

**SISTEM PENERJEMAH BAHASA ISYARAT OTOMATIS
MENGUNAKAN METODE *DEEP LEARNING MODEL*
*CONVOLUTIONAL NEURAL NETWORK***

SKRIPSI



Oleh

Rizkika Zakka Palindungan

NIM E41170164

**PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI JEMBER
2021**

**SISTEM PENERJEMAH BAHASA ISYARAT OTOMATIS
MENGUNAKAN METODE *DEEP LEARNING MODEL*
*CONVOLUTIONAL NEURAL NETWORK***

SKRIPSI



sebagai salah satu syarat untuk memperoleh gelar Sarjana Terapan Komputer
(S.Tr.Kom)
di Program Studi Teknik Informatika
Jurusan Teknologi Informasi

Oleh

Rizkika Zakka Palindungan

NIM E41170164

**PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI JEMBER
2021**

**KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET DAN
TEKNOLOGI
POLITEKNIK NEGERI JEMBER
JURUSAN TEKNOLOGI INFORMASI**

**Sistem Penerjemah Bahasa Isyarat Otomatis
Menggunakan Metode *Deep Learning*
*Model Convolutional Neural Network***

Rizkika Zakka Palindungan (NIM E41170164)

Ketua Penguji




Ratih Ayuninghemi, S.ST, M.Kom
NIP 19860802 201504 2 002

Sekretaris Penguji



Aji Seto Arifianto, S.ST., M.T.
NIP 19851128 200812 1 002

Anggota Penguji



Mukhamad Angga G., S. Pd., M. Eng.
NIP 19940812 201903 1 013

Dosen Pembimbing



Aji Seto Arifianto, S.ST., M.T.
NIP 19851128 200812 1 002



Mengesahkan,
Ketua Jurusan Teknologi Informasi



Hendri Yuni Riskiawan, S.Kom, M.Cs
NIP 19830203 200604 1 003

SURAT PERNYATAAN

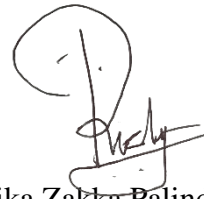
Saya yang bertandatangan dibawah ini :

Nama : Rizkika Zakka Palindungan

NIM : E41170164

Menyatakan dengan sebenar-benarnya bahwa segala pernyataan dalam Laporan Skripsi saya yang berjudul “Sistem Penerjemah Bahasa Isyarat Otomatis Menggunakan Metode *Deep Learning Model Convolutional Neural Network*” merupakan gagasan dan hasil karya saya sendiri dengan arahan komisi pembimbing, dan belum pernah diajukan dalam bentuk apapun pada perguruan tinggi manapun. Semua data dan informasi yang digunakan telah dinyatakan secara jelas dan dapat diperiksa kebenarannya. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan dari penulis lain telah disebutkan dalam naskah dan dicantumkan dalam daftar pustaka di bagian akhir Laporan Skripsi ini.

Jember, 20 September 2021



Rizkika Zakka Palindungan

NIM E41170164



**PERNYATAAN
PERSETUJUAN PUBLIKASI
KARYA ILMIAH UNTUK KEPENTINGAN
AKADEMIS**

Yang bertandatangan di bawah ini, saya:

Nama : Rizkika Zakka Palindungan
NIM : E41170164
Program Studi : Teknik Informatika
Jurusan : Teknologi Informasi

Demi pengembangan Ilmu Pengetahuan, saya menyetujui untuk memberikan kepada UPT. Perpustakaan Politeknik Negeri Jember, Hak Bebas Royalti NonEksklusif (Non-Exclusive Royalty Free Right) atas Karya Ilmiah **berupa Laporan Skripsi saya yang berjudul:**

**SISTEM PENERJEMAH BAHASA ISYARAT OTOMATIS
MENGUNAKAN METODE *DEEP LEARNING MODEL*
*CONVOLUTIONAL NEURAL NETWORK***

Dengan Hak Bebas Royalti Non-Eksklusif ini UPT. Perpustakaan Politeknik Negeri Jember berhak menyimpan, mengalih media atau format, mengelola dalam bentuk Pangkalan Data (Database), mendistribusikan karya dan menampilkan atau mempublikasikannya di Internet atau media lain untuk kepentingan akademis tanpa perlu meminta ijin dari saya selama tetap mencantumkan nama saya sebagai penulis atau pencipta.

Saya bersedia untuk menanggung secara pribadi tanpa melibatkan pihak Politeknik Negeri Jember, Segala bentuk tuntutan hukum yang timbul atas Pelanggaran Hak Cipta dalam Karya ilmiah ini.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Jember
Pada Tanggal : 20 September 2021

Yang menyatakan,



Nama : Rizkika Zakka Palindungan
NIM : E41170164

MOTTO

“If something’s important enough, you should try. Even if you the probable outcome is failure.”

-Elon Musk

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

-Albert Einstein

“Keberhasilan bukanlah milik Orang pintar. Namun Keberhasilan itu adalah milik mereka yang senantiasa berusaha.”

-B.J. Habibie

PERSEMBAHAN

Laporan skripsi ini saya persembahkan dengan rasa hormat kepada :

1. Orang tua tercinta terima kasih atas semua kasih sayang dan cintanya, dukungan baik moril maupun materil, serta doa yang tak henti dan pengorbanan yang tak terhingga.
2. Salsabila Qutrotu'ain yang selalu memberi semangat dan motivasi dalam mengerjakan skripsi ini.
3. Para pengajar Politeknik Negeri Jember khususnya dosen dan teknisi Program Studi Teknik Informatika yang telah memberikan banyak ilmu dan pengetahuan.
4. Almamater tercinta Politeknik Negeri Jember.

**Sistem Penerjemah Bahasa Isyarat Otomatis Menggunakan Metode Deep
Learning Model Convolutional Neural Network**
(Automatic Sign Language Translator System Using Deep Learning Model
Convolutional Neural Network Method)
Pembimbing (1 Orang).

Aji Seto Arifianto S.ST, M.T
Study Program of Informatics Engineering
Majoring of Information Technology
Program Studi Teknik Informatika
Jurusan Teknologi Informasi

ABSTRAK

Bisindo can be interpreted as a terminology used to refer to the natural sign language used by the deaf community in Indonesia. Bisindo functions as a tool to communicate using lips and body language, including facial expressions and eye gaze. According to research from the WHO, more than 5% of the world's population or 466 million people have hearing loss and is expected to continue to increase. While it is undeniable that generally sign language is difficult to understand by the wider community and makes the deaf community feel alienated in their environment. Therefore, the researcher created a system with the title "Automatic Sign Language Translator System Using Deep Learning Model Convolutional Neural Network Method". This research was conducted with a computer vision-based approach that uses a camera to capture sign language movements. Then carry out the process of collecting datasets, making training data models and image classification. The results of this study indicate that the training data has a test score of 0.4% and a test accuracy value of 99%. The analysis on the training data model that has been made by the researcher is categorized as NOT input-bound because only 1.3% of the total sample time is waiting for input. So it can be said that the model already has good efficiency.

Keywords : Sign Language, Bisindo , Deep Learning, Convolutional Neural Network

RINGKASAN

Sistem Penerjemah Bahasa Isyarat Otomatis Menggunakan Metode *Deep Learning Model Convolutional Neural Network*, Rizkika Zakka Palindungan, NIM E41170164, Tahun 2021, Teknologi Informasi, Politeknik Negeri Jember, Aji Seto Arifianto, S.ST, M.T. (Pembimbing).

Bisindo dapat diartikan sebagai sebuah terminologi yang digunakan untuk menunjuk pada bahasa isyarat alami yang digunakan oleh komunitas tuli di Indonesia. Bisindo berfungsi sebagai alat untuk berkomunikasi dengan menggunakan gerak bibir dan bahasa tubuh, termasuk ekspresi wajah dan pandangan mata. Menurut penelitian dari WHO lebih dari 5% populasi dunia atau 466 juta orang mengalami gangguan pendengaran dan diperkirakan akan terus mengalami kenaikan. Sedangkan tidak bisa dipungkiri pada umumnya bahasa isyarat sulit dipahami oleh masyarakat luas dan membuat komunitas tuli merasa terasingkan di lingkungannya. Oleh karena itu peneliti membuat sistem dengan judul “Sistem Penerjemah Bahasa Isyarat Otomatis Menggunakan Metode *Deep Learning Model Convolutional Neural Network*”. Penelitian ini dilakukan dengan pendekatan berbasis computer vision yang menggunakan kamera untuk menangkap gerakan bahasa isyarat. Kemudian melakukan proses mengumpulkan dataset, pembuatan model data training dan klasifikasi citra. Pada hasil penelitian ini menunjukkan data training memiliki nilai *test score* sebesar 0.4% dan nilai *test accuracy* sebesar 99%. Analisis pada model data training yang sudah dibuat peneliti dikategorikan TIDAK *input-bound* karena hanya 1,3% dari total waktu langkah sampel yang menunggu *input*. Sehingga dapat dikatakan model sudah memiliki efisiensi yang baik.

Kata Kunci : Bahasa Isyarat, Bisindo , *Deep Learning, Convolutional Neural Network*

PRAKATA

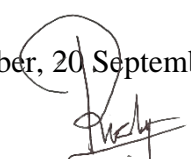
Puji syukur penulis panjatkan kehadirat Allah Swt. Atas berkat rahmat dan karunia-Nya sehingga penulisan laporan skripsi berjudul “Sistem Penerjemah Bahasa Isyarat Otomatis Menggunakan Metode *Deep Learning Model Convolutional Neural Network*” dapat diselesaikan dengan baik. Laporan ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Terapan (S.Tr. Kom) di Program Studi Teknik Informatika Jurusan Teknologi Informasi Politeknik Negeri Jember.

Penulis menyampaikan penghargaan dan ucapan terima kasih yang sebesar-besarnya sebagai berikut :

1. Direktur Politeknik Negeri Jember.
2. Bapak Hendra Yufit Riskiawan, S.Kom, M.Cs selaku Ketua Jurusan Teknologi Informasi.
3. Ibu Trismayanti Dwi Puspitasari, S.Kom. M.Cs. selaku Ketua Program Studi Teknik Informatika.
4. Bapak Aji Seto Arifianto, S.ST., M.T. selaku Pembimbing.
5. Ibu Ratih Ayuninghemi, S.ST, M.Kom dan Bapak Mukhamad Angga Gumilang, S. Pd., M. Eng. Selaku penguji.
6. Seluruh staf pengajar di program Studi Teknik Informatika.
7. Orang tua, kakak, dan seluruh keluarga besar, terima kasih atas semangat dan dukungannya.
8. Rekan-rekanku dan semua pihak yang telah ikut membantu dalam pelaksanaan penelitian dan penulisan laporan ini.

Laporan Skripsi ini masih kurang sempurna, mengharapkan kritik dan saran yang sifatnya membangun guna perbaikan di masa mendatang. Semoga tulisan ini bermanfaat.

Jember, 20 September 2021


Rizkika Zakka Palindungan
E41170164

DAFTAR ISI

	Halaman
JUDUL	ii
LEMBAR PENGESAHAN	iii
SURAT PERNYATAAN	iv
PERSETUJUAN PUBLIKASI.....	v
MOTTO	vi
PERSEMBAHAN.....	vii
ABSTRAK	viii
RINGKASAN	ix
PRAKATA	x
DAFTAR ISI.....	xi
DAFTAR GAMBAR.....	xv
DAFTAR TABEL	xvii
DAFTAR LAMPIRAN	xviii
BAB 1. PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan Penelitian	4
1.4 Manfaat Penelitian	4
BAB 2. TINJAUAN PUSTAKA.....	5
2.1 State Of The Art	5
2.2 Referensi dan Penelitian Yang Sudah Ada	6
2.3 Kekurangan Penelitian Sebelumnya	7

2.4	Kelebihan Dari Penelitian Yang Diambil.....	9
2.5	Bahasa Isyarat	9
2.5.1	Sejarah Bahasa Isyarat Indonesia (BISINDO).....	10
2.5.2	Isyarat Abjad Dalam Bisindo	11
2.5.3	Isyarat Nomor Dalam Bisindo	12
2.6	Citra Digital	13
2.7	Jenis Citra Digital.....	13
2.7.1	Citra RGB	13
2.7.2	Citra Grayscale.....	14
2.7.3	Citra Biner.....	15
2.8	Pengolahan Citra Digital	15
2.8.1	Preprocessing	16
2.8.2	Segmentasi citra	16
2.8.3	Normalisasi	16
2.9	Visi komputer	16
2.10	Gradient Descent.....	17
2.11	Artificial Neural Network	17
2.11.1	Single Perceptron	19
2.11.2	Multilayer Perceptron.....	21
2.11.3	Multi-class Classification.....	25
2.11.4	Multi-label Classification.....	26
2.11.5	Deep Neural Network	26
2.11.6	Regularization and Dropout	29
2.12	Convolutional Neural Network.....	30
2.12.1	Activation Layer.....	31

2.12.2	Convolution.....	32
2.12.3	Pooling	34
2.13	OpenCV	36
2.14	MediaPipe Hands.....	36
2.15	Keras Tensorflow	37
BAB 3.	METODE PENELITIAN.....	38
3.1	Waktu dan Tempat Penelitian	38
3.2	Alat dan Bahan	38
3.2.1	Alat Penelitian.....	38
3.2.2	Bahan Penelitian.....	39
3.3	Tahapan Penelitian.....	39
3.4	Jenis Data	42
3.4.1	Data Primer	42
3.5	Block Diagram Sistem.....	43
BAB 4.	HASIL DAN PEMBAHASAN.....	53
4.1	Studi Literatur	53
4.2	Pengumpulan Dataset	56
4.2.1	Pengambilan Dataset Langsung	56
4.2.2	Tahapan Pengolahan Citra Dataset	56
4.2.3	Hasil Dataset Gestur Bahasa Isyarat Indonesia.....	59
4.3	Perancangan dan Pembuatan Sistem	63
4.3.1	Sistem Pelatihan Dataset.....	63
4.3.2	Sistem Klasifikasi Citra / Testing	69
4.4	Analisis Hasil Penelitian	77
4.4.1	Performance Summary	78

4.4.2	Step-Time Graph.....	81
4.4.3	Memory Profile Tool.....	82
BAB 5.	KESIMPULAN DAN SARAN.....	85
5.1	Kesimpulan	85
5.2	Saran.....	85
	DAFTAR PUSTAKA	86
	LAMPIRAN.....	88

DAFTAR GAMBAR

	Halaman
Gambar 2.1 Gesture Bahasa Isyarat Abjad (BISINDO)	12
Gambar 2.2 Gesture Bahasa Isyarat Angka (BISINDO)	12
Gambar 2.3 Citra dan piksel penyusunnya	13
Gambar 2.4 Warna RGB	14
Gambar 2.5 Warna Grayscale	15
Gambar 2.6 Huruf “B” dan citra dalam biner	15
Gambar 2.7 Deep Neural Network.....	18
Gambar 2.8 Single Perceptron	19
Gambar 2.9 Multilayer Perceptron.....	22
Gambar 2.10 Multilayer Perceptron 2.....	24
Gambar 2.11 Proses latihan menggunakan backpropagation	25
Gambar 2.12 Ilustrasi desired output pada multi-class classification	26
Gambar 2.13 Ilustrasi desired output pada multi-label classification	26
Gambar 2.14 Deep Neural Network.....	27
Gambar 2.15 Proses latihan DNN menggunakan backpropagation.....	28
Gambar 2.16 Contoh successive learning	29
Gambar 2.17 Sliding window	32
Gambar 2.18 1D Convolution	33
Gambar 2.19 Konsep weight sharing	34
Gambar 2.20 Contoh pooling	35
Gambar 2.21 Convolution dan pooling	35
Gambar 2.22 Convolutional Neural Network	35
Gambar 3.1 Block diagram tahapan penelitian	40
Gambar 3.2 Struktur Folder Dataset	43
Gambar 3.3 Block Diagram Proses Mendapatkan Dataset Citra	44
Gambar 3.4 Block Diagram Proses Training Dataset Citra	47
Gambar 3.5 Block Diagram Proses Klasifikasi Citra	50

Gambar 4.1 Hasil Deteksi Objek Mediapipe	57
Gambar 4.2 Hasil Segmentasi Data Citra	58
Gambar 4.3 Distribution Plot Data Training.....	64
Gambar 4.4 Komparasi Hasil Preprocessing Pelatihan Dataset.....	65
Gambar 4.5 Grafik Akurasi Data Training	68
Gambar 4.6 Grafik Kesalahan Data Training	69
Gambar 4.7 Overview page TensorFlow Profiler	78
Gambar 4.8 Step-time Graph	81
Gambar 4.9 Memory profile tool	82
Gambar 4.10 Memory Timeline Graph.....	84

DAFTAR TABEL

	Halaman
Tabel 2.1 State Of The Art.....	5
Tabel 4.1 Klasifikasi Bahasa Isyarat Indonesia Abjad dan Nomor	59
Tabel 4.2 CNN Model Architecture.....	66
Tabel 4.3 Uji Coba Hasil Klasifikasi Citra	71
Tabel 4.4 Hasil Analisis Step-Time Breakdown.....	78
Tabel 4.5 Memory Profile Summary	83

DAFTAR LAMPIRAN

	Halaman
Lampiran 1 Proses Data Training	89
Lampiran 2 Top 10 TensorFlow operations on GPU	104

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Manusia adalah makhluk sosial yang saling berkomunikasi dengan menggunakan bahasa. Bentuk komunikasi ini biasanya dilakukan secara verbal/lisan, yang artinya komunikasi dengan menggunakan kata-kata. Akan tetapi *The World Health Organization* (WHO) memberikan pernyataan bahwa lebih dari 5% populasi dunia atau 466 juta orang mengalami gangguan pendengaran. Diperkirakan pada tahun 2050 lebih dari 900 juta orang atau satu dari setiap sepuluh orang akan mengalami gangguan pendengaran (Bridget Shield, 2019). Dengan informasi tersebut dapat disimpulkan bahwa akan terjadi peningkatan populasi yang mengalami gangguan pendengaran. Komunitas yang mengalami gangguan pendengaran disebut sebagai tunarungu. Komunitas ini memiliki cara berkomunikasi sendiri tanpa menggunakan bahasa lisan yaitu dengan bahasa isyarat. Sedangkan pada umumnya bahasa isyarat sulit dipahami oleh masyarakat dan membuat komunitas tersebut merasa terasingkan di lingkungannya. Bahasa isyarat itu sendiri merupakan bahasa yang digunakan untuk berkomunikasi dengan menggunakan gerak bibir dan bahasa tubuh, termasuk ekspresi wajah dan pandangan mata. Mengatasi keterbatasan dalam hal berkomunikasi, teknologi *computer vision* bisa menjadi solusi. Teknologi ini mampu menjembatani komunikasi antar manusia melalui mesin. *Computer vision* merupakan salah satu bidang yang memanfaatkan kecerdasan buatan untuk melatih komputer dalam menafsirkan dan memahami dunia visual. Teknologi ini memanfaatkan data dari kamera atau sensor yang kemudian diolah menggunakan model *deep learning*. Sehingga mesin mampu secara akurat mengidentifikasi dan mengklasifikasikan objek kemudian bereaksi terhadap apa yang mesin “lihat”.

Penelitian ini dilakukan dengan pendekatan berbasis *computer vision*. Pada pendekatan tersebut digunakan data yang berasal dari kamera untuk menangkap gerakan bahasa isyarat. Selanjutnya melakukan praproses seperti mengumpulkan data mentah dengan menggunakan kamera, deteksi objek, segmentasi, konversi warna, *reshaping*, *thresholding* dan operasi morfologi. Setelah data berhasil

didapatkan, kemudian melakukan normalisasi untuk pembuatan model dataset. Normalisasi ini memastikan bahwa data yang berhasil dimodelkan bisa dengan efektif dan efisien digunakan oleh sistem cerdas. Semua praproses tersebut dilakukan untuk memisahkan objek yang sangat penting dari *noise*. Sampai pada akhirnya sistem komputer dengan teknologi kecerdasan buatan mampu memanfaatkan model dari dataset tersebut dengan maksimal.

Untuk membuat kecerdasan buatan. Salah satu pendekatan yang bisa digunakan adalah menggunakan Jaringan Syaraf Tiruan yang terinspirasi dari jaringan syaraf pada manusia. Konsep tersebut kemudian dikembangkan lebih lanjut dalam *Deep Learning*. *Deep Learning* telah menjadi salah satu topik hangat dalam dunia *Machine Learning* karena kapabilitasnya yang signifikan dalam memodelkan berbagai data kompleks seperti citra dan suara. Metode *Deep Learning* yang saat ini memiliki hasil paling signifikan dalam pengenalan citra adalah *Convolutional Neural Network* (CNN). Hal tersebut dikarenakan CNN berusaha meniru sistem pengenalan citra pada visual *cortex* manusia sehingga memiliki kemampuan mengolah informasi citra.

Sebernarnya konsep penelitian dan pengembangan sistem komputer vision untuk penerjemah bahasa isyarat sudah ada beberapa yang telah dilakukan antara lain : Sistem Pengenalan Bahasa Isyarat Indonesia Dengan Menggunakan Metode Fuzzy K-Nearest Neighbor (Gafar, 2017), Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode Convolutional Neural Network (Bakti., 2019), Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network (Fadillah, 2020). Dari beberapa penelitian tersebut masih ditemukan keterbatasan dan kekurangan yang bisa diperbaiki seperti kemampuan sistem yang hanya bisa menerjemahkan bahasa isyarat sederhana, input data yang hanya berupa foto dari model peraga yang diambil secara manual atau tidak *realtime* berasal dari video kamera/webcam, tidak melakukan penghilangan *area* wajah dan leher didalam proses pengolahan citra untuk deteksi objek tangan, dan hasil akurasi yang rendah dalam menerjemahkan bahasa isyarat.

Sebagai perbaikan dan pengembangan sistem lebih lanjut, penelitian ini akan menggunakan Bahasa Isyarat Indonesia (BISINDO) sebagai jenis bahasa isyarat

yang dipakai didalam dataset. Selanjutnya peneliti akan menggunakan framework *MediaPipe* untuk mendeteksi objek tangan manusia dan mendapatkan landmark, hal ini bertujuan untuk meningkatkan kemampuan dan akurasi dalam mendeteksi bahasa isyarat. Kemudian pengambilan citra akan dilakukan dengan menggunakan satu kamera *webcam (selfie)* yang nantinya data citra akan diterjemahkan secara *realtime* ketika objek tangan terdeteksi oleh sistem. Pada proses pengolahan data citra dari kamera akan dilakukan proses pengaturan kecerahan gambar, membalik (*flip*) gambar, konversi warna, *reshaping* citra dan *cropping* data citra. Algoritma yang akan digunakan pada penelitian ini adalah CNN yaitu model arsitektur *Convolutional Neural Network*, algoritma ini dipilih berdasarkan tingkat kemampuan dan akurasi untuk menerjemahkan abjad dan nomor, hal ini dibuktikan dari penelitian berjudul “Automatic recognition of Colombian car license plates using convolutional neural networks and Chars74k database” dengan objek input berupa gambar karakter yang diambil dari pelat mobil Kolombia yang memiliki akurasi di atas 98% (D E Arroyo-Pérez., 2020).

1.2 Rumusan Masalah

Berdasarkan uraian dari latar belakang, terdapat beberapa permasalahan yang bisa dirumuskan yaitu sebagai berikut :

1. Bagaimana implementasi *framework MediaPipe* dalam mendeteksi objek tangan pada data citra dari kamera video?
2. Bagaimana proses pengolahan dan normalisasi data citra digital?
3. Bagaimana implementasi metode *Deep Learning Model Convolutional Neural Network* dalam menerjemahkan gerakan bahasa isyarat?

Batasan masalah yang ada dalam penelitian ini yaitu sebagai berikut :

1. Objek penelitian adalah daerah anggota badan manusia dari pergelangan sampai ujung jari tangan.
2. Pencahayaan ruangan harus terang. Objek penelitian berupa tangan harus masuk dalam frame kamera.
3. Kamera yang digunakan minimal memiliki resolusi sebesar 480 x 360 *pixels*.

1.3 Tujuan Penelitian

Adapun tujuan penelitian ini adalah sebagai berikut :

1. Untuk mengetahui implementasi *framework MediaPipe* dalam mendeteksi objek tangan.
2. Untuk mengetahui teknik pengolahan dan normalisasi data citra yang tepat.
3. Untuk mengetahui hasil metode *Deep Learning Model Convolutional Neural Network* dalam memahami dan menerjemahkan gerakan bahasa isyarat.

1.4 Manfaat Penelitian

Adapun manfaat penelitian ini adalah sebagai berikut :

1. Meningkatkan efisiensi komputasi pada sistem sehingga memiliki tingkat fungsional dan akurasi yang baik didalam menerjemahkan bahasa isyarat.
2. Memberikan data yang sesuai dengan lingkungan arsitektur dari machine learning dalam proses data training dan prediksi gerakan bahasa isyarat.
3. Mampu mengembangkan software untuk memudahkan masyarakat luas dalam memahami bahasa isyarat dengan bantuan machine learning.

BAB 2. TINJAUAN PUSTAKA

2.1 State Of The Art

Tabel 2.1 State Of The Art

	Penelitian 1	Penelitian 2	Penelitian 3	Penelitian 4
Judul	Sistem Pengenalan Bahasa Isyarat Indonesia dengan Menggunakan Metode Fuzzy K-Nearest Neighbor	Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode Convolutional Neural Network	Model Penerjemah Bahasa Isyarat Indonesia (BISINDO) Menggunakan Convolutional Neural Network	Sistem Penerjemah Bahasa Isyarat Otomatis Menggunakan Metode Deep Learning Model Convolutional Neural Network (CNN)
Penulis	Agum Agidutama Gafar	Mochamad Bagus Setiyo Bakti dan Yuliana Melita Pranoto	Riestiyya Zain Fadillah	Rizkika Zakka Palindungan
Tahun	2017	2019	2020	2021
Metode	Fuzzy K- Nearest Neighbor (FKNN)	Convolutional Neural Network (CNN)	Convolutional Neural Network (CNN)	Multichannel 2D Convolutional Neural Network (M2D CNN)
Objek	Citra Bahasa Isyarat Tangan (SIBI)	Citra Bahasa Isyarat Tangan (SIBI)	Citra Bahasa Isyarat Tangan (BISINDO)	Citra Bahasa Isyarat Tangan (BISINDO)

2.2 Referensi dan Penelitian Yang Sudah Ada

Konsep penelitian dan pengembangan sistem komputer vision untuk penerjemah bahasa isyarat yang telah dilakukan antara lain :

- a. Sistem Pengenalan Bahasa Isyarat Indonesia Dengan Menggunakan Metode Fuzzy K-Nearest Neighbor.

Proses pengenalan pada Sistem Pengenalan Bahasa Isyarat Indonesia Dengan Menggunakan Metode Fuzzy K-Nearest Neighbor (Gafar, 2017) dibangun menggunakan metode Fuzzy K-Nearest Neighbor (FKNN), metode ini memiliki dua keunggulan utama daripada algoritma K-Nearest Neighbor yaitu mampu mempertimbangkan sifat ambigu dari tetangga jika ada dan membuat sebuah interface akan memiliki derajat nilai keanggotaan pada setiap kelas sehingga akan lebih memberikan kekuatan atau kepercayaan suatu instance yang berada pada suatu kelas. Dengan menerapkan metode Fuzzy K-Nearest Neighbor (FKNN) pada proses klasifikasi bahasa isyarat, maka proses klasifikasi bisa dilakukan dengan lebih objektif. Penelitian ini memilih metode Fuzzy K-Nearest Neighbor (FKNN) untuk pengenalan Sistem Isyarat Bahasa Indonesia (SIBI). Hasil Sistem pengenalan bahasa isyarat Indonesia dengan menggunakan metode Fuzzy K-Nearest Neighbor (KNN) diperoleh nilai akurasi sebesar 88,8%. Dengan melakukan percobaan sebanyak 5 kali, masih ada beberapa huruf yang konsisten belum bisa dikenali, seperti huruf C, E, L, U dan V yang cenderung memiliki nilai kemiripan yang dekat.

