung bias menggunakan Persamaan 2.25.

$$b = -\frac{1}{2}\left[\sum_{i=1}^{m} \alpha_i y_i K(x_i, x^+) + \sum_{i=1}^{m} \alpha_i y_i K(x_i, x^-)\right]$$

$$b = -\frac{1}{2}[0,788417366 + (-0,847638305)]$$

$$b = -\frac{1}{2}[-0,059220939]$$

$$b = 0,02961047$$

Setelah tahap pelatihan data selesai, langkah selanjutnya adalah melakukan pengujian data dengan menghitung $f(x)$.Tahapan pertama yang dilakukan adalah menghitung nilai kernel pada data *testing* terhadap seluruh data pelatihan menggunakan rumus kernel linier sesuai dengan Persamaan 2.15. Berikut adalah contoh perhitungan dengan mengambil sampel data *testing* dengan kelas 'ae' positif.

$$K(x_{testing}, x_1) = ((0,005568693 \times 0,307855623) +$$

$$(0 \times 0,099364826)$$

$$+ (0 \times 0,088561086) + \cdots)$$

$$K(x_{testing}, x_1) = 2,52835272$$

Lakukan perhitungan tersebut secara berulang pada seluruh data *training*, dan Tabel 4.17 ini merupakan hasil

dari perhitungannya.

Tabel 4.17: Hasil Perhitungan Kernel Data *Testing* Terhadap Data Training

$K(x_{testing}, x_1)$	$K(x_{testing}, x_2)$	$K(x_{testing}, x_3)$
2,52835272	2,655340303	2,914737183

Setelah diperoleh nilai masing-masing kernel, kemudian dilakukan klasifikasi data dengan menggunakan Persamaan 2.26. sebagai berikut:

$$f(x_{testing}) = \sum_{i=1}^{m} \alpha_7 y_1 K(x_{testing}, x_i) + b$$

$$f(x_{testing}) = (0,1 \times 1 \times 2,52835272) + (0,1 \times 1 \times 2,655340303)$$

$$+ (0,1 \times (-1) \times 2,914737183) + 0,02961047$$

$$f(x_{testing}) = 0,517942584$$

Setelah diperoleh nilai $f(x)$ pada data *testing*, klasifikasi menghasilkan dua kelas yaitu kelas 'ae' dan kelas selain 'ae' dengan ketentuan yaitu untuk kelas 'ae' bernilai 1 dan kelas selain 'ae' bernilai -1. Jika nilai $f(x) > 0$, maka data tersebut masuk kedalam kategori kelas 1 yaitu kelas 'ae', sedangkan jika nilai $f(x) < 0$ maka data tersebut termasuk dalam kategori kelas selain 'ae'. Sehingga dengan nilai yang diperoleh dari $f(x_{testing})$ =0,517942584 , maka terprediksi sebagai kelas 'ae' dengan nilai aktual yang sama yaitu pada kelas 'ae'.

– **Optimasi Model Parameter**

Optimasi model dilakukan dengan menggunakan metode *Grid Search CV*. *Grid Search CV* adalah proses pemilihan kombinasi terbaik dari parameter dan hiperparameter model (Waladi dkk., 2024). Dengan melakukan pencarian parameter yang sistematis dan mengevaluasi setiap kombinasi menggunakan *Cross Validation*, model dengan kinerja terbaik dapat terpilih. *Hyperparameter* digunakan untuk meningkatkan performa model dengan melakukan pencarian parameter terbaik. Dalam penelitian ini, *hyperparameter* yang digunakan kernel linier mencakup *C* , *gamma*, dan kernel. Parameter *C* digunakan untuk mengatur kompleksitas model, dan kernel menentukan tipe fungsi untuk memetakan data ke ruang fitur yang berbeda. Kernel yang diuji dalam optimasi ini adalah kernel linier.

Metode *Cross Validation* digunakan untuk mengevaluasi kinerja setiap kombinasi parameter dengan membagi data menjadi lima bagian, di mana empat bagian digunakan untuk melatih model dan satu bagian untuk menguji model. Proses ini dilakukan lima kali untuk setiap 10 kombinasi hiperparameter, menghasilkan total 50 pengulangan. Skor akurasi dari setiap pengulangan dari pembagian dataset 70:30, 80:20, dan 90:10 digunakan sebagai pengukur kinerja model. Dalam pembagian 70:30, kombinasi *hyperparameter* terbaik yang terpilih adalah *C* = 0, 01, *gamma*= *scale*, dan kernel = linier dengan skor

0,43. Sedangkan dari pembagian dataset 80:20, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 0, 01, *gamma*= *scale* dan kernel = linier dengan skor 0,43. Pada pembagian dataset 90:10, kombinasi *hyperparameter* terbaik yang terpilih adalah *C* = 0, 1, *gamma*= *scale*, dan kernel = linier dengan skor 0,46. Berikut ini adalah 10 kombinasi *hyperparameter* untuk kernel linier yang diuji dalam optimasi, yang ditunjukkan pada Tabel 4.18.

Tabel 4.18: Kandidat Kombinasi *Hyperparameter* Tuning

C	Gamma	Kernel
0,001	scale	linier
0,001	auto	linier
0,01	scale	linier
0,01	auto	linier
0,1	scale	linier
0,1	auto	linier
1	scale	linier
1	auto	linier
10	scale	linier
10	auto	linier

Selanjutnya, terdapat dua metode berbeda untuk menghitung parameter gamma, yaitu *scale* dan *auto*, di mana metode terbaik akan dipilih melalui *hyperparameter tuning*. Dalam penelitian ini, perhitungan dilakukan berdasarkan karakteristik data, seperti jumlah fitur dan variasi data. Pemilihan metode ini akan memengaruhi cara perhitungan nilai sigma. Berikut adalah contoh perhitungan berdasarkan Persamaan 2.27 hingga 2.28.

$$\text{Scale} = \frac{1}{\text{jumlah dimensi fitur} \times \text{jumlah variasi fitur}}$$

$$\text{Scale} = \frac{1}{144 \times 2.0893}$$

$$\text{Scale} = 0,003328$$

$$\text{Auto} = \frac{1}{\text{jumlah dimensi fitur}}$$

$$\text{Auto} = \frac{1}{144}$$

$$\text{Auto} = 0,00694$$

– **Kernel *Polynomial***

Perhitungan kernel *polynomial* dilakukan berdasarkan Persamaan 2.16 dengan menggunakan beberapa sampel data yang terdapat pada Tabel 4.11, dengan penetapan nilai parameter yang diperoleh melalui optimasi model yang telah di jelaskan pada tahap optimasi model, dengan rincian C = 10, d= 3, λ=0,01724, ϵ= 0,00000001, dan γ=0,003328. Tabel 4.19 menyajikan contoh hasil dari perhitungan kernel *polynomial* untuk beberapa sampel. Berikut ini merupakan sampel perhitungan kernel *polynomial*.

$$K(x_1, x_1) = (x_1 \cdot x_1)^d$$

$$K(x_1, x_1) = ((0,307855623 \times 0,307855623) +$$

$$(0,099364826 \times 0,099364826)$$

$$+ (0,088561086 \times 0,088561086) + \cdot \cdot \cdot)^3$$

$$K(x_1, x_1) = 63,99999986.$$

Perhitungan ini dilakukan seacra berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti pada Tabel 4.19

Tabel 4.19: Sampel Hasil Perhitungan Kernel *Polynomial*

y / **D**	**1**	**1**	**-1**
1	63,99999986	4,664899524	10,84638982
1	4,664899524	63,99999994	11,57250071
-1	10,84638982	11,57250071	63,99999984

Setelah hasil perhitungan kernel *polynomial* didapatkan, dilanjutkan dengan melakukan perhitungan nilai matriks Hessian dari seluruh data latih dengan λ=0,01724 menggunakan Persamaan 2.21 sehingga diperoleh perhitungannya sebagai berikut ini.

$$D_{11} = (y_1)(y_1)\ \ K(x_i, x_j) + \lambda^2$$

$$D_{11} = (1)(1)\ \ 63,99999986 + (0,01724)^2$$

$$D_{11} = (1)(1)\ (64,00029708)$$

$$D_{11} = 64,00029708$$

Perhitungan ini dilakukan secara berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti pada Tabel 4.20 berikut ini.

Tabel 4.20: Sample Hasil Perhitungan Matriks Hessian

D	1	2	3
1	64,00029708	4,66E+00	-10,84668704
2	4,665196742	64,00029715	-11,57279792
3	-10,84668704	-11,57279792	64,00029706

Setelah hasil perhitungan matriks Hessian didapatkan, maka akan dilanjutkan dengan tahap selanjutnya yaitu menghitung nilai error terlebih dahulu dengan nilai awalnya adalah 0, menggunakan Persamaan 2.22 seperti berikut:

$$E_{11} = \sum_{j=1}^{n} \alpha_1 D_{1j}$$

$$E_{11} = (0 \times (64,00029708)) + (0 \times (4,66E+00))$$

$$+ (0 \times (-10,84668704))$$

$$E_{11} = 0$$

Lakukan perhitungan secara berulang hingga data terakhir, sehingga menghasilkan data seperti Tabel 4.21 dan 4.22 sebagai berikut.

Tabel 4.21: Sampel Hasil Perhitungan *Error*(Iterasi 1–7)

D	E						
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	1,92E-01	3,85E-01	0,769679999	1,539359999	3,078719998	6,16E+00
2	0	0,190004492	0,380008984	0,760017969	1,520035938	3,040071875	6,08014375
3	0	0,138380943	0,276761885	0,553523771	1,107047541	2,214095083	4,428190165

Tabel 4.22: Sampel Hasil Perhitungan *Error* (Iterasi 8–14)

D	E						
Iterasi	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	12,31487999	24,62975998	49,25951996	9,85E+01	1,97E+02	3,94E+02	5,78E+02
2	12,1602875	24,320575	48,64115	97,2823	194,5646	389,1292	570,9269597
3	8,856380331	17,71276066	35,42552132	70,85104265	141,7020853	283,4041706	415,808121

Selanjutnya, setelah nilai *error* (E_i) diperoleh, maka nilai selanjutrnya akan menghitung nilai *delta alpha* yang bertujuan untuk memperbarui nilai *alpha* sebagai parameter bobot setiap data, perhitungan dilakukan dengan nilai α mula - mulanya adalah 0 dan C=10, perhitungannya seperti pada Persamaan 2.23. sehinngga diperoleh:

$$\delta\alpha_1 = \min\{\max[\gamma(1 - E_1), \alpha_1], C - \alpha_1\}$$

$$\delta\alpha_1 = \min\{\max[0, 003328(1 - 0), 0], 10 - 0\}$$

$$\delta\alpha_1 = \min\{\max[0, 003328, 0], 10\}$$

$$\delta\alpha_1 = 0, 003328$$

Lakukan perhitungan secara berulang hingga data terakhir, sehingga menghasilkan data seperti Tabel 4.23 dan 4.24 sebagai berikut.

Tabel 4.23: Sampel Hasil Perhitungan *Delta Alpha* Iterasi 1-6

D	$\delta\alpha$					
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**
1	0,003328	0,003328	0,006656	0,013312	0,026624	0,053248
2	0,003328	0,003328	0,006656	0,013312	0,026624	0,053248
3	0,003328	0,003328	0,006656	0,013312	0,026624	0,053248

Tabel 4.24: Sampel Hasil Perhitungan *Delta Alpha* Iterasi 7-14

D	$\delta\alpha$							
Iterasi	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	0,106496	0,212992	0,425984	0,851968	1,703936	3,407872	3,184256	0
2	0,106496	0,212992	0,425984	0,851968	1,703936	3,407872	3,184256	0
3	0,106496	0,212992	0,425984	0,851968	1,703936	3,407872	3,184256	0

Perhitungan nilai *alpha* pada tahap ini bertujuan untuk melakukan pembaruan terhadap nilai *alpha* dengan menggunakan Persamaan 2.24. Perhitungan nilai *alpha* untuk data dengan nomor Sampel sama dengan huruf ae dapat dilihat pada perhitungan berikut ini :

$$\alpha_{12} = \alpha_{11} + \delta\alpha_{11}$$

$$\alpha_{12} = 0 + 0,003328$$

$$\alpha_{12} = 0,003328$$

Proses perhitungan dilakukan secara berulang hingga diperoleh nilai $\|\delta\alpha\| < \epsilon$, dengan $\epsilon = 0,00000001$. Jika kondisi ini belum tercapai, iterasi akan terus berlanjut dengan menghitung error untuk setiap data, menentukan nilai $\delta\alpha$ berikutnya,

dan memperbarui nilai *alpha* secara terus-menerus. Pada iterasi ke-14, kondisi telah terpenuhi dengan menunjukkan bahwa $\delta\alpha = 0$, sebagaimana ditampilkan pada Tabel 4.23. Dengan demikian, perhitungan dihentikan, dan diperoleh nilai $\alpha = 10$, seperti yang terlihat pada Tabel 4.25 dan Tabel 4.26.

Tabel 4.25: Sampel Hasil Perhitungan Pembaharuan *Alpha* Iterasi 1-6

D	α					
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**
1	0	0,003328	0,006656	0,013312	0,026624	0,053248
2	0	0,003328	0,006656	0,013312	0,026624	0,053248
3	0	0,003328	0,006656	0,013312	0,026624	0,053248

Tabel 4.26: Sampel Hasil Perhitungan Pembaharuan *Alpha* Iterasi 7-13

D	α							
Iterasi	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	0,106496	0,212992	0,425984	0,851968	1,703936	3,407872	6,815744	10
2	0,106496	0,212992	0,425984	0,851968	1,703936	3,407872	6,815744	10
3	0,106496	0,212992	0,425984	0,851968	1,703936	3,407872	6,815744	10

Setelah memperoleh nilai akhir *alpha* yang memenuhi kriteria, langkah selanjutnya adalah menghitung nilai bias menggunakan persamaan 2.25. Nilai bias, yang dilambangkan dengan *b*, menggambarkan bidang relatif terhadap koordinat pusat. Nilai $K(x_i, x^+)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label positif, sementara $K(x_i, x^-)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label negatif. Nilai $K(x_i, x^+)$ diambil dari data pertama, sementara $K(x_i, x^-)$ diambil dari

data ketiga. Nilai-nilai data ini diperoleh melalui perhitungan matriks Hessian seperti yang ditunjukkan dalam Tabel 4.20. Oleh karena itu, perhitungannya adalah sebagai berikut.

$$K(x_1, x^+) = \sum_{i=1}^{m} \alpha_{14} y_1 K(x_i, x^+)$$

$$K(x_1, x^+) = (10 \times 1 \times 64,00029708) + (10 \times 1 \times (4,66E + 00)) + (10 \times (-1) \times (10,84668704))$$

$$K(x_1, x^+) = 578,1361004$$

$$K(x_1, x^-) = \sum_{i=1}^{m} \alpha_{14} y_1 K(x_i, x^-)$$

$$K(x_1, x^-) = (10 \times 1 \times (-10,84668704)) + (10 \times 1 \times (-11,57279792)) + (10 \times (-1) \times (-64,00029706))$$

$$K(x_1, x^-) = -864,1978202$$

Selanjutnya menghitung bias menggunakan Persamaan 2.25. Berikut ini merupakan sampel perhitungan bias.

$$b = -\frac{1}{2}\left[\sum_{i=1}^{m} \alpha_i y_i K(x, x^+) + \sum_{i=1}^{m} \alpha_i y_i K(x, x^-)\right]$$

$$b = -\frac{1}{2}[578,1361004 + (-864,1978202)]$$

$$b = -\frac{1}{2}[-286,0617198]$$

$$b = 143,0308599$$

Setelah tahap pelatihan data selesai, langkah selanjutnya adalah melakukan pengujian data dengan menghitung $f(x)$.Tahapan pertama yang dilakukan adalah menghitung nilai kernel pada data *testing* terhadap seluruh data pelatihan menggunakan rumus *polynomial* sesuai dengan Persamaan 2.16. Berikut adalah contoh perhitungan dengan mengambil sampel data *testing* dengan kelas 'ae' positif.

$$K(x_{testing}, x_1) = ((0,005568693 \times 0,307855623) + (0 \times 0,099364826) + (0 \times 0,088561086) + \cdots)^3$$

$$K(x_{testing}, x_1) = 16,16266538.$$

Lakukan perhitungan tersebut secara berulang pada seluruh data *training*, dan Tabel 4.27 ini merupakan hasil dari perhitungannya.

Tabel 4.27: Hasil Perhitungan Kernel Data *Testing* Terhadap Data Training

$K(x_{testing}, x_1)$	$K(x_{testing}, x_2)$	$K(x_{testing}, x_3)$
16,16266538	18,72235871	24,76271183

Setelah mendapatkan nilai masing-masing kernel, kemudian dilakukan klasifikasi data dengan menggunakan Persamaan 2.26 sebagai berikut:

$$f(x_{testing}) = \sum_{i=1}^{m} \alpha_{14} y_1 K(x_{testing}, x_i) + b$$

$$= (10 \times 1 \times 16,16266538) + (10 \times 1 \times 18,72235871)$$

$$+ (10 \times (-1) \times 24,76271183) + 143,0308599$$

$$f(x_{testing}) = 244,2539825$$

Setelah diperoleh nilai $f(x)$ pada data *testing*, klasifikasi menghasilkan dua kelas yaitu kelas 'ae' dan kelas selain 'ae' dengan ketentuan yaitu untuk kelas 'ae' bernilai 1 dan kelas selain 'ae' bernilai -1. Jika nilai $f(x) > 0$, maka data tersebut masuk kedalam kategori kelas 1 yaitu kelas 'ae', sedangkan jika nilai $f(x) < 0$ maka data tersebut termasuk dalam kategori kelas selain 'ae'. Sehingga dengan nilai yang diperoleh dari $f(x_{testing}) = 244,2539825$, maka terprediksi sebagai

kelas 'ae' dengan nilai aktual yang sama yaitu pada kelas 'ae'.

* **Optimasi Model Parameter**

Optimasi model dilakukan dengan menggunakan metode *Grid Search CV*. *Grid Search CV* adalah proses pemilihan kombinasi terbaik dari parameter dan hiperparameter model (Waladi dkk., 2024). Dengan melakukan pencarian parameter yang sistematis dan mengevaluasi setiap kombinasi menggunakan *Cross Validation*, model dengan kinerja terbaik dapat terpilih. Dalam penelitian ini, *hyperparameter* yang digunakan mencakup *C*, *gamma* , *d* dan kernel. Parameter *C* digunakan untuk mengatur kompleksitas model, *gamma* berfungsi untuk mengontrol sejauh mana pengaruh suatu data terhadap hasil prediksi, ϵ menentukan margin kesalahan di sekitar fungsi prediksi, dan kernel menentukan tipe fungsi untuk memetakan data ke ruang fitur yang berbeda. Kernel yang diuji dalam optimasi ini adalah *polynomial*.

Metode *Cross Validation* digunakan untuk mengevaluasi kinerja setiap kombinasi parameter dengan membagi data menjadi lima bagian, di mana empat bagian digunakan untuk melatih model dan satu bagian untuk menguji model. Proses ini dilakukan lima kali untuk setiap 72 kombinasi hiperparameter, menghasilkan total 360 pengulangan. Skor akurasi dari setiap pengulangan

dari pembagian dataset 70:30, 80:20, dan 90:10 digunakan sebagai pengukur kinerja model.

Dalam penelitian ini, kombinasi hiperparameter terbaik yang terpilih dari pembagian dataset 70:30 adalah *C* = 10, *d*=3, *Gamma* = scale, dan kernel = *polynomial* dengan skor 0,63. Sedangkan dari pembagian dataset 80:20, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 100,*d*=3, *Gamma* = scale, dan kernel = *polynomial* dengan skor 0,65. Pada pembagian dataset 90:10, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 10, *Gamma* = scale, *d*=3, dan kernel = *polynomial* dengan skor 0,68. Berikut ini adalah 72 kombinasi hiperparameter untuk kernel *polynomial* yang diuji dalam optimasi, yang ditunjukkan pada Tabel 4.28 dan 4.29.

Tabel 4.28: Kandidat Kombinasi *Hyperparameter* Tuning

C	Gamma	Kernel	d(Derajat)
0,001	scale	*polynomial*	2
0,001	scale	*polynomial*	3
0,001	scale	*polynomial*	4
0,001	auto	*polynomial*	2
0,001	auto	*polynomial*	3
0,001	auto	*polynomial*	4
0,01	scale	*polynomial*	2
0,01	scale	*polynomial*	3
0,01	scale	*polynomial*	4
0,01	auto	*polynomial*	2
0,01	auto	*polynomial*	3
0,01	auto	*polynomial*	4
0,1	scale	*polynomial*	2
0,1	scale	*polynomial*	3
0,1	scale	*polynomial*	4

Tabel 4.29: Kandidat Kombinasi *Hyperparameter* Tuning

C	Gamma	Kernel	d(Derajat)
0,1	auto	*polynomial*	2
0,1	auto	*polynomial*	3
0,1	auto	*polynomial*	4
1	scale	*polynomial*	2
1	scale	*polynomial*	3
1	scale	*polynomial*	4
1	auto	*polynomial*	2
1	auto	*polynomial*	3
1	auto	*polynomial*	4
10	scale	*polynomial*	2
10	scale	*polynomial*	3
10	scale	*polynomial*	4
10	auto	*polynomial*	2
10	auto	*polynomial*	3
10	auto	*polynomial*	4
100	scale	*polynomial*	2
100	scale	*polynomial*	3
100	scale	*polynomial*	4
100	auto	*polynomial*	2
100	auto	*polynomial*	3
100	auto	*polynomial*	4
1000	scale	*polynomial*	2
1000	scale	*polynomial*	3
1000	scale	*polynomial*	4
1000	auto	*polynomial*	2
1000	auto	*polynomial*	3
1000	auto	*polynomial*	4

Selanjutnya, terdapat dua metode berbeda untuk menghitung parameter gamma, yaitu *scale* dan *auto*, di mana metode terbaik akan dipilih melalui *hyperparameter tuning*. Dalam penelitian ini, perhitungan dilakukan berdasarkan karakteristik data, seperti jumlah fitur dan variasi data. Pemilihan metode ini akan memengaruhi

cara perhitungan nilai sigma. Berikut adalah contoh perhitungan berdasarkan Persamaan 2.27 hingga 2.28.

$$\text{Scale} = \frac{1}{\text{jumlah dimensi fitur} \times \text{jumlah variasi fitur}}$$

$$\text{Scale} = \frac{1}{144 \times 2{,}0856}$$

Scale = 0, 00332971

$$\text{Auto} = \frac{1}{\text{jumlah dimensi fitur}}$$

$$\text{Auto} = \frac{1}{144}$$

Auto = 0, 00694

– **Kernel *Radial Basic Function* (RBF)**

Perhitungan nilai $\| x_i, x_j \|$ dilakukan dengan menggunakan konsep jarak Euclidean, yang kemudian digunakan sebagai *input* untuk mengitung fungsi kernel RBF. Berikut adalah hasil perhitungan nilai $\| x_i, x_j \|$ dengan menggunakan nilai fitur tiga sampel sesuai pada Tabel 4.11, perhitungan nilai sampel 1 ae terhadap sampel 1 ae pada pembagian 70:30 sesuai dengan Persamaan 2.19.

$$||x_i - x_j|| = \sqrt{(x_{1i} - x_{1j})^2 + (x_{2i} - x_{2j})^2 + \cdots + (x_{ki} - x_{kj})^2}$$

$$||x_1 - x_1|| = \sqrt{(0,30786 - 0,30786)^2 + (0,09936 - 0,09936)^2 + \sqrt{(0,08856 - 0,08856)^2 + \cdots + (x_{144} - x_{144})^2}}$$

$$||x_1 - x_1|| = \sqrt{0^2 + 0^2 + 0^2 + \cdots + 0^2}$$

$$||x_1 - x_1|| = 0$$

Tabel 4.30: Sampel Nilai Perhitungan $||x_i - x_j||$

x / **sampel**	**ae**	**ae**	**g**
ae	0	2,158291753	1,890196167
ae	0,305632066	0	1,864450494
g	1,890196167	1,864450494	0

Selanjutnya, langkah awal yang dilakukan adalah menginisialisasi sejumlah variabel parameter yang diperlukan dalam proses perhitungan. Nilai-nilai parameter ini diperoleh melalui optimasi model yang telah dijelaskan pada proses optimasi, dengan rincian $\lambda = 0,01725$, $\gamma = 0.00694$, $C = 10$, $\sigma = 8,488$ dan $\epsilon = 0,00000001$. Kemudian, menghitung nilai RBF kernel setiap data dengan menggunakan Persamaan 2.18.

$$K(x_1, x_1) = \exp\ -\frac{\|x_1 - x_1\|^2}{2\sigma^2}$$

$$K(x_1, x_1) = \exp\ -\frac{0}{2(8,488)^2}$$

$$K(x_1, x_1) = \exp(0)$$

$$K(x_1, x_1) = 1$$

Perhitungan ini dilakukan secara berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti pada Tabel 4.31 berikut.

Tabel 4.31: Sampel Matriks Hasil Kernel RBF

(*y*) / D	1	1	-1
1	1	0,968188912	0,975509375
1	0,999351938	1	0,976164026
-1	0,975509375	0,976164026	1

Selanjutnya, melakukan perhitungan nilai matriks Hessian dari seluruh data latih dengan λ = 0,01725 menggunakan Persamaan 2.21. sehingga diperoleh perhitungannya seperti berikut ini.

$$D_{11} = y_1 y_1\ \ K(x_1, x_1) + \lambda^2$$

$$D_{11} = (1)(1)\ \ (1) + (0,01725)^2$$

$$D_{11} = (1)(1)\ (1,000297218)$$

$$D_{11} = 1,000297218$$

Lakukan perhitungan secara berulang hingga data yang terakhir, sehingga akan menghasilkan data pada Tabel 4.32 seperti berikut ini.

Tabel 4.32: Sampel Hasil Perhitungan Matriks Hessian

D	1	2	3
1	1,000297218	9,68E-01	-0,975806593
2	0,999649156	1,000297218	-0,976461243
3	-0,975806593	-0,976461243	1,000297218

Langkah berikutnya adalah menghitung nilai *error* terlebih dahulu dengan nilai awal α adalah 0, menggunakan Persamaan 2.22 seperti berikut:

$$E_{11} = \sum_{j=1}^{n} \alpha_1 D_{1j}$$

$$E_{11} = (0 \times 1,000297218 + (0 \times 9,68E - 01)$$

$$+ (0 \times (-0,975806593))$$

$$E_{11} = 0$$

Lakukan perhitungan secara berulang hingga data terakhir, sehingga menghasilkan data seperti Tabel 4.33 sebagai berikut.

Tabel 4.33: Sampel Hasil Perhitungan *Error* Iterasi 1-6

D	E					
Iterasi	1	2	3	4	5	6
1	0	6,87E-03	0,01374441	0,027488821	0,054977641	0,109955282
2	0	0,007105328	0,014210656	0,028421312	0,056842623	0,113685247
3	0	-0,006606676	-0,013213352	-0,026426704	-0,052853409	-0,105706817

Tabel 4.34: Sampel Hasil Perhitungan *Error* Iterasi 7-13

D	E						
Iterasi	7	8	9	10	11	12	13
1	0,219910564	0,439821129	0,879642258	1,76E+00	3,518569031	7,04E+00	9,90E+00
2	0,227370493	0,454740987	0,909481973	1,818963947	3,637927894	7,275855788	10,23822467
3	-0,211413635	-0,42282727	-0,845654539	-1,691309079	-3,382618158	-6,765236315	-9,51970618

Setelah nilai *error* (*E*) didapatkan, langkah selanjutnya menghitung delta alfa yang berguna untuk memperbarui nilai alfa sebagai parameter bobot setiap data, perhitungan dilakukan dengan nilai α mula - mula adalah 0 dan C=10, perhitungan seperti Persamaan 2.23 sebagai berikut.

$$\delta\alpha_1 = \min\{\max[\gamma(1 - E_1), \alpha_1], C - \alpha_1\}$$

$$\delta\alpha_1 = \min\{\max[0,00694(1 - 0), 0], 10 - 0\}$$

$$\delta\alpha_1 = \min\{\max[(0,00694), 0], 10\}$$

$$\delta\alpha_1 = 0,00694$$

Lakukan perhitungan delta *alpha* berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti Tabel 4.35 dan 4.36 berikut.

Tabel 4.35: Sampel Hasil Perhitungan *Delta Alpha* Iterasi 1-7

D	$\delta\alpha$						
Iterasi	1	2	3	4	5	6	7
1	0,00694	0,00694	0,01388	0,02776	0,05552	0,11104	0,22208
2	0,00694	0,00694	0,01388	0,02776	0,05552	0,11104	0,22208
3	0,00694	0,00694	0,01388	0,02776	0,05552	0,11104	0,22208

Tabel 4.36: Sampel Hasil Perhitungan *Delta Alpha* Iterasi 8-13

D	$\delta\alpha$					
Iterasi	**8**	**9**	**10**	**11**	**12**	**13**
1	0,44416	0,88832	1,77664	3,55328	2,89344	0
2	0,44416	0,88832	1,77664	3,55328	2,89344	0
3	0,44416	0,88832	1,77664	3,55328	2,89344	0

Setelah mendapatkan nilai delta *alpha* pada setiap data, maka dapat dilakukan pembaruan nilai pada *alpha* dengan menggunakan Persamaan 2.24.

$$\alpha_{12} = \alpha_{11} + \delta\alpha_{11}$$

$$\alpha_{12} = 0 + 0,00694$$

$$\alpha_{12} = 0,00694$$

Proses perhitungan dilakukan secara berulang hingga diperoleh nilai $\|\delta\alpha\| < \epsilon$, dengan $\epsilon = 0,00000001$. Jika kondisi ini belum tercapai, iterasi akan terus berlanjut dengan menghitung error untuk setiap data, menentukan nilai $\delta\alpha$ berikutnya, dan memperbarui nilai *alpha* secara terus-menerus. Pada iterasi ke-13, kondisi telah terpenuhi dengan menunjukkan bahwa $\delta\alpha = 0$, sebagaimana ditampilkan pada Tabel 4.35. Dengan demikian, perhitungan dihentikan, dan diperoleh nilai $\alpha = 10$, seperti yang terlihat pada Tabel 4.37.

Tabel 4.37: Sampel Hasil Perhitungan Pembaharuan *Alpha* Bagian A

D	α						
Iterasi	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0,00694	0,01388	0,02776	0,05552	0,11104	0,22208
2	0	0,00694	0,01388	0,02776	0,05552	0,11104	0,22208
3	0	0,00694	0,01388	0,02776	0,05552	0,11104	0,22208

Tabel 4.38: Sampel Hasil Perhitungan Pembaharuan *Alpha* Bagian B

D	α					
Iterasi	**8**	**9**	**10**	**11**	**12**	**13**
1	0,44416	0,88832	1,77664	3,55328	7,10656	10
2	0,44416	0,88832	1,77664	3,55328	7,10656	10
3	0,44416	0,88832	1,77664	3,55328	7,10656	10

Setelah memperoleh nilai akhir *alpha* yang memenuhi kriteria, langkah selanjutnya adalah menghitung nilai bias yang dilambangkan dengan *b*, menggambarkan bidang relatif terhadap koordinat pusat. Nilai $K(x_i, x^+)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label positif, sementara $K(x_i, x^-)$ merujuk pada nilai SV *(Support Vector)* untuk kelas label negatif. Nilai $K(x_i, x^+)$ diambil dari data pertama, sementara $K(x_i, x^-)$ diambil dari data ketiga. Nilai-nilai data ini diperoleh melalui perhitungan matriks Hessian seperti yang ditunjukkan dalam Tabel 4.32. Oleh karena itu, perhitungannya adalah sebagai berikut.

$$K(x_1, x^+) = \sum_{i=1}^{m} \alpha_{13} y_1 K(x_i, x^+)$$

$$= (10 \times 1 \times 1, 000297218) + (10 \times 1 \times 9, 68E - 01)$$

$$+ (10 \times (-1) \times (-0, 975806593))$$

$$K(x_1, x^+) = 29, 46778773$$

$$K(x_1, x^-) = \sum_{i=1}^{m} \alpha_{13} y_1 K(x_i, x^-)$$

$$= (10 \times 1 \times (-0, 975806593)) + (10 \times$$

$$1 \times (-0, 976461243)) + (10 \times (-1) \times 1, 000297218)$$

$$K(x_1, x^-) = -29, 52565054$$

Selanjutnya menghitung bias menggunakan Persamaan 2.25 sebagai berikut:

$$b = -\frac{1}{2}\left[\sum_{i=1}^{m} \alpha_i y_i K(x_i, x^+) + \sum_{i=1}^{m} \alpha_i y_i K(x_i, x^-)\right]$$

$$= -\frac{1}{2}[29, 46778773 + (-29, 52565054)]$$

$$= -\frac{1}{2}[-0, 05786281]$$

$$b = 0, 028931405$$

Setelah proses pelatihan data selesai, langkah selanjutnya adalah melakukan pengujian pada data dengan menghitung fungsi $f(x)$. Langkah pertama yang dilakukan adalah menghitung nilai kernel pada data *testing* terhadap seluruh data pelatihan menggunakan rumus RBF sesuai dengan Persamaan 2.18. Berikut adalah contoh perhitungan dengan mengambil sampel data *testing* dengan kelas 'ae' positif.

$$K(x_{testing}, x_1) = \exp\left(-\frac{||x_{testing} - x_1||^2}{2\sigma^2}\right)$$

$$= \exp\left(-\frac{(1,715603263)^2}{2(8,488)^2}\right)$$

$$= \exp(-0,020426454)$$

$$K(x_{testing}, x_1) = 1,020636502$$

Lakukan perhitungan tersebut secara berulang pada seluruh data *training*, dan Tabel 4.39 ini merupakan hasil dari perhitungannya.

Tabel 4.39: Hasil Perhitungan Kernel Data *Testing* Terhadap Data Training

$K(x_{testing}, x_1)$	$K(x_{testing}, x_2)$	$K(x_{testing}, x_3)$
1,020636502	1,018839126	1,015177466

Setelah mendapatkan nilai masing-masing kernel, kemudian dilakukan klasifikasi data dengan menggunakan Persamaan 2.26. sebagai berikut:

$$f(x_{testing}) = \sum_{i=1}^{m+} \alpha_{14} y_1 K(x_{testing}, x_i) + b$$

$$= (10 \times 1 \times 1,020636502) + (10 \times 1 \times 1,018839126)$$

$$+ (10 \times (-1) \times 1,015177466) + 0,028931405$$

$$f(x_{testing}) = 9,905941085$$

Setelah diperoleh nilai $f(x)$ pada data *testing*, klasifikasi menghasilkan dua kelas yaitu kelas 'ae' dan kelas selain 'ae' dengan ketentuan yaitu untuk kelas 'ae' bernilai 1 dan kelas selain 'ae' bernilai -1. Jika nilai $f(x) > 0$, maka data tersebut masuk kedalam kategori kelas 1 yaitu kelas 'ae', sedangkan jika nilai $f(x) < 0$ maka data tersebut termasuk dalam kategori kelas selain 'ae'. Sehingga dengan nilai yang diperoleh dari $f(x_{testing})$ =9,905941085, maka terprediksi sebagai kelas 'ae' dengan nilai aktual yang sama yaitu pada kelas 'ae'.

* **Optimasi Model Parameter**

Optimasi model dilakukan dengan menggunakan metode *Grid Search CV*. *Grid Search CV* adalah proses pemilihan kombinasi terbaik dari parameter dan hiperparameter model (Waladi dkk., 2024). Dengan melakukan pencarian parameter yang sistematis dan mengevaluasi setiap kombinasi menggunakan *Cross Validation*,

model dengan kinerja terbaik dapat terpilih. Dalam penelitian ini, *hyperparameter* yang digunakan mencakup *C*, *gamma* , dan kernel. Parameter *C* digunakan untuk mengatur kompleksitas model, *gamma* berfungsi untuk mengontrol sejauh mana pengaruh suatu data terhadap hasil prediksi, ϵ menentukan margin kesalahan di sekitar fungsi prediksi, dan kernel menentukan tipe fungsi untuk memetakan data ke ruang fitur yang berbeda. Kernel yang diuji dalam optimasi ini adalah *RBF*.

Metode *Cross Validation* digunakan untuk mengevaluasi kinerja setiap kombinasi parameter dengan membagi data menjadi lima bagian, di mana empat bagian digunakan untuk melatih model dan satu bagian untuk menguji model. Proses ini dilakukan lima kali untuk setiap 20 kombinasi hiperparameter, menghasilkan total 100 pengulangan. Skor akurasi dari setiap pengulangan dari pembagian dataset 70:30, 80:20, dan 90:10 digunakan sebagai pengukur kinerja model. Dalam penelitian ini, kombinasi hiperparameter terbaik yang terpilih dari pembagian dataset 70:30 adalah *C* = 10, *Gamma* = auto, kernel = *rbf* dengan skor 0,65. Sedangkan dari pembagian dataset 80:20, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 100, *Gamma* = auto, kernel = *rbf* dengan skor 0,68. Pada pembagian dataset 90:10, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 100, *Gamma* = scale, kernel = *rbf*

dengan skor 0,69. Berikut ini adalah 20 kombinasi hiperparameter untuk kernel *rbf* yang diuji dalam optimasi, yang ditunjukkan pada Tabel 4.40.