- b. Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode Convolutional Neural Network.

Dalam Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode Convolutional Neural Network (Bakti., 2019) membahas pengenalan isyarat angka SIBI dengan menggunakan metode Convolutional Neural Network (CNN). Arsitektur CNN yang digunakan adalah arsitektur LeNet. Arsitektur CNN diproses dalam 3 tahap, 25 epoch, 50 epoch dan 100 epoch. Berdasarkan percobaan yang dilakukan nilai akurasi yang didapat terus meningkat dalam tiap tahapnya, mulai 67.66%,

89.44% sampai nilai akurasi tertinggi dalam proses training sebesar 96.44%. Begitupun dalam proses prediksi data juga mengalami kenaikan dalam tiap tahapnya, mulai 79.23%, 90.45% sampai didapatkan nilai akurasi tertinggi dalam prediksi data 98.89%. Dari total 90 data testingset, hanya sekali keasalahan dalam prediksi data.

- c. Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network.

Penelitian Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network (Fadillah, 2020) menghasilkan model penerjemah alfabet Bisindo menggunakan implementasi algoritma CNN dengan akurasi 94.38% dan 30% sebagai upaya untuk meningkatkan aksesibilitas Tuli pengguna Bisindo. Model penerjemah tersebut merupakan model terpilih dari dua model yang telah dikembangkan dengan menggunakan pendekatan yang berbeda. Model A dikembangkan menggunakan pemindahan arsitektur dari source model (Model ASL) dan Model B dikembangkan menggunakan pemindahan arsitektur beserta learning parameter yang tersimpan dalam source model tersebut. Dengan percobaan yang dilakukan dalam penelitian ini, pemindahan learning parameter dari Model ASL kepada Model B mengurangi durasi learning yang signifikan namun menyebabkan Model B hanya dapat mempelajari kemiripan fitur antara source dan target domain (huruf yang memiliki kemiripan antara ASL dan Bisindo). Oleh karena itu, pemindahan arsitektur tanpa menggunakan learning parameter merupakan metode pengembangan yang sesuai untuk kasus seperti ini, sehingga parameter-transfer masih belum dapat mengatasi keberagaman gestur ASL dan Bisindo.

2.3 Kekurangan Penelitian Sebelumnya

Dari penelitian dan karya tulis yang telah ada sebelumnya masih ditemukan keterbatasan dan kekurangan. Hal tersebut akan dijadikan sebagai acuan untuk

melakukan pengembangan dan penelitian yang lebih lanjut. Sehingga menjadi gagasan, ide dan inovasi dalam penelitian baru.

- a. Sistem Pengenalan Bahasa Isyarat Indonesia Dengan Menggunakan Metode Fuzzy K-Nearest Neighbor.

Dalam penelitian ini ditemukan beberapa kekurangan yang bisa dikembangkan untuk penelitian selanjutnya, yaitu sebagai berikut :

1. Tingkat akurasi dalam melakukan prediksi masih rendah meskipun bahasa isyarat yang dipakai sederhana.
2. Tidak ada objek deteksi dalam proses pengolahan citra.
3. Metode ekstraksi fitur sangat simple dan sedikit sehingga data / informasi yang diambil dari citra tidak efisien dan relevan sebagai parameter dalam proses klasifikasi.
4. Input citranya masih dalam bentuk file gambar / manual tidak secara *realtime* dengan menggunakan video kamera

- b. Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode Convolutional Neural Network.

Dalam penelitian ini ditemukan beberapa kekurangan yang bisa dikembangkan untuk penelitian selanjutnya, yaitu sebagai berikut :

1. Bahasa isyarat yang digunakan masih sederhana yaitu hanya bahasa isyarat angka dengan menggunakan satu tangan.
2. Sudut pandang kamera hanya satu yaitu dari depan objek / model peraga.
3. Input citranya masih dalam bentuk file gambar / manual tidak secara *realtime* dengan menggunakan video kamera

- c. Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network.

Dalam penelitian ini ditemukan beberapa kekurangan yang bisa dikembangkan untuk penelitian selanjutnya, yaitu sebagai berikut :

1. Metode ekstraksi fitur sangat simple dan sedikit serta input citra hanya dalam bentuk gambar tangan tidak menggunakan gambar model *full body*.

2. Input citranya masih dalam bentuk file gambar / manual tidak secara *realtime* dengan menggunakan video kamera.

2.4 Kelebihan Dari Penelitian Yang Diambil

Dalam proses pengembangan dan penelitian yang lebih lanjut. Peneliti akan melakukan upaya untuk memperbaiki kekurangan dari penelitian-penelitian sebelumnya. Adapun upaya tersebut dapat dijabarkan sebagai berikut :

- a. Input data citra yang digunakan dalam sistem akan diambil dan klasifikasi / terjemahkan secara *realtime* langsung dari kamera input atau webcam.
- b. Proses pengolahan citra akan lebih kompleks dan ditambah dengan adanya deteksi objek (MediaPipe) untuk menghilangkan *noise*, *background*, area wajah dan leher untuk memfokuskan dalam pengambilan objek tangan.
- c. Algoritma yang akan digunakan pada penelitian ini menggunakan model dari CNN yaitu model arsitektur *Convolutional Neural Network*, algoritma ini memiliki kemampuan dan akurasi untuk mengolah data input dalam bentuk 2 Dimensi.

2.5 Bahasa Isyarat

Bahasa isyarat merupakan jenis komunikasi non verbal karena merupakan bahasa yang tidak menggunakan suara tetapi menggunakan bentuk dan arah tangan, pergerakan tangan, bibir, badan serta ekspresi wajah untuk menyampaikan maksud dan pikiran dari seorang penutur. Kaum tunarungu adalah kelompok utama yang menggunakan bahasa ini. Bahasa isyarat biasanya pengkombinasian dari bentuk, orientasi dan gerak tangan, lengan, tubuh serta ekspresi wajah untuk mengungkapkan isi pikiran (Zuhir dan Amri, 2019).

Di Indonesia, ada dua sistem dari bahasa isyarat yang digunakan yaitu: Bahasa Isyarat Indonesia (BISINDO) dan Sistem Isyarat Bahasa Indonesia (SIBI). SIBI adalah bahasa isyarat yang diperkenalkan secara awal oleh Alm. Anton Widyatmoko yang merupakan mantan kepala sekolah SLB/B Widya Bakti di Semarang dalam proses penciptaannya tidak melalui musyawarah dan persetujuan dari Gerakan Kesejahteraan Tunarungu Indonesia (GERKATIN) tetapi

berkolaborasi dengan mantan kepala sekolah SLB/B di Jakarta dan Surabaya dengan hasil akhir sebuah kamus SIBI. BISINDO adalah bahasa isyarat yang mengadopsi nilai budaya asli Indonesia dan mudah dapat digunakan untuk berkomunikasi diantara kaum tunarungu dalam kehidupan sehari - hari. Kecepatan dan kepraktisannya dari BISINDO membuat lebih mudah untuk memahami dan mengerti bagi kaum tunarungu walaupun tidak mengikuti faedah tata bahasa dari bahasa Indonesia (Yunanda, 2018).

2.5.1 Sejarah Bahasa Isyarat Indonesia (BISINDO)

Menurut Laura Lesmana Wijaya selaku Ketua Pusat Bahasa Isyarat Indonesia (Pusbisindo), Bisindo dapat diartikan sebagai sebuah terminologi yang digunakan untuk menunjuk pada bahasa isyarat alami yang digunakan oleh komunitas Tuli di Indonesia. Berdasarkan penjelasan tersebut dapat disimpulkan bahwa sejarah Bisindo sejalan dengan kemunculan bahasa isyarat alami yang terdapat di Indonesia (Yohans, Arjawa dan Punia., 2019).

Kemunculan bahasa isyarat alami diyakini telah berlangsung sejak tahun 1933 ketika sekolah khusus Tuli pertama yaitu Sekolah Luar Biasa (SLB)/B Cicendo, Bandung, Jawa Barat berdiri. Selain itu, terdapat pula sekolah khusus Tuli lainnya yang berdiri pada tahun-tahun berikutnya seperti SLB/B Dena Upakara, Wonosobo, Jawa Tengah (sekolah khusus perempuan) pada tahun 1938, SLB/B Don Bosco, Wonosobo, Jawa tengah (sekolah khusus laki- laki) pada tahun 1955, dan SLB/B Santi Rama (Jakarta) pada tahun 1970 (Tim Produksi Bahasa Isyarat Jakarta, 2014: vii). Penjelasan ini diperkuat dengan keberadaan bahasa isyarat Jakarta yang variasinya berasal dari pencampuran bahasa isyarat asli, termasuk bahasa isyarat yang digunakan oleh orang-orang Tuli yang pernah mendapatkan pendidikan formal di sekolah khusus Tuli tersebut (Tim Produksi Bahasa Isyarat Jakarta, 2014: vii).

Perkembangan bahasa isyarat alami di Indonesia tidak serta merta mendapatkan pengakuan oleh pemerintah Indonesia. Sistem Isyarat Bahasa Indonesia (SIBI) merupakan sarana komunikasi yang terlebih dahulu diakui oleh pemerintah Indonesia. Pengakuan dan pembakuan atas penggunaan SIBI secara

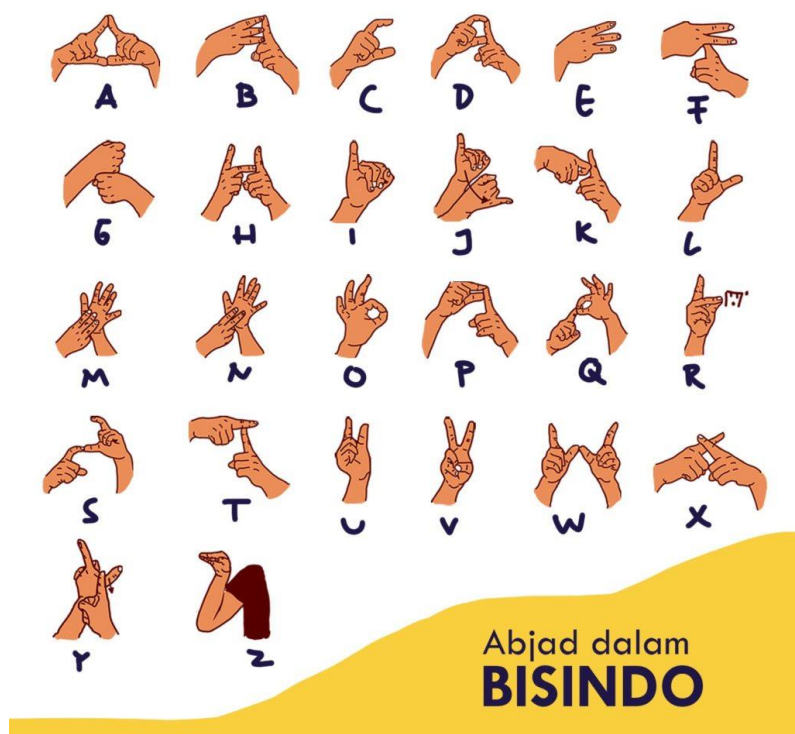
resmi ditetapkan pada tahun 1994 melalui Keputusan Mendikbud RI Nomor 0161/U/1994.

Disebarluaskan dan dibakukannya penggunaan SIBI sebagai sistem isyarat buatan yang bersifat nasional tidak sepenuhnya diterima oleh komunitas Tuli. Komunitas Tuli menilai bahwa keberadaan SIBI tidak merepresentasikan bahasa isyarat asli Indonesia, terdapat berbagai bentuk isyarat yang tidak sesuai dengan isyarat yang berkembang di komunitas Tuli. Salah satu isyarat yang banyak diterapkan pada kamus SIBI, yaitu isyarat yang terdapat pada sistem isyarat American Sign Language (ASL).

Permasalahan dan pertentangan atas penggunaan SIBI menjadi salah satu faktor yang memengaruhi munculnya penggunaan istilah Bisindo. Laura Lesmana Wijaya menjelaskan bahwa penggunaan istilah Bahasa Isyarat Indonesia atau disingkat Bisindo dimulai pada awal tahun 2000. Istilah tersebut muncul melalui pelaksanaan kongres yang dilaksanakan oleh Gerkatin. Penentuan istilah tersebut digunakan untuk menunjuk pada bahasa isyarat alami yang berkembang di komunitas Tuli.

2.5.2 Isyarat Abjad Dalam Bisindo

Berikut adalah gambar gerakan bahasa isyarat dari abjad mulai A hingga Z yang diterjemahkan kedalam Bahasa Isyarat Indonesian (BISINDO).



Gambar 2.1 Gesture Bahasa Isyarat Abjad (BISINDO)

2.5.3 Isyarat Nomor Dalam Bisindo

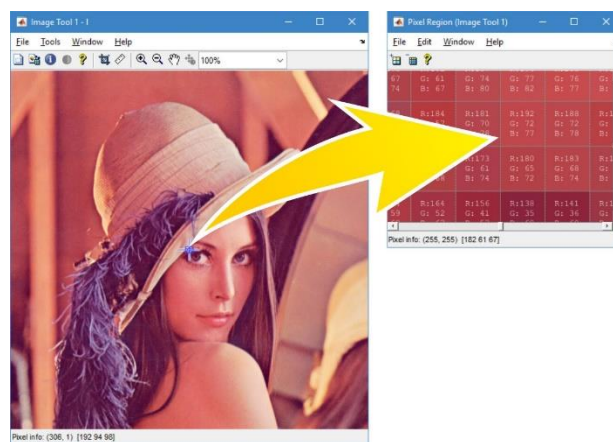
Berikut adalah gambar gerakan bahasa isyarat dari nomor mulai 1 hingga 10 yang diterjemahkan kedalam Bahasa Isyarat Indonesian (BISINDO).



Gambar 2.2 Gesture Bahasa Isyarat Angka (BISINDO)

2.6 Citra Digital

Citra digital merupakan representasi dari fungsi intensitas cahaya dalam bentuk diskrit pada bidang dua dimensi. Citra tersusun oleh sekumpulan piksel (*picture element*) yang memiliki koordinat (x,y) dan amplitudo $f(x,y)$. Koordinat (x,y) menunjukkan letak / posisi piksel dalam suatu citra, sedangkan amplitudo $f(x,y)$ menunjukkan nilai intensitas warna citra. Representasi citra digital beserta piksel penyusunnya ditunjukkan pada Gambar berikut ini.



Gambar 2.3 Citra dan piksel penyusunnya

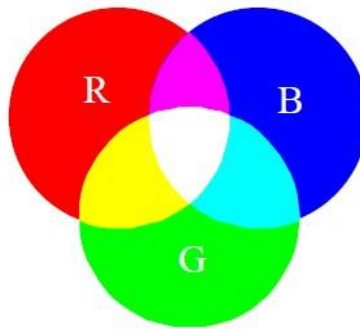
2.7 Jenis Citra Digital

Nilai suatu *pixel* memiliki nilai dalam rentang tertentu, dari nilai minimum sampai nilai maksimum. Jangkauan yang digunakan berbedabeda tergantung dari jenis warnanya. Berikut adalah jenis-jenis citra berdasarkan nilai *pixel*-nya, sebagai berikut:

2.7.1 Citra RGB

RGB (*Red, Green, Blue*) merupakan model perpaduan warna yang didasari pada tiga warna dasar yaitu *Red, Green, Blue* yang kemudian dikombinasikan bersama- sama untuk menghasilkan perpaduan warna.

Jangkauan warna RGB adalah mulai dari range 0 – 255. Penggunaan warna RGB untuk warna yang berbasiskan elektronik seperti *Camera*, Komputer, Televisi dan berbagai gadget lainnya.



Gambar 2.4 Warna RGB

Dari range warna yang dihasilkan oleh masing – masing $R = 0 - 255$, $G = 0-255$ dan $B = 0 - 255$ yang apabila digabungkan komposisinya menghasilkan warna yang baru. Sehingga warna yang mampu dihasilkan oleh ketiga kombinasi di atas adalah 256^3 sebanyak 16.777.216 kombinasi warna. Y. Ming (1988) yang memperkenalkan sebuah metode Normalisasi RGB. Dimana warna dari sebuah pixel diproporsikan dengan semua nilai RGB. Konsep ini digunakan untuk mengatasi adanya perbedaan intentitas yang terdapat pada objek yang sama. Y. Ming merumuskan konsep Normalisasi RGB ini sebagai berikut :

$$r = \frac{R}{R+G+B} \quad 2.1$$

$$g = \frac{G}{R+G+B} \quad 2.2$$

$$b = \frac{B}{R+G+B} \quad 2.3$$

Selanjutnya Michael J. Jones James M. Rehg pada tahun 1999 memperkenalkan warna model histogram untuk membedakan antara warna kulit manusia atau bukan warna kulit manusia.

2.7.2 Citra Grayscale

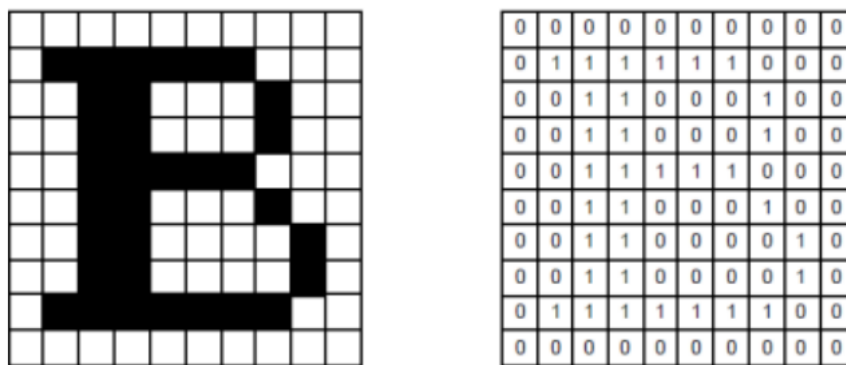
Citra grayscale mempunyai kemungkinan warna hitam untuk nilai minimal dan warna putih untuk nilai maksimal. Banyaknya warna tergantung pada jumlah bit yang disediakan di memori untuk menampung kebutuhan warna tersebut. Semakin besar jumlah bit warna yang disediakan di memori, maka semakin halus gradasi warna yang terbentuk. Nilai warna untuk Grayscale (derajat keabuan) adalah nilai warna yang menunjukkan kehitaman dengan nilai $28 - 1 = 255$ untuk image 8 bit.



Gambar 2.5 Warna Grayscale

2.7.3 Citra Biner

Citra biner adalah citra yang hanya mempunyai dua nilai derajat keabuan, hanya hitam dan putih. Piksel-piksel objek bernilai 1 dan piksel-piksel latar belakang bernilai 0. Pada waktu menampilkan gambar, 1 adalah putih sedangkan 0 adalah hitam (Erwin, Dedi, dan Ikhwan., 2015).



Gambar 2.6 Huruf “B” dan citra dalam biner

2.8 Pengolahan Citra Digital

Pengolahan Citra Digital adalah merupakan proses yang bertujuan untuk memanipulasi dan menganalisis citra dengan bantuan komputer. Baik citra yang berdimensi 2 atau citra 3 dimensi. Kegiatan yang dilakukan dalam pengolahan citra dibagi menjadi dua bagian, yang pertama perbaikan kualitas terhadap sebuah citra agar mata manusia mampu menginterpretasi dengan baik. Perbaikan ini termasuk Image Enhancement agar mendapatkan citra yang lebih baik dari citra sebelum dilakukan pengolahan. Dan Kedua pengolahan citra bekerja untuk mendapatkan dan mengolah informasi yang terdapat pada suatu citra untuk keperluan pengenalan objek secara otomatis. Sebagai contoh aplikatifnya, image detection, skin detection, Pengenalan Pola dan masih banyak lainnya.

2.8.1 Preprocessing

Preprocessing adalah proses awal dilakukannya perbaikan suatu citra untuk menghilangkan noise. *Preprocessing* merupakan suatu proses untuk menghilangkan bagian-bagian yang tidak diperlukan pada gambar input untuk proses selanjutnya. Beberapa proses yang dapat dilakukan pada tahap *preprocessing* antara lain, proses binerisasi, segmentasi, dan normalisasi.

2.8.2 Segmentasi citra

Segmentasi citra adalah proses pengolahan citra yang bertujuan memisahkan wilayah (*region*) objek dengan wilayah latar belakang agar objek mudah dianalisis dalam rangka mengenali objek yang banyak melibatkan persepsi visual (Destyningtias, 2010).

2.8.3 Normalisasi

Normalisasi adalah proses mengubah ukuran citra, baik menambah atau mengurangi, menjadi ukuran yang ditentukan tanpa menghilangkan informasi penting dari citra tersebut. Dengan adanya proses normalisasi maka ukuran semua citra yang akan diproses menjadi seragam.

2.9 Visi komputer

Visi komputer (*Computer Vision*) adalah salah satu teknologi yang paling banyak dipakai pada zaman ini. Teknologi visi komputer ini merupakan salah satu bidang dari teknologi *Artificial Intelligence*. Visi komputer juga merupakan dan kumpulan dari metode-metode untuk mendapatkan, memproses, menganalisis suatu gambar atau dalam arti lain visi komputer, merupakan kumpulan metode-metode yang digunakan untuk menghasilkan angka-angka atau simbol-simbol yang didapat dari gambar yang diambil dari dunia nyata agar komputer dapat mengerti apa makna dari gambar tersebut. Inti dari teknologi visi komputer adalah untuk menduplikasi kemampuan penglihatan manusia kedalam benda elektronik sehingga benda elektronik dapat memahami dan mengerti arti dari gambar yang dimasukkan (Milan Sonka, Vaclav Hlavac and Roger Boyle - 2008).

2.10 Gradient Descent

Gradient Descent merupakan salah satu algoritma yang paling populer dalam melakukan optimasi pada model jaringan syaraf tiruan (*Artificial neural network* / ANN). Algoritma ini adalah cara yang paling sering dipakai dalam berbagai macam model pembelajaran. Ketika akan melatih sebuah model, akan dibutuhkan sebuah *loss function* yang dapat memungkinkan peneliti untuk mengukur kualitas dari setiap bobot atau parameter tertentu. Tujuan dari pengoptimalan ini yaitu untuk menentukan parameter manakah yang mampu meminimalkan *loss function* (Ruder, 2017).

Gradient Descent bekerja dengan meminimalkan fungsi (θ) yang mempunyai parameter θ dengan memperbarui parameter ke suatu arah yang menurun. *Gradient descent* mempunyai *learning rate* (η) yang digunakan untuk menentukan langkah yang akan diambil untuk mencapai pada titik minimum. Hal ini, dapat digambarkan bahwa suatu objek akan seperti menuruni sebuah bukit dengan langkah tersebut sehingga mencapai pada bagian lembah (titik minimum). *Stochastic Gradient Descent* (SGD) merupakan metode *gradient descent* yang melakukan *update parameter* untuk setiap data pelatihan $x(i)$ dan label $y(i)$ dan mempunyai persamaan dasar berikut :

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^i; y^i) \quad 2.4$$

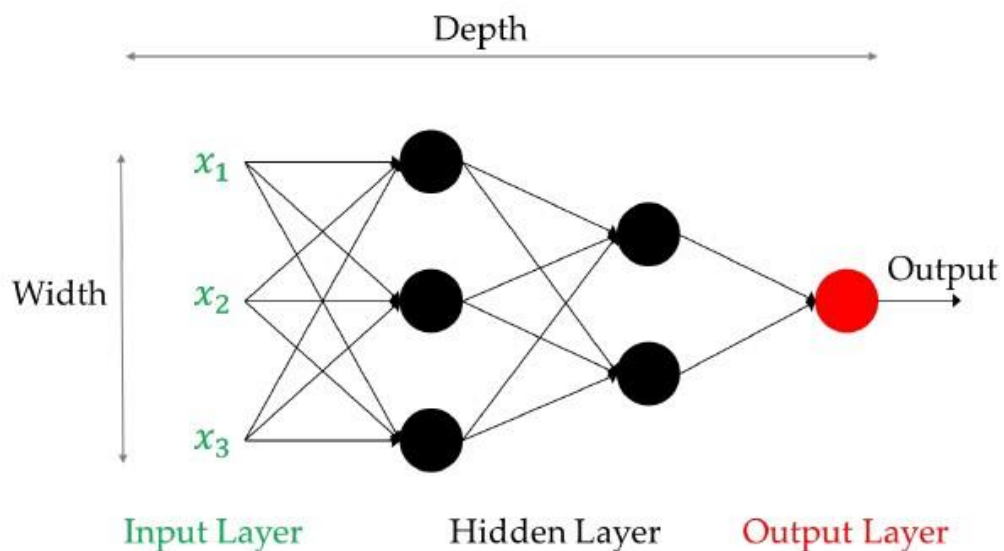
SGD seringkali melakukan *update*/pembaruan dengan varians yang tinggi, sehingga menyebabkan fungsi objektif meningkat secara tidak beraturan. Di satu sisi, hal ini dapat membuat *loss function* melompat ke titik minimal yang baru dan mempunyai potensi untuk melompat ke nilai minimum yang tidak pasti. Namun, hal ini dapat dicegah dengan mengurangi nilai *learning rate*, dan hasil SGD akan menuruni *loss function* ke titik minimum dengan optimal.

2.11 Artificial Neural Network

Artificial neural network adalah salah satu algoritma *supervised learning* yang populer dan bisa juga digunakan untuk *semi-supervised* atau *unsupervised learning* (Amir Atiya, 1994). Walaupun tujuan awalnya adalah untuk mensimulasikan jaringan saraf biologis, jaringan tiruan ini sebenarnya simulasi

yang terlalu disederhanakan, artinya simulasi yang dilakukan tidak mampu menggambarkan kompleksitas jaringan biologis manusia.

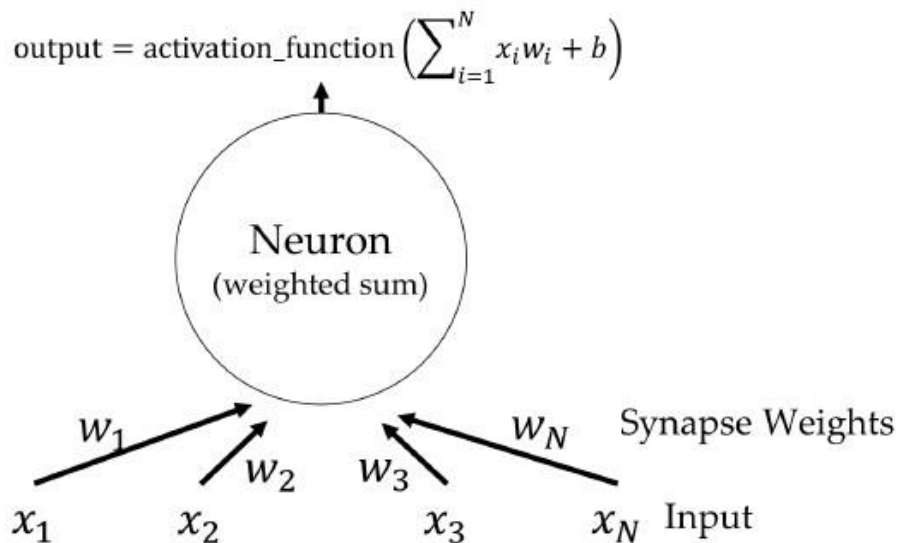
Artificial Neural Network (selanjutnya disingkat ANN), menghasilkan model yang sulit dibaca dan dimengerti oleh manusia karena memiliki banyak layer (kecuali *single perceptron*) dan sifat **non-linear** (merujuk pada fungsi aktivasi). Pada bidang riset ini, ANN disebut agnostik - kita percaya, tetapi sulit membuktikan kenapa konfigurasi parameter yang dihasilkan *training* bisa benar. Konsep matematis ANN itu sendiri cukup *solid*, tetapi *interpretability* (tingkat pemahaman dan 'insight') model rendah menyebabkan kita tidak dapat menganalisa proses inferensi(penyimpulan) yang terjadi pada model ANN. Secara matematis, ANN ibarat sebuah graf. ANN memiliki neuron/*node* (*vertex*), dan sinapsis (*edge*). Karena memiliki struktur seperti graf, operasi pada ANN mudah dijelaskan dalam notasi aljabar linear. Sebagai gambaran, ANN berbentuk seperti Gambar 2.7 (*deep neural network*, salah satu varian arsitektur). *Depth* (kedalaman) ANN mengacu pada jumlah layer. Sementara *width* (lebar) ANN mengacu pada jumlah unit pada *layer*.