Tabel 4.40: Kandidat Kombinasi *Hyperparameter* Tuning

C	**Gamma**	**Kernel**
0.001	scale	rbf
0.001	auto	rbf
0.01	scale	rbf
0.01	auto	rbf
0.1	scale	rbf
0.1	auto	rbf
1	scale	rbf
1	auto	rbf
10	scale	rbf
10	auto	rbf
100	scale	rbf
100	auto	rbf
1000	scale	rbf
1000	auto	rbf

Selanjutnya, terdapat dua metode berbeda untuk menghitung parameter gamma, yaitu *scale* dan *auto*, di mana metode terbaik akan dipilih melalui *hyperparameter tuning*. Dalam penelitian ini, perhitungan dilakukan berdasarkan karakteristik data, seperti jumlah fitur dan variasi data. Pemilihan metode ini akan memengaruhi cara perhitungan nilai sigma. Berikut adalah contoh perhitungan berdasarkan Persamaan 2.27. hingga 2.30.

$$Scale = \frac{1}{\text{jumlah dimensi fitur} \times \text{jumlah variasi fitur}}$$

$$Scale = \frac{1}{144 \times 2,0864}$$

$$Scale = 0,003328$$

$$Auto = \frac{1}{\text{jumlah dimensi fitur}}$$

$$Auto = \frac{1}{144}$$

$$Auto = 0,00694$$

Setelah mengetahui nilai gamma dari metode scale, nilai tersebut dapat digunakan untuk menghitung sigma menggunakan Persamaan 2.29 dan 2.30. Berikut ini merupakan perhitungan sigma.

$$\sigma = \sqrt{\frac{1}{(\text{scale} \times 2)}}$$

$$\sigma = \sqrt{\frac{1}{0,003328 \times 2}}$$

$$\sigma = \sqrt{\frac{1}{0,006656}}$$

$$\sigma = 12,25726$$

Selain itu, jika menggunakan nilai gamma dari metode *auto*, perhitungannya adalah sebagai berikut.

$$\sigma = \sqrt{\frac{1}{(\text{auto} \times 2)}}$$

$$\sigma = \sqrt{\frac{1}{(0{,}00694 \times 2)}}$$

$$\sigma = 8{,}488$$

– **Kernel *Sigmoid***

Perhitungan kernel *sigmoid* dilakukan dengan menggunakan Persamaan 2.20 dan menggunakan beberapa sampel data pada Tabel 4.11, dengan penentuan nilai C=1, $\gamma = 0{,}00694$, $coef = 0$, $\epsilon = 0{,}00000001$ dan σ=8,488. Tabel 4.19 menampilkan contoh hasil dari perhitungan kernel *sigmoid*. Contoh perhitungan kernel *sigmoid* pada nilai sampel 1 ae terhadap sampel 1 ae pada pembagian 70:30, sesuai dengan Persamaan 2.20 pada perhitungan sebagai berikut:

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(\sigma(\boldsymbol{x_i} \cdot \boldsymbol{x_j}) + coef)$$

$$= \tanh(8, 488((0, 307855623 \times 0, 307855623)+$$

$$(0, 099364826 \times 0, 099364826) + (0, 088561086$$

$$\times 0, 088561086) + \cdot \cdot \cdot) + 0)$$

$$= \tanh(8, 488(3, 999999997 + 0))$$

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = 1$$

Perhitungan ini dilakukan seacra berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti pada Tabel 4.41 berikut.

Tabel 4.41: Sampel Hasil Perhitungan Kernel *Sigmoid*

y / **D**	**1**	**1**	**-1**
1	1	1	1
1	1	1	1
-1	1	1	1

Selanjutnya, melakukan perhitungan nilai matriks Hessian dari seluruh data latih dengan λ = 0,01724 menggunakan Persamaan 2.21, sehingga di peroleh perhitungannya seperti berikut ini.

$$D_{11} = y_1 y_1 \quad K(x_1, x_1) + \lambda^2$$

$$D_{11} = (1)(1) \quad (1) + (0{,}01724)^2$$

$$D_{11} = (1)(1)\ (1{,}\ 000297218)$$

$$D_{11} = 1{,}\ 000297218$$

Lakukan perhitungan secara berulang hingga data yang terakhir, sehingga akan menghasilkan data pada Tabel 4.42 seperti berikut ini.

Tabel 4.42: Sampel Hasil Perhitungan Matriks Hessian

D	1	2	3
1	1,000297218	1,000297218	-1,000297218
2	1,000297218	1,000297218	-1,000297218
3	-1,000297218	-1,000297218	1,000297218

Langkah berikutnya adalah menghitung nilai error terlebih dahulu dengan nilai awal α adalah 0, menggunakan Persamaan 2.22 seperti berikut:

$$E_{11} = \sum_{j=1}^{n} \alpha_1 D_{1j}$$

$$E_{11} = ((0 \times 1{,}\ 000297218) + (0 \times 1{,}\ 000297218)$$

$$+ (0 \times (-1{,}\ 000297218))$$

$$E_{11} = 0$$

Lakukan perhitungan secara berulang hingga data terakhir, sehingga menghasilkan data seperti Tabel 4.43 dan 4.44 sebagai berikut.

Tabel 4.43: Sampel Hasil Perhitungan *Error* Iterasi 1-5

D	**E**				
Iterasi	**1**	**2**	**3**	**4**	**5**
1	0	6,95E-03	1,39E-02	0,027784256	0,055568511
2	0	0,006946064	0,013892128	0,027784256	0,055568511
3	0	-0,006946064	-0,013940376	-0,027880751	-0,055761502

Tabel 4.44: Sampel Hasil Perhitungan *Error* Iterasi 6-10

D	**E**				
Iterasi	**6**	**7**	**8**	**9**	**10**
1	1,11E-01	2,22E-01	0,444548088	8,89E-01	1,00E+00
2	0,111137022	0,222274044	0,444548088	0,889096177	1,000297218
3	-0,111523005	-0,223046009	-0,446092018	-0,892184036	-1,000297218

Setelah nilai *error* (*E*) didapatkan, langkah selanjutnya menghitung delta alfa yang berguna untuk memperbarui nilai alfa sebagai parameter bobot setiap data, perhitungan dilakukan dengan nilai α mula - mula adalah 0 dan C=1, perhitungan seperti Persamaan 2.23 sebagai berikut.

$$\delta\alpha_1 = \min\{\max[\gamma(1 - E_1), \alpha_1], C - \alpha_1\}$$

$$\delta\alpha_1 = \min\{\max[0,006944(1 - 0), 0], 1 - 0\}$$

$$\delta\alpha_1 = \min\{\max[(0,006944), 0], 1\}$$

$$\delta\alpha_1 = 0,006944$$

Lakukan perhitungan delta *alpha* berulang hingga data yang terakhir, sehingga akan menghasilkan data seperti Tabel 4.45 dan Tabel 4.46 berikut.

Tabel 4.45: Sampel Hasil Perhitungan *Delta Alpha* Iterasi 1-5

D	*δα*				
Iterasi	**1**	**2**	**3**	**4**	**5**
1	0,006944	0,006944	0,013888	0,027776	0,055552
2	0,006944	0,006944	0,013888	0,027776	0,055552
3	0,006944	0,006992233	0,013936233	0,027872467	0,055744934

Tabel 4.46: Sampel Hasil Perhitungan *Delta Alpha* Iterasi 6-10

D	*δα*				
Iterasi	**6**	**7**	**8**	**9**	**10**
1	0.111104	0.222208	0.444416	0.111168	0
2	0.111104	0.222208	0.444416	0.111168	0
3	0.111490	0.222980	0.445959	0.108081	0

Setelah mendapatkan nilai delta *alpha* pada setiap data, maka dapat dilakukan pembaruan nilai *alpha* dengan menggunakan Persamaan 2.24.

$$\alpha_{12} = \alpha_{11} + \delta\alpha_{11}$$

$$\alpha_{12} = 0 + 0,006944$$

$$\alpha_{12} = 0,006944$$

Proses perhitungan dilakukan secara berulang hingga diperoleh nilai $\|\delta\alpha\| < \epsilon$, dengan $\epsilon = 0,00000001$. Jika kondisi ini belum tercapai, iterasi akan terus berlanjut dengan menghitung error untuk setiap data, menentukan nilai $\delta\alpha$ berikutnya,

dan memperbarui nilai *alpha* secara terus-menerus. Pada iterasi ke-10, kondisi telah terpenuhi dengan menunjukkan bahwa $\delta\alpha = 0$, sebagaimana ditampilkan pada Tabel 4.46. Dengan demikian, perhitungan dihentikan, dan diperoleh nilai $\alpha = 1$, seperti yang terlihat pada Tabel 4.47 dan Tabel 4.48.

Tabel 4.47: Sampel Hasil Perhitungan Pembaharuan *Alpha* Iterasi 1-5

D	α				
Iterasi	**1**	**2**	**3**	**4**	**5**
1	0	0,006944	0,013888	0,027776	0,055552
2	0	0,006944	0,013888	0,027776	0,055552
3	0	0,006944	0,013936	0,027872	0,055745

Tabel 4.48: Sampel Hasil Perhitungan Pembaharuan *Alpha* Iterasi 6-10

D	α				
Iterasi	**6**	**7**	**8**	**9**	**10**
1	0,111104	0,222208	0,444416	0,888832	1
2	0,111104	0,222208	0,444416	0,888832	1
3	0,111490	0,222980	0,445959	0,891919	1

Setelah diperoleh nilai akhir *alpha* yang memenuhi kriteria, selanjutnya menghitung nilai bias yang dilambangkan dengan *b*, menggambarkan bidang relatif terhadap koordinat pusat. Nilai $K(x_i, x^+)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label positif, sementara $K(x_i, x^-)$ merujuk pada nilai SV *(Support Vector)* untuk kelas dengan label negatif. Nilai $K(x_i, x^+)$ diambil dari data pertama, sementara $K(x_i, x^-)$ diambil dari data ketiga. Nilai-nilai data ini diperoleh melalui perhitungan matriks Hessian seperti yang ditunjukkan dalam Tabel 4.42 sebagai berikut.

$$K(x_1, x^+) = \sum_{i=1}^{m} \alpha_{10} y_1 K(x_i, x^+)$$

$$= (1 \times 1 \times 1{,}000297218) + (1 \times 1 \times 1{,}000297218) + (1 \times (-1) \times (-1{,}000297218))$$

$$K(x_1, x^+) = 3{,}000297218$$

$$K(x_1, x^-) = \sum_{i=1}^{m} \alpha_{10} y_1 K(x_i, x^-)$$

$$= (1 \times 1 \times (-1{,}000297218)) + (1 \times 1 \times (-1{,}000297218)) + (1 \times (-1) \times 1{,}000297218)$$

$$K(x_1, x^-) = -3{,}000891654$$

Selanjutnya, menghitung bias menggunakan Persamaan 2.25 sebagai berikut:

$$b = -\frac{1}{2}\left[\sum_{i=1}^{m} \alpha_i y_i K(x_i, x^+) + \sum_{i=1}^{m} \alpha_i y_i K(x_i, x^-)\right]$$

$$= -\frac{1}{2}[3{,}000891654 + (-3{,}000891654)]$$

$$= -\frac{1}{2}[0]$$

$$b = 0$$

Setelah proses pelatihan data selesai, langkah selanjutnya adalah melakukan pengujian pada data dengan menghitung fungsi $f(x)$. Langkah pertama yang dilakukan adalah menghitung nilai kernel pada data *testing* terhadap seluruh data pelatihan menggunakan rumus *sigmoid* sesuai dengan Persamaan 2.20. Berikut adalah contoh perhitungan dengan mengambil sampel data *testing* dengan kelas 'ae' positif.

$$K(\boldsymbol{x_{testing}}, \boldsymbol{x_1}) = \tanh(\sigma(\boldsymbol{x_{testing}} \cdot \boldsymbol{x_1}) + 0)$$

$$= \tanh(8,488((0,005568693 \times 0,307855623) + (0 \times 0,099364826) + (0 \times 0,088561086) + \cdots) + 0)$$

$$= \tanh(8,488(2,52835272) + 0)$$

$$K(\boldsymbol{x_{testing}}, \boldsymbol{x_1}) = 1$$

Lakukan perhitungan tersebut secara berulang pada seluruh data *training*, dan Tabel 4.49 ini merupakan hasil dari perhitungannya.

Tabel 4.49: Hasil Perhitungan Kernel Data *Testing* Terhadap Data Training

$K(x_{testing}, x_1)$	$K(x_{testing}, x_2)$	$K(x_{testing}, x_3)$
1	1	1

Setelah mendapatkan nilai masing-masing kernel, kemudian dilakukan klasifikasi data dengan

menggunakan Persamaan 2.26. sebagai berikut:

$$f(x_{testing}) = \sum_{i=1}^{m} \alpha_{10} y_1 K(x_{testing}, x_i) + b$$

$$= (1 \times 1 \times 1) + (1 \times 1 \times 1)$$

$$+ (1 \times (-1) \times 1) + 0$$

$$f(x_{testing}) = 1$$

Setelah diperoleh nilai $f(x)$ pada data *testing*, klasifikasi menghasilkan dua kelas yaitu kelas 'ae' dan kelas selain 'ae' dengan ketentuan yaitu untuk kelas 'ae' bernilai 1 dan kelas selain 'ae' bernilai -1. Jika nilai $f(x) > 0$, maka data tersebut masuk kedalam kategori kelas 1 yaitu kelas 'ae', sedangkan jika nilai $f(x) < 0$ maka data tersebut termasuk dalam kategori kelas selain 'ae'. Sehingga dengan nilai yang diperoleh dari $f(x_{testing}) = 1$, maka terprediksi sebagai kelas 'ae' dengan nilai aktual yang sama yaitu pada kelas 'ae'.

* **Optimasi Model Parameter**

  Optimasi model dilakukan dengan menggunakan metode *Grid Search CV*. Dalam penelitian ini, *hyperparameter* yang digunakan mencakup *C*, *gamma*, *coef* dan kernel. Parameter *C* digunakan untuk mengatur kompleksitas model, *gamma* berfungsi untuk mengontrol sejauh mana pengaruh suatu data terhadap hasil prediksi, ϵ digunakan untuk menentukan margin kesalahan

di sekitar fungsi prediksi, dan kernel menentukan tipe fungsi untuk memetakan data ke ruang fitur yang berbeda. Kernel yang diuji dalam optimasi ini adalah *sigmoid*.

Metode *Cross Validation* digunakan untuk mengevaluasi kinerja setiap kombinasi parameter dengan membagi data menjadi lima bagian, di mana empat bagian digunakan untuk melatih model dan satu bagian untuk menguji model. Proses ini dilakukan lima kali untuk setiap 10 kombinasi hiperparameter, menghasilkan total 50 pengulangan. Skor akurasi dari setiap pengulangan dari pembagian dataset 70:30, 80:20, dan 90:10 digunakan sebagai pengukur kinerja model. Dalam penelitian ini, kombinasi hiperparameter terbaik yang terpilih dari pembagian dataset 70:30 adalah *C* = 1, *coef* = 0, *Gamma* = auto, kernel = *sigmoid* dengan skor 0,29. Sedangkan dari pembagian dataset 80:20, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 1, *coef* = 0, *Gamma* = auto, kernel = *sigmoid* dengan skor 0,27. Pada pembagian dataset 90:10, kombinasi hiperparameter terbaik yang terpilih adalah *C* = 0*,* 1, *coef* = 0, *Gamma* = scale, kernel = *sigmoid* dengan skor 0,33. Berikut ini adalah 10 kombinasi hiperparameter untuk kernel *sigmoid* yang diuji dalam optimasi, yang ditunjukkan pada Tabel 4.50.

Tabel 4.50: Kandidat Kombinasi *Hyperparameter* Tuning

C	Gamma	Coef	Kernel
0.001	scale	0.0	*sigmoid*
0.001	auto	0.1	*sigmoid*
0.001	scale	0.5	*sigmoid*
0.001	auto	1.0	*sigmoid*
0.01	scale	0.0	*sigmoid*
0.01	auto	0.1	*sigmoid*
0.01	scale	0.5	*sigmoid*
0.01	auto	1.0	*sigmoid*
0.1	scale	0.0	*sigmoid*
0.1	auto	0.1	*sigmoid*
0.1	scale	0.5	*sigmoid*
0.1	auto	1.0	*sigmoid*
1	scale	0.0	*sigmoid*
1	auto	0.1	*sigmoid*
1	scale	0.5	*sigmoid*
1	auto	1.0	*sigmoid*
10	scale	0.0	*sigmoid*
10	auto	0.1	*sigmoid*
10	scale	0.5	*sigmoid*
10	auto	1.0	*sigmoid*

Selanjutnya, terdapat dua metode berbeda untuk menghitung parameter gamma, yaitu *scale* dan *auto*, di mana metode terbaik akan dipilih melalui *hyperparameter tuning*. Dalam penelitian ini, perhitungan dilakukan berdasarkan karakteristik data, seperti jumlah fitur dan variasi data. Pemilihan metode ini akan memengaruhi cara perhitungan nilai sigma. Berikut adalah contoh perhitungan berdasarkan Persamaan 2.27

hingga 2.30.

$$\text{Scale} = \frac{1}{\text{jumlah dimensi fitur} \times \text{jumlah variasi fitur}}$$

$$\text{Scale} = \frac{1}{144 \times 2{,}0864}$$

$$\text{Scale} = 0,\ 003328$$

$$\text{Auto} = \frac{1}{\text{jumlah dimensi fitur}}$$

$$\text{Auto} = \frac{1}{144}$$

$$\text{Auto} = 0,\ 00694$$

Setelah mengetahui nilai *gamma* dari metode *scale*, nilai tersebut dapat digunakan untuk menghitung sigma menggunakan Persamaan 2.29 dan 2.30. Berikut ini merupakan perhitungan menentukan sigma.

$$\sigma = \sqrt{\frac{1}{(\text{scale} \times 2)}}$$

$$\sigma = \sqrt{\frac{1}{0,003328 \times 2}}$$

$$\sigma = \sqrt{\frac{1}{0,006656}}$$

$$\sigma = 12,25726$$

Selain itu, jika menggunakan nilai gamma dari metode *auto*, perhitungannya adalah sebagai berikut.

$$\sigma = \sqrt{\frac{1}{(\text{auto} \times 2)}}$$

$$\sigma = \sqrt{\frac{1}{(0,00694 \times 2)}}$$

$$\sigma = 8,488$$

2. Implementasi

- **Pelatihan Model**

Pelatihan model merupakan proses membangun model dengan metode SVM, di mana model *machine learning* dilatih menggunakan seluruh data yang tersedia

dan dilakukan penyetelan *hyperparameter tuning* untuk memperoleh kinerja model yang optimal. Proses penyetelan *hyperparameter* pada model SVM menggunakan teknik *Grid Search CV* mempunyai `parametergrid` yang akan disetel, seperti `C`, `gamma`, dan `kernel`. *Hyperparameter* `C` berfungsi untuk menyeimbangkan akurasi pada data *training* dengan kemampuan model dalam melakukan generalisasi. `gamma` bertugas untuk mengatur sejauh mana data *training* yang jauh dari titik keputuan mempengaruhi model, sedangkan `kernel` digunakan untuk memilih fungsi kernel yang paling tepat.

Selanjutnya, membuat *GridSearch* dari *library scikit-learn*, yang digunakan dalam *machine learning*. Objek ini memiliki beberapa parameter, di antaranya adalah `SVC`, yang merupakan objek untuk model SVM dasar, `param_grid` yang berisi daftar *hyperparameter* yang akan dioptimalkan untuk menemukan kombinasi terbaik, `refit` yang digunakan untuk melatih ulang model terbaik, serta `verbose` yang mengontrol seberapa detail informasi yang ditampilkan selama proses pencarian model. Berikut ini adalah implementasi kode untuk pelatihan SVM dan optimasi model dari masing - masing kernel. Proses ini dimulai dengan melatih model SVM menggunakan metode `fit`, yang memerlukan dua parameter, yaitu fitur data HOG yang tersimpan dalam variabel *fitur_train_scaled* serta label data yang akan dilatih, yang disimpan dalam *label_encoded_train*.

- Kernel Linier

```
# Hyperparameter tuning dengan GridSearchCV untuk
kernel linier
   param_grid = {
       'C': [0.001, 0.01, 0.1, 1, 10],
       'kernel': ['linear']
       'gamma': ['scale', 'auto']
   }

   grid_search = GridSearchCV(SVC(tol=nilai_tol,
   probability=True), param_grid, cv=5, refit=True, verbose=3)
   grid_search.fit(fitur_train_scaled, label_encoded_train)

   best_params = grid_search.best_params_
   print(f'Parameter terbaik yang ditemukan: {best_params}',
   file=f)
```

Gambar 4.52: *Source Code* Pelatihan Model SVM dengan Kernel Linier

Grid Search CV yang ditunjukkan pada Gambar 4.52 akan mencari kombinasi parameter yang terbaik. Berikut ini merupakan tampilan proses pelatihan model beserta optimasi parameter dengan masing - masing pembagian 70:30, 80:20, dan 90:10.

1. Pembagian rasio 70:30

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END C=0.001, gamma=scale, kernel=linear;, score=0.332
total time=  15.8s
[CV 2/5] END C=0.001, gamma=scale, kernel=linear;, score=0.320
total time=  16.5s
.......
[CV 5/5] END ...C=10, gamma=auto, kernel=linear;, score=0.365
total time= 2.8min
Parameter terbaik yang ditemukan: {'C': 0.01, gamma=scale,
'kernel': 'linear'}
```

Gambar 4.53: Proses Pelatihan dan Optimizer Model SVM dengan Kernel Linier Rasio 70:30

Gambar 4.53 menunjukkan bahwa model yang diuji dengan *5-fold cross-validation* untuk setiap kombinasi dari 10 kandidat parameter, sehingga total yang dilakukan sebanyak 50 eksperimen. Parameter yang diuji mencakup *C* (*regulazation parameter*), *gamma*,dan kernel. Setiap parameter yang diuji lima kali, dengan skor evaluasi dari masing - masing *fold* menunjukkan bahwa performa model meningkat seiring dengan peningkatan nilai *C*, tetapi waktu pelatihan yang dibutuhkan akan bertambah secara signifikan, seperti parameter *C*=10, yang membutuhkan waktu hingga 2,8 menit per *fold*.Setelah semua kombinasi diujikan, parameter terbaik pada pembagian rasio 70:30 yang diperoleh *C* = 0.01, *gamma*= *auto* dan kernel= Linier yang memberikan keseimbangan baik antara performa dan efisiensi waktu pelatihan. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

2. Pembagian rasio 80:20

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END C=0.001, gamma=scale, kernel=linear;, score=0.334
total time=  22.1s
[CV 2/5] END C=0.001, gamma=scale, kernel=linear;, score=0.326
total time=  23.1s
[CV 3/5] END C=0.001, gamma=scale, kernel=linear;, score=0.344
total time=  22.6s
.......
[CV 4/5] END ...C=10, gamma=auto, kernel=linear;, score=0.370
total time= 5.2min
[CV 5/5] END ...C=10, gamma=auto, kernel=linear;, score=0.411
total time= 5.2min
Parameter terbaik yang ditemukan: {'C': 0.01, gamma=scale,
'kernel': 'linear'}
```

Gambar 4.54: Proses Pelatihan dan Optimizer Model SVM dengan Kernel Linier Rasio 80:20

Gambar 4.54 menunjukkan bahwa model diuji dengan *5-fold cross-validation* untuk setiap kombinasi dari 10 kandidat parameter, sehingga total dilakukan sebanyak 50 eksperimen. Parameter yang diuji mencakup *C* (*regularization parameter*), *gamma*,dan kernel. Setiap parameter diuji lima kali, dengan skor evaluasi dari masing-masing *fold* menunjukkan bahwa performa model meningkat seiring dengan peningkatan nilai *C*, tetapi waktu pelatihan yang dibutuhkan bertambah secara signifikan. Misalnya, saat parameter $C = 10$, waktu yang dibutuhkan mencapai 5,4 menit per *fold*. Setelah semua kombinasi diuji, parameter terbaik pada pembagian rasio 80:20 diperoleh pada $C = 0.01$, *gamma= scale* dan kernel linier, yang memberikan

keseimbangan baik antara performa dan efisiensi waktu pelatihan. Saat proses pelatihan dan optimasi selesai, model disimpan dalam bentuk file `.joblib` agar dapat digunakan kembali tanpa perlu melakukan pelatihan ulang saat ingin melakukan prediksi.

3. Pembagian rasio 90:10

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END C=0.001, gamma=scale, kernel=linear;, score=0.344
total time=  23.9s
[CV 2/5] END C=0.001, gamma=scale, kernel=linear;, score=0.351
total time=  23.6s
[CV 3/5] END C=0.001, gamma=scale, kernel=linear;, score=0.340
total time=  23.7s
.......
[CV 5/5] END ...C=10, gamma=auto, kernel=linear;, score=0.400
total time= 5.4min
Parameter terbaik yang ditemukan: {'C': 0.1, gamma=scale,
'kernel': 'linear'}
```

Gambar 4.55: Proses Pelatihan dan Optimizer Model SVM dengan Kernel Linier Rasio 90:10

Gambar 4.55 menunjukkan bahwa model diuji dengan *5-fold cross-validation* untuk setiap kombinasi dari 10 kandidat parameter, sehingga total dilakukan sebanyak 50 eksperimen. Parameter yang diuji mencakup *C* (*regularization parameter*), *gamma*, dan kernel. Setiap parameter diuji lima kali, dengan skor evaluasi dari masing-masing *fold* menunjukkan bahwa performa model meningkat seiring dengan peningkatan nilai *C*, tetapi waktu pelatihan yang dibutuhkan bertambah secara signifikan.

Misalnya, pada parameter C = 10, waktu yang dibutuhkan mencapai 5,4 menit per *fold*. Setelah semua kombinasi diuji, parameter terbaik pada pembagian rasio 90:10 diperoleh pada C = 0.1, *gamma*= *scale* dan kernel linier, yang memberikan keseimbangan baik antara performa dan efisiensi waktu pelatihan. Selain itu, setelah proses pelatihan dan optimasi selesai, model disimpan dalam bentuk file `.joblib`,sehingga dapat digunakan kembali tanpa perlu melakukan pelatihan ulang saat ingin melakukan prediksi.

- Kernel RBF *(Radial Basis Function)*

```
# Hyperparameter tuning dengan GridSearchCV
#untuk mencari C dan gamma
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000,
    1000, 5000, 10000, 20000],
    'gamma': ['scale', 'auto'],
    'kernel': ['rbf']
}
grid_search = GridSearchCV(
    SVC(tol=nilai_tol, probability=True),
    param_grid,
    cv=5,
    refit=True,
    verbose=3
)
grid_search.fit(fitur_train_scaled, label_encoded_train)

best_params = grid_search.best_params_
print(f'Parameter terbaik yang ditemukan:
{best_params}', file=f)
```

Gambar 4.56: *Source Code* Pelatihan Model SVM dengan Kernel RBF

Grid Search CV yang ditunjukkan pada Gambar 4.56 akan memilih kombinasi parameter yang terbaik.

Berikut ini merupakan tampilan proses pelatihan model beserta optimasi parameter dengan masing - masing pembagian 70:30, 80:20, dan 90:10.

1. Pembagian rasio 70:30

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[CV 1/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.337
total time=  44.2s
[CV 2/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.333
total time=  45.8s
.......
[CV 4/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.624
total time=  42.3s
[CV 5/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.638
total time=  39.8s
Parameter terbaik yang ditemukan: {'C': 10, 'gamma': 'auto',
'kernel': 'rbf'}
```

Gambar 4.57: Proses Pelatihan dan Optimizer Model SVM dengan Kernel RBF Rasio 70:30

Gambar 4.57 menunjukkan bahwa model SVM diuji dengan *5-fold cross-validation* untuk setiap kombinasi dari 14 kandidat parameter, sehingga total eksperimen yang dilakukan berkisar 70 kali. Parameter yang diuji mencakup *C* (*regularization parameter*), *gamma*, dan kernel. Hasil evaluasi menunjukkan bahwa nilai *C* = 0.001 memberikan skor paling rendah, yaitu 0.333. Sementara itu, parameter dengan skor tertinggi diperoleh pada *C* = 1000 dan *gamma=auto*, dengan nilai sebesar 0.638, yang menunjukkan peningkatan performa model. Selama proses evaluasi, parameter terbaik pada pembagian rasio 70:30 diperoleh pada *C* = 10, *gamma=scale*, dan kernel RBF, yang

memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan *overfitting*. Setelah proses pelatihan dan optimasi selesai, model akan disimpan dalam bentuk file `.joblib` sehingga dapat digunakan kembali tanpa perlu melakukan pelatihan ulang saat melakukan prediksi.

2. Pembagian rasio 80:20

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[CV 1/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.342
total time=  57.0s
[CV 2/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.336
total time=  58.2s
[CV 3/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.348
total time=  56.5s
.......
[CV 3/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.665
total time=  46.8s
[CV 4/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.632
total time=  45.1s
[CV 5/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.661
total time=  45.2s
Parameter terbaik yang ditemukan: {'C': 100, 'gamma': 'auto',
'kernel': 'rbf'
```

Gambar 4.58: Proses Pelatihan dan Optimizer Model SVM dengan Kernel RBF Rasio 80:20

Gambar 4.58 menunjukkan bahwa parameter terbaik pada pembagian rasio 80:20 yang diperoleh *C* = 100 dan *gamma*= *auto*, kernel= RBF yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib`

dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

3. Pembagian rasio 90:10

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
[CV 1/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.350
total time= 1.2min
[CV 2/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.350
total time= 1.1min
[CV 3/5] END ..C=0.001, gamma=scale, kernel=rbf;, score=0.329
total time= 1.1min
.......
[CV 4/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.650
total time=  49.4s
[CV 5/5] END ....C=1000, gamma=auto, kernel=rbf;, score=0.668
total time=  50.7s
Parameter terbaik yang ditemukan: {'C': 100, 'gamma': 'scale',
'kernel': 'rbf'}
```

Gambar 4.59: Proses Pelatihan dan Optimizer Model SVM dengan Kernel RBF Rasio 90:10

Gambar 4.59 menunjukkan bahwa parameter terbaik pada pembagian rasio 90:10 yang diperoleh *C* = 100 dan *gamma*= *scale*, kernel= RBF yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

- Kernel *Polynomial*

```
# Hyperparameter tuning dengan GridSearchCV
untuk kernel polinomial
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
            'degree': [2, 3, 4],
            'gamma': ['scale', 'auto'],
            'kernel': ['poly']
    }

    grid_search = GridSearchCV(SVC(tol=nilai_tol,
    probability=True), param_grid, cv=5, refit=True,verbose=3)
    grid_search.fit(fitur_train_scaled, label_encoded_train)

    best_params = grid_search.best_params_
    print(f'Parameter terbaik yang ditemukan: {best_params}',
    file=f)
```

Gambar 4.60: *Source Code* Pelatihan Model SVM dengan Kernel *Polynomial*l

Grid Search CV yang ditunjukkan pada Gambar 4.60 akan memilih kombinasi parameter yang terbaik. Berikut ini merupakan tampilan proses pelatihan model beserta optimasi parameter dengan masing - masing pembagian 70:30, 80:20, dan 90:10.

1. Pembagian rasio 70:30

   Gambar 4.61 menunjukkan bahwa parameter terbaik pada pembagian rasio 70:30 yang diperoleh *C* = 10 ,*gamma*= *scale*, *degree*=3, dan kernel= *Polynomial* yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat

digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
[CV 1/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.281 total time=  26.8s
[CV 2/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.276 total time=  26.8s
[CV 3/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.277 total time=  26.8s
.......
[CV 4/5] END C=1000, degree=4, gamma=auto, kernel=poly;,
score=0.579 total time=  32.0s
[CV 5/5] END C=1000, degree=4, gamma=auto, kernel=poly;,
score=0.558 total time=  33.4s
Parameter terbaik yang ditemukan: {'C': 10,
'degree': 3, 'gamma': 'scale', 'kernel': 'poly'}
```

Gambar 4.61: Proses Pelatihan Model dan Optimizer SVM dengan Kernel *Polynomial* Rasio 70:30

2. Pembagian rasio 80:20

   Gambar 4.62 menunjukkan bahwa parameter terbaik pada pembagian rasio 80:20 yang diperoleh *C* = 100, *degree*=3 dan *gamma*= *scale*, kernel= *polynomial* yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
[CV 1/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.284 total time=  41.5s
[CV 2/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.273 total time=  42.5s
[CV 3/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.270 total time=  43.1s
.......
[CV 4/5] END C=1000, degree=4, gamma=auto, kernel=poly;,
score=0.585 total time=  48.2s
[CV 5/5] END C=1000, degree=4, gamma=auto, kernel=poly;,
score=0.591 total time=  49.4s
Parameter terbaik yang ditemukan: {'C': 100,
'degree': 3, 'gamma': 'scale', 'kernel': 'poly'}
```

Gambar 4.62: Proses Pelatihan Model dan Optimizer SVM dengan Kernel *Polynomial* dengan Rasio 80:20

3. Pembagian rasio 90:10

   Gambar 4.63 menunjukkan bahwa parameter terbaik pada pembagian rasio 90:10 yang diperoleh *C* = 10, *degree*=3, dan *gamma*= *scale*, kernel= *polynomial* yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
[CV 1/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.298 total time=  47.0s
[CV 2/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.284 total time=  49.2s
[CV 3/5] END C=0.001, degree=2, gamma=scale, kernel=poly;,
score=0.276 total time=  48.3s
.......
[CV 4/5] END C=1000, degree=4, gamma=auto, kernel=poly;,
score=0.591 total time=  53.7s
[CV 5/5] END C=1000, degree=4, gamma=auto, kernel=poly;,
score=0.604 total time=  54.3s
Parameter terbaik yang ditemukan: {'C': 10,
'degree': 3, 'gamma': 'scale', 'kernel': 'poly'}
```

Gambar 4.63: Proses Pelatihan Model dan Optimizer SVM dengan Kernel *Polynomial* dengan Rasio 90:10

- Kernel *Sigmoid*

```
# Hyperparameter tuning dengan
GridSearchCV untuk kernel sigmoid
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10],
        'gamma': ['scale', 'auto'],
        'coef0': [0.0, 0.1, 0.5, 1.0],
        'kernel': ['sigmoid']
    }
    grid_search = GridSearchCV(SVC(tol=nilai_tol
    , probability=True),
    param_grid, cv=5, refit=True, verbose=3)
    grid_search.fit(fitur_train_scaled, label_encoded_train)

    # Menyimpan model terbaik
    best_params = grid_search.best_params_
    print(f"Parameter terbaik yang ditemukan: {best_params}",
    file=f)
```

Gambar 4.64: *Source Code* Pelatihan Model SVM dengan Kernel *Sigmoid*

Grid Search CV yang ditunjukkan pada Gambar 4.64 akan memilih kombinasi parameter yang terbaik.

Berikut ini merupakan tampilan proses pelatihan model beserta optimasi parameter dengan masing - masing pembagian 70:30, 80:20, dan 90:10.

1. Pembagian rasio 70:30

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[CV 1/5] END C=0.001, coef0=0.0, gamma=scale,
kernel=sigmoid;, score=0.217 total time=  36.9s
[CV 2/5] END C=0.001, coef0=0.0, gamma=scale,
kernel=sigmoid;, score=0.203 total time=  37.3s
[CV 3/5] END C=0.001, coef0=0.0, gamma=scale,
kernel=sigmoid;, score=0.213 total time=  37.3s
.......
[CV 3/5] END C=10, coef0=1.0, gamma=auto, kernel=sigmoid;,
score=0.079 total time=  23.2s
[CV 4/5] END C=10, coef0=1.0, gamma=auto, kernel=sigmoid;,
score=0.089 total time=  22.4s
[CV 5/5] END C=10, coef0=1.0, gamma=auto, kernel=sigmoid;,
score=0.092 total time=  23.2s
Parameter terbaik yang ditemukan: {'C': 1, 'coef':0.0,
'gamma':'auto', 'kernel': 'sigmoid'
```

Gambar 4.65: Proses Pelatihan Model dan Optimizer SVM dengan Kernel *Sigmoid* Rasio 70:30

Gambar 4.65 menunjukkan bahwa parameter terbaik pada pembagian rasio 70:30 yang diperoleh *C* = 1 , *coef* = 0, *gamma*= *auto*, dan kernel= sigmoid yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

2. Pembagian rasio 80:20

```
Fitting 5 folds for each of 40 candidates,totalling 200 fits
[CV 1/5] END C=0.001, coef0=0.0, gamma=scale,
kernel=sigmoid;, score=0.220 total time=  55.3s
[CV 2/5] END C=0.001, coef0=0.0, gamma=scale,
kernel=sigmoid;, score=0.207 total time=  55.5s
[CV 3/5] END C=0.001, coef0=0.0, gamma=scale,
kernel=sigmoid;, score=0.219 total time=  55.3s
.......
[CV 4/5] END ..C=10, coef=1.0, gamma=auto, kernel=sigmoid;,
score=0.190 total time=  27.6s
[CV 5/5] END ..C=10, coef=1.0, gamma=auto, kernel=sigmoid;,
score=0.205 total time=  28.0s
Parameter terbaik yang ditemukan: {'C': 1, 'coef':0.0,
'gamma': 'auto', 'kernel': 'sigmoid'}
```

Gambar 4.66: Proses Pelatihan Model dan Optimizer SVM dengan Kernel *Sigmoid* dengan Rasio 80:20

Gambar 4.66 menunjukkan bahwa parameter terbaik pada pembagian rasio 80:20 yang diperoleh *C* = 1, *coef* = 0, *gamma*= *auto*, dan kernel= *Sigmoid* yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatihan ulang saat melakukan proses prediksi.