Gambar 2.7 Deep Neural Network

2.11.1 Single Perceptron

Bentuk terkecil (minimal) sebuah ANN adalah *single perceptron* yang hanya terdiri dari sebuah neuron. Sebuah neuron diilustrasikan pada Gambar 2.8. Secara matematis, terdapat *feature vector* x yang menjadi *input* bagi neuron tersebut (*feature vector* merepresentasikan suatu *data point*, *event* atau *instans*). Neuron akan memproses *input* x melalui perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*, yang dilewatkan pada fungsi non linear. Pada *training*, yang dioptimasi adalah nilai *synapse weight* (*learning parameter*). Selain itu, terdapat juga bias b sebagai kontrol tambahan (materi *steepest gradient descent*). Output dari neuron adalah hasil fungsi aktivasi dari perhitungan jumlah perkalian antara nilai *input* dan *synapse weight*. Ada beberapa macam fungsi aktivasi, misal *step function*, *sign function*, *rectifier* dan *sigmoid function*. Bila diplot menjadi grafik, fungsi ini memberikan bentuk seperti huruf S.



Gambar 2.8 Single Perceptron

$$\sigma(u) = \frac{1}{1+e^{-u}} \quad 2.5$$

Perhatikan kembali, Gambar 2.8 sesungguhnya adalah operasi aljabar linear. Single perceptron dapat dituliskan kembali sebagai berikut :

$$o = f(x \cdot w + b) \quad 2.6$$

Dimana o adalah *output* dan f adalah fungsi non-linear yang dapat diturunkan secara matematis (*differentiable non-linear function* - selanjutnya disebut “fungsi non linear” saja.). Bentuk ini tidak lain dan tidak bukan adalah persamaan model linear yang ditransformasi dengan fungsi non-linear. Secara filosofis, ANN bekerja mirip dengan model linear, yaitu mencari *decision boudary*. Apabila beberapa model non-linear ini digabungkan, maka kemampuannya akan menjadi lebih hebat. Yang menjadikan ANN "spesial" adalah penggunaan fungsi non-linear.

Untuk melakukan pembelajaran *single perceptron*, *training* dilakukan menggunakan ***perceptron training rule***. Prosesnya sebagai berikut:

1. Inisiasi nilai *synapse weights*, bisa *random* ataupun dengan aturan tertentu.
2. Lewatkan input pada neuron, kemudian kita akan mendapatkan nilai *output*. Kegiatan ini disebut ***feedforward***.
3. Nilai *output* (*actual output*) tersebut dibandingkan dengan *desired output*.
4. Apabila nilai *output* sesuai dengan *desired output*, tidak perlu mengubah apa-apa.
5. Apabila nilai *output* tidak sesuai dengan *desired output*, hitung nilai *error* (*loss*) kemudian lakukan perubahan terhadap *learning parameter* (*synapse weight*).
6. Ulangi langkah-langkah ini sampai tidak ada perubahan nilai *error*, nilai *error* kurang dari sama dengan suatu *threshold* (biasanya mendekati 0), atau sudah mengulangi proses latihan sebanyak **T** kali (*threshold*).

Error function diberikan pada persamaan 2.7 (dapat diganti dengan absolute value) dan perubahan *synapse weight* diberikan pada persamaan 2.8, dimana y melambangkan *desired output*, $o = f(x \cdot w + b)$ melambangkan actual output untuk x sebagai input. μ disebut sebagai learning rate.

$$E(w) = (y - o)^2 \quad 2.7$$

$$\Delta w_i = \mu(y - o)x_1 \quad 2.8$$

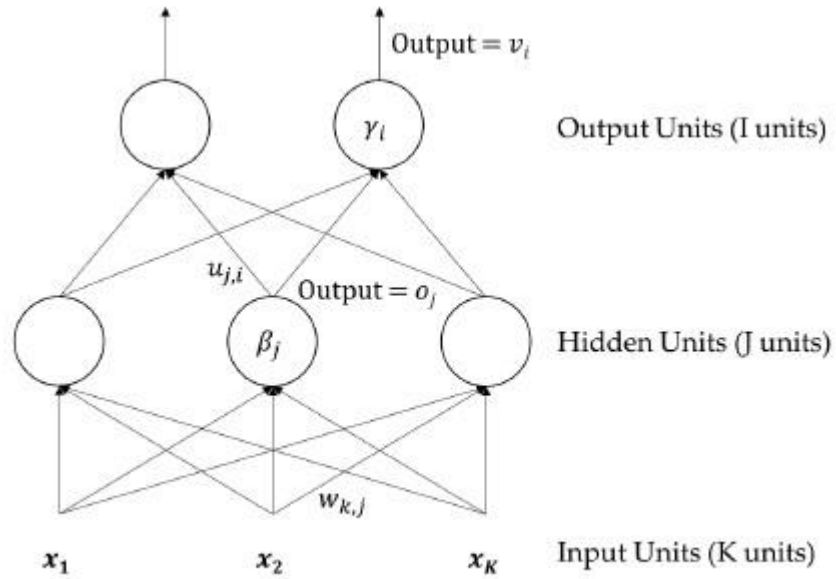
Hasil akhir pembelajaran (*learning*) adalah konfigurasi *synapse weight*. Saat klasifikasi, kita melewatkan *input* baru pada jaringan yang telah dibangun, kemudian tinggal mengambil hasilnya. Pada contoh kali ini, seolah-olah *single perceptron* hanya dapat digunakan untuk melakukan *binary classification* (hanya ada dua kelas, nilai 0 dan 1). Untuk multi-class classification, kita dapat

menerapkan berbagai strategi, misal thresholding, i.e., nilai output 0 - 0.2 mengacu pada kelas pertama, 0.2 - 0.4 untuk kelas kedua, dst.

2.11.2 Multilayer Perceptron

Multilayer perceptron (MLP) yang juga dikenal sebagai *feedforward neural network* secara literal memiliki beberapa *layers*. Pada lecture note ini, secara umum ada tiga *layers*: *input*, *hidden*, dan *output layer*. *Input layer* menerima *input* (tanpa melakukan operasi apapun), kemudian nilai *input* (tanpa dilewatkan ke fungsi aktivasi) diberikan ke *hidden units*. Pada *hidden units*, *input* diproses dan dilakukan perhitungan hasil fungsi aktivasi untuk tiap-tiap neuron, lalu hasilnya diberikan ke layer berikutnya. *Output* dari *input layer* akan diterima sebagai *input* bagi *hidden layer*. Begitupula seterusnya *hidden layer* akan mengirimkan hasilnya untuk *output layer*.

Kegiatan ini dinamakan *feed forward*. Hal serupa berlaku untuk *artificial neural network* dengan lebih dari tiga *layers*. Parameter neuron dapat dioptimisasi menggunakan metode *gradient-based optimization*. Perlu diperhatikan, MLP adalah gabungan dari banyak fungsi *non-linear*. Gabungan banyak fungsi *non-linear* ini lebih hebat dibanding single perceptron. Masing-masing neuron terkoneksi dengan semua neuron pada *layer* berikutnya. Konfigurasi ini disebut sebagai ***fully connected***. MLP pada umumnya menggunakan konfigurasi *fully connected*.



Gambar 2.9 Multilayer Perceptron

$$o_j = \sigma\left(\sum_{k=1}^K x_k w_{k,j} + \beta_j\right) \quad 2.9$$

$$v_i = \sigma\left(\sum_{j=1}^J o_j u_{j,i} + \gamma_i\right) = \sigma\left(\sum_{j=1}^J \sigma\left(\sum_{k=1}^K x_k w_{k,j} + \beta_j\right) u_{j,i} + \gamma_i\right) \quad 2.10$$

Perhatikan persamaan 2.9 dan 2.10 untuk menghitung *output* pada *layer* yang berbeda. u, w adalah *learning parameters*. β, γ melambangkan *noise* atau *bias*. K adalah banyaknya *input units* dan J adalah banyaknya *hidden units*. Persamaan 2.10 dapat disederhanakan penulisannya sebagai persamaan 2.11. Persamaan 2.11 terlihat relatif lebih “elegant”. Sehingga ANN dapat direpresentasikan dengan notasi operasi aljabar.

$$V = \sigma(oU + \gamma) = \sigma(\sigma(xU + \beta)U + \gamma) \quad 2.11$$

Untuk melatih MLP, algoritma yang umumnya digunakan adalah **backpropagation**. Arti kata *backpropagation* sulit untuk diterjemahkan ke dalam bahasa Indonesia. Peneliti memperbaharui parameter (*synapse weights*) secara bertahap (dari *output* ke *input* layer, karena itu disebut *backpropagation*) berdasarkan *error/loss* (*output* dibandingkan dengan *desired output*). Intinya adalah mengoreksi *synapse weight* dari output layer ke *hidden layer*, kemudian *error* tersebut dipropagasi ke layer sebelum-sebelumnya. Artinya, perubahan *synapse weight* pada suatu layer dipengaruhi oleh perubahan *synapse weight* pada layer

setelahnya. Backpropagation tidak lain dan tidak bukan adalah metode *gradient-based optimization* yang diterapkan pada ANN.

Pertama-tama diberikan pasangan *input* (x) dan *desired output* (y) sebagai *training data*. Untuk meminimalkan *loss*, algoritma *backpropagation* menggunakan prinsip *gradient descent*. Cara menurunkan *backpropagation* menggunakan teknik *gradient descent*, yaitu menghitung *loss* ANN pada Gambar 2.9 yang menggunakan fungsi aktivasi sigmoid.

Error, untuk MLP diberikan oleh persamaan 2.12 (untuk satu data point), dimana I adalah banyaknya *output unit* dan θ adalah kumpulan *weight matrices* (semua *parameter* pada MLP). Kami ingatkan kembali perhitungan *error* bisa juga menggunakan nilai absolut.

$$E(\theta) = \frac{1}{2} \sum_{i=1}^I (y_i - v_i)^2 \quad 2.12$$

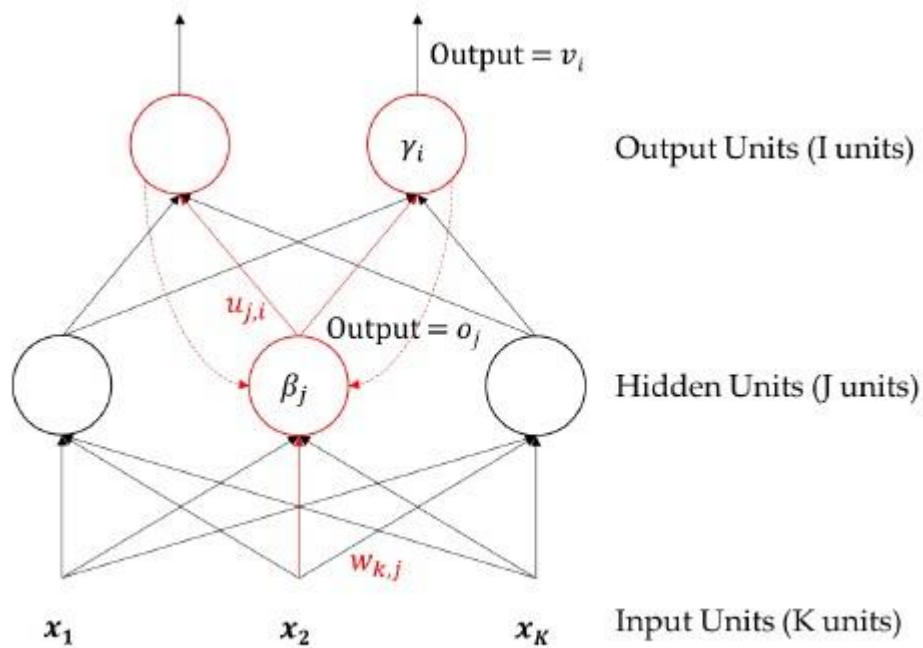
Proses penurunan untuk melatih MLP, *error/loss* diturunkan terhadap tiap learning parameter. Diferensial $u_{j,i}$ diberikan oleh turunan sigmoid function.

$$\begin{aligned} \frac{\delta E(\theta)}{\delta u_{j,i}} &= (y_i - v_i) \frac{\delta v_i}{\delta u_{j,i}} \\ &= (y_i - v_i) v_i (1 - v_i) o_j \end{aligned} \quad 2.13$$

Diferensial $w_{k,j}$ diberikan oleh turunan *sigmoid function*

$$\begin{aligned} \frac{\delta E(\theta)}{\delta w_{k,j}} &= \sum_{i=1}^I (y_i - v_i) \frac{\delta v_i}{\delta w_{k,j}} \\ &= \sum_{i=1}^I (y_i - v_i) \frac{\delta v_i}{\delta o_j} \frac{\delta o_j}{\delta w_{k,j}} \\ &= \sum_{i=1}^I (y_i - v_i) (v_i (1 - v_i) u_{j,i}) (o_j (1 - o_j) x_k) \end{aligned} \quad 2.14$$

Perhatikan, *diferensial* $w_{k,j}$ memiliki \sum sementara $u_{j,i}$ tidak ada. Hal ini disebabkan karena $u_{j,i}$ hanya berkorespondensi dengan satu *output* neuron. Sementara $w_{k,j}$ berkorespondensi dengan banyak *output* neuron. Dengan kata lain, nilai $w_{k,j}$ mempengaruhi hasil operasi yang terjadi pada banyak *output* neuron, sehingga banyak neuron mempropagasi error kembali ke $w_{k,j}$. Ilustrasi diberikan pada Gambar 2.10.



Gambar 2.10 Multilayer Perceptron 2

Metode penurunan serupa dapat juga digunakan untuk menentukan perubahan β dan γ . Jadi proses *backpropagation* untuk kasus Gambar 2.9 dapat diberikan seperti pada Gambar 2.11 dimana η adalah *learning rate*. Untuk *artificial neural network* dengan lebih dari 3 layers juga bisa menurunkan persamaannya. Secara umum, proses melatih ANN (apapun variasi arsitekturnya) mengikuti *framework perceptron training rule*.

(2) Hidden to Output

$$v_i = \sigma \left(\sum_{j=1}^I o_j u_{j,i} + \gamma_i \right)$$

(3) Output to Hidden

$$\delta_i = (y_i - v_i)v_i(1 - v_i)$$

$$\Delta u_{j,i} = -\eta(t)\delta_i o_j$$

$$\Delta \gamma_i = -\eta(t)\delta_i$$

(1) Input to Hidden Layer

$$o_j = \sigma \left(\sum_{k=1}^K x_k w_{k,j} + \beta_j \right)$$

(4) Hidden to Input

$$\varphi_j = \sum_{i=1}^I \delta_i u_{j,i} o_j (1 - o_j)$$

$$\Delta w_{k,j} = -\eta(t)\varphi_j x_k$$

$$\Delta \beta_j = -\eta(t)\varphi_j$$

Gambar 2.11 Proses latihan menggunakan backpropagation

2.11.3 Multi-class Classification

Multilayer perceptron dapat memiliki *output unit* berjumlah lebih dari satu. Seumpama mempunyai empat kelas, dengan demikian peneliti dapat merepresentasikan keempat kelas tersebut sebagai empat *output units*. Kelas pertama direpresentasikan dengan *unit* pertama, kelas kedua dengan *unit* kedua, dst. Untuk C kelas, kita dapat merepresentasikannya dengan C *output units*. Kita dapat merepresentasikan data harus dimasukkan ke kelas mana menggunakan *sparse vector*, yaitu bernilai 0 atau 1. Elemen ke-i bernilai 1 apabila data masuk ke kelas c_i , sementara nilai elemen lainnya adalah 0 (ilustrasi pada Gambar 2.12). Output ANN dilewatkan pada suatu fungsi softmax yang melambangkan probabilitas *class-assignment*, i.e., kita ingin output agar semirip mungkin dengan *sparse vector* (*desired output*). Pada kasus ini, *output* ANN adalah sebuah distribusi yang melambangkan input di-assign ke kelas tertentu. Cross entropy cocok digunakan sebagai utility function ketika output berbentuk distribusi.

$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Kelas pertama
	Kelas kedua
	Kelas ketiga
	Kelas keempat

Gambar 2.12 Ilustrasi desired output pada multi-class classification

2.11.4 Multi-label Classification

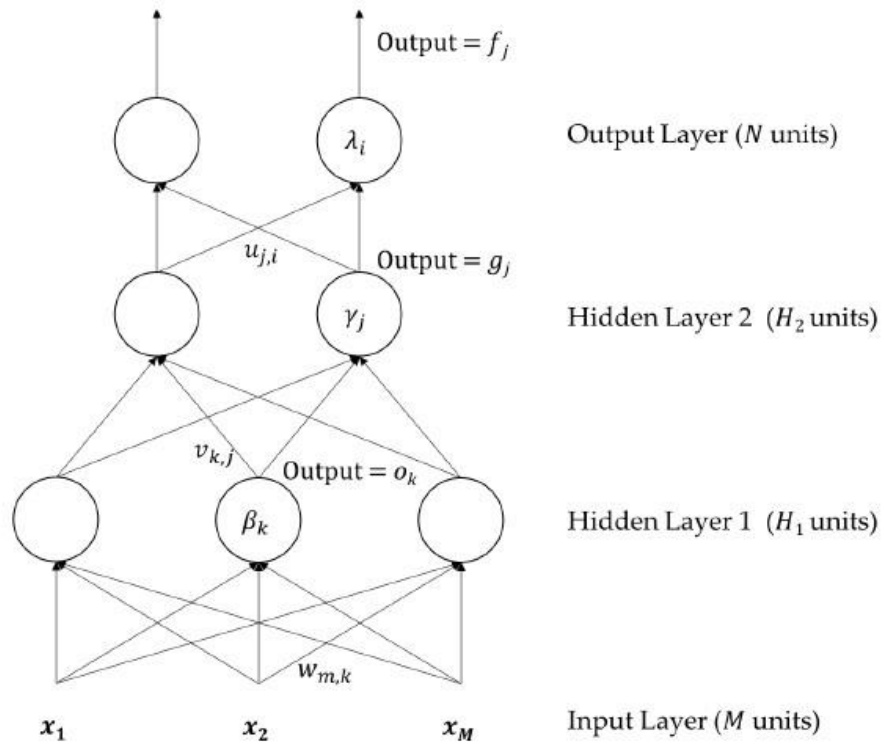
Seperti halnya *multi-class classification*, kita dapat menggunakan sejumlah C neuron untuk merepresentasikan C kelas pada *multi-label classification*. Perbedaan *multi-class* dan *multilabel* terletak pada cara interpretasi *output* dan evaluasi *output*. Pada umumnya, *layer* terakhir diaktivasi dengan fungsi sigmoid, dimana tiap neuron n_i merepresentasikan probabilitas suatu dapat diklasifikasikan sebagai kelas c_i atau tidak. *Cross entropy* juga cocok untuk mengevaluasi (dan melatih) *multi-label classification*.

c_1	c_2	c_3	c_4	
1	0	1	0	Label = c_1, c_3
0	1	0	0	Label = c_2
1	0	0	1	Label = c_1, c_4
0	1	1	1	Label = c_2, c_3, c_4

Gambar 2.13 Ilustrasi desired output pada multi-label classification

2.11.5 Deep Neural Network

Deep Neural Network (DNN) adalah *artificial neural network* yang memiliki banyak *layer*. Pada umumnya, deep neural network memiliki lebih dari 3 *layers* (*input layer*, N *hidden layers*, *output layer*), dengan kata lain adalah MLP dengan lebih banyak *layer*. Karena ada relatif banyak *layer*, disebutlah *deep*. Proses pembelajaran pada DNN disebut sebagai *deep learning*¹¹. Jaringan neural network pada DNN disebut *deep neural network*. Perhatikan Gambar 2.14 yang memiliki 4 *layers*.



Gambar 2.14 Deep Neural Network

Cara menghitung *final output* sama seperti MLP, diberikan pada persamaan berikut dimana adalah *noise* atau *bias*.

$$f_i = \left(\sum_{j=1}^{H_2} u_{j,i} \sigma \left(\sum_{k=1}^{H_1} u_{k,j} \sigma \left(\sum_{m=1}^M x_m w_{m,k} + \beta_k \right) + \gamma_j \right) + \lambda_i \right) \quad 2.15$$

Cara melatih *deep neural network*, salah satunya dapat menggunakan backpropagation. Hanya perlu menurunkan rumusnya saja. Hasil proses penurunan dapat dilihat pada Gambar 2.15.

(3) Hidden 2 to Output

$$f_i = \sigma \left(\sum_{j=1}^{H_2} g_j u_{j,i} + \lambda_i \right)$$

(4) Output to Hidden 2

$$\begin{aligned} \delta_i &= (y_i - f_i) f_i (1 - f_i) \\ \Delta u_{j,i} &= -\eta(t) \delta_i g_j \\ \Delta \lambda_i &= -\eta(t) \delta_i \end{aligned}$$

(2) Hidden 1 to Hidden 2

$$g_j = \sigma \left(\sum_{k=1}^{H_1} o_k v_{k,j} + \gamma_j \right)$$

(5) Hidden 2 to Hidden 1

$$\begin{aligned} \varphi_j &= \sum_{i=1}^N \delta_i u_{j,i} g_j (1 - g_j) \\ \Delta v_{k,j} &= -\eta(t) \varphi_j o_k \\ \Delta \gamma_j &= -\eta(t) \varphi_j \end{aligned}$$

(1) Input to Hidden Layer

$$o_k = \sigma \left(\sum_{m=1}^M x_m w_{m,k} + \beta_k \right)$$

(6) Hidden 1 to Input

$$\begin{aligned} \mu_k &= \sum_{j=1}^{H_2} \varphi_j v_{k,j} o_k (1 - o_k) \\ \Delta w_{m,k} &= -\eta(t) \mu_k x_m \\ \Delta \beta_k &= -\eta(t) \beta_k \end{aligned}$$

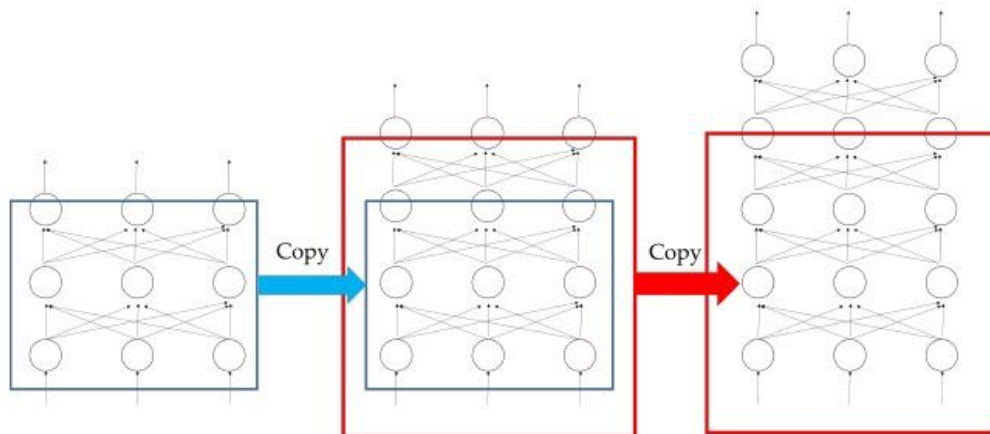
Gambar 2.15 Proses latihan DNN menggunakan backpropagation

Deep network terdiri dari banyak *layer* dan *synapse weight*, karenanya estimasi parameter susah dilakukan. Arti filosofisnya adalah susah/lama untuk menentukan relasi antara *input* dan *output*. Walaupun deep learning sepertinya kompleks, tetapi entah kenapa dapat bekerja dengan baik untuk permasalahan praktis. *Deep learning* dapat menemukan relasi "tersembunyi" antara input dan output, yang tidak dapat diselesaikan menggunakan *multilayer perceptron* (3 layers).

Banyak orang percaya *deep neural network* lebih baik dibanding *neural network* yang lebar tapi sedikit layer, karena terjadi lebih banyak transformasi. Maksud lebih banyak transformasi adalah kemampuan untuk merubah input menjadi suatu representasi (tiap hidden layer dapat dianggap sebagai salah satu bentuk representasi input) dengan langkah hierarchical. Seperti contoh permasalahan XOR, permasalahan *non-linearly separable* pun dapat diselesaikan apabila kita dapat mentransformasi data (representasi data) ke dalam bentuk *linearly separable* pada ruang yang berbeda. Keuntungan utama deep learning adalah mampu merubah data dari *non-linearly separable* menjadi *linearly separable* melalui serangkaian transformasi (*hidden layers*). Selain itu, deep

learning juga mampu mencari *decision boundary* yang berbentuk *non-linear*, serta mensimulasikan interaksi *non-linear* antar fitur.

Karena memiliki banyak parameter, proses latihan ANN pada umumnya lambat. Ada beberapa strategi untuk mempercepat pembelajaran menggunakan deep learning, misalnya: regularisasi, *successive learning*, dan penggunaan *autoencoder*. Sebagai contoh, arti *successive learning* adalah jaringan yang dibangun secara bertahap. Misal kita latih ANN dengan 3 *layers*, kemudian kita lanjutkan 3 *layers* tersebut menjadi 4 *layers*, lalu kita latih lagi menjadi 5 *layers*, dst (mulai dari hal kecil). Ilustrasinya dapat dilihat pada Gambar 2.16. Menggunakan *deep learning* harus hati-hati karena pembelajaran cenderung *divergen* (artinya, *minimum squared error* belum tentu semakin rendah seiring berjalannya waktu - *swing* relatif sering).



Gambar 2.16 Contoh successive learning

2.11.6 Regularization and Dropout

Pada model linear. Model harus mampu menggeneralisasi dengan baik (kinerja baik pada *training* data dan *unseen examples*). Peneliti dapat menambahkan fungsi regularisasi untuk mengontrol kompleksitas ANN. Regularisasi pada ANN cukup *straightforward* seperti regularisasi pada model linear.

Selain itu, agar ANN tidak “bergantung” pada satu atau beberapa *synapse weights* saja, peneliti dapat menggunakan *dropout*. **Dropout** berarti me-nol-kan nilai *synapse weights* dengan nilai rate tertentu. Misalkan me-nol-kan nilai 30%

synapse weights (dropout rate= 0,3) secara random. Hal ini dapat dicapai dengan teknik *masking*, yaitu mengalikan *synapse weights* dengan suatu *mask*.

Ingat kembali ANN secara umum, persamaan 2.16 dimana W adalah *synapse weights*, x adalah input (dalam pembahasan saat ini, dapat merepresentasikan *hidden state* pada suatu *layer*), b adalah *bias* dan f adalah fungsi aktivasi (*non-linear*). Peneliti bisa buat suatu *mask* untuk *synapse weights* seperti pada persamaan 2.17, dimana p adalah vektor dan $p_i = [0,1]$ merepresentasikan *synapse weight* diikutsertakan atau tidak. $r\%$ (*dropout rate*) elemen vektor p bernilai 0. Biasanya p diambil dari *bernoulli distribution*. Kemudian, saat *feed forward*, mengganti *synapse weights* menggunakan *mask* seperti pada persamaan 2.18. Saat menghitung *backpropagation*, turunan fungsi juga mengikut sertakan *mask* (*gradient di-mask*). Teknik *regularization* dan *dropout* sudah menjadi metode yang cukup "standar" dan diaplikasikan pada berbagai macam arsitektur.

$$o = f(x \cdot W + b) \quad 2.16$$

$$w' = p \cdot W \quad 2.17$$

$$o = f(x \cdot W' + b) \quad 2.18$$

2.12 Convolutional Neural Network

Kemampuan utama *convolutional neural network* (CNN) adalah arsitektur yang mampu mengenali informasi prediktif suatu objek (gambar, teks, potongan suara, dsb) walaupun objek tersebut dapat diposisikan dimana saja pada input. Kontribusi CNN adalah pada *convolution* dan *pooling layer*. *Convolution* bekerja dengan prinsip *sliding window* dan *weight sharing* (mengurangi kompleksitas perhitungan). *Pooling layer* berguna untuk merangkum informasi informatif yang dihasilkan oleh suatu *convolution* (mengurangi dimensi). Pada ujung akhir CNN, kita lewatkan satu vektor hasil beberapa operasi *convolution* dan *pooling* pada *multilayer perceptron* (*feed-forward neural network*), dikenal juga sebagai *fully connected layer*, untuk melakukan suatu pekerjaan, e.g., klasifikasi. Pada umumnya CNN tidak berdiri sendiri, dalam artian CNN biasanya digunakan (dikombinasikan) pada arsitektur yang lebih besar.

2.12.1 Activation Layer

Activation layer pada umumnya merupakan sebuah fungsi aktivasi di bagian atas sebuah *output layer*. Tujuan utama dari penggunaan fungsi aktivasi ini adalah untuk membentuk *non-linearity* (ketidaklinieran) pada *neural network*. Tanpa adanya fungsi aktivasi, *neural network* hanya akan melakukan transformasi linier dari input ke output. Secara matematis, fungsi aktivasi ditulis sebagai berikut.

$$x^l = f(x^{l-1}) \quad 2.19$$

Beberapa jenis fungsi aktivasi yang umum digunakan adalah ReLU (*Rectified-Linear Unit*), Sigmoid dan TanH (*Hyperbolic Tangent*).

Sigmoid

Fungsi Sigmoid akan memetakan input ke interval [0, 1]. Fungsi sigmoid didefinisikan sebagai berikut.

$$\sigma(u) = \frac{1}{1+\exp^{-u}} \quad 2.20$$

TanH

Fungsi TanH atau *hyperbolic tangent function* merupakan fungsi transformasi linier dari sigmoid ke interval [-1, 1]. Fungsi TanH dapat didefinisikan sebagai berikut.

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad 2.21$$

ReLU

Permasalahan yang ada pada fungsi sigmoid dan tanh yaitu output yang mudah jenuh ke angka 0 atau 1 pada sigmoid dan +1 dan -1 pada tanh. Output yang jenuh akan membuat hilangnya gradien dan menyebabkan turunnya kecepatan training dan memungkinkan untuk menjebak model pada *area local minimum*. ReLU diperkenalkan pada tahun 2010 untuk meningkatkan kecepatan *konvergensi* dari *training neural networks* (V. Nair, 2010). ReLU memiliki kelebihan tidak akan jenuh pada suatu nilai dan memungkinkan untuk melakukan *training neural network* berukuran besar layaknya CNN. Relu didefinisikan sebagai berikut.

$$f(x) = \max(0, x) \quad 2.22$$

Pada CNN, layer output pada umumnya merupakan *feature maps* dimensi dua. Pada persamaan 2.22, x adalah matrix dan \max adalah elemen *twice* yang

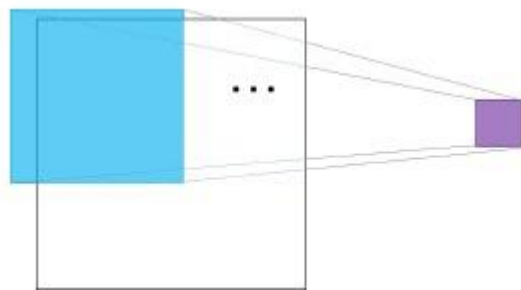
diterapkan pada setiap elemen matriks. Varian lain dari ReLU yaitu LeakyReLU memungkinkan adanya nilai yang kecil, bukan nol meski unit sedang tidak dalam keadaan aktif. Leaky ReLU didefinisikan sebagai berikut.

$$\begin{cases} x, & x > 0 \\ 0.01x & \text{selain } x > 0 \end{cases} \quad 2.23$$

2.12.2 Convolution

Motivasi CNN adalah untuk mampu mengenali aspek yang informatif pada regional tertentu (lokal). Dibanding meng-copy mesin pembelajaran beberapa kali untuk mengenali objek pada banyak regional, ide lebih baik adalah untuk menggunakan *sliding window*. Setiap operasi pada window bertujuan untuk mencari aspek lokal yang paling informatif.

Ilustrasi diberikan oleh Gambar 2.17. Warna biru merepresentasikan satu *window*, kemudian kotak ungu merepresentasikan aspek lokal paling informatif (disebut *filter*) yang dikenali oleh *window*. Dengan kata lain, kita mentransformasi suatu *window* menjadi suatu nilai numerik (*filter*). Kita juga dapat mentransformasi suatu *window* (regional) menjadi d nilai numerik (d channels, setiap elemen berkorespondensi pada suatu *filter*). *Window* ini kemudian digeser-geser sebanyak T kali, sehingga akhirnya kita mendapatkan vektor dengan panjang d x T. Keseluruhan operasi ini disebut sebagai *convolution*.

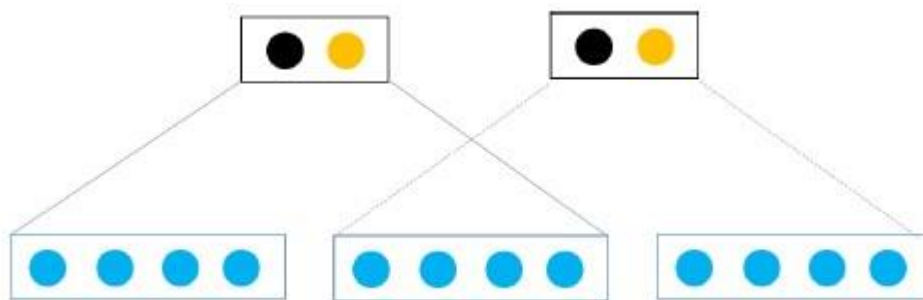


Gambar 2.17 Sliding window

Pada Gambar 2.18 menggunakan *window* selebar 2, satu *window* mencakup 2 data; i.e., $window_1 = (X_1, X_2)$, $window_2 = (X_2, X_3)$, Untuk suatu input X. Kita juga dapat mempergunakan *stride* sebesar s, yaitu seberapa banyak data yang digeser untuk *window* baru. Contoh yang diberikan memiliki *stride* sebesar satu.

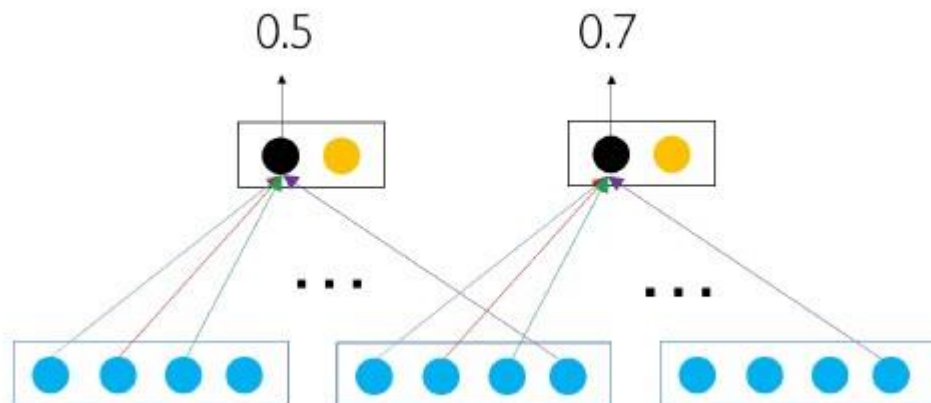
Apabila memiliki $stride=2$, maka akan menggeser sebanyak 2 data setiap langkah, i.e., $window_1 = (X_1, X_2)$, $window_2 = (X_3, X_4)$,

Berikut contoh dalam bentuk 1-D pada Gambar 2.18. Warna biru merepresentasikan *feature vector (regional)* untuk suatu input (e.g., *regional* pada suatu gambar, kata pada kalimat, dsb). Pada contoh ini, setiap 2 input ditransformasi menjadi vektor berdimensi 2 (*2-channels*); menghasilkan vektor berdimensi 4 ($2 \text{ window} \times 2$).



Gambar 2.18 1D Convolution

Selain *sliding window* dan *filter*, *convolutional layer* juga mengadopsi prinsip *weight sharing*. Artinya, *synapse weights* untuk suatu filter adalah sama walau filter tersebut dipergunakan untuk berbagai *window*. Sebagai ilustrasi, perhatikan Gambar 2.19, warna yang sama pada *synapse weights* menunjukkan *synapse weights* bersangkutan memiliki nilai (*weight*) yang sama. Tidak hanya pada *filter* hitam, hal serupa juga terjadi pada *filter* berwarna oranye (i.e., *filter* berwarna oranye juga memenuhi prinsip *weight sharing*). Walaupun memiliki konfigurasi bobot *synapse weights* yang sama, unit dapat menghasilkan output yang berbeda untuk input yang berbeda. Konsep *weight sharing* ini sesuai dengan pernyataan bahwa konfigurasi parameter untuk mengenali karakteristik informatif untuk satu objek bernilai sama walau pada lokasi yang berbeda. Dengan *weight sharing*, parameter neural network juga menjadi lebih sedikit dibanding menggunakan *multilayer perceptron (feed-forward neural network)*.



Gambar 2.19 Konsep weight sharing

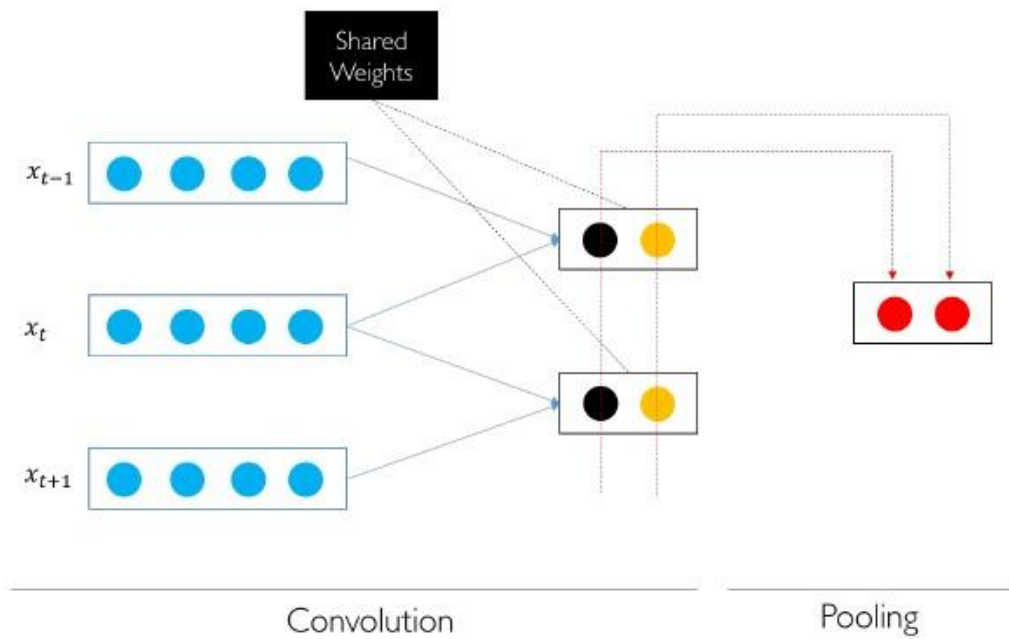
2.12.3 Pooling

Pada tahap convolution, setiap k -sized window diubah menjadi satu vektor berdimensi d (yang dapat disusun menjadi matriks D). Semua vektor yang dihasilkan pada tahap sebelumnya dikombinasikan (pooled) menjadi satu vektor c . Ide utamanya adalah mengekstrak informasi paling informatif (semacam meringkas). Ada beberapa teknik *pooling*, diantaranya: *max pooling*, *average pooling*, dan *K-max pooling*; diilustrasikan pada Gambar 2.20. *Max pooling* mencari nilai maksimum untuk setiap dimensi vektor. *Average pooling* mencari nilai rata-rata tiap dimensi. *K-max pooling* mencari K nilai terbesar untuk setiap dimensinya (kemudian hasilnya digabungkan). Gabungan operasi *convolution* dan *pooling* secara konseptual diilustrasikan pada Gambar 2.21.

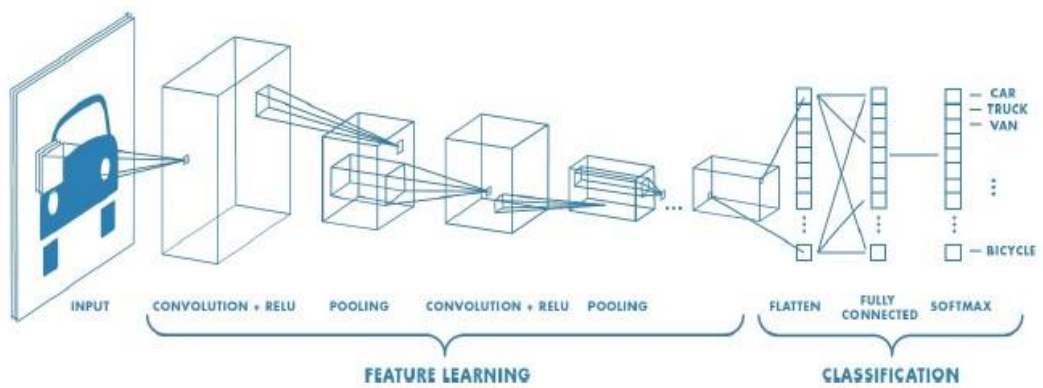




Gambar 2.20 Contoh pooling



Gambar 2.21 Convolution dan pooling



Gambar 2.22 Convolutional Neural Network

Setelah melewati berbagai operasi *convolution* dan *pooling*, pembuat akan memiliki satu vektor yang kemudian dilewatkan pada *multilayer perceptron (fully connected)* untuk melakukan sesuatu (tergantung permasalahan), misal klasifikasi gambar, klasifikasi sentimen, dsb (Ilustrasi pada Gambar 2.22).

2.13 OpenCV

OpenCV (Open Computer Vision) merupakan sebuah API (Application Programming Interface) library open source yang sangat cocok digunakan untuk image processing. OpenCV pertama kali dibuat oleh Intel pada tahun 1999 oleh Gary Bradsky dan mulai dirilis keluar pada tahun 2000. Pada tahun 2005, OpenCV berhasil memenangkan DARPA Grand Challenge. Saat ini, OpenCV telah mendukung banyak algoritma yang terkait dengan Computer Vision dan Machine Learning. Selain itu, saat ini OpenCV juga dapat digunakan dalam berbagai macam bahasa pemrograman, seperti C++, Python, Java, dan lain sebagainya. Tidak hanya itu, OpenCV juga tersedia dalam berbagai platform, seperti Windows, Linux, OSX, Android, IOS, dan lain sebagainya. OpenCV juga dapat melakukan operasi highspeed GPU dengan menggunakan antarmuka-antarmuka berbasis CUDA dan OpenCL. Kombinasi terbaik untuk dapat melakukan operasi berkecepatan tinggi tersebut adalah dengan perpaduan antara OpenCV, C++ API dan bahasa pemrograman Python.

2.14 MediaPipe Hands

MediaPipe adalah kerangka kerja lintas platform yang dapat diimplementasikan untuk membangun jaringan pipa untuk memproses data persepsi dari berbagai format (audio dan video). MediaPipe Hands adalah solusi deteksi tangan dan jari dengan ketelitian tinggi. Dalam Hanya satu frame di dapat menyimpulkan hingga 21 Landmark tangan 3D. Ini adalah hibrida antara model deteksi telapak tangan / tangan yang beroperasi pada gambar penuh dan mengembalikan data *bounding box* tangan yang berorientasi dan model tengara tangan yang beroperasi pada gambar yang dipangkas wilayah yang ditentukan oleh detektor telapak tangan, yang mengembalikan tangan 3D fidelitas tinggi poin kunci.

Ini mendeteksi tengara dari satu tangan atau kedua tangan tergantung pada jenis modul (Sankeerthana., 2021).

2.15 Keras Tensorflow

Keras merupakan suatu jaringan saraf API (application program interface) tingkat tinggi yang ditulis dengan bahasa Python dan dapat berjalan diatas TensorFlow. Keras juga mengkompilasi model dengan fungsi loss dan optimizer dalam proses pelatihan. Dalam keras terdapat salah satu istilah yang disebut sebagai “Backend”, dimana istilah tersebut dapat diketahui saat menjalankan program dari keras itu sendiri. Istilah tersebut merupakan perhitungan tingkat rendah dalam tensor dan konvolusi dengan bantuan library dari tensorflow. Sedangkan TensorFlow merupakan suatu multidimensi array yang dikenal sebagai tensor dan juga digunakan sebagai jaringan saraf dalam operasi yang berbeda. Selain dikenal sebagai suatu multidimensi array, tensorflow juga merupakan suatu tool perpustakaan perangkat lunak yang memiliki tugas pada pembelajaran dalam machine learning oleh jaringan saraf maupun pemrograman diferensial.

BAB 3. METODE PENELITIAN

3.1 Waktu dan Tempat Penelitian

Penelitian dilakukan selama rentang waktu 12 bulan (satu tahun), dimulai dari bulan Agustus 2020 sampai Juli 2021 di Lab Rekayasa Perangkat Lunak Jurusan Teknologi Informasi Politeknik Negeri Jember. Penelitian hanya dilakukan di satu tempat yaitu di Gedung Jurusan Teknologi Informasi Politeknik Negeri Jember.

3.2 Alat dan Bahan

3.2.1 Alat Penelitian

Pada penelitian ini digunakan alat berupa perangkat keras dan perangkat lunak sebagai berikut :

a. Perangkat Keras

Perangkat keras yang digunakan antara lain satu unit laptop, pc dan kamera webcam dengan detail sebagai berikut :

1. Acer Aspire E5-476G-58V
 - a) Screen Size 14 inch
 - b) Processor Intel(R) Core™ i5-8250 1.6 GHz with Turbo Boost up to 3.4 GHz
 - c) NVIDIA (R) Geforce(R) MX130 with 2 GB VRAM
 - d) 8 GB DDR4 Memory
 - e) 1000 GB HDD
2. Webcam :
 - a) Internal HDR webcam Acer Aspire E5-476G-58V
 - b) Eksternal USB webcam

b. Perangkat Lunak

Perangkat lunak yang digunakan antara lain :

1. Operating System Windows 10 64-bit
2. JetBrains PyCharm Edu x64
3. Python 3.7 ++
4. OpenCV 4.5.2.54

5. Keras 2.4.3
6. Matplotlib 3.4.2
7. MediaPipe 0.8.6
8. Numpy 1.19.5
9. Tensorflow 2.5.0
10. Sklearn

3.2.2 Bahan Penelitian

Data sampel (training dan validasi) citra Tangan Bahasa Isyarat Indonesia (BISINDO) dari huruf A-Z dan 1-10.

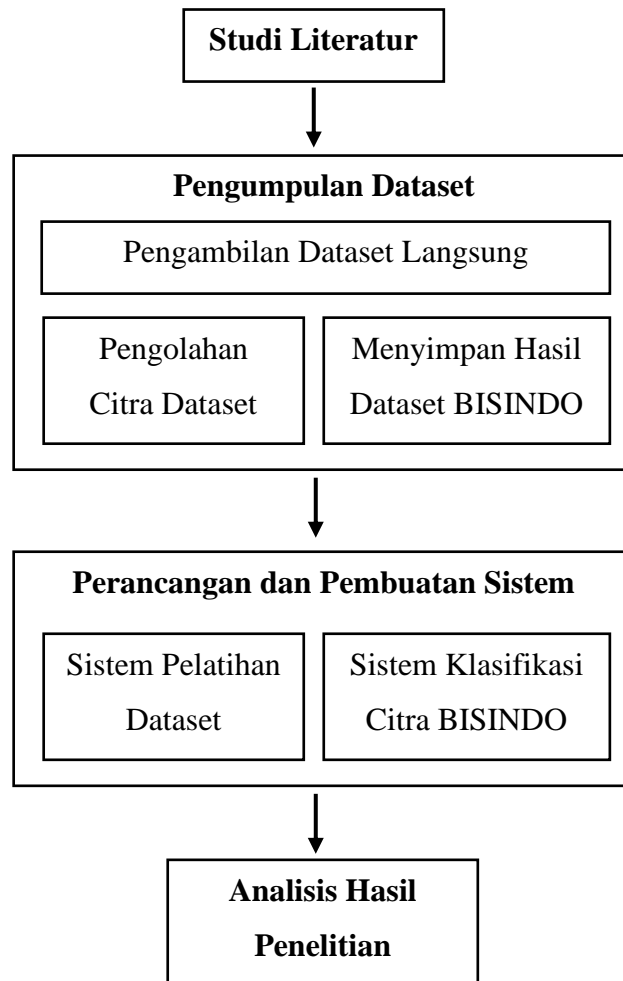
3.3 Tahapan Penelitian

Pada metode penelitian ini peneliti melakukan beberapa tahapan yaitu studi literatur, pengumpulan data, tahapan pengembangan sistem, hasil, dan analisis penelitian seperti pada Gambar 3.1. Tahapannya terdiri atas studi literatur, pengumpulan data dan tahapan pengembangan sistem. Penjelasan mengenai setiap tahapan akan dijelaskan sebagai berikut :

a. Studi Literatur

Penelitian ini diawali dengan tahap studi literatur untuk mencari referensi-referensi atau teori-teori yang sesuai dengan permasalahan dan solusi penelitian. Adapun referensi yang peneliti pelajari meliputi :

1. Informasi tentang sejarah dan bentuk gerak Bahasa Isyarat Indonesia (BISINDO).
2. Proses pengolahan data citra (*preprocessing*, segmentasi dan augmentasi).
3. Konsep *Image Classification* dan penelitian yang mirip dalam pengembangan sistem komputer vision untuk penerjemah bahasa isyarat.
4. Konsep Metode Deep Learning Model Convolutional Neural Network.



Gambar 3.1 Block diagram tahapan penelitian

b. Pengumpulan Data

Penelitian ini menggunakan data dan informasi akurat untuk menunjang proses penelitian agar berjalan efektif dan efisien. Berikut ini metode pengumpulan data yang dilakukan peneliti :

1. Pengambilan Dataset Langsung

Pada tahap ini peneliti melakukan pengambilan dataset secara langsung berupa citra Bahasa Isyarat Indonesia (BISINDO) untuk digunakan sebagai data penelitian. Jenis gestur bahasa isyarat abjad dan angka yang digunakan dalam dataset dibedakan menjadi 36 kelas yaitu abjad A-Z dan angka 1-10.

2. Pengolahan Citra Dataset.

Pengolahan citra dataset bertujuan untuk mengolah data citra mentah yang berasal dari kamera / *webcam* agar dataset citra memiliki nilai informasi yang lebih baik dan konsisten. Pada tahap ini dilakukan proses deteksi objek, preprocessing data citra, dan segmentasi data citra. Penjelasan lebih detail akan dijelaskan didalam Block Diagram Sistem pada Gambar 3.3.

3. Menyimpan Hasil Dataset BISINDO.

Hasil proses pengolahan citra dataset kemudian dikelompokkan dengan dimasukkan kedalam folder berdasarkan kelas / label yang telah ditentukan.

c. Perancangan dan Pembuatan Sistem

Tahapan kegiatan ini terdiri dari rancangan yang digunakan dalam membangun sistem pelatihan dataset / dataset *training* dan sistem klasifikasi citra BISINDO / *testing*.

1. Sistem Pelatihan Dataset.

Proses pelatihan dataset / data training merupakan tahapan dimana CNN dilatih untuk memperoleh akurasi yang tinggi dari klasifikasi yang dilakukan. Peneliti merancang sistem pelatihan data yang dijelaskan didalam Gambar 3.4 Block Diagram Proses Training Dataset Citra yang terdiri dari proses load file dataset citra, split dan shuffle data, preprocessing dan augmentasi gambar data citra, one hot encode, mendefinisikan dan memulai training data dengan model CNN dan menyimpan hasil training data (model dataset CNN).

2. Sistem Klasifikasi Citra BISINDO.

Proses sistem klasifikasi / testing merupakan proses klasifikasi menggunakan bobot dan bias dari hasil proses training. Proses ini tidak jauh berbeda dengan proses training yang membedakannya tidak terdapat proses backpropagation setelah proses feedforward. Sehingga hasil akhir dari proses ini menghasilkan beberapa data

yaitu akurasi dari klasifikasi yang dilakukan, data yang gagal diklasifikasi, nomor citra yang gagal diklasifikasi, dan bentuk network yang terbentuk dari proses feedforward. Keseluruhan proses tahap ini dijelaskan pada Gambar 3.5 Block Diagram Proses Klasifikasi Citra.

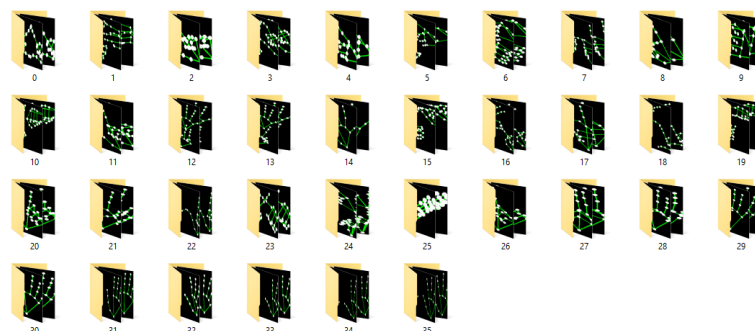
d. Analisis Hasil Penelitian

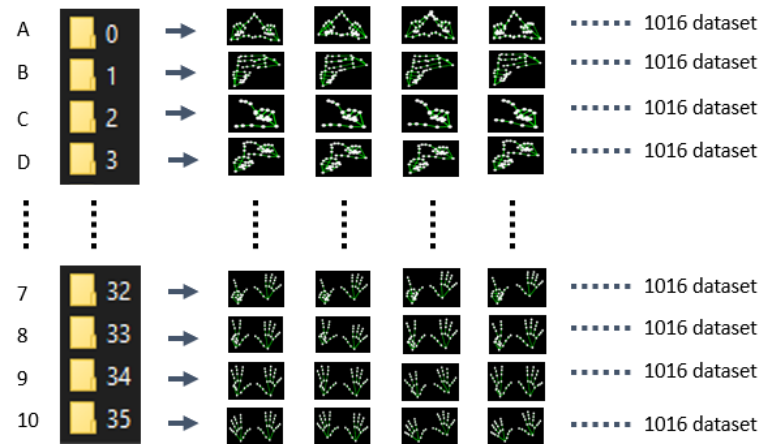
Algoritma *machine learning* biasanya mahal secara komputasi. Oleh karena itu, sangat penting untuk mengukur kinerja aplikasi *machine learning* yang sudah dibuat dan untuk memastikan bahwa peneliti menjalankan versi model yang paling optimal. Peneliti menggunakan Tensorflow Profiler untuk memantau pelatihan model. Profiler mencatat waktu yang diperlukan untuk menjalankan operasi, waktu yang diperlukan untuk menyelesaikan langkah-langkah yang di eksekusi, mengumpulkan data tentang pemanfaatan sumber daya dalam hal waktu dan memori dan memberikan visualisasi untuk memahami informasi tersebut.