3. Pembagian rasio 90:10

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END C=0.001, coef0=0.0, gamma=scale, kernel=sigmoid;,
score=0.211 total time= 1.2min
.......
[CV 5/5] END ..C=10, coef0=1.0, gamma=auto, kernel=sigmoid;,
score=0.198 total time=  31.8s
Parameter terbaik yang ditemukan: {'C': 0.1, 'coef':0.0,
'gamma': 'scale', 'kernel': 'sigmoid'}
```

Gambar 4.67: Proses Pelatihan Model dan Optimizer SVM dengan Kernel *Sigmoid* dengan Rasio 90:10

Gambar 4.67 menunjukkan bahwa parameter terbaik pada pembagian rasio 90:10 yang diperoleh *C* = 0,1, *coef* = 0 dan *gamma= scale*, kernel= *sigmoid* yang dapat memberikan keseimbangan terbaik antara akurasi dan generalisasi model tanpa menyebabkan terjadinya *overfitting*. Pada saat proses pelatihan dan optimasi selesai dijalankan, model akan disimpan dalam bentuk file `.joblib` dengan tujuan model dapat digunakan kembali tanpa melakukan pelatiihan ulang saat melakukan proses prediksi.

- **Evaluasi Model**

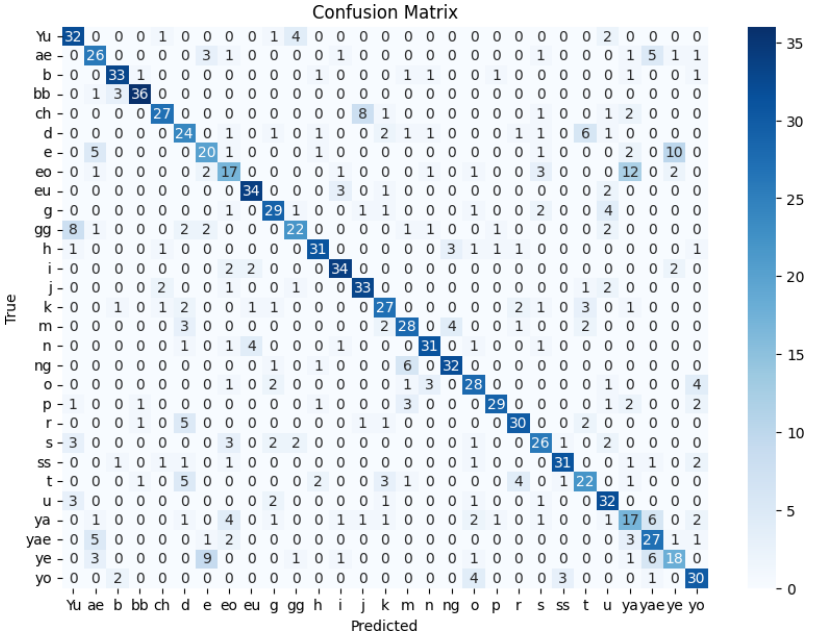
Setelah melakukan pelatihan dan optimizer model, selanjutnya adalah melakukan evaluasi model dengan melakukan pengujian terhadap tingkat *accuracy*, *recall*, *precission* dan *F1-score*. Pengujian ini dilakukan dengan menggunakan *confusion matrix* sesuai dengan Persamaan 2.37 sampai 2.40. Pada Gambar 4.68 menunjukkan beberapa fungsi *surce code* yang digunakan dalam evaluasi

model, antara lainnya adalah `ypredict` yang berfungsi untuk menyimpan hasil prediksi model untuk setiap sampel dalam data uji, `confusionmatrix(actual, predicted)` berfungsi untuk membandingkan label asli `(labelencodedtest)` dengan prediksi *(ypred)*, `print(..., file=f)` digunakan untuk mencetak teks dan *confusion matrix* ke dalam *file* (f) alih-alih ke konsol, `sns.heatmap()` digunakan untuk membuat `heatmap` dari *confusion matrix* menggunakan `Seaborn`, `annot=True` digunakan untuk menampilkan angka didalam `heapmap`, `fmt='d'` merupakan format angka dalam bentuk *integer*, *xticklabels=encoderlabel.classes* merupakan label sumbu *x* pada nama kelas, `targetnames=encoderlabel.classes` merupakan nama kelas yang digunakan dalam *report*. Berikut ini merupakan *source code* evaluasi model SVM.

```
# Evaluasi model pada data testing
y_pred = best_model.predict(fitur_test_scaled)
conf_matrix = confusion_matrix(label_encoded_test, y_pred)