3.4 Jenis Data

3.4.1 Data Primer

Data Primer dalam penelitian ini adalah data gambar atau sampel gambar Bahasa Isyarat Indonesia (BISINDO) dengan proses pengambilan langsung yang dilakukan oleh peneliti.

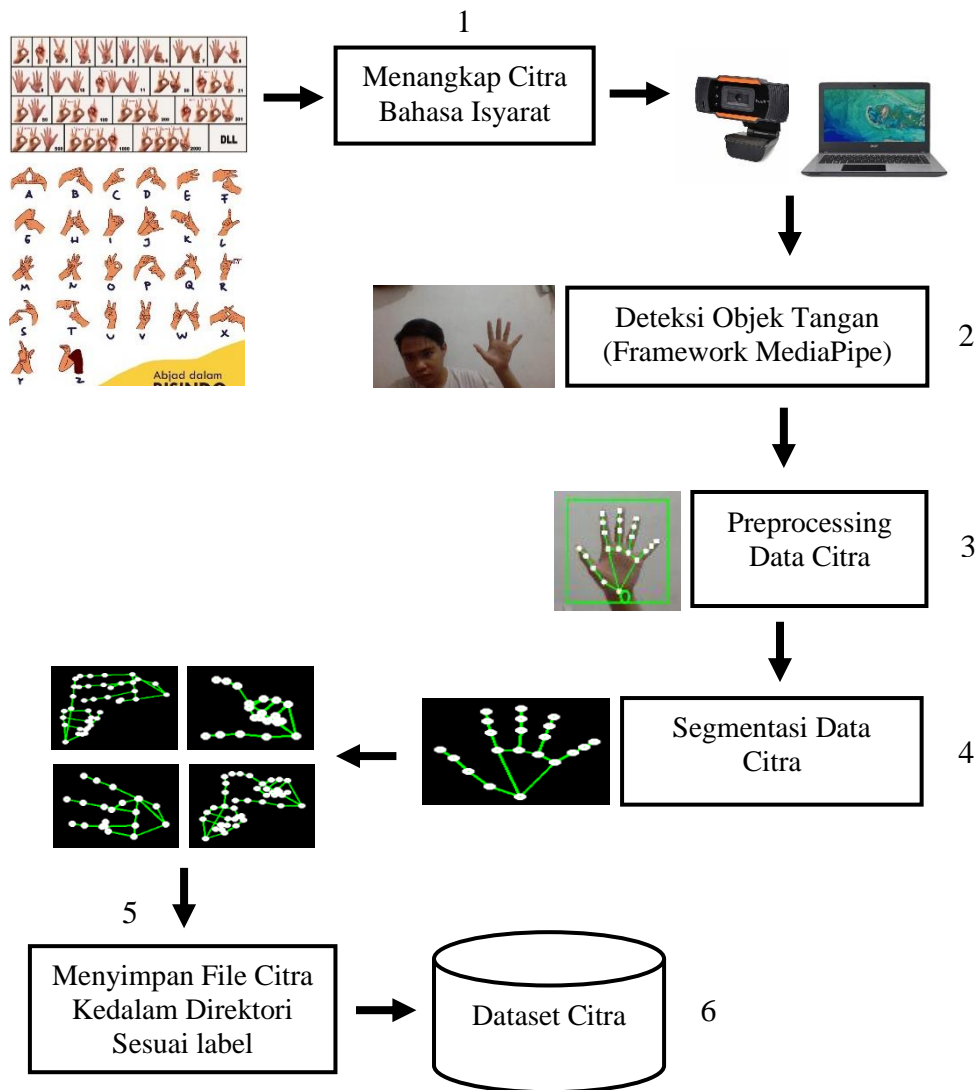




Gambar 3.2 Struktur Folder Dataset

3.5 Block Diagram Sistem

Dalam sistem ini hanya terdapat user tunggal sebagai aktornya. Alur proses sistem ini dapat digambarkan dalam bentuk diagram block. Berikut ini digambarkan mengenai diagram *block* sistem pada Gambar 3.3, Gambar 3.4 dan Gambar 3.5.



Gambar 3.3 Block Diagram Proses Mendapatkan Dataset Citra

Pada Gambar 3.3 Block diagram proses mendapatkan dataset citra memiliki beberapa tahapan, seperti berikut :

a. Tahap Pertama

Kamera webcam dan laptop menangkap *gesture* Bahasa Isyarat indonesia yang diperagakan oleh peneliti. Terdapat satu kamera yang dipakai yaitu kamera webcam eksternal. Posisi kamera berada didepan user atau disebut *selfie*.

b. Tahap Kedua

Untuk melakukan pelacakan tangan dan jari dengan ketelitian tinggi, peneliti menggunakan framework machine learning bernama “MediaPipe”. Framework ini menggunakan pembelajaran mesin (ML) untuk menyimpulkan 21 landmark 3D tangan hanya dari satu bingkai secara realtime, yang nantinya 21 landmark tersebut akan digunakan untuk proses-proses penting seperti mendapatkan *region of interest* pada gambar.

c. Tahap Ketiga

Sistem melakukan proses preprocessing pada data citra yang berhasil ditangkap oleh kamera. Preprocessing adalah proses dimana citra yang telah masuk ke sistem akan diproses melalui beberapa tahap, meliputi :

1. Crop Image

Crop image merupakan penghapusan bagian sudut dari suatu gambar untuk memotong/mengambil/mengeluarkan sebagian isi dari gambar guna memperoleh hasil yang diinginkan.

2. Resize

Proses resize citra digunakan untuk mengurangi jumlah piksel dari citra inputan.

3. Operator Laplacian

Operator Laplacian merupakan operator turunan yang digunakan untuk mencari edge pada suatu citra. Laplacian sering diterapkan pada gambar yang telah dihaluskan terlebih dahulu dengan filter Gaussian seperti untuk mengurangi sensitivitasnya terhadap noise. Operator ini biasanya menggunakan satu gambar keabuan sebagai input dan output.

d. Tahap Keempat

Melakukan proses segmentasi citra untuk memisahkan objek penelitian dengan latar yang terdapat dalam sebuah citra. Proses tersebut meliputi mendapatkan data landmark hasil proses *hand detection* MediaPipe, membuat *bounding box* disekitar objek tangan yang sudah terdeteksi, dan

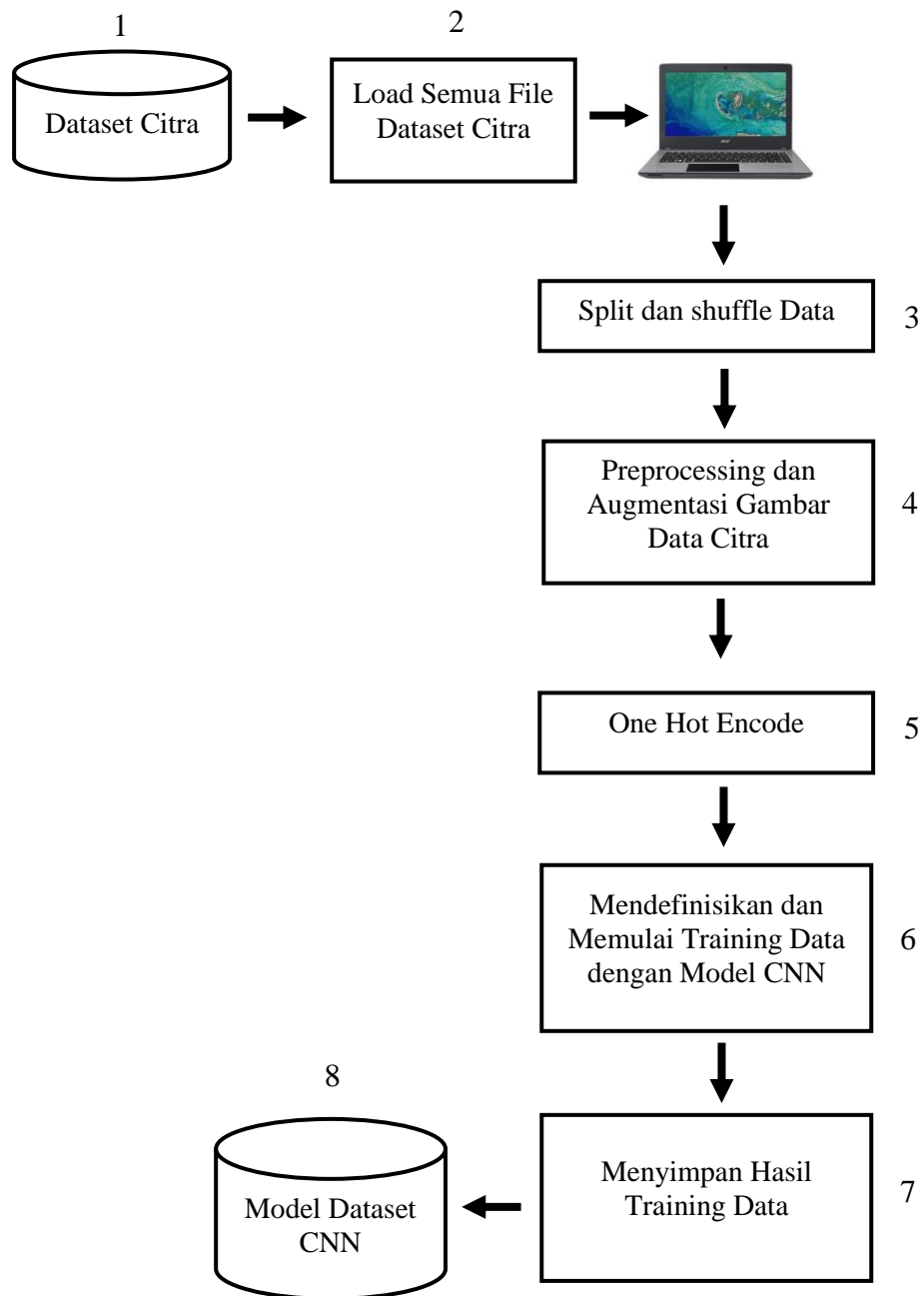
membuat ulang gambar landmark tangan dengan background hitam sesuai *bounding box* dan hasil proses *hand detection* MediaPipe.

e. Tahap Kelima

Menyimpan file gambar hasil proses *preprocessing* dan segmentasi citra kedalam folder yang bernama sesuai dengan label (kelas/kelompok *gesture*).

f. Tahap Keenam

Dataset merupakan kumpulan data citra yang akan digunakan untuk proses data training dalam pembuatan data model deep learning (CNN). Pada penelitian ini terdapat 36 direktori/kelas dimana setiap kelas berisi 1016 data citra sehingga total dataset yang digunakan sebanyak 36.576.



Gambar 3.4 Block Diagram Proses Training Dataset Citra

Pada Gambar 3.4 Block diagram proses training dataset citra memiliki beberapa tahapan, seperti berikut :

a. Tahap Pertama

Pada penelitian ini terdapat 36 direktori/kelas dimana setiap kelas berisi 1016 data citra sehingga total dataset yang digunakan sebanyak 36.576.

b. Tahap Kedua

Memanggil semua file beserta dengan labelnya yang ada dalam direktori dataset, kemudian sistem akan membuat array dataset dan array label secara berurutan. Array tersebut akan digunakan untuk proses komputasi oleh komputer.

c. Tahap Ketiga

Dengan bantuan fungsi pada *scikit-learn*, peneliti akan memisahkan *array* dataset menjadi 3 bagian utama yaitu *training* (23.424), *testing* (7.320) dan *validation* (5.856).

d. Tahap Keempat

Sistem melakukan proses preprocessing pada data citra yang berhasil ditangkap oleh kamera. Preprocessing adalah proses dimana citra yang telah masuk ke sistem akan diproses melalui beberapa tahap, meliputi :

1. Resize

Proses resize citra digunakan untuk mengurangi jumlah piksel dari citra inputan.

2. Reshaping Image Array

Dikarenakan Keras Tensorflow hanya menerima data gambar berupa gambar dengan 1 channel maka diperlukan proses untuk merubah bentuk array. Bentuk array adalah jumlah elemen dalam setiap dimensi. Dengan reshaping array peneliti bisa dapat menambah atau menghapus dimensi atau mengubah jumlah elemen di setiap dimensi.

3. Konversi Citra Grayscale

Citra yang ditampilkan dari citra jenis ini terdiri atas warna abu-abu, bervariasi pada warna hitam pada bagian yang intensitas terlemah dan warna putih pada intensitas terkuat. Citra grayscale disimpan dalam format 8 bit untuk setiap sample pixel, yang memungkinkan sebanyak 256 intensitas. Format ini sangat membantu dalam pemrograman karena manipulasi bit yang tidak terlalu banyak. Untuk mengubah citra berwarna yang mempunyai nilai matrik masing-masing R, G dan B menjadi citra grayscale dengan nilai X,

maka konversi dapat dilakukan dengan mengambil rata-rata dari nilai R, G dan B.

4. Histogram Equalization

Histogram Equalization adalah suatu metode dalam pengolahan citra dengan penyesuaian kontras menggunakan histogram citra. Metode ini biasanya meningkatkan kontras global dari banyak gambar. OpenCV memiliki fungsi untuk melakukan ini, `cv2.equalizeHist()`. Inputnya hanya gambar skala abu-abu dan outputnya adalah gambar histogram yang disamakan.

5. Augmentasi Data

Augmentasi data adalah suatu proses dalam pengolahan data gambar, augmentasi merupakan proses mengubah atau memodifikasi gambar sedemikian rupa sehingga komputer akan mendeteksi bahwa gambar yang diubah adalah gambar yang berbeda, namun manusia masih dapat mengetahui bahwa gambar yang diubah tersebut adalah gambar yang sama (L. Perez and J. Wang, 2017). Augmentasi dapat meningkatkan akurasi dari model CNN yang dilatih karena dengan augmentasi model mendapatkan data-data tambahan yang dapat berguna untuk membuat model yang dapat melakukan generalisasi dengan lebih baik.

e. Tahap Kelima

One Hot Encoding adalah salah satu metode encoding. Metode ini merepresentasikan data bertipe kategori sebagai vektor biner yang bernilai integer 0 dan 1, dimana semua elemen akan bernilai 0 kecuali satu elemen yang bernilai 1, yaitu elemen yang memiliki nilai kategori tersebut.

f. Tahap Keenam

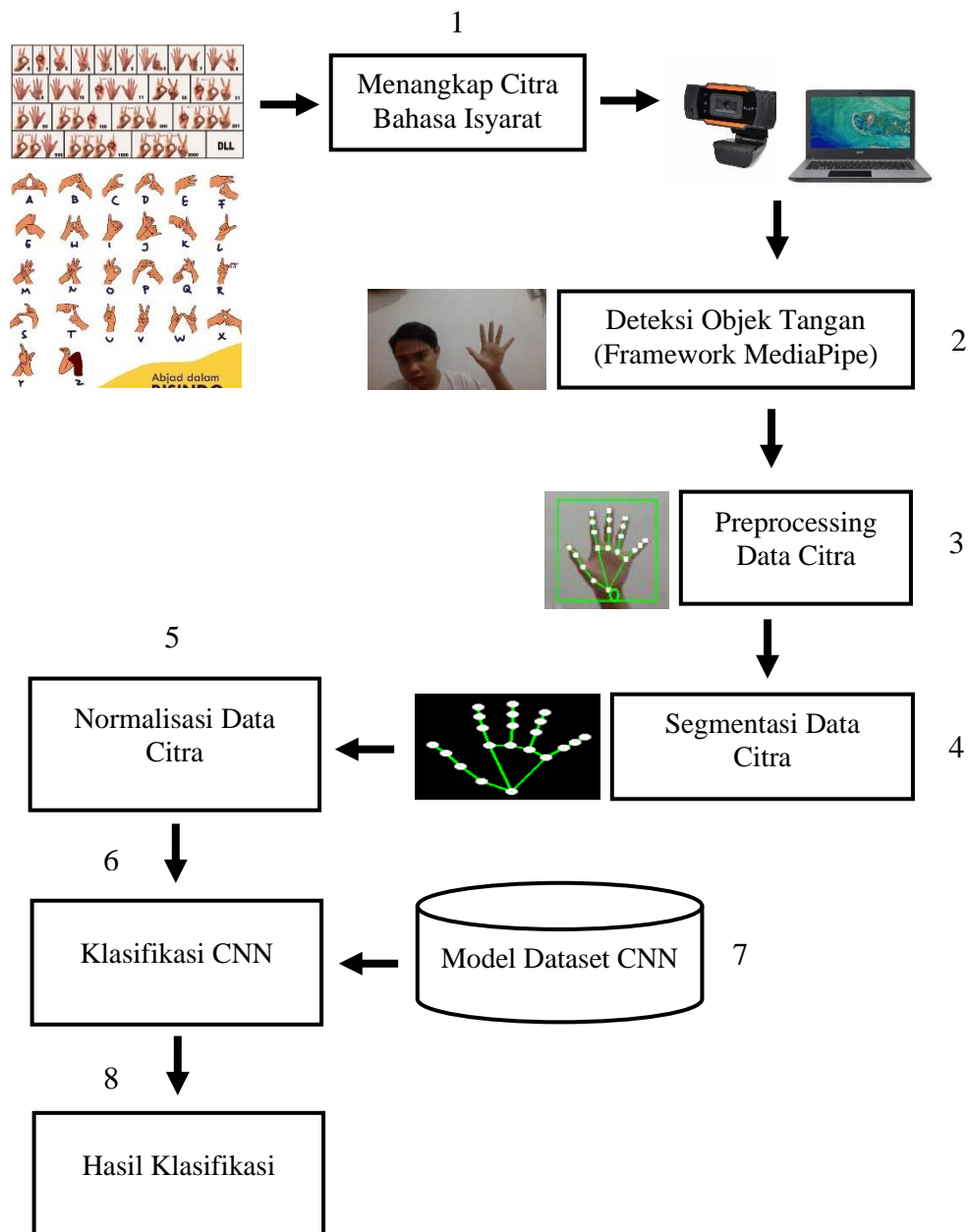
Melakukan Proses Training data dengan menggunakan metode algoritma CNN (Convolutional Neural Network).

g. Tahap Ketujuh

Menyimpan hasil proses training data dengan format file tertentu (`model_trained_all.h5`).

h. Tahap Kedelapan

File model yang sudah berhasil dibuat selanjutnya bisa digunakan oleh machine learning untuk proses klasifikasi tanpa harus melakukan training data lagi.



Gambar 3.5 Block Diagram Proses Klasifikasi Citra

Pada Gambar 3.5 Block diagram proses klasifikasi citra dengan menggunakan algoritma model CNN memiliki beberapa tahapan, seperti berikut :

a. Tahap Pertama

Kamera webcam dan laptop menangkap gesture Bahasa Isyarat Indonesia yang diperagakan oleh peneliti. Terdapat satu kamera yang dipakai yaitu kamera webcam eksternal. Posisi kamera berada didepan user atau disebut selfie.

b. Tahap Kedua

Untuk melakukan pelacakan tangan dan jari dengan ketelitian tinggi, peneliti menggunakan framework machine learning bernama “MediaPipe”. Framework ini menggunakan pembelajaran mesin (ML) untuk menyimpulkan 21 landmark 3D tangan hanya dari satu bingkai secara realtime, yang nantinya 21 landmark tersebut akan digunakan untuk proses-proses penting seperti mendapatkan region of interest pada gambar.

c. Tahap Ketiga

Sistem melakukan proses preprocessing pada data citra yang berhasil ditangkap oleh kamera. Preprocessing adalah proses dimana citra yang telah masuk ke sistem akan diproses melalui beberapa tahap, meliputi :

1. Crop Image

Crop image merupakan penghapusan bagian sudut dari suatu gambar untuk memotong/mengambil/mengeluarkan sebagian isi dari gambar guna memperoleh hasil yang diinginkan.

2. Resize

Proses resize citra digunakan untuk mengurangi jumlah piksel dari citra inputan.

3. Operator Laplacian

Operator Laplacian merupakan operator turunan yang digunakan untuk mencari edge pada suatu citra. Laplacian sering diterapkan pada gambar yang telah dihaluskan terlebih dahulu dengan filter Gaussian seperti untuk mengurangi sensitivitasnya terhadap noise. Operator ini biasanya menggunakan satu gambar keabuan sebagai input dan output.

d. Tahap Keempat

Melakukan proses segmentasi citra untuk memisahkan objek penelitian dengan latar yang terdapat dalam sebuah citra. Proses tersebut meliputi mendapatkan data landmark hasil proses hand detection MediaPipe, membuat bounding box disekitar objek tangan yang sudah terdeteksi, dan membuat ulang gambar landmark tangan dengan background hitam sesuai bounding box dan hasil proses hand detection MediaPipe.

e. Tahap Kelima

Dikarenakan Keras Tensorflow hanya menerima data gambar berupa gambar dengan 1 channel maka diperlukan proses untuk merubah bentuk array. Bentuk array adalah jumlah elemen dalam setiap dimensi. Dengan reshaping array peneliti bisa dapat menambah atau menghapus dimensi atau mengubah jumlah elemen di setiap dimensi.

f. Tahap Keenam

Inisialisasi proses prediksi oleh deep learning CNN terhadap citra input dan model dataset untuk mengklasifikasikan citra dan menerjemahkannya kedalam bentuk tulisan.

g. Tahap Ketujuh

File model yang sudah berhasil dibuat bisa digunakan oleh mechine learning untuk proses klasifikasi tanpa harus melakukan training data lagi.

h. Tahap Kedelapan

Hasil klasifikasi yang didapatkan akan disimpan dan ditampilkan di layar monitor.

CNN Model Architecture.

Jaringan saraf yang dirancang untuk pengenalan karakter bahasa isyarat Indonesia (Abjad dan Nomor) menggunakan model CNN LeNeT sebagai dasar pembuatan arsitektur. Arsitektur yang dirancang terdiri dari 4 lapisan konvolusi diikuti oleh fungsi aktivasi Relu dan 2 lapisan Maxpooling. Diikuti dengan *dropout*, *flatten*, *density layer* yang diakhiri dengan *softmax activation layer* dengan 36 output (26 huruf dan 10 angka), yang akan menunjukkan persentase klasifikasi yang diperoleh untuk setiap kelas.

BAB 4. HASIL DAN PEMBAHASAN

4.1 Studi Literatur

Proses studi literatur yang dilakukan pada penelitian ini adalah mengumpulkan data konsep berupa jurnal-jurnal penelitian. Jurnal tersebut berkaitan dengan pembahasan Bahasa Isyarat Indonesia (BISINDO), Pengolahan Citra Digital, *Image Classification*, dan Metode *Deep Learning Model Convolutional Neural Network*. Salah satu judul jurnal yang dipakai sebagai literasi adalah jurnal dengan judul “Bahasa Isyarat Indonesia Dalam Proses Interaksi Sosial Tuli Dan Masyarakat Dengar Di Kota Denpasar”. Dalam jurnal tersebut menyebutkan bahwa Bisindo dapat diartikan sebagai sebuah terminologi yang digunakan untuk menunjuk pada bahasa isyarat alami yang digunakan oleh komunitas Tuli di Indonesia. Berdasarkan penjelasan tersebut dapat disimpulkan bahwa sejarah Bisindo sejalan dengan kemunculan bahasa isyarat alami yang terdapat di Indonesia. Jenis penelitian ini digunakan untuk memaparkan berbagai faktor terkait implementasi Bisindo dalam proses interaksi sosial serta memaparkan analisis makna-makna Bisindo dalam proses interaksi sosial Tuli dan “masyarakat dengar” berdasarkan teori interaksionisme simbolik Blumer. Ketika memproduksi suatu isyarat dalam penggunaan Bisindo terdapat lima parameter yang perlu diperhatikan, yaitu bentuk tangan (*handshapes*), orientasi (*orientation*), lokasi (*locations*), gerakan (*movements*), dan ekspresi wajah (*non-manual expression*) (Tim Produksi Bahasa Isyarat Jakarta, 2014: 1). Berdasarkan hal tersebut dapat kita ketahui bahwa parameter dalam memproduksi isyarat merupakan faktor penting yang tidak dapat dikesampingkan ketika seseorang berinteraksi dengan menggunakan bahasa isyarat. Pengabaian terhadap parameter-parameter tersebut dapat membuat isyarat yang disampaikan memiliki makna yang tidak sesuai dengan apa yang ingin disampaikan.

Kemudian pada literatur berjudul “Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode *Convolutional Neural Network*” (Bakti., 2019). Dalam paper ini membahas pengenalan isyarat angka SIBI dengan

menggunakan metode Convolutional Neural Network (CNN). Arsitektur CNN yang digunakan adalah arsitektur LeNet. Arsitektur CNN dapat dimodifikasi sesuai kebutuhan. LeNet menjadi salah satu struktur CNN yang sesuai dalam pemrosesan gambar. Terbukti dengan menggunakan arsitektur LeNet tingkat akurasi training dan testing mencapai 90% bahkan lebih. Dalam proses training setiap tahapnya dapat mempengaruhi nilai akurasi. Terbukti dari 25 sampai 100 epoch yang telah dilakukan, nilai akurasi dalam training data terus meningkat hingga mencapai 96.44% dengan nilai loss 0.13%. Hal tersebut juga dibuktikan dalam hasil keluaran dari proses testing data. Data test dapat diprediksi dengan baik hingga akurasinya mencapai 98,89%. Dari percobaan yang telah dilakukan, proses prediksi data pada kelas angka 2 selalu mengalami kesalahan prediksi data.

Kemudian pada literatur berjudul “Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network” (Fadillah, 2020). Penelitian ini mengusulkan pengembangan model penerjemah Bahasa Isyarat Indonesia (Bisindo) dengan memanfaatkan teknologi machine learning khususnya Convolutional Neural Network (CNN). Berbeda dengan penelitian yang sudah ada, Bisindo adalah bahasa isyarat yang relatif banyak digunakan Tuli namun tidak resmi, sehingga dataset yang diperlukan hampir tidak ada. Tujuan dari penelitian ini yaitu bertambahnya jumlah penerjemah Bisindo elektronik untuk meningkatkan aksesibilitas Tuli. Selain itu, penelitian ini juga bertujuan untuk mengevaluasi metode parameter-transfer untuk mengatasi keterbatasan jumlah data pada dataset dan implementasi pada sistem bahasa isyarat yang berbeda. Pengembangan model Bisindo dilakukan dengan memanfaatkan arsitektur Model American Sign Language (ASL) melalui dua pendekatan berdasarkan pemanfaatan parameter model tersebut, yang kemudian disebut Model A dan Model B. Model A dikembangkan tanpa menggunakan parameter dari Model ASL (tanpa parameter-transfer), sedangkan Model B dikembangkan dengan menggunakan parameter Model ASL serta knowledge parameter di dalamnya (dengan parameter-transfer). Metode yang digunakan dalam pengembangan adalah eksperimen dan analisis data untuk menentukan model terbaik secara kuantitatif dengan analisis variabel durasi training, akurasi saat testing dan F1 Score saat testing. Model ASL di penelitian

memiliki akurasi testing sebesar 96.40%. Hasil penelitian menunjukkan bahwa Model A menghasilkan akurasi testing sebesar 94.38% sedangkan Model B sebesar 30%. Dapat disimpulkan bahwa parameter-transfer pada Model B memerlukan waktu training yang lebih sedikit dibandingkan Model A serta mampu mempelajari fitur Bisindo yang memiliki kemiripan dengan fitur ASL. Namun, untuk mempelajari keseluruhan alfabet Bisindo, pemanfaatan arsitektur tanpa parameter-transfer merupakan pendekatan yang lebih baik dibandingkan menggunakan parameter-transfer.