print(f"\n===== Confusion Matrix =====", file=f)
print(conf_matrix, file=f)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=encoder_label.classes_,
yticklabels=encoder_label.classes_)
 class_report = classification_report
(label_encoded_test, y_pred,
target_names=encoder_label.classes_)
print(f"\n===== Laporan Klasifikasi =====", file=f)
print(class report, file=f)
```

Gambar 4.68: *Source Code* Evaluasi Model

Dalam skenario pada pengujian ini, sampel parameter yang ditampilkan adalah rasio 90% :10%, yang sudah diaugmentasi sebesar 10440 data latih dan 1160 data uji. Pengujian ini dilakukan dengan menggunakan parameter yaitu dengan kernel = RBF, *C*= 100, *gamma*= scale, kernel= rbf. Berikut ini merupakan hasil pengujian model dalam bentuk *confusion matrix*.



Gambar 4.69: *Confusion Matrix dengan Rasio 90:10*

Pada Gambar 4.69 menunjukkan bahwa *confusion matrix* mempunyai adanya hubungan antara *true label* dan *predicted label*. Berdasarkan *confusion matrix* tersebut, perhitungan manual nilai *accuracy*, *recall*, *precision*, dan *F1-Score* dengan menggunakan sampel kelas "Yu" sebagai

berikut:

- *Accuracy score*

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$
$$= \frac{32 + 1104}{32 + 1104 + 16 + 8} \times 100\%$$

$$Accuracy = 97,93\%$$

- *Recall*

$$Recall = \frac{TP}{TP + FN} \times 100\%$$
$$= \frac{32}{32 + 8} \times 100\%$$

$$Recall = 80\%$$

- *Precision*

$$Precision = \frac{TP}{TP + FP} \times 100\%$$
$$= \frac{32}{32 + 16} \times 100\%$$

$$Precision = 66,67\%$$

- *F1-Score*

$$F1\text{-}Score = 2 \times \left(\frac{Recall \times Precision}{Recall + Precision} \times 100\%\right)$$

$$= 2 \times \left(\frac{80 \times 66,67}{80 + 66,67}\right) \times 100\%$$

$$F1\text{-}Score = 72,72\%$$

Setelah melakukan perhitungan manual dengan sampel salah satu karakter tulisan Korea *(Hangeul)* yaitu kelas "Yu". Berikut merupakan tabel hasil dari evaluasi model dengan pembagian 90%:10%.

Tabel 4.51: Hasil Evaluasi Model dengan Pembagian 90:10

Kelas	Precision	Recall	F1-score	Support
Yu	0.67	0.80	0.73	40
ae	0.60	0.65	0.63	40
b	0.82	0.82	0.82	40
bb	0.90	0.90	0.90	40
ch	0.82	0.68	0.74	40
d	0.55	0.60	0.57	40
e	0.54	0.50	0.52	40
eo	0.47	0.42	0.45	40
eu	0.83	0.85	0.84	40
g	0.72	0.72	0.72	40
gg	0.71	0.55	0.62	40
h	0.82	0.78	0.79	40
i	0.81	0.85	0.83	40
j	0.75	0.82	0.79	40
k	0.68	0.68	0.68	40
m	0.60	0.70	0.68	40
n	0.82	0.78	0.79	40
ng	0.82	0.80	0.81	40
o	0.67	0.70	0.68	40
p	0.88	0.72	0.79	40
r	0.77	0.75	0.76	40
s	0.67	0.65	0.66	40
ss	0.86	0.78	0.82	40
t	0.61	0.55	0.58	40
u	0.63	0.80	0.70	40
ya	0.39	0.42	0.40	40
yae	0.59	0.68	0.63	40
ye	0.53	0.45	0.49	40
yo	0.68	0.75	0.71	40
Akurasi			**0.69**	
Macro Avg	0.70	0.69	0.69	1160
Weighted Avg	0.70	0.69	0.69	1160

Berdasarkan hasil evaluasi model pada Tabel 4.51 dan Gambar 4.69 menunjukkan bahwa model yang dilatih dengan pembagian data 90%:10% menghasilkan nilai akurasi keseluruhan sebesar 69 %, *recall* sebesar 69%, *precision* sebesar 70%, dan *f1-score* sebesar 69%. Hal ini menunjukkan bahwa model yang di latih dapat mengidentifikasi sebagian besar karakter tulisan Korea *(Hangeul)* dengan cukup baik. Akan tetapi, terdapat beberapa kelas yang mempunyai nilai *recall* dan *precision* yang rendah, sehingga dapat menyebabkan kesalahan dalam mengidentifikasi karakter yang sering tertukar.

Salah satu kelas yang mempunyai performa terendah adalah kelas "ya", yang mempunyai nilai *precision* sebesar 39%, *recall* sebesar 42%, dan *F1-Score* sebesar 40%, yang menunjukkan bahwa terdapat banyak sampel dari kelas "ya" diklasifikasikan secara salah ke kelas lain. Selain itu, kelas "eo" mempunyai nilai *precision* rendah sebesar 47% dan *recall* sebesar 42%, yang menunjukkan bahwa kesulitan model dalam mengenali karakter ini secara akurat.

Model yang menujukkan dengan performa terbaik dalam mengenali karakter tulisan Korea *Hangeul* adalah kelas "bb", dengan nilai *precision*, *recall* sebesar 90%, dan *F1-Score* sebesar 90%. Hal ini menunjukkan bahwa model dapat mengidentifikasi semua sampel pada kelas ini dengan tingkat kesalahan yang sangat rendah. Selain itu, kelas "eu" mempunyai performa tinggi dengan nilai *F1-score* sebesar 84%, *precision* sebesar 83%, *recall* sebesar 85%, yang menunjukkan bahwa model dapat mengenali karakter kelas "eu" dengan baik.

Berikut ini merupakan hasil evaluasi model secara keseluruhan dengan pembagian 70:30, 80:20, dan 90:10 dapat dilihat pada Tabel 4.52 sebagai berikut.

Tabel 4.52: Hasil Seluruh Pengujian

Split Data		Kernel	C	Gamma	Akurasi (%)	Precision (%)	Recall (%)	F1-Score (%)
Training	Testing							
70	30	RBF	10	*auto*	65	65	65	65
		Linier	0,01	*scale*	43	43	43	43
		Polynomial	10	*scale*	63	64	63	63
		Sigmoid	1	*auto*	29	29	29	28
80	20	RBF	100	*auto*	68	68	68	68
		Linier	0,1	*scale*	44	46	44	44
		Polynomial	100	scale	65	66	65	65
		Sigmoid	1	*auto*	27	27	27	27
90	10	RBF	100	*scale*	69	70	69	69
		Linier	0,1	*scale*	46	47	46	46
		Polynomial	10	*scale*	68	68	68	67
		Sigmoid	0,1	*scale*	33	34	33	32

Pada Tabel 4.52 menunjukkan bahwa hasil pengujian dari berbagai konfigurasi model SVM berdasarkan berbagai rasio pembagian data, jenis kernel, serta parameter "*C*" dan "*Gamma*". Dalam penelitian ini, data dibagi menjadi 3 skenario, antara lainnya 70%:30% , 80%:20% dan 90%:10%.

Dari hasil pengujian, kernel RBF *(Radial Basis Function)* mempunyai performa terbaik diantara kernel yang lainnya, terutama pada rasio 90%:10% dengan parameter *C*=100 dan *Gamma=scale* yang menghasilkann akurasi tertinggi sebesar 69%. Selain itu, kernel *polynomial* mempunyai performa yang cukup baik dengan akurasi mencapai 68% pada skenario yang sama. Namun, terdapat kernel linier mempunyai akurasi lebih rendah dibandingkan RBF dan *Polynomial*, tetapi mempunyai performa lebih baik daripada kernel *sigmoid*, yang mempunyai performa yang sangat buruk dengan akurasi hanya berkisar antara 27% - 33%.

Hal ini menunjukkan bahwa pemilihan kernel yang tepat dapat berpengaruh sangat besar terhadap kinerja model SVM. Sehingga, terbukti bahwa kernel RBF pada SVM mempunyai performa baik dalam menangani pola data pada kasus pengenalan karakter tulisan Korea *(Hangeul)*.

H. Perbandingan *Convolutional Neural Network*(CNN) dan *Support Vector Machine*(SVM)

Perbandingan hasil klasifikasi SVM *(Support Vector Machine)* dan CNN *(Convolutional Neural Network)* ditunjukkan pada Tabel 4.53.

Tabel 4.53: Perbandingan CNN dan SVM dari Hasil yang Terbaik

Uuran Evaluasi	SVM	CNN
Rasio	90%:10%	90%:10%
Akurasi	69%	86%
Precision	70 %	86,57%
Recall	69%	86,21%
F1-Score	69%	86,08%

Berdasarkan Tabel 4.53 menunjukkan bahwa Metode CNN *(Convolutional Neural Network)* mendapatkan akurasi sebesar 86%, *precision* sebesar 86,57%, *Recall* sebesar 86,21%, dan *F1-Score* sebesar 86,08%. Sedangkan, metode SVM *(Support Vector Machine)* memperoleh akurasi sebesar 69%, *precision* sebesar 70%, *Recall* sebesar 69%, dan *F1-Score* sebesar 69%. Dari kedua metode tersebut metode yang paling unggul adalah metode CNN *(Convolutional Neural Network)* untuk nilai akurasi, *precision*, *recall*, dan *f1-score* dengan mempunyai selisih akurasi sebesar 17% dari metode SVM *(Support Vector Machine)*, hal ini dapat

dinyatakan bahwa metode CNN *(Convolutional Neural Network)* yang lebih akurat dan unggul dalam kasus pengenalan karakter tulisan Korea *(Hangeul)*.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan pada hasil pengujian, yang diperoleh mengenai proses klasifikasi pada metode CNN *(Convolutional Neural Network)* dan SVM *(Support Vector Machine)* dari pembahasan yang telah dilakukan, maka dapat disimpulkan bahwa:

1. Pengenalan karakter tulisan Korea *(Hangeul)* menggunakan metode CNN *(Convolutional Neural Network)* mempunyai nilai akurasi tertinggi 86%, nilai *precision* sebesar 86,57%, *recall* sebesar 86,21%, dan *F1-Score* sebesar 86,08 % dengan rasio pembagian 90%:10%. Dengan demikian, rasio pembagian data 90%:10% mempunyai hasil akurasi lebih tinggi dibandingkan pada rasio 70%:30% dan 80%:20%. Pengujian dilakukan dengan parameter *epoch* sebanyak 15, *learning rate* sebesar 0,001, dan optimizenya adalah RMSProp.

2. Pengenalan karakter tulisan Korea *(Hangeul)* menggunakan metode SVM *(Support Vector Machine)* mempunyai akurasi tertinggi 69%, *precision* sebesar 70%, *recall* sebesar 69%, *F1-Score* sebesar 69% dan kernel RBF dengan skenario pembagian data 90%:10%. Dengan demikian, rasio pembagian data 90%:10% dengan kernel RBF *(Radial Basis Function)* mempunyai akurasi lebih tinggi dibandingkan pada rasio 70%:30% dan 80%:20% dengan pengujian kernel lainnya. Selain itu, parameter yang terbaik yang diperoleh

pembagian 90%:10% dengan kernel RBF adalah parameter *C* =100 dan *Gamma* = *scale* pada kasus pengenalan karakter tulisan Korea *(Hangeul)*.

3. Metode CNN *(Convolutional Neural Network)* merupakan metode paling unggul daripada metode SVM *(Support Vector Machine)* dengan akurasinya selisih 17%. Dengan demikian, metode CNN *(Convolutional Neural Network)* mempunyai akurasi lebih akurat dibandingkan dengan SVM *(Support Vector Machine* dalam pengenalan karakter tulisan Korea *(Hangeul)*.

B. Saran

Berdasarkan hasil penelitian yang sudah dilakukan, terdapat beberapa saran:

1. Kualitas citra karakter tulisan Korea *(Hangeul)* dapat ditingkatkan dengan menggunakan citra beresolusi tinggi.
2. Metode CNN *(Convolutional Neural Network)* dapat ditingkatkan dengan memanfaatkan berbagai arsitektur, seperti AlexNet, YOLO, VGG-16, VGG-19, Xception, EfficientNet, ResNet101, dll.
3. Klasifikasi menggunakan metode SVM dapat ditingkatkan dengan ekstraksi fitur lainnya seperti *Haralick Features*, kombinasi *SURF* dengan *Histogram Of Oriented*, kombinasi *Hu Moments (Moment Invariants)* dengan *edge detection*, dll.
4. Pengenalan karakter tulisan Korea *(Hangeul)* dapat menggunakan teknik *deep learning* lainnya seperti *Random*

Multimodel Deep Learning (RMDL), *Graph Neural Networks (GNN)*, dan *Self-Supervised Learning*.

5. Pengenalan karakter tulisan Korea *(Hangeul)* dapat menggunakan teknik *machine learning* lainnya seperti SSD *(Single Shot Multibox Detector)*, YOLO *(You Only Look Once)*, dll.

DAFTAR PUSTAKA

Adelia, R., Khairunisa, N., S Zulfiqri, R. (2024). Implementasi *Convolutional Neural Network* (CNN) Dalam Mendeteksi Sampah Organik, Plastik, dan Kertas. *JUTIM (Jurnal Teknik Informatika Musirawas)*, 9(1), 29 -37.

Akbar, L. N., S Utaminingrum, F. (2019). Klasifikasi Golongan Kendaraan Berdasarkan Fitur *Histogram of Oriented Gradients* (HOG) Menggunakan metode *K-Nearest Neighbors* (K-NN) Berbasis Raspberry PI 3. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 3(9), 8531-8538.

Amri, I., Nurcahyo, A. C., Cahyaningtyas, C., S Salfarini, E. M. (2024). Implementasi algoritma Convolutional Neural Network untuk menerjemahkan bahasa isyarat. *Kohesi: Jurnal Sains dan Teknologi*, 2(9), 70-87.

Awalullaili, F. O., Inspriyanti, D., S Widiharih, T. (2022). Klasifikasi Penyakit Hipertensi Menggunakan Metode SVM Grid Search dan SVM Genetic Algorithm (GA). *Jurnal Gaussian*, 11(4), 488-498.

Ayuni,D. P., Jasril,Irsyad, M., Yanto, F., S Sanjaya, S. (2023). Augmentasi Data Pada Implementasi *Convolutional Neural Network* Arsitektur Efficientnet-B3 untuk Klasifikasi Penyakit Padi. *Zonasi Jurnal Sistem Informasi*, 5(2), 239-249.

Azmi, B. N., Hermawan, A., S Avianto, D. (2023). Analisis pengaruh komposisi data training dan data testing pada penggunaan PCA dan algoritma decision tree untuk

klasifikasi penderita penyakit liver. *JTIM: Jurnal Teknologi Informasi dan Multimedia*, 4(4), 281–290.

Balqis, Z. (2012). *Bahasa Korea Super Mudah*. Yogyakarta: Familia Publisher.

Beysolow, T. (2017). *Introduction to Deep Learning Using R: A Step-by-Step Guide to Learning and Implementing Deep Learning Models Using R*. California: Apress Media.

Bi, Y., Xue, B., S Zhang, M. (2021). *Genetic Programming for Image Classification: An Automated Approach to Feature Learning*. Cham, Switzerland: Springer.

Bishop, C., Heckerman, D., Jordan, M., S Kearns, M. (2002). *Learning with Kernels*. Chambridge: MIT Press.

Boser, B., Guyon, I., S Vapnik, V. (1992). *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT, 144–152.

Brownlee, J. (2018). *Master Machine Learning: Discover How They Work and Implement Them From Scratch*. Australia: Jason Brownlee.

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*.

Chandra, M. M., S Yoannita. (2023). Klasifikasi jenis bunga menggunakan metode SVM berdasarkan citra dengan fitur HSV. *JIST*, 4(2), 255-264.

Campbell, C., S Ying, Y. (2011). *Learning with Support Vector Machines*. USA: Morgan S Claypool.

Chaki, J., S Dey, N. (2019). *A Beginner's Guide to Image Preprocessing Techniques*. United Kingdom: Taylor and Francis Group.

Darmanto, H. (2019). Pengenalan Spesies Ikan Berdasarkan Kontur Otolith Menggunakan *Convolutional Neural Network*. *JOINED Journal*, 2(1), 41-59.

Deng, N., Tian, Y., S Zhang, C. (2013). *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions*. Minnesota: CRC Press.

Devita,D., Randy, C. W.,S Widodo, A. W. (2019). Ekstraksi Ciri Untuk Klasifikasi Gender Berbasis Citra Wajah Menggunakan Metode Histogram of Oriented Gradients. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 3(8), 8002-8011.

Dewi, D. D., Qisthi, N., Lestari, S. S. S., S Putri, Z. H. S. (2023). Perbandingan metode Neural Network dan Support Vector Machine dalam klasifikasi diagnosa penyakit diabetes. *Cerdika: Jurnal Ilmiah Indonesia*, 3(9), 828-839.

Donuk, K., Ari, A., ÖZDEMİR, M. F., S Hambay, D. (2023). Deep Feature Selection for Facial Emotion Recognition Based on BPSO and SVM. *JOURNAL of POLYTECHNIC*, 26(1), 131-142.

Dutt, S., Chandramouli, S., S Dass, A. K. (2018). *Machine Learning*. Indian: Pearson.

Efendi, E., Aviatri, F., S Sipahutar, I. M. (2023). Peran Siaran Radio, Televisi, dan Multimedia (Internet) dalam Pengembangan

Dakwah. *Dawatuna: Journal of Communication and Islamic Broadcasting*, 3(4), 780-793.

Elwirehardja, G. N., Suparyanto, T., S Pardamean, B. (2023). *Pengenalan Konsep Machine Learning untuk Pemula*. Yogyakarta: INSTIPER PRESS.

Fadillah, R. Z., Irawan, A., S Susanty, M. (2024). Data augmentasi untuk mengatasi keterbatasan data pada model penerjemah bahasa isyarat Indonesia (BISINDO). *Jurnal Informatika*, 8(2), 208–214.

Fajarendra, Y. I., Fauzan, Y. R., S 'Uyun, S. (2024). Deteksi Ikan Molly Menggunakan Metode BLOB Dan HSV Pada Peternakan Ikan CSA Sidoarjo. *JATI: Jurnal Mahasiswa Teknik Informatika*, 8(4), 7754-7761. Rafael C. Gonzalez dan Richard E. Woods

Gonzales, R. C. S Woods, R. E. (2018). *Digital Image Processing*. New York: Pearson.

Goodfellow, I., Bengio, Y., S Courville, A. (2016). *Deep Learning*. Cambridge: MIT Press.

Hanif, A. R., Nasrullah, E., S Setyawan, F. X. A. (2023). Deteksi karakter plat nomor kendaraan dengan menggunakan metode Optical Character Recognition (OCR). *JITET: Jurnal Informatika dan Teknik Elektro Terapan*, 1(11), 109-117.

Harahap, M., Aguatia, E. S., Lubis, M. M., Apriand, S Anggara, A. (2020). Deteksi objek manusia pada image dengan metode thinning berdasarkan local maxima. *Jurnal Teknik Informatika dan Sistem Informasi*, 7(3), 617–623.

Heryadi, Y., S Wahyono, T. (2020). *Machine Learning: Konsep dan Implementasi*. Yogyakarta: Gava Media.

Heryadi, Y., S Wahyono, T. (2021). *Dasar-Dasar Deep Learning dan Implementasinya*. Yogyakarta: Gava Media.

Heryanto, W. A., Artama, Kurniawan, M. W. S., S Gunadi, G. A.(2020). Segmentasi Warna dengan Metode *Thresholding*. *Wahana Matematika dan Sains: Jurnal Matematika, Sains, dan Pembelajarannya*, 14(1),54-64.

Hwa, A. K., Yong, C. H., Adinda, R. N., Agung, S., S Hutagalung, F. (2013). *Bahasa Korea Terpadu Untuk Orang Indonesia Jilid 1*. Korea: Yu Hyun-seok.

Irfani, M. H., S Gasim. (2024). Segmentasi teks pada citra tulisan tangan kalimat menggunakan metode median filtering dan Otsu. *Teknosains: Media Informasi dan Teknologi*, 18(1), 88–97.

Isnaeni, Sudarmin,S Rais, Z. (2022). Analisis *Support Vector Regression* (SVR) dengan *Kernel Radial Basis Function* (RBF) untuk Memprediksi Laju Inflasi Di Indonesia. *Journal of Statistics and Its Application on Teaching and Research* , 4(1), 30-38.

JAMESCASIA. (2023). Handwritten Hangul Characters. Retrieved from `https://www.kaggle.com/jamesasia/handwritten-hangul-character`. Diakses 07 Juni 2024.

Julianto, A., Sunyoto, A.,S Wibowo, F. W. (2022). Optimasi *Hyperparameter Convolutional Neural Network* untuk

Klasifikasi Penyakit Tanaman Padi. *TEKNIMEDIA*, 3(2), 98-105.

Ketkar, N., S Moolayil, J. (2021). *Deep Learning With Python*. New York: Apress Media LLC.

Lederer, J. (2021). *Activation Functions in Artificial Neural Networks: A Systematic Overview*.

Lu, G. (1999). *Multimedia Database Management Systems (Artech House Computing Library)*. California: Artech House.

Munawaroh, K., S Alamsyah. (2022). Performance Comparison of SVM, Naïve Bayes, and KNN Algorithms for Analysis of Public Opinion Sentiment Against COVID-19 Vaccination on Twitter. *Journal of Advances in Information Systems and Technology*, 4(2), 113-125.

Maulani, A. A., Winarno, S., Zeniarja, J., Putri, R. T. E., S Cahyani, A. N. (2024). Comparison of Hyperparameter Optimization Techniques in Hybrid CNN-LSTM Model for Heart Disease Classification. *Sinkron: Jurnal dan Penelitian Teknik Informatika*, 8(1), 455-465.

Mellyssa, W., Misriana, M., Suryati,S., S Milawarni, M. (2020). Perbandingan Penggunaan Metode Otsu Thresholding dan *Adaptive Thresholding* pada Proses Binerisasi Sistem Dokumentasi Buku Tugas Akhir.*Proceeding Seminar Nasional Politeknik Negeri Lhokseumawe*, 4(1), 49-54.

Muchtar, M., S Muchtar, R. A. (2024). Perbandingan Metode KNN dan SVM dalam Klasifikasi Kematangan Buah Mangga

Berdasarkan Citra HSV dan Fitur Statistik. *JITET (Jurnal Informatika dan Teknik Elektro Terapan)*,12(2), 876-884.

Mudhofar, N., S Agustin, S. (2024). Klasifikasi penyakit daun apel menggunakan ekstraksi fitur warna RGB. *Publikasi Teknik Informatika dan Jaringan*, 2(3), 147–156.

Naufal, M. A., S Gasim. (2023). Identifikasi kadar ikan pada pempek menggunakan fitur GLCM dan SVM. *Jurnal Algoritme*, 3(2), 199-211.

Nazar, K., Saeed, Y., Ali, A., Algarni, A. D., Soliman, N. F., Ateya, A. A., Muthanna, M. S. A., S Jamil, F. (2022). Towards Intelligent Zone-Based Content Pre-Caching Approach in VANET for Congestion Control. *Sensors*, 22(1), 1-29.

Nurafiya, A. Y., S Chandra, A. Y. (2024). Analisis Performa Akurasi Klasifikasi Citra Jenis Sayur Salada Menggunakan Arsitektur VGG16. *Jurnal Ilmu Komputer*, 9(1), 33-48.

Nugraha, W., S Sasongko, A. (2022). Hyperparameter Tuning pada Algoritma Klasifikasi dengan *Grid Search*. *SISTEMASI: Jurnal Sistem Informasi*, 11(2), 391–401.

Octariadi, B. C., S Brianorman, Y.(2019). Pengenalan Pola Tanda Tangan Menggunakan Metode *Support Vector Machine*. *Infotech Journal*, 5(2), 16-22.

Oktaviana, N. E., Sari, Y. A., S Indriati(2022). Analisis Sentimen Terhadap Kebijakan Kuliah Daring Selama Pandemi Menggunakan Pendekatan *Lexicon Based Features* dan *Support Vector Machine*. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, 9(2), 357-362.

Pal, S., Trang, L. H., Hieu,V. T., Nguyen, D. D., Vu, D. Q., S Prakash, I.(2024). *Investigation of Support Vector Machines with Different Kernel Functions for Prediction of Compressive Strength of Concrete*. *Journal of Science and Transport Technology*, 4(2), 55-68.

Permana, A. A., Wahyuddin, Santoso, L. W., Wibowo, G. W. N., Wardhani, A. K., Rahmaddeni, Wahidin, A. J., Yuliastuti, G. E., Elisawati, Wijayanti, R. R., S Abdurrasyid. (2022). *Machine Learning*. Padang: PT Global Eksekutif Teknologi.

Prasetyo, E. (2012). *Data Mining: Konsep dan Aplikasi menggunakan MATLAB, 1st ed*. Yogyakarta: Andi Offset.

Prasetyo, E. (2014). *Data Mining: Mengolah data menjadi informasi menggunakan matlab*. Yogyakarta: Andi Offset.

Pratiwi, A. O. C. (2023). Klasifikasi Jenis Anggur Berdasarkan Bentuk Daun Menggunakan *Convolutional Neural Network* Dan *K-Nearest Neighbor*. *Jurnal Ilmiah Teknik Informatika dan Komunikasi*, 3(2), 201-224.

Primartha, R. (2021). *Algoritma Machine Learning*. Bandung: Informatika.

Putra, D. (2010). *Pengolahan Citra Digital*. Yogyakarta: CV Andi Offset.

Putra, A. P. H., Purwandari, E. P., S Andreswari, D. (2020). Identifikasi Pola Iris Mata Menggunakan Metode Support Vector Machine Dengan Ekstraksi Ciri Hue Saturation Value (HSV) Histogram, Gabor Filter dan Wavelet Transform. *Jurnal Rekursif*, 8(1), 23-32.

Putra, I. K. N. P., Dewi, N. P. D. A. S., Mupu, D. N., S Pusparani, D. A. (2024). Identifikasi Tanda Tangan Lansia dengan Metode HOG dan *K-Nearest Neighbors Classifier*. *JURNAL FASILKOM*, 14(1), 95-100.

Putra, F. A. I. A., Sulaksono, A. G., Utomo, L. T.,S Khamdani, A. R. (2023). Klasifikasi Buah dan Sayur menggunakan fitur ekstraksi HOG dan Metode KNN. *JIP (Jurnal Informatika Polinema)*, 10(1), 45–52.

Putra, I. K. N., Dewi, N. P. D. A. S., Mupu, D. N., dkk. (2024). Identifikasi tanda tangan lansia dengan metode HOG dan K-nearest neighbors classifier. *Jurnal Fasilkom*, 14(1), 95–100.

Putra, G. P. S., Fatkhiyah,E., S Ariyana, R. Y. (2024). Analisis Perbandingan Algoritma *Local Binary Patterns Histogram* (LBPH) dan Algoritma *Convolutional Neural Network* (CNN) Pada Sistem Pengenalan Wajah. *Jurnal SCRIPT* , 12(1), 39–48.

Rabbani, S., Safitri, D., Ramadhan, N., Fadillah, A. A., S Anam, M. K. (2023). Perbandingan Evaluasi Kernel SVM untuk Klasifikasi Sentimen dalam Analisis Kenaikan Harga BBM. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, 3(2), 153-160.

Rachmawati, D. W., Rahman, E. T., S Ahyani, H. (2023). *Bahasa Korea*. Bandung: Widina Media Utama.

Rahmadwati, Imran, A. Z., Aswin, M., S Ferdiana, K. (2024).Identifikasi Penyakit Katarak berdasarkan Citra Fundus menggunakan Siamese *Convolutional Neural*

Network. *ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika*, 12(4), 838 - 851.

Rivan, M. E. A., Irsyad, H., Kevin, SNarta, A. T. (2020).Implementasi LDA Pada Fitur HOG Untuk Klasifikasi ASL Menggunakan K-NN. *Jurnal Teknik Informatika dan Sistem Informasi*, 7(2), 214-225.

Rizky, M., S Andarsyah, R. (2023). *Komparasi Performa Model Terhadap Klasifikasi Sinyal MIT-BIH ARRHYTHMIA Database*. Bandung: Buku Pedia.

Rostineu. (2014). *30 Hari Belajar Bahasa Korea Dengan Mudah & Lancar*. Jakarta: TransMedia Pustaka.

Samsuri (2022). Penentuan Kelayakan dan Besaran Pinjaman Pada Koperasi Di Banjarmasin Memanfaatkan *Support Vector Machine* (SVM) Dan Regresi Linier Berganda. *Jurnal Sains Komputer dan Teknologi Informasi*, 4(2), 49-58.

Sari, M., Nurcahyo, A. C., Cahyaningtyas, C., S Salfarini, E. M. (2024). Pengenalan pola aksara Dunging Kalbar menggunakan metode *Learning Vector Quantization (LVQ)*. *Journal of Information Technology*, 4(1), 143-149.

Sakhdiah, M., Salma, A., Permana, D., S Fitria, D. (2024). *Sentiment Analysis Using Support Vector Machine (SVM) of ChatGPT Application Users in Play Store*. *UNP Journal of Statistics and Data Science*, 2(2), 151-158.

Sam, K. (2023). *Ilsang Hangugeo: Cepat jago bahasa Korea dari nol*. Yogyakarta: Anak Hebat Indonesia.

Santony, J. (2020). Ekstraksi objek minutiae pada citra sidik jari menggunakan metode morfologi dan Gabor filter. *Jurnal KomtekInfo*, 7(1), 032–040.

Satrio, M., S Agung, T. W. (2021). Klasifikasi Tanaman Aglaonema Berdasarkan Citra Daun Menggunakan Metode *Convolutional Neural Network (CNN)*. *e-Proceeding of Engineering*, 8(5), 10621-10636.

Sendhilkumar, S., S Geetha, T. V. (2023). *Machine Learning: Concepts, Techniques, and Applications*. Florida: CRC Press.

Swasono, D. I., Wijaya, M. A. R., S Hidayat, M. A. (2023). Klasifikasi Penyakit pada Citra Buah Jeruk Menggunakan *Convolutional Neural Networks* (CNN) dengan Arsitektur Alexnet. *Informatics Journal*, 8(1), 68-75.

Supriyadi, Panggabean, T., Dewi, S. R., Yusra, S., S Ramadhan, N. D. (2024). Penerapan Metode *Deteksi Tepi Sederhana Pada Citra Menggunakan Operator Sobel*. *Repeater : Publikasi Teknik Informatika dan Jaringan*, 2(3), 43-56.

Suyanto, Ramadhan, K. N., S Mandala, S. (2019). *Deep Learning: Modernisasi Machine Learning Untuk Big Data*. Bandung: Informatika.

Tang, X., Wang, X., Hou, J., Wu, H., S Liu, D. (2020). *An Improved Sobel Face Gray Image Edge Detection Algorithm*. *Proceedings of the 39th Chinese Control Conference*,6639-6643.

Talib, S., Sudin, S., S Suratin, M. D. (2024). Penerapan Metode *Support Vector Machine* (SVM) Pada Klasifikasi Jenis Cengkeh

Berdasarkan Fitur Tekstur Daun. *Jurnal PROSISKO*, 11(1), 26-34.

Toyib, M., Pratama, T. D. K., S Aqil, I. (2024). Penerapan algoritma CNN untuk mendeteksi tulisan tangan angka Romawi dengan augmentasi data. *Algoritma: Jurnal Matematika, Ilmu Pengetahuan Alam, Kebumian, dan Angkasa*, 2(3), 108–120.

Tumiwa, V., Batubara, M. Z., Stephani, G., Sembiring, L. A., S Novelitia, J. (2024). Gejolak budaya Korea melalui K-pop dan drama Korea terhadap kehidupan mahasiswa Universitas Palangka Raya. *JISPAR: Jurnal Ilmu Sosial, Politik dan Pemerintahan*, 13(1), 302-311.

Umam, A. K., Ngastiti, P. T. B., Alfan, A., Shahadah, Z., S Muamalah, A. F. (2024). Transformasi wavelet untuk denoising citra. *Mathunesa: Jurnal Ilmiah Matematika*, 12(02), 374–380.

Utami, M. (2021). Identifikasi Text Meteran Air Menggunakan Metode Run-Length Smearing Algorithm (RLSA). *JUKOMIKA: Jurnal Ilmu Komputer dan Informatika*, 4(2), 98-106.

Waladi, A., Perdana, Y., Iftitah, H. S Hanum, N. R. (2024). Stock Price Prediction Using Machine Learning - Based on RNN Algorithms. *Media Journal of Information System and Informatics*, 1(1), 1-8.

Wardana, M. W., Rahmat, B., S Maulana, H. (2024). Klasifikasi Citra Eurosat. *JATI: Jurnal Mahasiswa Teknik Informatika*, 8(4), 7754-7761.

Wardani, K. R. R., S Leonardi, L. (2023). Klasifikasi Penyakit pada Daun Anggur menggunakan Metode *Convolutional Neural NetworkJurnal Tekno Insentif*, 17(2), 112-126.

Wulandari, N. P., S Fitrianah, D. (2021). Analisa Perbandingan Algoritma CNN Dan MLP Dalam Mendeteksi Penyakit COVID-19 Pada Citra X-Ray Paru. *Sains, Aplikasi, Komputasi dan Teknologi Informasi*, 3(2), 44-52.

Yohannes, R., S Rivan, M. E. A. (2022). Klasifikasi Jenis Kanker Kulit Menggunakan CNN-SVM. *Jurnal Algoritme*, 2(2), 133-144 .

Yafis, B., S Rijal, H. (2023). Analisa Respon Frekuensi Citra Digital Menggunakan Metode Transformasi Fourier Diskrit. *JETI: Jurnal Elektronika dan Teknologi Informasi*, 4(1), 25-33 .

Zhang, A., Lipton, Z. C., Li, M., S Smola, A. J. (2020). *Dive into Deep Learning*. Release 0.14.3. MXNet.

Zhi, T. Y., Tseng, S. F., Tsou, S. X,S Ho, W. T. (2024). *Using an LED Light Source Coupled With Spectral Image Analysis for Non-Invasive Glucose Detection*. *IEEE Photonics Journal*, 16(3), 1-6.

Lampiran 1. *Source Code* Python Tahap *RGB to Grayscale* Karakter Tulisan Korea *(Hangeul)*

```
def rgb_to_grayscale_manual(image_path):
    """
    Mengubah gambar RGB menjadi grayscale secara manual.

    Args:
        image_path: Path ke gambar RGB.