Kemudian pada literatur berjudul “Sistem Pengenalan Bahasa Isyarat Indonesia Dengan Menggunakan Metode Fuzzy K-Nearest Neighbor” (Gafar, 2017). Proses pengenalan pada penelitian ini dibangun menggunakan metode Fuzzy K-Nearest Neighbor (FKNN), metode ini memiliki dua keunggulan utama daripada algoritma K-Nearest Neighbor. Pertama, algoritma Fuzzy K-Nearest Neighbor (FKNN) mampu mempertimbangkan sifat ambigu dari tetangga jika ada. Algoritma ini sudah dirancang sedemikian rupa agar tetangga yang ambigu tidak memainkan peranan penting dalam klasifikasi. Keunggulan kedua, yaitu sebuah interface akan memiliki derajat nilai keanggotaan pada setiap kelas sehingga akan lebih memberikan kekuatan atau kepercayaan suatu instance yang berada pada suatu kelas. Dengan menerapkan metode Fuzzy K-Nearest Neighbor (FKNN) pada proses klasifikasi bahasa isyarat, maka proses klasifikasi bisa dilakukan dengan lebih objektif. Untuk mendukung peningkatan tingkat akurasi dari penelitian sebelumnya, maka penelitian ini memilih metode Fuzzy K-Nearest Neighbor (FKNN) untuk pengenalan Sistem Isyarat Bahasa Indonesia (SIBI). Hasil Sistem pengenalan bahasa isyarat Indonesia dengan menggunakan metode Fuzzy K-Nearest Neighbor (KNN) diperoleh nilai akurasi sebesar 88,8%, hal ini menunjukkan metode yang diajukan mampu melakukan mengenali model bahasa isyarat dengan baik. Dengan melakukan percobaan sebanyak 5 kali, masih ada beberapa huruf yang konsisten belum bisa dikenali, seperti huruf C, E, L, U dan V yang cenderung memiliki nilai kemiripan yang dekat.

4.2 Pengumpulan Dataset

4.2.1 Pengambilan Dataset Langsung

Pengumpulan data yang dilakukan pada penelitian ini menggunakan proses pengambilan langsung melalui webcam. Peneliti memperagakan gestur bahasa isyarat indonesia secara langsung dengan posisi webcam berada didepan user yang berjarak sekitar 50-100 cm. Kondisi lingkungan berada didalam ruangan yang memiliki pencahayaan lampu cukup terang dengan *background* tembok berwarna putih. Pengambilan data dilakukan untuk mendapatkan data citra gestur bahasa isyarat abjad dan angka kemudian melakukan proses pengolahan citra digital sampai menjadi dataset yang siap digunakan oleh *mechine learning*. Jenis gestur bahasa isyarat abjad dan angka yang digunakan dalam dataset dibedakan menjadi 36 kelas yaitu abjad A-Z dan angka 1-10.

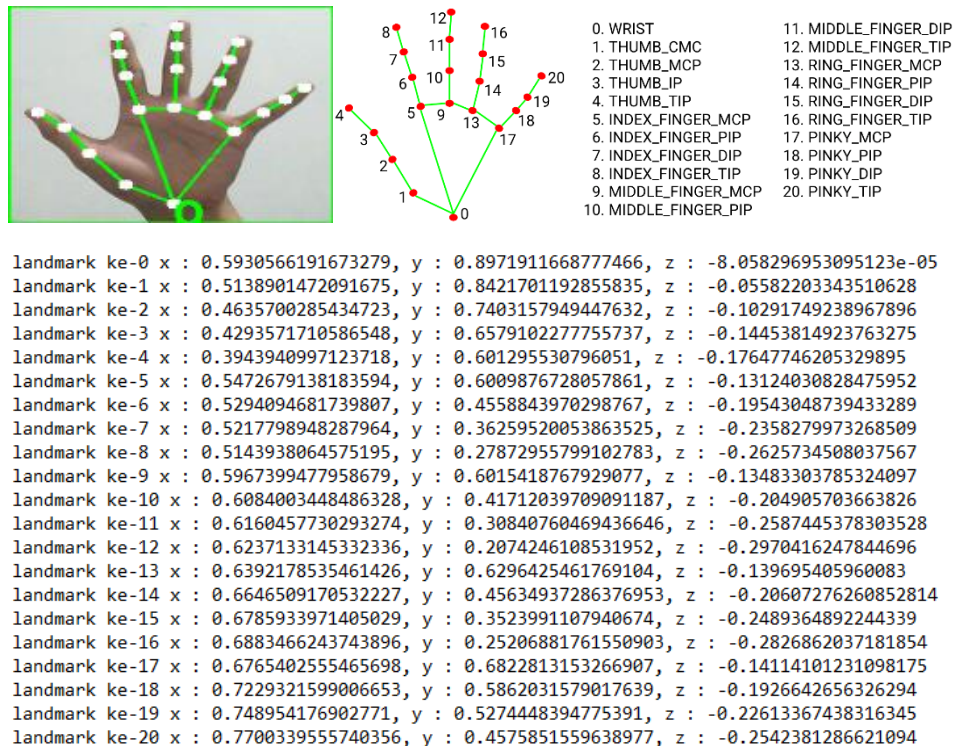
4.2.2 Tahapan Pengolahan Citra Dataset

Sebelum data citra bahasa isyarat disimpan, data citra akan melalui beberapa tahap proses pengolahan citra digital. Proses tersebut berupa deteksi objek, *preprocessing*, dan segmentasi data citra. Tahapan dari pengolahan citra akan dijelaskan di subbab berikut :

a. Deteksi objek

Objek penelitian yang digunakan adalah kedua tangan manusia. Maka dari itu peneliti menggunakan MediaPipe sebagai framework bantuan untuk mendeteksi tangan dan jari. Dalam implementasinya citra yang berasal dari *camera / webcam* harus dalam bentuk RGB dan akan diproses oleh MediaPipe untuk memprediksi dan menggambar objek / landmark tangan manusia. Hasil output dari deteksi objek oleh Mediapipe berupa *multihand landmarks* yaitu kumpulan hasil deteksi / pelacakan tangan, di mana setiap tangan direpresentasikan sebagai daftar 21 *landmark* tangan dan setiap *landmark* terdiri dari x, y, dan z. x dan y dinormalisasi ke [0.0, 1.0] masing-masing dengan lebar dan tinggi gambar (ratio image). z mewakili kedalaman *landmark* dengan kedalaman di pergelangan tangan sebagai titik asal, dan semakin kecil nilainya, semakin dekat *landmark* itu ke kamera.

Besarnya z menggunakan skala yang kira-kira sama dengan x . Data hasil deteksi objek akan dimanfaatkan untuk membantu melakukan *preprocessing* data citra, terutama untuk membuat bounding box disekitar telapak tangan manusia. Berikut adalah hasil dari deteksi objek yang dilakukan oleh Mediapipe :



Gambar 4.1 Hasil Deteksi Objek Mediapipe

b. Preprocessing Data Citra

Preprocessing adalah proses dimana citra yang telah masuk ke sistem akan diproses melalui beberapa tahap, meliputi :

1. Crop Image

Proses *crop image* akan dilakukan untuk menghapus/memotong semua bagian gambar / frame diluar lokasi *bounding box* yang sudah didapatkan diproses deteksi objek. Hal ini bertujuan untuk mendapatkan *region of interest* berupa gambar telapak tangan manusia. Berikut adalah hasil dari proses *crop image* :

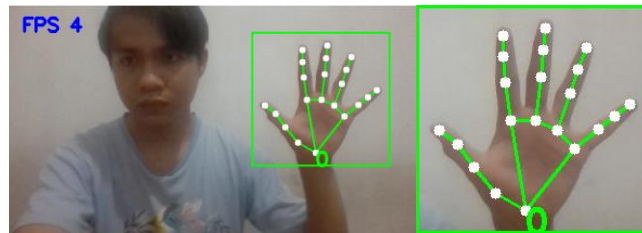


Figure 4.1 Hasil Crop Image Deteksi Objek

2. Resize

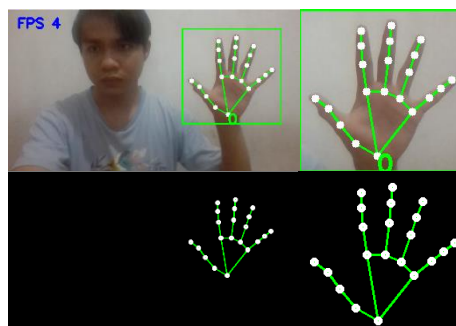
Sebelum data citra disimpan kedalam folder *dataset*, data citra harus melalui proses *resize* / merubah ukuran citra (180 pixels * 120 pixels) dengan tujuan keseragaman / konsistensi dataset citra.

3. Operator Laplacian

Operator Laplacian digunakan untuk mendapatkan nilai *blur* dari suatu citra. Peneliti ingin melakukan *filter* pada citra yang disimpan, data citra yang disimpan haruslah memiliki nilai minimal *blur* sebesar 500.

c. Segmentasi Data Citra

Proses segmentasi citra digunakan untuk memisahkan objek penelitian dengan latar yang terdapat dalam sebuah citra. Proses tersebut meliputi mendapatkan data landmark hasil proses hand detection MediaPipe, membuat bounding box disekitar objek tangan yang sudah terdeteksi, dan membuat ulang gambar landmark tangan dengan background hitam sesuai bounding box dan hasil proses hand detection MediaPipe.


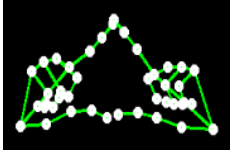



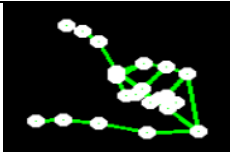

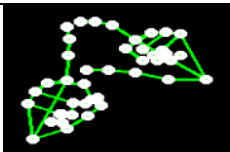

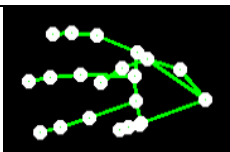

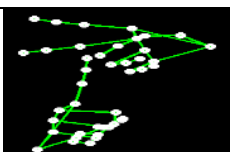





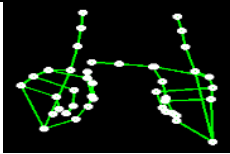

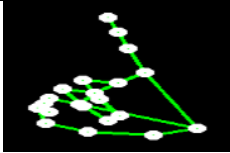

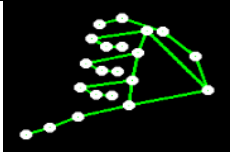

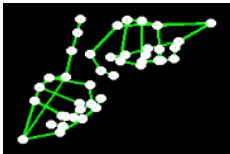

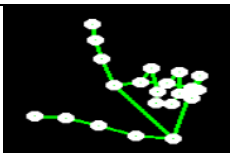

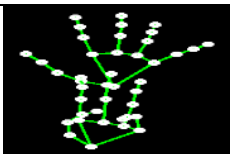



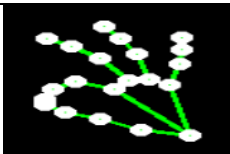

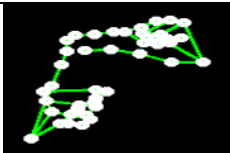
Gambar 4.2 Hasil Segmentasi Data Citra


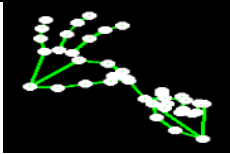

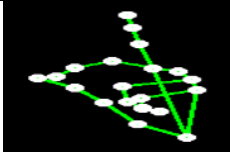

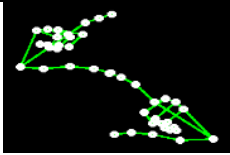

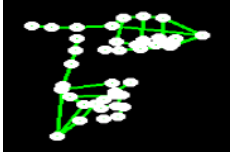

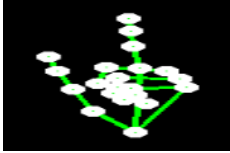

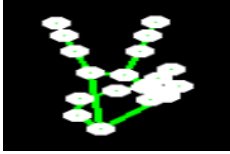

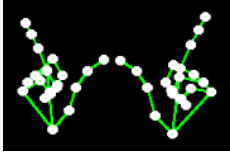



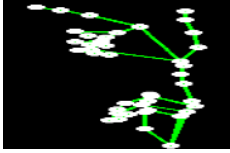
4.2.3 Hasil Dataset Gestur Bahasa Isyarat Indonesia




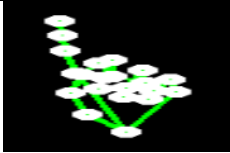

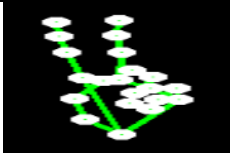

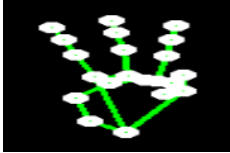

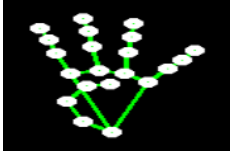

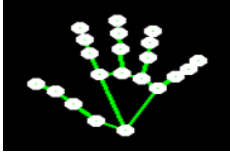

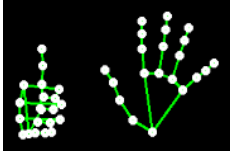

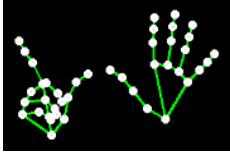

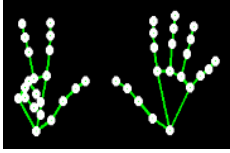
Hasil proses pengolahan citra dataset kemudian dikelompokkan dengan dimasukkan kedalam folder berdasarkan kelas / label yang telah ditentukan. Berikut adalah hasil dari pengumpulan dataset citra digital bahasa isyarat indonesia :


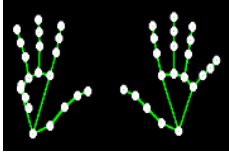

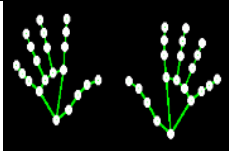
Tabel 4.1 Klasifikasi Bahasa Isyarat Indonesia Abjad dan Nomor

No	Jenis Klasifikasi	Label	Data Citra	
			Sebelum PCD	Setelah PCD
1	A	0		
2	B	1		
3	C	2		
4	D	3		
5	E	4		
6	F	5		
7	G	6		

8	H	7		
9	I	8		
10	J	9		
11	K	10		
12	L	11		
13	M	12		
14	N	13		
15	O	14		
16	P	15		

17	Q	16		
18	R	17		
19	S	18		
20	T	19		
21	U	20		
22	V	21		
23	W	22		
24	X	23		
25	Y	24		

26	Z	25		
27	1	26		
28	2	27		
29	3	28		
30	4	29		
31	5	30		
32	6	31		
33	7	32		
34	8	33		

35	9	34		
36	10	35		

Berdasarkan Tabel 4.1 Klasifikasi Bahasa Isyarat Indonesia Abjad dan Nomor terdapat 36.576 dataset citra bahasa isyarat yang digunakan, terdiri dari 36 kelas yaitu klasifikasi abjad A-Z dan nomor 1-10. Pada proses pembuatan sistem jumlah data *training* yang digunakan adalah sebanyak 23.424, jumlah data yang digunakan untuk proses *testing* adalah 7.320 dan jumlah data yang digunakan untuk proses *validation* adalah 5.856. Jumlah data testing tersebut berasal dari pembagian 20% dari total dataset, kemudian sisa dataset digunakan 20% lagi untuk data *validation* dan sisanya digunakan untuk data *training*. Tidak ada aturan khusus mengenai jumlah data yang digunakan. Namun, semakin banyak data *training* akan semakin baik jaringan syaraf tiruan mengenali gestur yang akan diuji.

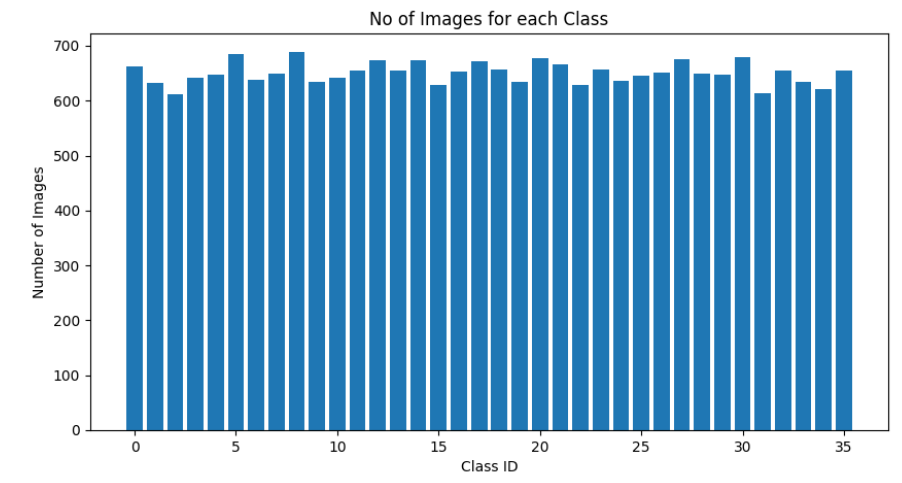
4.3 Perancangan dan Pembuatan Sistem

4.3.1 Sistem Pelatihan Dataset

a. Proses Import Dataset kedalam Sistem Training

Data latih yang digunakan adalah 64%, data *testing* sebanyak 20% dan data *validation* sebanyak 16% dari total keseluruhan data. Sehingga didapat data latih sebanyak 23.424 gambar dengan masing-masing kelas kurang lebih sebanyak 600 data sampel. Pada tahap ini sistem secara otomatis melakukan pelacakan dataset citra dan memasukkan semua dataset citra beserta pasangan labelnya kedalam *array* gambar dan label. Selanjutnya terjadi proses pemisahan acak keseluruhan dataset menjadi dataset pelatihan, pengujian dan validasi yang dilakukan dengan bantuan fungsi *train_test_split*. Fungsi tersebut merupakan salah satu fungsi didalam

library python bernama scikit-learn. Berikut adalah hasil dari pembagian data latih citra (Distribution Plot).



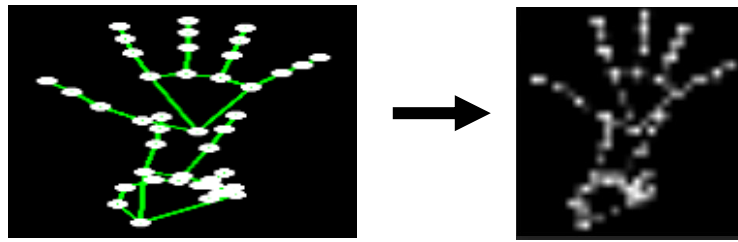
Gambar 4.3 Distribution Plot Data Training

b. Pengolahan Data Citra dan One Hot Encode Dataset

Setiap dataset pelatihan, pengujian dan validasi akan melalui beberapa tahapan proses sebelum dilakukannya pelatihan data. Tahapan ini bertujuan untuk menyesuaikan data citra dan label dengan sistem arsitektur Keras Tensorflow CNN. Proses tersebut berupa *Preprocessing*, *Image Augmentation* dan *One Hot Encode*. Proses tersebut akan dijelaskan di dalam subbab berikut :

1. Preprocessing

Preprocessing adalah tahap awal proses pengolahan citra. Salah satu proses tersebut meliputi fungsi *resize image* untuk merubah ukuran pada setiap dataset citra dari ukuran 180x120 pixels menjadi citra ukuran 32x32 pixels. Selanjutnya yaitu proses konversi RGB ke *grayscale* dan membuat efek kontras dengan bantuan metode *equalizeHist* openCV yang berfungsi untuk menyeimbangkan histogram dari setiap gambar. Kemudian bentuk citra akan dirubah menjadi *single channel image* / citra biner dengan bantuan *library array numpy* yaitu fungsi *reshape*. Berikut adalah komparasi sebelum dan sesudah proses *preprocessing* :



Gambar 4.4 Komparasi Hasil Preprocessing Pelatihan Dataset

2. Image Augmentation

Augmentasi citra adalah proses yang dapat meningkatkan akurasi dari model CNN yang dilatih karena proses pelatihan data akan mendapatkan data-data tambahan yang dapat berguna untuk membuat model yang dapat melakukan generalisasi dengan lebih baik. Proses Augmentasi citra pada tahap ini dilakukan dengan menggunakan fungsi dari `keras.preprocessing.image` yaitu `ImageDataGenerator`. Parameter yang digunakan adalah :

- a) `width_shift_range=0.1`

Menggeser gambar sepanjang dimensi lebar.

- b) `height_shift_range=0.1`

Menggeser gambar sepanjang dimensi ketinggian.

- c) `zoom_range=0.2`

zoom secara acak. Jika float, [bawah, atas] = [1-zoom_range, 1+zoom_range].

- d) `shear_range=0.1`

Intensitas Geser yang membuat gambar akan terdistorsi sepanjang sumbu (Sudut geser berlawanan arah jarum jam dalam derajat).

- e) `rotation_range=10`

Rentang derajat untuk rotasi acak.

3. One Hot Encode

One Hot Encoding adalah salah satu metode encoding yang merepresentasikan data bertipe kategori sebagai vektor biner

bernilai integer, 0 dan 1. Peneliti menggunakan bantuan library `keras.utils.np_utils` yaitu fungsi `to_categorical`. Berikut adalah contoh hasil implementasi pada salah satu label dataset :

Label : 18

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

c. Mendefinisikan dan Memulai Pelatihan Data

Arsitektur CNN LeNeT yang dirancang terdiri dari 4 lapisan konvolusi diikuti oleh fungsi aktivasi Relu dan 2 lapisan Maxpooling. Diikuti dengan *dropout*, *flatten*, *density layer* yang diakhiri dengan *softmax activation layer* dengan 36 output (26 huruf dan 10 angka), yang akan menunjukkan persentase klasifikasi yang diperoleh untuk setiap kelas. Proses data training menggunakan parameter sebagai berikut :

1. `batch_size=50`

Batch Size adalah jumlah sampel data yang disebarkan ke Neural Network.

2. `steps_per_epoch= 468`

Steps Per Epoch adalah jumlah iterasi batch sebelum periode pelatihan dianggap selesai.

3. `epochs=156`

Epoch adalah ketika seluruh dataset sudah melalui proses training pada Neural Network sampai dikembalikan ke awal untuk sekali putaran.

4. `shuffle=1`

Mengacak data pelatihan sebelum melakukan setiap epoch.

Tabel 4.2 CNN Model Architecture

Layer (type)	Output Shape	Parameter
conv2d (Conv2D)	(None, 28, 28, 60)	1560
conv2d_1 (Conv2D)	(None, 24, 24, 60)	90060

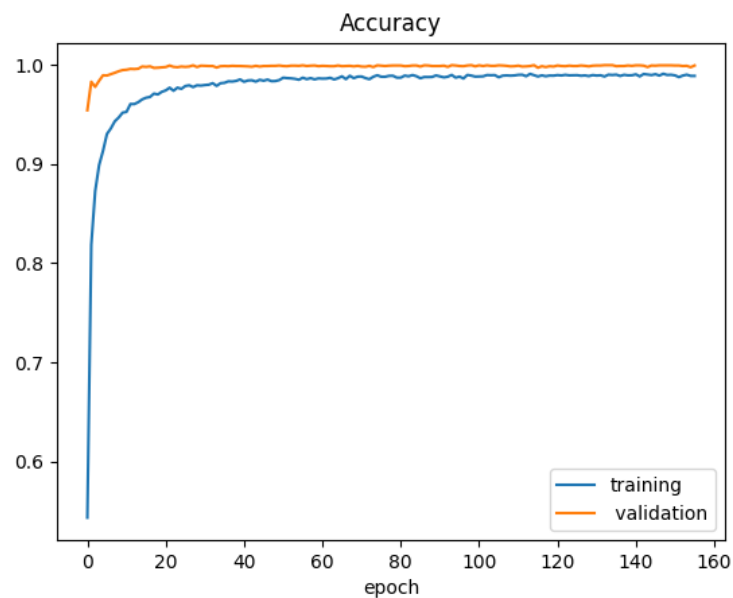
max_pooling2d (MaxPooling2D)	(None, 12, 12, 60)	0
conv2d_2 (Conv2D)	(None, 10, 10, 30)	16230
conv2d_3 (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 30)	0
dropout (Dropout)	(None, 4, 4, 30)	0
flatten (Flatten)	(None, 480)	0
dense (Dense)	(None, 500)	240500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 36)	18036
Total parameters: 374,516		
Trainable parameters: 374,516		
Non-trainable parameters: 0		

Dilihat dari informasi Tabel 4.2 CNN Model Architecture, Model arsitektur tersebut menggunakan gambar input 32x32 piksel *single channel*, operasi konvolusi pertama diterapkan dengan *filter* 5x5 untuk menghasilkan 60 peta fitur ukuran 28x28. Operasi konvolusi kedua diterapkan yang *filter*-nya 5x5 untuk menghasilkan 60 peta fitur ukuran 24x24. *Filter* max pooling 2x2 diterapkan dan menghasilkan peta fitur ukuran 12x12. Selanjutnya melakukan 2 operasi konvolusi terakhir dengan ukuran dan jumlah *filter* yang sama yaitu 3x3 dan 30, menghasilkan peta fitur 30 dengan ukuran 8x8. Kemudian menerapkan *filter max pooling* 2x2 menghasilkan peta fitur 4x4. Setelah tahap ekstraksi fitur, dilakukan klasifikasi dari *layer flatten*, *dense*, *dropout* dan *fully connected*. CNN ini diakhiri dengan fungsi *softmax*, yang

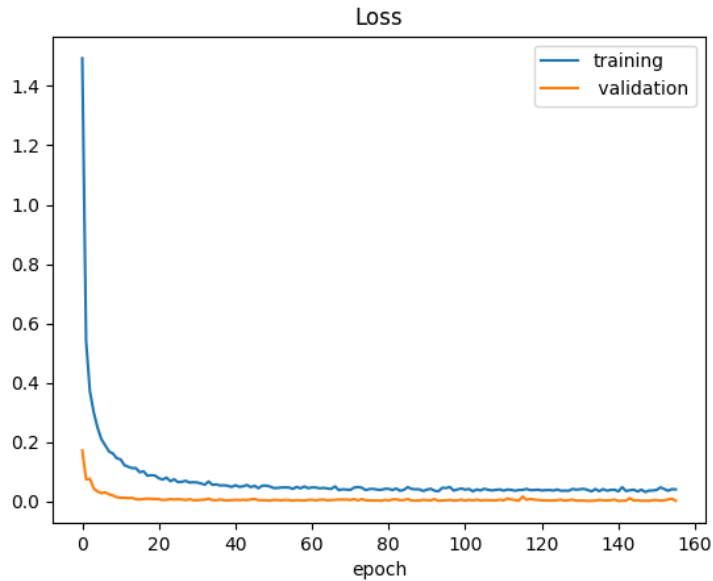
memberikan probabilitas untuk melakukan klasifikasi masing-masing karakter.

d. Hasil Training Data

Hasil training memberikan akurasi yang baik. Grafik dari akurasi dan kesalahan proses training disajikan pada Gambar 4.5 Grafik Akurasi Data Training dan Gambar 4.6 Grafik Kesalahan Data Training. Data Training memiliki nilai test score sebesar 0.004841047804802656 dan nilai test accuracy sebesar 0.9989070892333984.



Gambar 4.5 Grafik Akurasi Data Training



Gambar 4.6 Grafik Kesalahan Data Training

4.3.2 Sistem Klasifikasi Citra / Testing

Penelitian ini menggunakan metode *deep learning* CNN untuk memproses citra *input* dan model hasil dataset *training* untuk mengklasifikasikan citra dan menerjemahkannya kedalam bentuk tulisan. Ada beberapa tahapan utama didalam sistem klasifikasi, berikut adalah penjelasannya :

a. Menangkap dan Memproses Citra Secara Realtime

Peneliti memperagakan gestur bahasa isyarat Indonesia, kemudian sistem menangkap citra / vidio secara *realtime* melalui *webcam* komputer. Citra inputan akan melaui tahapan preprocessing dan normalisasi citra dengan konsep dan alur yang sama seperti didalam sistem pengumpulan dataset dan sistem pelatihan dataset. Berikut adalah daftar proses pengolahan citra, sebelum dilakukannya proses klasifikasi dengan metode CNN :

1. Deteksi Objek Tangan.
2. Preprocessing Data Citra
 - a) Crop Image
 - b) Resize
 - c) Operator Laplacian
3. Segmentasi Data Citra

4. Normalisasi Data Citra

b. Proses Klasifikasi CNN

Proses *testing* merupakan proses klasifikasi menggunakan bobot dan bias dari hasil proses *training* atau bisa disebut model data *training*. Dengan bobot dan bias yang baru proses *feedforward* diterapkan yang kemudian menghasilkan lapisan *output*. Lapisan *output* sudah *fully connected* dengan label yang disediakan. Hasil *fully connected* tersebut diperoleh data yang gagal dan berhasil diklasifikasi. Berikut adalah implementasi kode program dengan menggunakan Keras tensorflow untuk melakukan klasifikasi metode deep learning CNN :

```
from tensorflow import keras

# Load Model Data Training
self.model_all = keras.models.load_model(
    self.basicTools.getBaseUrl() +
    '/Resources/model/model_trained_all.h5')

# Prediction
all_classIndex = int(self.model_all.predict_classes(imgRoi))
all_predictions = self.model_all.predict(imgRoi)
all_proVal = np.amax(all_predictions)

classIndex = all_classIndex
predictions = all_predictions
proVal = all_proVal
predictionType = 'ALL'

return classIndex, predictions, proVal, predictionType
```

c. Hasil Klasifikasi Citra

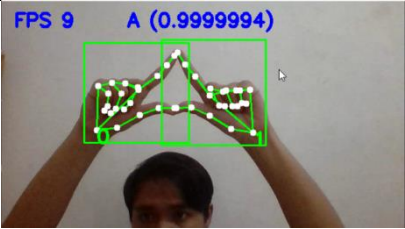
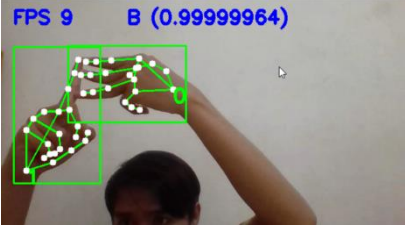
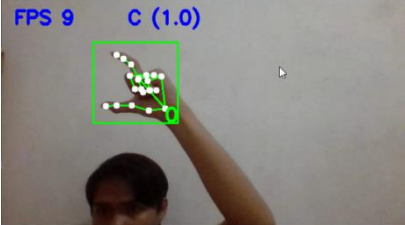
Nilai *return* dari proses klasifikasi citra oleh keras tensorflow berupa *Index* Kelas, nilai prediksi, dan nilai *possibility* / kemungkinan prediksi dari citra *input*-an. Peneliti memberi batas ambang nilai *possibility* sebesar 0.9 atau 90% untuk menentukan dan memunculkan hasil prediksi dari perhitungan metode CNN. Berikut adalah potongan kode program :

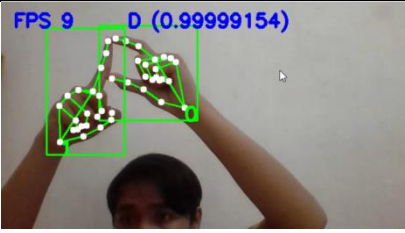
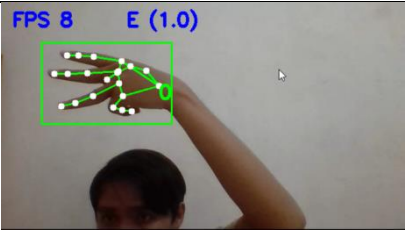
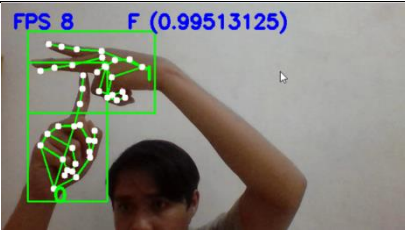
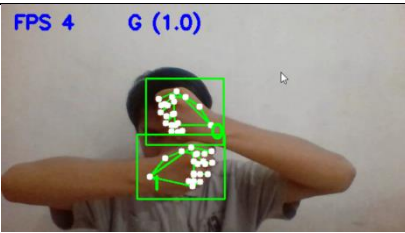
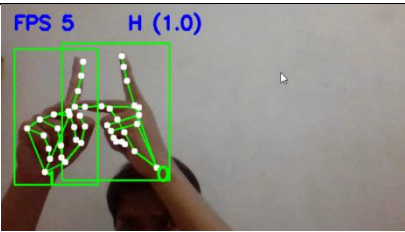
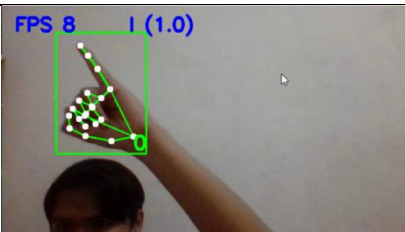

```
# Predict
classIndex, predictions, proVal, predictionType =
prediction.predictAll(imgRoi)

# Show Prediction
if proVal >= threshold:
    cv2.putText(img, translation.mapper(classIndex,
predictionType) + ' (' + str(proVal) + ')', (200, 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, globalColor, 3)
```

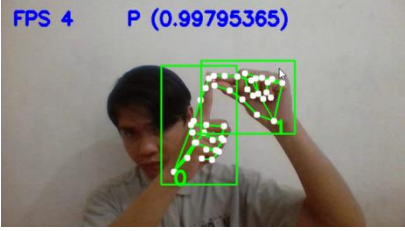
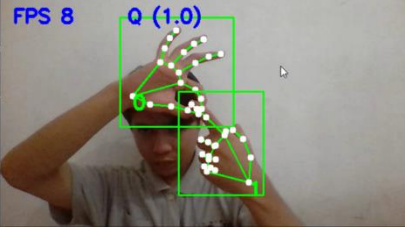
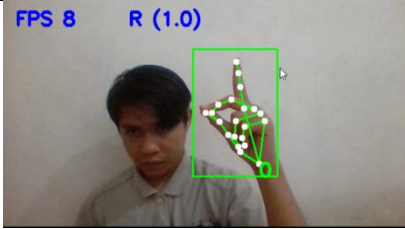
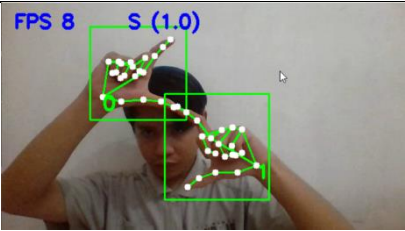
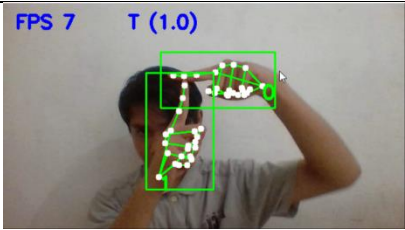
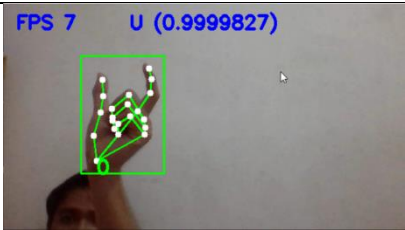
Nilai hasil prediksi akan dimunculkan secara *realtime* didalam layar. Berikut adalah hasil uji coba sistem klasifikasi terhadap semua label dataset :

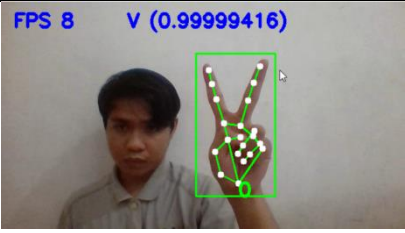
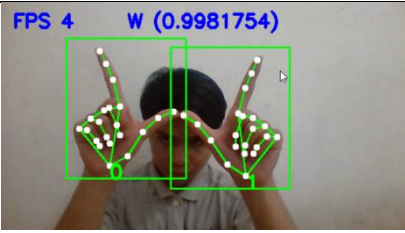
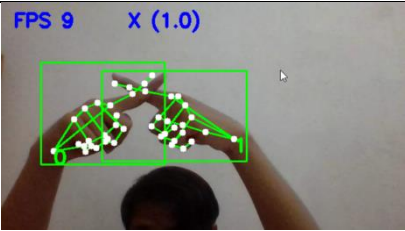
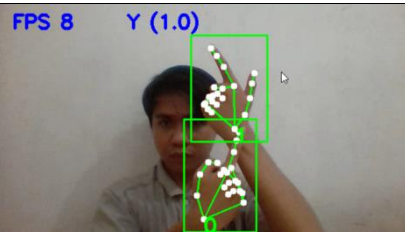
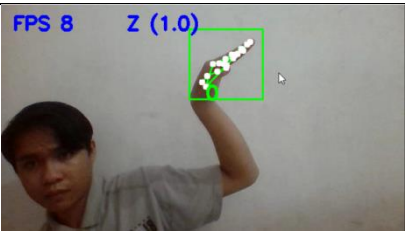

Tabel 4.3 Uji Coba Hasil Klasifikasi Citra

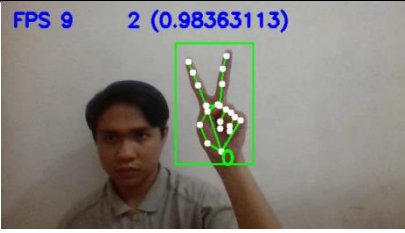
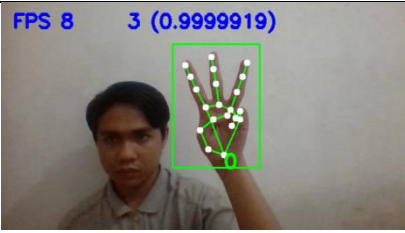
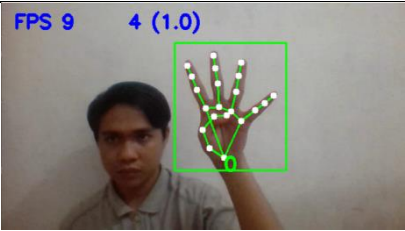
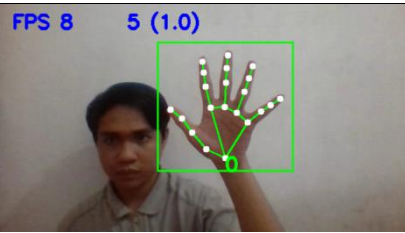
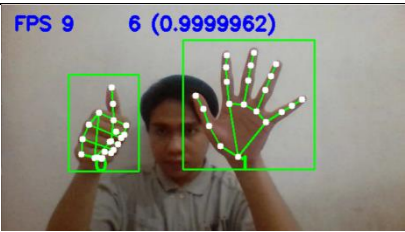
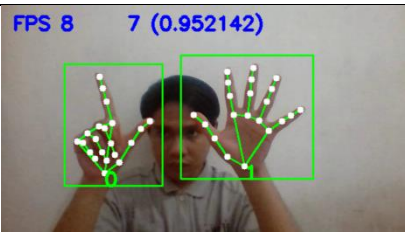
No	Label	Output Layar	Nilai Possibility %
1	A		99.9%
2	B		99.9%
3	C		100%

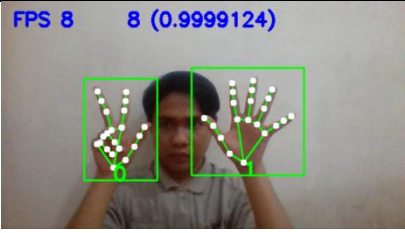
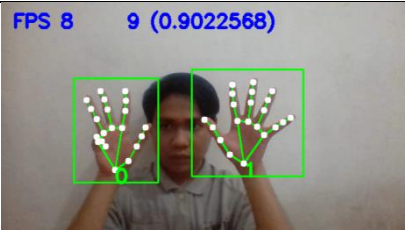
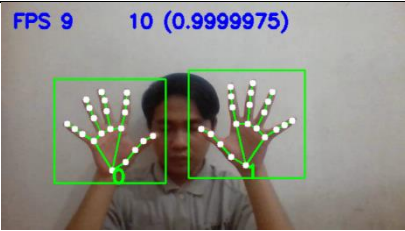
4	D		99.9%
5	E		100%
6	F		99.5%
7	G		100%
8	H		100%
9	I		100%

10	J	<p>FPS 9 J (1.0)</p>	100%
11	K	<p>FPS 8 K (0.9999999)</p>	99.9%
12	L	<p>FPS 8 L (0.9995047)</p>	99.9%
13	M	<p>FPS 8 M (0.99971515)</p>	99.9%
14	N	<p>FPS 8 N (0.9999994)</p>	99.9%
15	O	<p>FPS 8 O (1.0)</p>	100%

16	P		99.7%
17	Q		100%
18	R		100%
19	S		100%
20	T		100%
21	U		99.9%

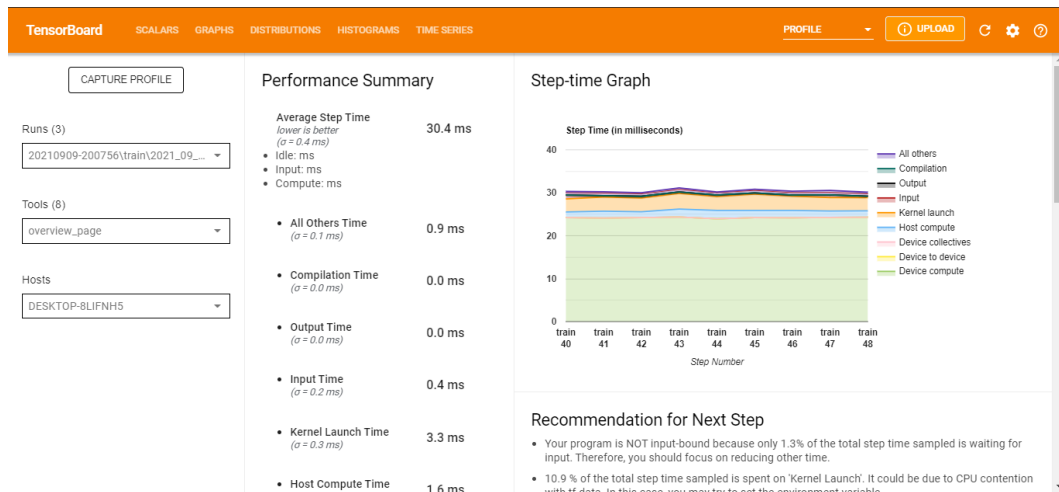
22	V	 <p>FPS 8 V (0.99999416)</p>	99.9%
23	W	 <p>FPS 4 W (0.9981754)</p>	99.8%
24	X	 <p>FPS 9 X (1.0)</p>	100%
25	Y	 <p>FPS 8 Y (1.0)</p>	100%
26	Z	 <p>FPS 8 Z (1.0)</p>	100%
27	1	 <p>FPS 9 1 (0.99998474)</p>	99.9%

28	2		98.3%
29	3		99.9%
30	4		100%
31	5		100%
32	6		99.9%
33	7		95.2%

34	8		99.9%
35	9		90.2%
36	10		99.9%

4.4 Analisis Hasil Penelitian

Pada analisis proses data *training* peneliti menggunakan alat yang tersedia dengan TensorFlow Profiler untuk melacak kinerja model. TensorFlow Profiler mempelajari kinerja hasil model data *training* pada *host* (CPU), perangkat (GPU), atau pada kombinasi *host* dan perangkat. TensorFlow Profiler membantu memahami konsumsi sumber daya perangkat keras (waktu dan memori) dari berbagai operasi TensorFlow dalam hasil model data *training* dan mengatasi kemacetan kinerja dan pada akhirnya membuat model dijalankan lebih cepat. Dalam melakukan analisis, peneliti menentukan batch yang akan dianalisis menggunakan Tensorflow Profiler yaitu kinerja model dari *batch* 40 hingga 48 dari total 50 *batch* yang dilakukan. Penentuan tersebut dilakukan dengan fungsi/argumen *profile_batch* yang fungsinya memberitahu tensorflow untuk merekam kinerja eksekusi dan membuat profilnya.



Gambar 4.7 Overview page TensorFlow Profiler

4.4.1 Performance Summary

Menampilkan ringkasan performa pada model tensorflow yang sudah dibuat. Kunci untuk menganalisis gambaran / ringkasan performa adalah berapa banyak waktu langkah dari eksekusi perangkat yang sebenarnya, persentase operasi yang ditempatkan pada perangkat *vs host* dan berapa banyak *kernel* yang menggunakan fp16 (16bit). Berikut adalah penjelasan beberapa bagian ringkasan performa :

- a. **Step-time breakdown.** Mengurai rata-rata waktu proses menjadi beberapa kategori. Dalam kasus yang ideal peneliti ingin model menghabiskan sebagian besar waktunya untuk benar-benar melakukan “*Data Training*”, yaitu membuat GPU sesibuk mungkin (Waktu komputasi perangkat harus yang tertinggi, dan semua *overhead* lainnya harus serendah mungkin). Berikut adalah detail hasil analisis *step-time breakdown* :

Tabel 4.4 Hasil Analisis Step-Time Breakdown

No	Kategori Langkah / Proses	Rata Rata Waktu
1	All Others Time ($\sigma = 0.1$ ms)	0.9 ms
2	Compilation Time ($\sigma = 0.0$ ms)	0.0 ms
3	Output Time ($\sigma = 0.0$ ms)	0.0 ms

4	Input Time ($\sigma = 0.2$ ms)	0.4 ms
5	Kernel Launch Time ($\sigma = 0.3$ ms)	3.3 ms
6	Host Compute Time ($\sigma = 0.2$ ms)	1.6 ms
7	Device Collective Communication Time ($\sigma = 0.0$ ms)	0.0 ms
8	Device to Device Time ($\sigma = 0.0$ ms)	0.0 ms
9	Device Compute Time ($\sigma = 0.1$ ms)	24.2 ms

Hasilnya adalah rata - rata waktu Langkah / proses sebesar **30.4 ms** (semakin sedikit semakin efisien) dengan $\sigma = 0.4$ ms. Sebagian besar waktu untuk setiap proses (yaitu 24.2 ms) dihabiskan untuk "*Device Compute Time*". Ini berarti bahwa GPU tetap sibuk dengan operasi seperti menghitung peta aktivasi dan kemudian menghitung gradien dan menyebarkannya kembali. Karena "*Input Time*" adalah 0.4 ms, ini berarti model tidak benar-benar diam saat data dimuat ke GPU. Ini adalah pertanda baik karena kapasitas komputasi sedang digunakan sepenuhnya. Pada proses "*Kernel Launch Time*" membutuhkan waktu yang cukup tinggi dibanding yang lain, tetapi sebenarnya itu dapat diperbaiki dengan menyetel flag *TF_GPU_THREAD_MODE* ke *gpu_private*. Berikut adalah penjelasan singkat mengenai kategori langkah / proses :

1. All others, including Python overhead : Semua proses yang lain, termasuk overhead Python.
2. Compilation : Waktu yang dihabiskan untuk mengkompilasi kernel.
3. Output : Waktu yang dihabiskan untuk membaca data output.
4. Input : Waktu yang dihabiskan untuk membaca data input.
5. Kernel launch : Waktu yang dihabiskan oleh host untuk meluncurkan *kernel*.
6. Host compute time : Waktu komputasi *host* (CPU).
7. Device-to-device communication time : Waktu komunikasi antara perangkat.

8. On-device compute time : Waktu komputasi di perangkat.

b. TF Op Placement. *Op TF Placement* menunjukkan berapa persen operasi yang dijalankan pada *host* (CPU) vs perangkat (GPU). Untuk memaksimalkan penggunaan GPU, Perangkat *Op Placement* harus dimaksimalkan. Berikut adalah hasil presentase *TF Op Placement* (umumnya ingin memiliki lebih banyak operasi di perangkat / GPU) :

1. Host: 10.2%
2. Device: 89.8%

Hasil data menunjukkan bahwa sebagian besar operasi sudah berada di bagian perangkat / GPU. Salah satu alasan untuk menempatkan sebagian besar operasi pada GPU adalah untuk mencegah salinan memori yang berlebihan antara *host* dan perangkat (salinan memori untuk data *input/output* model antara host dan perangkat yang diharapkan). Hal ini karena salinan terkadang dapat merusak kinerja jika mereka memblokir *kernel* GPU dari eksekusi.

c. Op Time Spent on Eager Execution. Waktu yang dihabiskan pada proses *eager execution*. *TensorFlow's eager execution* adalah lingkungan pemrograman imperatif yang mengevaluasi operasi dengan segera, tanpa membuat grafik atau bisa dibilang operasi mengembalikan nilai konkret alih-alih membuat grafik komputasi untuk dijalankan nanti. Berikut adalah hasil nilai *Op Time Spent on Eager Execution* (semakin sedikit semakin bagus) :

1. Host: 1.3%
2. Device: 0.0%

Persentase yang lebih rendah menunjukkan bahwa kode dioptimalkan dengan menyematkan instruksi ke dalam struktur data berbasis grafik yang dapat dioptimalkan.

d. Device compute precisions. Melaporkan persentase waktu komputasi perangkat yang menggunakan komputasi 16 dan 32-bit. Berikut adalah hasil presentase *device compute precisions* (dari Total Waktu Perangkat):

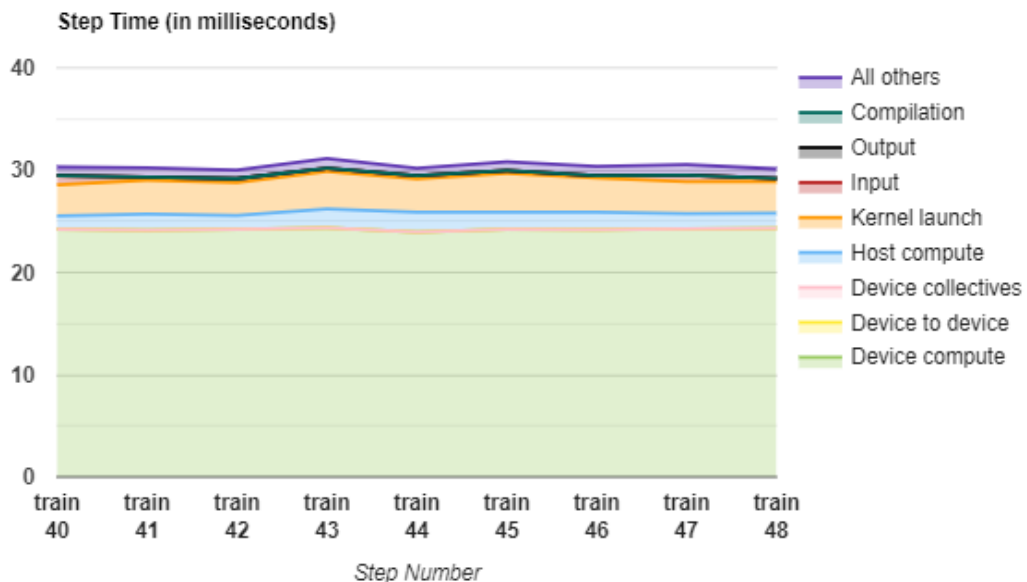
1. 16-bit: 0.0%

2. 32-bit: 100.0%

Dari hasil *device compute precisions* Hanya 0.0% dari perhitungan perangkat adalah 16 bit. Jadi, untuk meningkatkan kinerja (jika akurasi yang berkurang dapat diterima) peneliti bisa mengganti lebih banyak Ops 32 bit dengan Ops 16-bit.

4.4.2 Step-Time Graph.

Grafik waktu langkah memiliki sumbu x yang merupakan plot dari nomor langkah, sedangkan sumbu y (waktu langkah) adalah waktu yang diperlukan untuk melakukan langkah yang sesuai dengan sumbu x. Sehingga grafik menampilkan waktu langkah perangkat (dalam milidetik) untuk semua langkah yang diambil sampelnya (*batch* ke 40-48 dari total 50 *batch*). Setiap langkah dipecah menjadi beberapa kategori (dengan warna berbeda) di mana waktu dihabiskan. Area merah sesuai dengan porsi waktu langkah perangkat dalam keadaan diam menunggu data input dari host. Area hijau menunjukkan berapa lama perangkat benar-benar berfungsi.



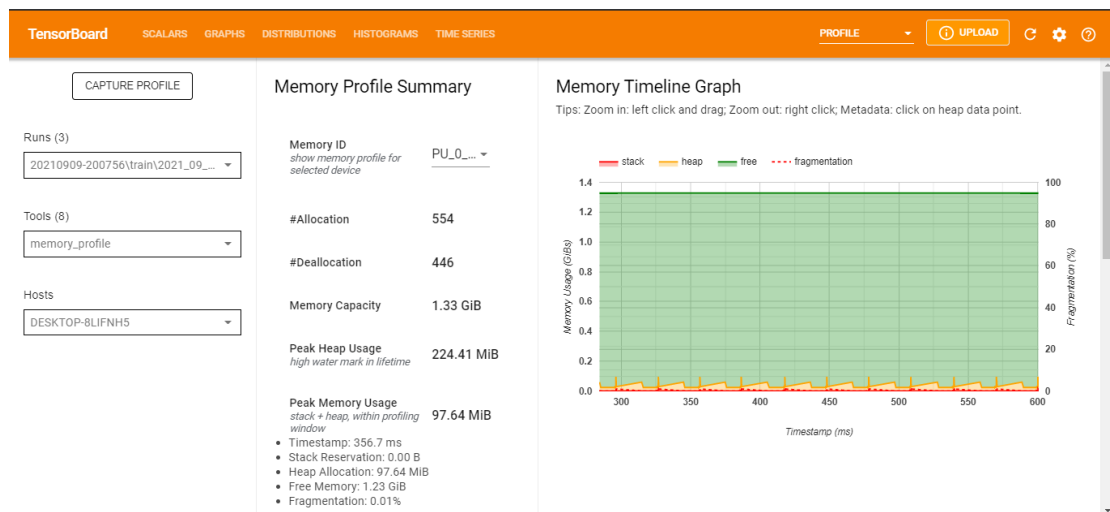
Gambar 4.8 Step-time Graph

Melihat grafik di atas, dapat dikatakan bahwa model yang sudah dibuat peneliti dikategorikan TIDAK *input-bound* karena hanya 1,3% dari total waktu

langkah sampel yang menunggu *input*. Ini adalah pertanda baik, sehingga peneliti bisa fokus untuk mengurangi waktu dari proses yg lain.

4.4.3 Memory Profile Tool

Memory profile tool adalah alat yang membuat profil penggunaan memori GPU. Alat profil memori dapat digunakan untuk melakukan *debug* masalah kehabisan memori (OOM) dengan menunjukkan penggunaan memori puncak dan alokasi memori yang sesuai untuk operasi TensorFlow, juga dapat men-*debug* masalah OOM yang mungkin muncul saat peneliti menjalankan *multi-tenancy inference* dan bisa melakukan debug masalah fragmentasi memori.