    Returns:
        Matriks grayscale dan gambar grayscale.
    """
    image = cv2.imread(image_path)
    if image is None:
        print(f"Gambar tidak bisa dibaca: {image_path}")
        return None, None

    height, width, _ = image.shape
    grayscale_matrix = np.zeros((height, width),
    dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            r, g, b = image[i, j]
            grayscale_value = int(0.2989 * r + 0.5870 * g +
            0.1140 * b)  # Rumus grayscale
            grayscale_matrix[i, j] = grayscale_value

    return grayscale_matrix, grayscale_matrix

def process_images_in_folder(folder_path, save_folder):
    """
    Memproses semua gambar dalam folder,
    mengubahnya menjadi
    grayscale,
    dan menyimpan hasilnya di folder lain.
```

```
    Args:
        folder_path: Path ke folder utama.
        save_folder: Path ke folder tempat menyimpan hasil.
    """
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.endswith(('.jpg', '.jpeg', '.png')):
                image_path = os.path.join(root, file)
                grayscale_matrix, grayscale_image =
                rgb_to_grayscale_manual(image_path)

                if grayscale_image is not None:
                    # Menyimpan gambar grayscale
                    relative_path = os.path.relpath(root,
                    folder_path)
                    save_path = os.path.join(save_folder,
                    relative_path)
                    os.makedirs(save_path, exist_ok=True)
                    save_image_path = os.path.join(save_path,
                    file)

                    cv2.imwrite(save_image_path,
                    grayscale_image)
                    print(f"Gambar disimpan:
                    {save_image_path}")

                    # Menampilkan gambar dan matriks grayscale
                    plt.imshow(grayscale_image, cmap='gray')
                    plt.title(f'Gambar Grayscale - {file}')
                    plt.show()

                    print("Matriks Grayscale:")
                    print(grayscale_matrix)

# Path ke folder utama dan folder tujuan di Google Drive
folder_utama = '/content/drive/MyDrive/Data Korea'
folder_save = '/content/drive/MyDrive/Hasil Grayscale3'
# Memproses semua gambar
process_images_in_folder(folder_utama, folder_save)
```

Lampiran 2. *Source Code* Python Tahap *Denoising* Karakter Tulisan Korea *(Hangeul)*

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

def median_filter_manual(image_matrix, kernel_size):
    """
    Melakukan median filter secara manual.

    Args:
        image_matrix: Matriks gambar.
        kernel_size: Ukuran kernel (misalnya, 3 untuk
        kernel 3x3).

    Returns:
        Matriks gambar yang telah difilter.
    """
    height, width = image_matrix.shape
    filtered_matrix = np.copy(image_matrix)

    padding = kernel_size // 2

    for i in range(padding, height - padding):
        for j in range(padding, width - padding):
            neighborhood = []
            for x in range(i - padding, i + padding + 1):
                for y in range(j - padding, j + padding + 1):
                    neighborhood.append(image_matrix[x, y])

            neighborhood.sort()
            filtered_matrix[i, j] = neighborhood
            [len(neighborhood) // 2]

    return filtered_matrix
```

```
def rgb_to_grayscale_manual(image_path):
    """
    Mengubah gambar RGB menjadi grayscale secara manual.

    Args:
        image_path: Path ke gambar.

    Returns:
        Matriks grayscale dan gambar grayscale.
    """
    image = cv2.imread(image_path)
    if image is None:
        return None, None

    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return grayscale_image, grayscale_image

def process_images_with_median_filter(folder_path,
save_folder, kernel_size):
    """
    Memproses semua gambar dalam folder, mengubahnya menjadi
    grayscale, melakukan median filter,
    dan menyimpan hasilnya di folder lain.

    Args:
        folder_path: Path ke folder utama.
        save_folder: Path ke folder tempat menyimpan hasil.
        kernel_size: Ukuran kernel untuk median filter.
    """
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.endswith(('.jpg', '.jpeg', '.png')):
                image_path = os.path.join(root, file)
                grayscale_matrix, grayscale_image =
                rgb_to_grayscale_manual(image_path)

                if grayscale_image is not None:
                    print(f"Matriks Grayscale - {file}:\n",
                    grayscale_matrix)
```

```
                    denoised_matrix = median_filter_manual
                    (grayscale_matrix, kernel_size)

                    print(f"Matriks Setelah Median Filter -
                    {file}:\n", denoised_matrix)

                    relative_path = os.path.relpath
                    (root, folder_path)
                    save_path = os.path.join
                    (save_folder, relative_path)
                    os.makedirs(save_path, exist_ok=True)
                    save_image_path = os.path.join
                    (save_path, file)

                    cv2.imwrite(save_image_path,
                    denoised_matrix)
                    print(f"Gambar setelah
                    median filter disimpan:
                    {save_image_path}")

                    plt.figure(figsize=(10, 5))
                    plt.subplot(1, 2, 1)
                    plt.imshow(grayscale_matrix, cmap='gray')
                    plt.title(f'Gambar Asli - {file}')

                    plt.subplot(1, 2, 2)
                    plt.imshow(denoised_matrix, cmap='gray')
                    plt.title(f'Gambar Setelah Median Filter
                    - {file}')
                    plt.show()

# folder input dan folder output  di Google Drive
folder_utama = '/content/drive/MyDrive/Hasil Grayscale3'
folder_save = '/content/drive/MyDrive/Hasil Median Filter4'
# Memproses semua gambar dengan median filter
#(kernel size 3x3)
process_images_with_median_filter(folder_utama, folder_save,
kernel_size=3)
# Memproses semua gambar
process_images_in_folder(folder_utama,  folder_save)
```

Lampiran 3. *Source Code* Python Tahap Binerisasi Karakter Tulisan Korea *(Hangeul)*

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

def process_image_with_otsu(image_path, save_path):
    """
    Memproses gambar menggunakan thresholding Otsu,
    mengonversinya menjadi matriks biner,
    dan menyimpan gambar hasil thresholding ke Google Drive.

    Args:
        image_path: Path ke gambar asli.
        save_path: Path untuk menyimpan gambar hasil
        thresholding.
    """
    # Membaca gambar dalam skala abu-abu
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Gambar tidak dapat dibaca: {image_path}")
        return
    # Thresholding Otsu
    ret, thresh = cv2.threshold(img, 0, 255,
    cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
    # Menampilkan pembagian level threshold sebagai
    nilai desimal
    print(f"Pembagian Level (Threshold): {ret/255:.2f}
    untuk gambar {os.path.basename(image_path)}")
    # Konversi gambar biner menjadi matriks angka biner
    (0 untuk
    foreground, 1 untuk background)
    binary_matrix = (thresh < 128).astype(np.uint8)
    # Tampilan matriks biner
    print("Matriks Biner:")
    print(binary_matrix)
```

```
    # Menyimpan gambar hasil thresholding Otsu
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    cv2.imwrite(save_path, thresh)
    print(f"Gambar hasil thresholding disimpan di:
    {save_path}")
    # Visualisasi
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title("Gambar Asli")
    plt.imshow(img, cmap='gray')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.title("Gambar Biner (Otsu)")
    plt.imshow(thresh, cmap='gray')
    plt.axis('off')
    plt.show()
def process_images_from_folder(folder_path, save_folder):
    """
    Memproses semua gambar dari folder, menerapkan
    thresholding Otsu, dan menyimpan hasilnya di Drive.
    Args:
        folder_path: Path ke folder utama yang berisi
        gambar-gambar.
        save_folder: Path ke folder tempat menyimpan hasil
        gambar setelah thresholding.
    """
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.endswith(('.jpg', '.jpeg', '.png')):
                image_path = os.path.join(root, file)
                relative_path = os.path.relpath
                (root, folder_path)
                save_path = os.path.join(save_folder,
                relative_path, file)
                # Memproses setiap gambar
                process_image_with_otsu(image_path, save_path)
# folder input dan output
folder_utama = '/content/drive/MyDrive/Hasil Median Filter4'
folder_save = '/content/drive/MyDrive/Hasil Otsu4'
process_images_from_folder(folder_utama, folder_save)
```

Lampiran 4. *Source Code* Python Tahap Invers Karakter Tulisan Korea *(Hangeul)*

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

def invert_binary_manual(binary_matrix):
    """
    Melakukan invers biner secara manual pada matriks biner.

    Args:
        binary_matrix: Matriks biner (numpy array) yang
        akan di-invers.

    Returns:
        Matriks biner yang telah di-invers.
    """
    inverted_matrix = 1 - binary_matrix
    return inverted_matrix

def process_and_invert_image(image_path, save_path):
    """
    Memproses gambar dari folder utama, melakukan
    thresholding Otsu, dan invers biner secara manual.
    Hasil gambar yang telah di-invers disimpan ke Google Drive.

    Args:
        image_path: Path ke gambar asli.
        save_path: Path untuk menyimpan gambar yang di-invers.
    """
    # Membaca gambar dalam skala abu-abu
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Gambar tidak dapat dibaca: {image_path}")
        return
```

```
# Thresholding Otsu
ret, thresh = cv2.threshold
(img, 0, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Konversi gambar biner menjadi
matriks angka biner (0 untuk foreground,
1 untuk background)
binary_matrix = (thresh < 128).astype(np.uint8)

# Melakukan invers biner secara manual
inverted_matrix = invert_binary_manual(binary_matrix)

# Menampilkan matriks biner asli dan yang telah di-invers
print("Matriks Biner Asli:")
print(binary_matrix)
print("\nMatriks Biner Setelah Invers:")
print(inverted_matrix)

# Menyimpan gambar hasil invers
os.makedirs(os.path.dirname(save_path), exist_ok=True)
cv2.imwrite(save_path, inverted_matrix * 255)
# Konversi kembali ke format gambar
print(f"Gambar hasil invers disimpan di: {save_path}")

# Visualisasi
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Gambar Biner Asli")
plt.imshow(thresh, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("Gambar Biner Setelah Invers")
plt.imshow(inverted_matrix, cmap='gray')
plt.axis('off')

plt.show()
```

```
def process_images_from_folder(folder_path, save_folder):
    """
    Memproses semua gambar dari folder utama, melakukan
    thresholding Otsu, dan invers biner.
    Hasilnya disimpan ke Google Drive.

    Args:
        folder_path: Path ke folder utama yang
        berisi gambar-gambar.
        save_folder: Path ke folder tempat menyimpan hasil
        gambar setelah invers biner.
    """
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.endswith(('.jpg', '.jpeg', '.png')):
                image_path = os.path.join(root, file)
                relative_path = os.path.relpath
                (root, folder_path)
                save_path = os.path.join(save_folder,
                relative_path, file)

                # Memproses setiap gambar
                process_and_invert_image(image_path, save_path)

# Folder input dan output
hasil
folder_utama = '/content/drive/MyDrive/Hasil Median Filter4'
folder_save = '/content/drive/MyDrive/Hasil Invers5'

# Memproses semua gambar dari folder utama
process_images_from_folder(folder_utama, folder_save)
```

Lampiran 5. *Source Code* Python Tahap *Thinning* Karakter Tulisan Korea *(Hangeul)*

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.morphology import skeletonize
from google.colab import drive
import os

# Fungsi untuk melakukan invers biner secara manual
def invert_binary_manual(binary_matrix):
    inverted_matrix = 1 - binary_matrix
    return inverted_matrix

# Fungsi untuk menyimpan hasil gambar ke Google Drive
def save_image(image_matrix, save_path):
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    # Membuat folder jika belum ada
    cv2.imwrite(save_path, image_matrix * 255)
    # Simpan gambar dalam skala 0-255
    print(f"Gambar berhasil disimpan di {save_path}")

# input dan output
folder_utama = '/content/drive/MyDrive/Data Korea'
folder_save = '/content/drive/MyDrive/Hasil thinned6'
# Memproses semua gambar dalam folder utama secara rekursif
for root, dirs, files in os.walk(folder_utama):
    for file in files:
        if file.endswith('.jpg') or file.endswith('.png'):
        # Filter hanya file gambar
            image_path = os.path.join(root, file)
            save_path = os.path.join(folder_save,
            os.path.relpath
            (image_path, folder_utama))  # Menyimpan dengan
            struktur yang sama

            # Membaca gambar dalam skala abu-abu
            img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
        # Jika gambar tidak terbaca, tampilkan pesan
        kesalahan
        if img is None:
            print(f"Gambar tidak dapat
            dibaca dari path: {image_path}")
            continue
        # Thresholding Otsu untuk binarisasi gambar
        ret, thresh = cv2.threshold
        (img, 0, 255, cv2.THRESH_BINARY_INV +
        cv2.THRESH_OTSU)
        # Konversi gambar biner menjadi matriks angka biner
        (0 untuk foreground, 1 untuk background)
        binary_matrix = (thresh < 128).astype(np.uint8)
        # Melakukan invers biner secara manual
        inverted_matrix = invert_binary_manual
        (binary_matrix)

        #thinning menggunakan skeletonize
        dari skimage
        skeleton = skeletonize(inverted_matrix)

        # Konversi hasil skeletonize menjadi matriks biner
        (0 dan 1)
        thinned_binary_matrix = skeleton.astype(np.uint8)
        # Menyimpan hasil ke Google Drive
        save_image(thinned_binary_matrix, save_path)

        # Menampilkan matriks biner setelah thinning
        print(f"Matriks Biner Setelah Thinning untuk
        gambar {file}:")
        print(thinned_binary_matrix)

        # visualisasi data gambar
        plt.imshow(thinned_binary_matrix, cmap='gray')
        plt.title(f"Gambar Setelah Thinning
        (Matrix Biner) - {file}")
        plt.axis('off')
        plt.show()
```

Lampiran 6. *Source Code* Python Tahap Pembagian Data Karakter Tulisan Korea *(Hangeul)*

```
import numpy as np
import os
import cv2
from sklearn.model_selection import train_test_split
# Direktori data dan kelas karakter
base_dir = "/content/drive/MyDrive/augmented_data"
# Direktori untuk menyimpan hasil split dataset
#di Google Drive
split_output_dir = "/content/drive/MyDrive/Split_Data3"
os.makedirs(split_output_dir, exist_ok=True)
# Fungsi untuk memuat, membagi, dan menyimpan dataset
def load_split_and_save_data(base_dir, classes, test_size=0.2,
split_output_dir=None):
    images, labels, image_paths = [], [], []
    for idx, class_name in enumerate(classes):
        class_dir = os.path.join(base_dir, class_name)
        if not os.path.exists(class_dir):
            print(f"Direktori untuk kelas '{class_name}'
            tidak ditemukan: {class_dir}")
            continue
        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)
            img = cv2.imread(img_path)
            if img is None:
                print(f"Gambar '{img_name}' pada kelas '
                {class_name}' gagal dimuat.")
                continue

            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, (28, 28))
            images.append(img)
            labels.append(idx)
            image_paths.append(img_path)
```

```
    images = np.array(images, dtype='float32') / 255.0
    labels = np.array(labels)
    X_train, X_test, y_train, y_test, train_paths,
    test_paths = train_test_split(
        images, labels, image_paths,
        test_size=test_size, random_state=42)
    if split_output_dir:
        save_split_data(train_paths,
        X_train, y_train, "train",
        split_output_dir, classes)
        save_split_data(test_paths,
        X_test, y_test, "test",
        split_output_dir, classes)
    return X_train, X_test, y_train, y_test
# Fungsi untuk menyimpan data yang sudah di-split
def save_split_data(image_paths, images, labels, split_type,
split_output_dir, classes):
    split_dir = os.path.join(split_output_dir, split_type)
    os.makedirs(split_dir, exist_ok=True)
    for img, label, path in zip(images, labels, image_paths):
        class_name = classes[label]
        class_dir = os.path.join(split_dir, class_name)
        os.makedirs(class_dir, exist_ok=True)
        img_name = os.path.basename(path)
        save_path = os.path.join(class_dir, img_name)
        img = (img * 255).astype(np.uint8)
        cv2.imwrite(save_path, cv2.cvtColor
        (img, cv2.COLOR_RGB2BGR))
# Membagi dataset sesuai rasio yang berbeda
# 80:20 Split
print("Membagi dan menyimpan dataset 80:20...")
split_80_20_dir = os.path.join(split_output_dir, '80-20')
os.makedirs(split_80_20_dir, exist_ok=True)
X_train_80, X_test_80, y_train_80, y_test_80 = load_split_
and_save_data(
    base_dir, char_classes, test_size=0.20,
    split_output_dir=split_80_20_dir)
print(f'80:20 Split: {len(X_train_80)} sampel pelatihan,
{len(X_test_80)} sampel pengujian')
```

```
# 70:30 Split
print("Membagi dan menyimpan dataset 70:30...")
split_70_30_dir = os.path.join(split_output_dir, '70-30')
os.makedirs(split_70_30_dir, exist_ok=True)
X_train_70, X_test_70, y_train_70, y_test_70 =
load_split_and_save_data(
    base_dir, char_classes, test_size=0.30,
    split_output_dir=split_70_30_dir)
print(f'70:30 Split: {len(X_train_70)} sampel pelatihan,
{len(X_test_70)} sampel pengujian')

# 90:10 Split
print("Membagi dan menyimpan dataset 90:10...")
split_90_10_dir = os.path.join(split_output_dir, '90-10')
os.makedirs(split_90_10_dir, exist_ok=True)
X_train_90, X_test_90, y_train_90, y_test_90 =
load_split_and_save_data(
    base_dir, char_classes, test_size=0.10,
    split_output_dir=split_90_10_dir)
print(f'90:10 Split: {len(X_train_90)} sampel pelatihan,
{len(X_test_90)} sampel pengujian')
```

Lampiran 7. *Source Code* Python Proses Augmentasi Data Karakter Tulisan Korea *(Hangeul)*

```
from tensorflow.keras.preprocessing.image
import ImageDataGenerator, load_img, img_to_array
import os
import cv2
import numpy as np

# Label kelas
char_classes = ["Yu", "ae", "b", "bb",
"ch", "d", "e", "eo", "eu", "g", "gg", "h", "i", "j", "k",
                "m", "n", "ng", "o", "p", "r", "s",
                "ss", "t", "u", "ya", "yae", "ye", "yo"]

# Augmentasi Data
datagen = ImageDataGenerator(
    rotation_range=40,  # Rotasi gambar hingga 40 derajat
    width_shift_range=0.2,  # Geser gambar secara horizontal
    height_shift_range=0.2,  # Geser gambar secara vertikal
    shear_range=0.2,  # Geser gambar secara miring
    zoom_range=0.2,  # Zoom ke dalam gambar
    horizontal_flip=True,  # Membalik gambar secara horizontal
    fill_mode="nearest"
)

# simpan hasil augmentasi
output_dir = "/content/drive/MyDrive/augmented_data"

# memastikan direktori output ada
os.makedirs(output_dir, exist_ok=True)

# Fungsi untuk mengaugmentasi dan menyimpan gambar per kelas
def augment_and_save_images(datagen, input_dir, output_dir,
num_samples=5):
    for class_name in char_classes:
        class_input_dir = os.path.join(input_dir, class_name)
        class_output_dir = os.path.join(output_dir, class_name)
```

```
      # Periksa apakah direktori input kelas ada
        if not os.path.exists(class_input_dir):
            print(f"Peringatan: Folder {class_input_dir}
            tidak ada. Melewati kelas ini.")
            continue
        # Memastikan direktori output kelas ada
        os.makedirs(class_output_dir, exist_ok=True)
        #   di direktori kelas
        image_paths = [os.path.join(class_input_dir, img_name)
        for img_name in os.listdir(class_input_dir)]
    # Inisialisasi penghitung gambar yang diaugmentasi
        aug_count = 0
        for img_path in image_paths:
            img = load_img(img_path)  # Muat gambar
             # Ubah menjadi array numpy
            img_array = img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0)
            #  generator untuk augmentasi
            img_gen = datagen.flow(img_array, batch_size=1)
            #simpan gambar yang diaugmentasi
             # Buat num_samples per gambar
            for _ in range(num_samples):
            # sampel gambar yang diaugmentasi
                aug_img = next(img_gen)[0].astype(np.uint8)
                aug_img_name = f"{class_name}_aug_
                {aug_count}.jpg"
                aug_save_path = os.path.join(class_output_dir,
                aug_img_name)
                cv2.imwrite(aug_save_path, cv2.cvtColor
                (aug_img, cv2.COLOR_RGB2BGR))
                aug_count += 1
                print(f"Gambar yang diaugmentasi ke-{aug_count}
                telah disimpan untuk kelas {class_name}.")
# Inputan


input_dir = "/content/drive/MyDrive/Hasil thinned6"
# Augmentasi dan simpan gambar untuk data pelatihan
print("Menyimpan gambar yang diaugmentasi...")
augment_and_save_images(datagen, input_dir, output_dir,
num_samples=5)
```

Lampiran 8. *Source Code* Python Proses Ekstraksi Fitur *Histogram of Oriented Gradients* (HOG) Karakter Tulisan Korea *(Hangeul)*

```
import os
import numpy as np
import cv2
from skimage.feature import hog
import pandas as pd
from tqdm import tqdm  # Progress bar untuk melihat progress

# Folder input dan output
main_dir = '/content/drive/MyDrive/Split_Data3/80-20/train/'
output_dir = '/content/drive/MyDrive/processed67/80_20'

# Menelusuri semua folder dan file di dalam direktori
for root, dirs, files in os.walk(main_dir):
 # Progress bar
    for file in tqdm(files, desc="Memproses gambar"):
        try:
            # Memastikan hanya memproses file gambar
            if file.endswith('.jpg') or file.endswith('.png'):
                image_path = os.path.join(root, file)
                print(f"Memproses gambar: {image_path}")

                # Membaca dan mengubah gambar menjadi
                grayscale dengan OpenCV
                image = cv2.imread(image_path,
                cv2.IMREAD_GRAYSCALE)

                #Ukuran gambar
                gray_image_resized = cv2.resize(image, (28, 28)
                , interpolation=cv2.INTER_AREA)
                  # Membuat folder output untuk gambar
                yang telah diproses
                class_name = os.path.basename(root)
```

```
                # Mengambil nama folder sebagai nama kelas
                output_folder = os.path.join
                (output_dir, class_name)
                os.makedirs(output_folder, exist_ok=True)
                # Menyimpan gambar grayscale yang
                #diresize ke PNG
                gray_image_resized_path = os.path.join
                (output_folder, f'{file}_gray_resized.png')
                cv2.imwrite(gray_image_resized_path,
                gray_image_resized)

                # Ekstraksi fitur HOG dan visualisasi
                hog_features, hog_image = hog(
                    gray_image_resized,
                    orientations=9,  # Jumlah bin orientasi
                    pixels_per_cell=(8, 8), # Ukuran sel
                    cells_per_block=(2, 2), # Ukuran blok
                    block_norm='L2-Hys', # Normalisasi L2-Hys
                    visualize=True  # Menghasilkan gambar HOG
                )

                # HOG hasil ekstraksi fitur ke PNG
                hog_image_path = os.path.join(output_folder,
                f'{file}_hog_image.png')
                cv2.imwrite(hog_image_path, hog_image)

                # simpan vektor fitur HOG asli (144x1)keCSV
                hog_features_path = os.path.join(output_folder,
                f'{file}_hog_features.csv')
                pd.DataFrame(hog_features).to_csv
                (hog_features_path, index=False, header=False)

                # Normalisasi L2-Hys dan mengubah
                #ukuran menjadi 36x1
                normalized_hog_features = hog_features /
                np.linalg.norm(hog_features, ord=2)
```

```
                # Memastikan ukuran matriks fitur HOG yang
                dinormalisasi menjadi 36x1
                desired_size = 36
                if normalized_hog_features.size !=
                desired_size:
                    if normalized_hog_features.size >
                    desired_size:
                        normalized_hog_features =
                        normalized_hog_features[:desired_size]
                    else:
                        normalized_hog_features =
                        np.pad(normalized_hog_features,
                        (0, desired_size -
                        normalized_hog_features.size),
                        'constant')

                # simppanvektor HOG yang dinormalisasi (36x1)
                normalized_hog_features_path =
                os.path.join(output_folder, f'{file}
                _hog_normalized_features.csv')
                pd.DataFrame(normalized_hog_features).
                to_csv(normalized_hog_features_path,
                index=False, header=False)
```

Lampiran 9. *Source Code* Python Metode SVM *(Support Vector Machine)*

```
import os
import numpy as np
import pandas as pd
classification_report, roc_curve, auc
from sklearn.model_selection import GridSearchCV
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
# Menentukan direktori
direktori_train = '/content/drive/MyDrive/processed67/70_30'
nilai_tol = = 0.00000001

# Fungsi untuk membaca data dari direktori
def load_data(direktori):
    fitur, label = [], []
    # Mencari file yang berakhiran _hog_features.csv
    di dalam direktori dan subdirektori
    file_csv = [os.path.join(root, file)
                for root, _, files in os.walk(direktori)
                for file in files if file.endswith
                ('_hog_features.csv')]

    if not file_csv:
        print(f"Tidak ada file ditemukan di {direktori}.
        Silakan periksa direktori atau nama file.")
        return np.array(fitur), np.array(label)
```

```
    # Iterasi melalui file dan memuat data
    for file in tqdm(file_csv, desc="Memuat fitur HOG",
   leave=False):
        try:
            nama_kelas = os.path.basename
            (os.path.dirname(file))
            fitur_hog = pd.read_csv(file, header=None).values.
            flatten()
            fitur.append(fitur_hog)
            label.append(nama_kelas)
        except Exception as e:
            print(f"Terjadi kesalahan saat memuat {file}: {e}")

    return np.array(fitur), np.array(label)

# Memuat data training
fitur_train, label_train = load_data(direktori_train)
if len(fitur_train) == 0 or len(label_train) == 0:
    raise ValueError("Data training tidak berhasil dimuat.
    Periksa direktori dan file input.")

# Memuat data testing
fitur_test, label_test = load_data(direktori_test)
if len(fitur_test) == 0 or len(label_test) == 0:
    raise ValueError("Data testing tidak berhasil dimuat.
    Periksa direktori dan file input.")

# Encode label sebagai integer
encoder_label = LabelEncoder()
label_encoded_train = encoder_label.fit_transform(label_train)
label_encoded_test = encoder_label.transform(label_test)
# Normalisasi data
scaler = StandardScaler()
fitur_train_scaled = scaler.fit_transform(fitur_train)
fitur_test_scaled = scaler.transform(fitur_test)
```

Kernel Polinomial

```
output_file = '/content/drive/MyDrive/
svmhurr_outputlalalala7030.txt'
best_model_path = f"/content/drive/MyDrive/
svmhuurr_model_70_30lala.joblib"

if not os.path.exists(best_model_path):
    with open(output_file, 'w') as f:
        print(f"\n===== Memulai Pelatihan SVM =====",
        file=f)

        print(f"Nilai tol yang digunakan: {nilai_tol}",
        file=f)

        # Menampilkan jumlah fitur HOG dan variasinya
        print(f"\n===== Statistik Fitur HOG (Training) =====",
        file=f)
        print(f"Jumlah fitur HOG: {fitur_train.shape[1]}",
        file=f)
        print(f"Jumlah variasi fitur HOG: {np.var(fitur_train,
        axis=0).sum():.4f}", file=f)

       # Hyperparameter tuning dengan GridSearchCV
      param_grid = {
            'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
            'degree': [2, 3, 4],
            'gamma': ['scale', 'auto'],
            'kernel': ['poly']
        }
```

Kernel Sigmoid

```
  output_file = '/content/drive/MyDrive/
svmhurr_outputlalalala7030.txt'
best_model_path = f"/content/drive/MyDrive/
svmhuurr_model_70_30lala.joblib"

if not os.path.exists(best_model_path):
    with open(output_file, 'w') as f:
        print(f"\n===== Memulai Pelatihan SVM =====",
        file=f)

        print(f"Nilai tol yang digunakan: {nilai_tol}",
        file=f)

        # Menampilkan jumlah fitur HOG dan variasinya
        print(f"\n===== Statistik Fitur HOG (Training) =====",
        file=f)
        print(f"Jumlah fitur HOG: {fitur_train.shape[1]}",
        file=f)
        print(f"Jumlah variasi fitur HOG: {np.var(fitur_train,
        axis=0).sum():.4f}", file=f)
 # Hyperparameter tuning
GridSearchCV untuk kernel sigmoid
      param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10],
        'coef0': [0.0, 0.1, 0.5, 1.0],
        'gamma': ['scale', 'auto'],
        'kernel': ['sigmoid']
    }
```

Kernel Linier

```
output_file = '/content/drive/MyDrive/
svmhurr_outputlinier7030.txt'
best_model_path = f"/content/drive/MyDrive/
svmhuurr_model_70_30lala_linier.joblib"
if not os.path.exists(best_model_path):
    with open(output_file, 'w') as f:
        print(f"\n===== Memulai Pelatihan SVM =====",
        file=f)
        print(f"Nilai tol yang digunakan: {nilai_tol}",
        file=f)
        # Menampilkan jumlah fitur HOG dan variasinya
        print(f"\n===== Statistik Fitur HOG (Training) =====",
        file=f)
        print(f"Jumlah fitur HOG: {fitur_train.shape[1]}",
        file=f)
        print(f"Jumlah variasi fitur HOG:
        {np.var(fitur_train, axis=0).sum():.4f}", file=f)
 # Hyperparameter tuning
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10],
        'gamma': ['scale', 'auto'],
        'kernel': ['linear']
    }
```

Kernel RBF

```
output_file = '/content/drive/MyDrive/
svmhurr_outputlalalala7030.txt'
best_model_path = f"/content/drive/MyDrive/
svmhuurr_model_70_30lala_rbf.joblib"

if not os.path.exists(best_model_path):
    with open(output_file, 'w') as f:
        print(f"\n===== Memulai Pelatihan SVM =====",
        file=f)

        print(f"Nilai tol yang digunakan: {nilai_tol}",
        file=f)

        # Menampilkan jumlah fitur HOG dan variasinya
        print(f"\n===== Statistik Fitur HOG (Training) =====",
        file=f)
        print(f"Jumlah fitur HOG: {fitur_train.shape[1]}",
        file=f)
        print(f"Jumlah variasi fitur HOG:
        {np.var(fitur_train, axis=0).sum():.4f}", file=f)
 # Hyperparameter tuning dengan
 #GridSearchCV untuk mencari C dan gamma
             param_grid = {
             'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
            'gamma': ['scale', 'auto'],
            'kernel': ['rbf']
        }
```

Evaluasi Model

```
        grid_search = GridSearchCV(SVC(tol=nilai_tol,
              probability=True),
        param_grid, cv=5, refit=True, verbose=3)
        grid_search.fit(fitur_train_scaled,
        label_encoded_train)
        best_params = grid_search.best_params_
        print(f'Parameter terbaik yang ditemukan:
        {best_params}',  file=f)

        # Menyimpan model terbaik
        best_grid_model = grid_search.best_estimator_
        joblib.dump(best_grid_model, best_model_path)
        print(f"\nModel terbaik disimpan di:
        {best_model_path}", file=f)

        # Evaluasi model pada data testing
        y_pred = best_grid_model.predict(fitur_test_scaled)
        conf_matrix = confusion_matrix(label_encoded_test,
        y_pred)

        print(f"\n===== Confusion Matrix =====", file=f)
        print(conf_matrix, file=f)

        plt.figure(figsize=(10, 7))
        sns.heatmap(conf_matrix, annot=True, fmt='d',
        cmap='Blues', xticklabels=encoder_label.classes_,
        yticklabels=encoder_label.classes_)
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.title('Confusion Matrix')
        plt.show()
        class_report = classification_report
        (label_encoded_test, y_pred,
        target_names=encoder_label.classes_)
        print(f"\n===== Laporan Klasifikasi =====", file=f)
        print(class_report, file=f)
    print(f"Model sudah ada di {best_model_path},
    proses pelatihan dilewati.")
    print(f"Output berhasil disimpan di: {output_file}")
```

Lampiran 10. *Source Code* Python Metode CNN *(Convolutional Neural Network)*

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,
confusion_matrix
from tensorflow.keras.utils import to_categorical
from skimage.feature import hog
from skimage.color import rgb2gray
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D,
MaxPooling2D, Flatten, Dense, Concatenate
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping

# Fungsi untuk memuat dan memproses gambar
def load_images_from_folder(folder):
    images, labels = [], []
    class_names = os.listdir(folder)

    for class_name in class_names:
        class_path = os.path.join(folder, class_name)
        if os.path.isdir(class_path):
            for filename in os.listdir(class_path):
                img_path = os.path.join(class_path, filename)
                img = Image.open(img_path).resize((28, 28))
                img = np.array(img) / 255.0
                images.append(img)
                labels.append(class_name)

    return np.array(images), np.array(labels)
```

```
# dataset
train_path = '/content/drive/MyDrive/Split_Data3/70-30/train/'
test_path = '/content/drive/MyDrive/Split_Data3/70-30/test/'

x_train, y_train = load_images_from_folder(train_path)
x_test, y_test = load_images_from_folder(test_path)

# Encoding label
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
num_classes = len(label_encoder.classes_)
y_train = to_categorical(y_train_encoded, num_classes)
y_test = to_categorical(y_test_encoded, num_classes)

# Fungsi untuk mengekstrak fitur HOG
def extract_hog_features(images):
    return np.array([
        hog(rgb2gray(image, channel_axis=-1), pixels_per_cell
        =(8, 8),
            cells_per_block=(2, 2), visualize=False)
        for image in images
    ])

x_train_hog = extract_hog_features(x_train)
x_test_hog = extract_hog_features(x_test)

# Fungsi untuk membuat model CNN + HOG
def create_model(hog_feature_size):
    input_image = Input(shape=(28, 28, 3))
    x = Conv2D(32, (3, 3), activation='relu')(input_image)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
```

```
 input_hog = Input(shape=(hog_feature_size,))
    combined_features = Concatenate()([x, input_hog])

    z = Dense(128, activation='relu')(combined_features)
    z = Dropout(0.3)(z)  # Mengurangi overfitting
    #dengan dropout 30%
    z = Dense(64, activation='relu')(z)
    output = Dense(num_classes, activation='softmax')(z)

    model = Model(inputs=[input_image, input_hog],
    outputs=output)
    return model

# Hyperparameter tuning
optimizers = [(Adam, "Adam"), (RMSprop, "RMSprop")]
learning_rates = [0.001, 0.0001]
epochs_list = [15, 25]
batch_sizes = [32, 64]

early_stopping = EarlyStopping(monitor='val_loss',
patience=3, restore_best_weights=True)

results = []
best_model = None
best_accuracy = 0

for optimizer_class, optimizer_name in optimizers:
    for lr in learning_rates:
        for epochs in epochs_list:
            for batch_size in batch_sizes:
                model = create_model(x_train_hog.shape[1])
                optimizer = optimizer_class(learning_rate=lr)
                model.compile(optimizer=optimizer,
                loss='categorical_crossentropy',
                metrics=['accuracy'])

                print(f"Training dengan {optimizer_name},
                LR: {lr}, Epochs: {epochs},
                Batch: {batch_size}")
```

```
history = model.fit(
                    [x_train, x_train_hog], y_train,
                    epochs=epochs, batch_size=batch_size,
                    validation_data=([x_test, x_test_hog],
                    y_test),
                    callbacks=[early_stopping], verbose=1
                )



                test_loss, test_accuracy = model.evaluate
                ([x_test, x_test_hog], y_test)
                print(f"Test Accuracy: {test_accuracy
                * 100:.2f}%")

                results.append([optimizer_name, lr, epochs,
                batch_size, test_accuracy])

                # Simpan model terbaik
                if test_accuracy > best_accuracy:
                    best_accuracy = test_accuracy
                    best_model = model

                y_pred = model.predict([x_test, x_test_hog])
                y_pred_classes = np.argmax(y_pred, axis=1)
                y_true_classes = np.argmax(y_test, axis=1)

                print("Classification Report:")
                print(classification_report(y_true_classes,
                y_pred_classes, target_names=
                label_encoder.classes_))

                precision, recall, f1, _ =
                precision_recall_fscore_support(y_true_classes,
                y_pred_classes, average='weighted')
                print(f"Precision: {precision:.4f},
                Recall: {recall:.4f}, F1-score: {f1:.4f}")
                conf_matrix = confusion_matrix(y_true_classes,
                y_pred_classes)
                plt.figure(figsize=(8, 6))
                sns.heatmap(conf_matrix, annot=True, fmt='d',
                cmap='Blues',xticklabels=label_encoder
                .classes_,
```

```
yticklabels=label_encoder.classes_)
                plt.xlabel('Predicted')
                plt.ylabel('Actual')
                plt.title(f'CM {optimizer_name}, LR: {lr},
                Epochs: {epochs}, Batch: {batch_size}')
                plt.show()

# Simpan hasil ke CSV
results_df = pd.DataFrame(results, columns=
["Optimizer", "Learning Rate", "Epochs", "Batch Size",
"Test Accuracy"])
results_df.to_csv("/content/drive/MyDrive/
hyperparameter7000_results.csv", index=False)

# Cari model terbaik berdasarkan akurasi
best_model_idx = results_df["Test Accuracy"].idxmax()
best_params = results_df.iloc[best_model_idx]
print("Best Hyperparameters:", best_params)

# Simpan model terbaik dalam format H5
if best_model:
    save_model_h5(best_model, "/content/drive/MyDrive/
    best70000_cnn_hog_model.h5")
```

Lampiran 11. Daftar Riwayat Hidup

DAFTAR RIWAYAT HIDUP

Nama	: Muhammad Fikri Riyanto
TTL	: Kudus, 31 Maret 2003
Alamat Rumah	: Desa Langgardalem Rt 02 Rw 03, Kota Kudus, Provinsi Jawa Tengah
Email	: 2108046041@student.walisongo.ac.id
No. HP	: +6287814535194
Jenis Kelamin	: Laki - Laki
Agama	: Islam
Kewarganegaraan	: Indonesia

RIWAYAT PENDIDIKAN:

2007-2009	: TK Muslimat NU Nawa Kartika
2009-2015	: SD NU Nawa Kartika
2015-2018	: MTSN 1 Kudus
2018-2021	: SMA Negeri 2 Kudus
2021-sekarang	: S1 Matematika Jurusan Matematika UIN Walisongp Semarang

Semarang, 19 MARET 2025

Hormat Saya,

Muhammad Fikri Riyanto

NIM: 2108046041