Gambar 4.9 Memory profile tool

Memory profile tool menampilkan data dalam beberapa bagian :

- Memory Profile Summary
- Memory Timeline Graph

Memory Profile Summary :

Memory Profile Summary menunjukkan ringkasan profil memori yang dilakukan aplikasi TensorFlow yang dibuat. Berikut adalah hasil pada *memory profile summary* :

Memory ID : PU_0_bfc

Tabel 4.5 Memory Profile Summary

No.	Memory profile tool	Nilai
1.	Allocation	554
2.	Deallocation	446
3.	Memory Capacity	1.33 GiB
4.	Peak Heap Usage	224.41 MiB
5.	Peak Memory Usage	97.64 MiB

Peak Memory Usage :

- a. Timestamp: 356.7 ms
- b. Stack Reservation: 0.00 B
- c. Heap Allocation: 97.64 MiB
- d. Free Memory: 1.23 GiB
- e. Fragmentation: 0.01%

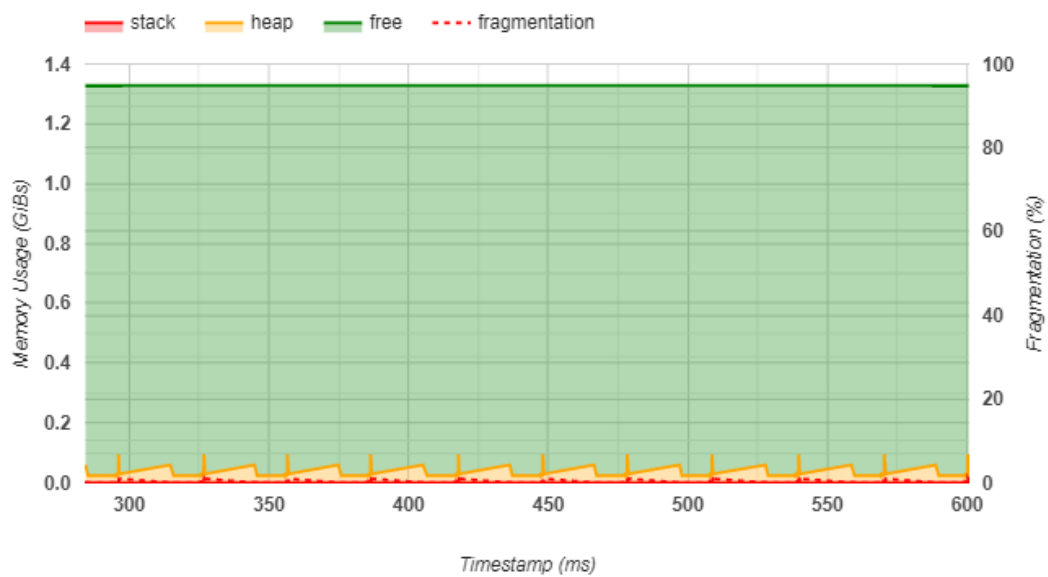
Berikut adalah penjelasan singkat ringkasan profil memori (memory profile summary) :

- a. Memory ID : Dropdown yang mencantumkan semua sistem memori perangkat yang tersedia.
- b. Allocation : Jumlah alokasi memori yang dibuat selama interval pembuatan profil.
- c. Deallocation : Jumlah dealokasi memori dalam interval pembuatan profil.
- d. Memory Capacity : Kapasitas total (dalam GiBs) dari sistem memori yang dipilih.
- e. Peak Heap Usage : Penggunaan memori puncak (dalam GiBs) sejak model mulai berjalan.
- f. Peak Memory Usage : Penggunaan memori puncak (dalam GiBs) dalam interval pembuatan profil. Bidang ini berisi sub-bidang berikut :
 1. Timestamp : Catatan digital ketika penggunaan memori puncak terjadi pada Timeline Grafik.
 2. Stack Reservation : Jumlah memori yang dicadangkan pada stack (dalam GiBs).

3. Heap Allocation : Jumlah memori yang dialokasikan pada heap (dalam GiBs).
4. Free Memory : Jumlah memori bebas (dalam GiBs). Kapasitas Memori adalah jumlah total dari Reservasi Stack, Alokasi Heap, dan Memori Bebas.
5. Fragmentation : Persentase fragmentasi (lebih rendah lebih baik). Ini dihitung sebagai persentase $(1 - \text{Ukuran potongan terbesar dari memori bebas} / \text{Total memori bebas})$.

Memory Timeline Graph

Bagian ini menampilkan plot penggunaan memori (dalam GiBs) dan persentase fragmentasi terhadap waktu (dalam ms). Berikut adalah Memory Timeline Graph :



Gambar 4.10 Memory Timeline Graph

BAB 5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan analisis hasil penelitian sistem penerjemah bahasa isyarat otomatis menggunakan metode *deep learning* model *convolutional neural network* diperoleh kesimpulan bahwa dalam implementasi framework MediaPipe citra RGB akan diproses untuk memprediksi dan menggambar objek / 21 landmark tangan manusia. Data hasil deteksi objek tangan manusia akan dimanfaatkan untuk membantu melakukan proses pengolahan dan normalisasi data citra digital. Terdapat subproses pada pengolahan citra digital yaitu *image preprocessing*, segmentasi gambar, import dataset kedalam sistem training, *image augmentation*, dan *one hot encode*. Pada data training hasil implementasi metode *deep learning* model *convolutional neural network* memiliki nilai *test score* sebesar 0.4% dan nilai *test accuracy* sebesar 99%. Analisis pada model data training yang sudah dibuat peneliti dikategorikan TIDAK *input-bound* karena hanya 1,3% dari total waktu langkah sampel yang menunggu *input*. Sehingga dapat dikatakan model sudah memiliki efisiensi yang baik.

5.2 Saran

Berdasarkan hasil yang diperoleh dari pelaksanaan skripsi maka saran yang dapat dilakukan guna mengembangkan skripsi ini antara lain :

1. Dalam proses klasifikasi yang dilakukan secara realtime masih terjadi penurunan fps yang cukup signifikan dari 30 fps menjadi 4-9 fps. Hal ini terjadi karena masih belum dilakukan optimasi model data training setelah analisis menggunakan Tensorflow Profiler.
2. Dataset citra yang dikumpulkan perlu adanya penambahan dataset yang berlabel negatif atau dataset yang nilai klasifikasinya bukan sebagai bahasa isyarat sehingga hasil output klasifikasi *realtime* lebih relevan dan fungsional.
3. Penerapan model *convolutional neural network* cukup baik jika ingin melakukan klasifikasi satu frame / per gambar tetapi tidak cocok jika melakukan proses klasifikasi pada data input berupa gerakan.

DAFTAR PUSTAKA

Arrofiqoh EN, Harintaka H. Implementasi Metode Convolutional Neural Network Untuk Klasifikasi Tanaman Pada Citra Resolusi Tinggi. *Geomatika*. 2018;24(2):61. doi:10.24895/jig.2018.24-2.810

Arroyo-Pérez DE, Alvarez-Canchila OI, Patño-Saucedo A, Rostro González H, Patño-Vanegas A. Automatic recognition of Colombian car license plates using convolutional neural networks and Chars74k database. *J Phys Conf Ser*. 2020;1547(1). doi:10.1088/1742-6596/1547/1/012024

Bagus M, Bakti S, Pranoto YM. Pengenalan Angka Sistem Isyarat Bahasa Indonesia Dengan Menggunakan Metode Convolutional Neural Network. *Semin Nas Inov Teknol*. Published online 2019:11-16.

Bhatnagar S, Agrawal S. Hand Gesture Recognition for Indian Sign Language: A Review. *Int J Comput Trends Technol*. 2015;21(3):121-122. doi:10.14445/22312803/ijctt-v21p122

Breva Yunanda A, Mandita F, Primasetya Armin A. Pengenalan Bahasa Isyarat Indonesia (BISINDO) Untuk Karakter Huruf Dengan Menggunakan Microsoft Kinect. *Fountain Informatics J*. 2018;3(2):41. doi:10.21111/fij.v3i2.2469

Eka Putra WS. Klasifikasi Citra Menggunakan Convolutional Neural Network (CNN) pada Caltech 101. *J Tek ITS*. 2016;5(1). doi:10.12962/j23373539.v5i1.15696

Fadillah RZ. Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network Model Penerjemah Bahasa Isyarat Indonesia (Bisindo) Menggunakan Convolutional Neural Network. *Fak Sains dan Ilmu Komputer, Progr Stud Ilmu Komputer, Univ Pertamina*. Published online 2020.

Gafar AA, Sari JY. Sistem Pengenalan Bahasa Isyarat Indonesia dengan Menggunakan Metode Fuzzy K-Nearest Neighbor. *J Ultim*. 2018;9(2):122-128. doi:10.31937/ti.v9i2.671

Gumelar G, Hafiar H, Subekti P. Bahasa Isyarat Indonesia Sebagai Budaya Tuli Melalui Pemaknaan Anggota Gerakan Untuk Kesejahteraan Tuna Rungu. *Informasi*. 2018;48(1):65.

Hu J, Kuang Y, Liao B, Cao L, Dong S, Li P. A Multichannel 2D Convolutional Neural Network Model for Task-Evoked fMRI Data Classification. *Comput Intell Neurosci*. 2019;2019(i). doi:10.1155/2019/5065214

Lecun Y, Bottou L, Bengio Y, Ha P. LeNet. *Proc IEEE*. 1998;(November):1-46.

Pinto RF, Borges CDB, Almeida AMA, Paula IC. Static Hand Gesture Recognition Based on Convolutional Neural Networks. *J Electr Comput Eng*. 2019;2019. doi:10.1155/2019/4167890

Rahim MA, Islam MR, Shin J. Non-touch sign word recognition based on dynamic hand gesture using hybrid segmentation and CNN feature fusion. *Appl Sci*. 2019;9(18). doi:10.3390/app9183790

Riska Budiana, Effendi Hasan. *Jurnal Ilmiah Mahasiswa FISIP Unsyiah* Volume 4, Nomor 3, Agustus 2019 www.jim.unsyiah.ac.id/FISIP. *J Ilm Mhs FISIP Unsyiah*. 2019;4:3. www.jim.unsyiah.ac.id/FISIP

Shield B. Hearing Loss - Numbers and Costs Evaluation of the social and economic costs of hearing impairment. A report for Hear-it. Brunel Univ London. 2018;(October):159.

Yohanes JA, Arjawa IGPBS, Punia IN. Bahasa Isyarat Indonesia Dalam Proses Interaksi Sosial Tuli dan “Masyarakat Dengar” di Kota Denpasar. *OJS Unud*. Published online 2013:1-15.

LAMPIRAN

Lampiran 1 Proses Data Training

Epoch 1/156

468/468 - 39s 38ms/step - loss: 2.2485 - accuracy: 0.3468 - val_loss: 0.1710 - val_accuracy: 0.9542

Epoch 2/156

468/468 - 16s 34ms/step - loss: 0.6028 - accuracy: 0.7977 - val_loss: 0.0737 - val_accuracy: 0.9829

Epoch 3/156

468/468 - 16s 34ms/step - loss: 0.3941 - accuracy: 0.8657 - val_loss: 0.0752 - val_accuracy: 0.9778

Epoch 4/156

468/468 - 16s 33ms/step - loss: 0.3190 - accuracy: 0.8913 - val_loss: 0.0427 - val_accuracy: 0.9839

Epoch 5/156

468/468 - 16s 34ms/step - loss: 0.2566 - accuracy: 0.9104 - val_loss: 0.0325 - val_accuracy: 0.9894

Epoch 6/156

468/468 - 16s 34ms/step - loss: 0.2073 - accuracy: 0.9314 - val_loss: 0.0268 - val_accuracy: 0.9892

Epoch 7/156

468/468 - 16s 34ms/step - loss: 0.1930 - accuracy: 0.9336 - val_loss: 0.0302 - val_accuracy: 0.9906

Epoch 8/156

468/468 - 17s 36ms/step - loss: 0.1580 - accuracy: 0.9458 - val_loss: 0.0228 - val_accuracy: 0.9920

Epoch 9/156

468/468 - 16s 35ms/step - loss: 0.1684 - accuracy: 0.9450 - val_loss: 0.0192 - val_accuracy: 0.9935

Epoch 10/156

468/468 - 16s 33ms/step - loss: 0.1486 - accuracy: 0.9489 - val_loss: 0.0136 - val_accuracy: 0.9947

Epoch 11/156

468/468 - 16s 34ms/step - loss: 0.1545 - accuracy: 0.9492 - val_loss: 0.0116 - val_accuracy: 0.9950

Epoch 12/156

468/468 - 16s 35ms/step - loss: 0.1206 - accuracy: 0.9602 - val_loss: 0.0114 - val_accuracy: 0.9959

Epoch 13/156

468/468 - 16s 34ms/step - loss: 0.1245 - accuracy: 0.9581 - val_loss: 0.0104 - val_accuracy: 0.9957

Epoch 14/156

468/468 - 16s 34ms/step - loss: 0.1136 - accuracy: 0.9632 - val_loss: 0.0113 - val_accuracy: 0.9961

Epoch 15/156

468/468 - 16s 35ms/step - loss: 0.1128 - accuracy: 0.9653 - val_loss: 0.0067 - val_accuracy: 0.9983

Epoch 16/156

468/468 - 16s 35ms/step - loss: 0.1032 - accuracy: 0.9653 - val_loss: 0.0056 - val_accuracy: 0.9980

Epoch 17/156

468/468 - 16s 35ms/step - loss: 0.1063 - accuracy: 0.9667 - val_loss: 0.0077 - val_accuracy: 0.9985

Epoch 18/156

468/468 - 16s 35ms/step - loss: 0.0841 - accuracy: 0.9718 - val_loss: 0.0083 - val_accuracy: 0.9969

Epoch 19/156

468/468 - 16s 35ms/step - loss: 0.0841 - accuracy: 0.9707 - val_loss: 0.0077 - val_accuracy: 0.9971

Epoch 20/156

468/468 - 16s 35ms/step - loss: 0.0856 - accuracy: 0.9728 - val_loss: 0.0070 - val_accuracy: 0.9974

Epoch 21/156

468/468 - 16s 35ms/step - loss: 0.0717 - accuracy: 0.9762 - val_loss: 0.0068 - val_accuracy: 0.9978

Epoch 22/156

468/468 - 16s 34ms/step - loss: 0.0739 - accuracy: 0.9770 - val_loss: 0.0039 - val_accuracy: 0.9993

Epoch 23/156

468/468 - 16s 35ms/step - loss: 0.0812 - accuracy: 0.9720 - val_loss: 0.0048 - val_accuracy: 0.9980

Epoch 24/156

468/468 - 17s 36ms/step - loss: 0.0585 - accuracy: 0.9797 - val_loss: 0.0068 - val_accuracy: 0.9976

Epoch 25/156

468/468 - 16s 35ms/step - loss: 0.0729 - accuracy: 0.9770 - val_loss: 0.0055 - val_accuracy: 0.9983

Epoch 26/156

468/468 - 16s 35ms/step - loss: 0.0604 - accuracy: 0.9790 - val_loss: 0.0065 - val_accuracy: 0.9980

Epoch 27/156

468/468 - 16s 35ms/step - loss: 0.0640 - accuracy: 0.9797 - val_loss: 0.0057 - val_accuracy: 0.9983

Epoch 28/156

468/468 - 16s 35ms/step - loss: 0.0713 - accuracy: 0.9769 - val_loss: 0.0038 - val_accuracy: 0.9995

Epoch 29/156

468/468 - 16s 35ms/step - loss: 0.0597 - accuracy: 0.9800 - val_loss: 0.0065 - val_accuracy: 0.9978

Epoch 30/156

468/468 - 16s 35ms/step - loss: 0.0637 - accuracy: 0.9779 - val_loss: 0.0035 - val_accuracy: 0.9991

Epoch 31/156

468/468 - 16s 35ms/step - loss: 0.0659 - accuracy: 0.9790 - val_loss: 0.0032 - val_accuracy: 0.9990

Epoch 32/156

468/468 - 16s 35ms/step - loss: 0.0644 - accuracy: 0.9784 - val_loss: 0.0041 - val_accuracy: 0.9988

Epoch 33/156

468/468 - 16s 35ms/step - loss: 0.0586 - accuracy: 0.9812 - val_loss: 0.0053 - val_accuracy: 0.9988

Epoch 34/156

468/468 - 16s 35ms/step - loss: 0.0676 - accuracy: 0.9794 - val_loss: 0.0083 - val_accuracy: 0.9973

Epoch 35/156

468/468 - 16s 35ms/step - loss: 0.0593 - accuracy: 0.9801 - val_loss: 0.0037 - val_accuracy: 0.9986

Epoch 36/156

468/468 - 16s 35ms/step - loss: 0.0543 - accuracy: 0.9832 - val_loss: 0.0032 - val_accuracy: 0.9988

Epoch 37/156

468/468 - 16s 35ms/step - loss: 0.0502 - accuracy: 0.9838 - val_loss: 0.0056 - val_accuracy: 0.9986

Epoch 38/156

468/468 - 16s 35ms/step - loss: 0.0574 - accuracy: 0.9827 - val_loss: 0.0032 - val_accuracy: 0.9990

Epoch 39/156

468/468 - 16s 35ms/step - loss: 0.0523 - accuracy: 0.9843 - val_loss: 0.0026 - val_accuracy: 0.9988

Epoch 40/156

468/468 - 16s 35ms/step - loss: 0.0488 - accuracy: 0.9856 - val_loss: 0.0033 - val_accuracy: 0.9988

Epoch 41/156

468/468 - 16s 34ms/step - loss: 0.0568 - accuracy: 0.9816 - val_loss: 0.0044 - val_accuracy: 0.9986

Epoch 42/156

468/468 - 16s 35ms/step - loss: 0.0510 - accuracy: 0.9832 - val_loss: 0.0037 - val_accuracy: 0.9985

Epoch 43/156

468/468 - 16s 35ms/step - loss: 0.0478 - accuracy: 0.9853 - val_loss: 0.0047 - val_accuracy: 0.9981

Epoch 44/156

468/468 - 16s 35ms/step - loss: 0.0524 - accuracy: 0.9841 - val_loss: 0.0034 - val_accuracy: 0.9990

Epoch 45/156

468/468 - 16s 35ms/step - loss: 0.0434 - accuracy: 0.9857 - val_loss: 0.0060 - val_accuracy: 0.9983

Epoch 46/156

468/468 - 16s 35ms/step - loss: 0.0529 - accuracy: 0.9831 - val_loss: 0.0073 - val_accuracy: 0.9988

Epoch 47/156

468/468 - 16s 35ms/step - loss: 0.0392 - accuracy: 0.9866 - val_loss: 0.0034 - val_accuracy: 0.9988

Epoch 48/156

468/468 - 17s 35ms/step - loss: 0.0611 - accuracy: 0.9821 - val_loss: 0.0032 - val_accuracy: 0.9990

Epoch 49/156

468/468 - 16s 35ms/step - loss: 0.0452 - accuracy: 0.9858 - val_loss: 0.0028 - val_accuracy: 0.9990

Epoch 50/156

468/468 - 16s 35ms/step - loss: 0.0510 - accuracy: 0.9833 - val_loss: 0.0021 - val_accuracy: 0.9993

Epoch 51/156

468/468 - 16s 35ms/step - loss: 0.0465 - accuracy: 0.9865 - val_loss: 0.0044 - val_accuracy: 0.9988

Epoch 52/156

468/468 - 16s 34ms/step - loss: 0.0423 - accuracy: 0.9878 - val_loss: 0.0039 - val_accuracy: 0.9986

Epoch 53/156

468/468 - 17s 36ms/step - loss: 0.0448 - accuracy: 0.9870 - val_loss: 0.0028 - val_accuracy: 0.9991

Epoch 54/156

468/468 - 16s 35ms/step - loss: 0.0453 - accuracy: 0.9867 - val_loss: 0.0042 - val_accuracy: 0.9991

Epoch 55/156

468/468 - 16s 35ms/step - loss: 0.0402 - accuracy: 0.9867 - val_loss: 0.0037 - val_accuracy: 0.9990

Epoch 56/156

468/468 - 16s 35ms/step - loss: 0.0465 - accuracy: 0.9862 - val_loss: 0.0017 - val_accuracy: 0.9995

Epoch 57/156

468/468 - 16s 35ms/step - loss: 0.0469 - accuracy: 0.9856 - val_loss: 0.0037 - val_accuracy: 0.9988

Epoch 58/156

468/468 - 16s 35ms/step - loss: 0.0496 - accuracy: 0.9852 - val_loss: 0.0031 - val_accuracy: 0.9991

Epoch 59/156

468/468 - 16s 35ms/step - loss: 0.0463 - accuracy: 0.9861 - val_loss: 0.0020 - val_accuracy: 0.9993

Epoch 60/156

468/468 - 16s 35ms/step - loss: 0.0385 - accuracy: 0.9868 - val_loss: 0.0043 - val_accuracy: 0.9986

Epoch 61/156

468/468 - 16s 35ms/step - loss: 0.0451 - accuracy: 0.9849 - val_loss: 0.0047 - val_accuracy: 0.9990

Epoch 62/156

468/468 - 16s 35ms/step - loss: 0.0458 - accuracy: 0.9862 - val_loss: 0.0028 - val_accuracy: 0.9988

Epoch 63/156

468/468 - 16s 35ms/step - loss: 0.0418 - accuracy: 0.9874 - val_loss: 0.0048 - val_accuracy: 0.9986

Epoch 64/156

468/468 - 16s 35ms/step - loss: 0.0430 - accuracy: 0.9855 - val_loss: 0.0054 - val_accuracy: 0.9986

Epoch 65/156

468/468 - 16s 35ms/step - loss: 0.0464 - accuracy: 0.9856 - val_loss: 0.0029 - val_accuracy: 0.9993

Epoch 66/156

468/468 - 16s 35ms/step - loss: 0.0350 - accuracy: 0.9893 - val_loss: 0.0037 - val_accuracy: 0.9986

Epoch 67/156

468/468 - 16s 35ms/step - loss: 0.0497 - accuracy: 0.9857 - val_loss: 0.0048 - val_accuracy: 0.9988

Epoch 68/156

468/468 - 16s 35ms/step - loss: 0.0390 - accuracy: 0.9884 - val_loss: 0.0054 - val_accuracy: 0.9990

Epoch 69/156

468/468 - 16s 35ms/step - loss: 0.0388 - accuracy: 0.9866 - val_loss: 0.0052 - val_accuracy: 0.9985

Epoch 70/156

468/468 - 16s 35ms/step - loss: 0.0380 - accuracy: 0.9879 - val_loss: 0.0055 - val_accuracy: 0.9990

Epoch 71/156

468/468 - 16s 34ms/step - loss: 0.0378 - accuracy: 0.9885 - val_loss: 0.0040 - val_accuracy: 0.9983

Epoch 72/156

468/468 - 16s 35ms/step - loss: 0.0456 - accuracy: 0.9862 - val_loss: 0.0068 - val_accuracy: 0.9981

Epoch 73/156

468/468 - 16s 35ms/step - loss: 0.0463 - accuracy: 0.9853 - val_loss: 0.0027 - val_accuracy: 0.9990

Epoch 74/156

468/468 - 16s 35ms/step - loss: 0.0456 - accuracy: 0.9878 - val_loss: 0.0070 - val_accuracy: 0.9978

Epoch 75/156

468/468 - 16s 35ms/step - loss: 0.0375 - accuracy: 0.9905 - val_loss: 0.0030 - val_accuracy: 0.9995

Epoch 76/156

468/468 - 16s 35ms/step - loss: 0.0376 - accuracy: 0.9889 - val_loss: 0.0024 - val_accuracy: 0.9991

Epoch 77/156

468/468 - 16s 35ms/step - loss: 0.0359 - accuracy: 0.9897 - val_loss: 0.0022 - val_accuracy: 0.9988

Epoch 78/156

468/468 - 16s 35ms/step - loss: 0.0435 - accuracy: 0.9884 - val_loss: 0.0023 - val_accuracy: 0.9991

Epoch 79/156

468/468 - 16s 35ms/step - loss: 0.0405 - accuracy: 0.9881 - val_loss: 0.0015 - val_accuracy: 0.9993

Epoch 80/156

468/468 - 16s 34ms/step - loss: 0.0467 - accuracy: 0.9851 - val_loss: 0.0042 - val_accuracy: 0.9993

Epoch 81/156

468/468 - 16s 35ms/step - loss: 0.0476 - accuracy: 0.9861 - val_loss: 0.0025 - val_accuracy: 0.9993

Epoch 82/156

468/468 - 16s 35ms/step - loss: 0.0352 - accuracy: 0.9906 - val_loss: 0.0058 - val_accuracy: 0.9986

Epoch 83/156

468/468 - 16s 35ms/step - loss: 0.0447 - accuracy: 0.9879 - val_loss: 0.0062 - val_accuracy: 0.9988

Epoch 84/156

468/468 - 16s 35ms/step - loss: 0.0409 - accuracy: 0.9890 - val_loss: 0.0030 - val_accuracy: 0.9993

Epoch 85/156

468/468 - 16s 35ms/step - loss: 0.0385 - accuracy: 0.9886 - val_loss: 0.0038 - val_accuracy: 0.9993

Epoch 86/156

468/468 - 16s 35ms/step - loss: 0.0484 - accuracy: 0.9869 - val_loss: 0.0072 - val_accuracy: 0.9983

Epoch 87/156

468/468 - 16s 35ms/step - loss: 0.0427 - accuracy: 0.9870 - val_loss: 0.0021 - val_accuracy: 0.9995

Epoch 88/156

468/468 - 16s 35ms/step - loss: 0.0449 - accuracy: 0.9864 - val_loss: 0.0020 - val_accuracy: 0.9993

Epoch 89/156

468/468 - 16s 35ms/step - loss: 0.0397 - accuracy: 0.9875 - val_loss: 0.0018 - val_accuracy: 0.9988

Epoch 90/156

468/468 - 16s 35ms/step - loss: 0.0392 - accuracy: 0.9881 - val_loss: 0.0046 - val_accuracy: 0.9988

Epoch 91/156

468/468 - 16s 35ms/step - loss: 0.0362 - accuracy: 0.9889 - val_loss: 0.0032 - val_accuracy: 0.9988

Epoch 92/156

468/468 - 16s 35ms/step - loss: 0.0424 - accuracy: 0.9871 - val_loss: 0.0027 - val_accuracy: 0.9991

Epoch 93/156

468/468 - 16s 35ms/step - loss: 0.0434 - accuracy: 0.9856 - val_loss: 0.0067 - val_accuracy: 0.9980

Epoch 94/156

468/468 - 16s 35ms/step - loss: 0.0397 - accuracy: 0.9891 - val_loss: 0.0010 - val_accuracy: 0.9997

Epoch 95/156

468/468 - 16s 35ms/step - loss: 0.0415 - accuracy: 0.9871 - val_loss: 0.0025 - val_accuracy: 0.9993

Epoch 96/156

468/468 - 16s 35ms/step - loss: 0.0395 - accuracy: 0.9896 - val_loss: 0.0027 - val_accuracy: 0.9988

Epoch 97/156

468/468 - 16s 35ms/step - loss: 0.0544 - accuracy: 0.9851 - val_loss: 0.0046 - val_accuracy: 0.9986

Epoch 98/156

468/468 - 16s 35ms/step - loss: 0.0411 - accuracy: 0.9889 - val_loss: 0.0018 - val_accuracy: 0.9993

Epoch 99/156

468/468 - 16s 35ms/step - loss: 0.0349 - accuracy: 0.9896 - val_loss: 0.0027 - val_accuracy: 0.9997

Epoch 100/156

468/468 - 16s 35ms/step - loss: 0.0448 - accuracy: 0.9878 - val_loss: 0.0054 - val_accuracy: 0.9985

Epoch 101/156

468/468 - 16s 35ms/step - loss: 0.0436 - accuracy: 0.9872 - val_loss: 0.0016 - val_accuracy: 0.9995

Epoch 102/156

468/468 - 16s 35ms/step - loss: 0.0363 - accuracy: 0.9890 - val_loss: 0.0048 - val_accuracy: 0.9988

Epoch 103/156

468/468 - 16s 35ms/step - loss: 0.0306 - accuracy: 0.9908 - val_loss: 0.0021 - val_accuracy: 0.9995

Epoch 104/156

468/468 - 16s 35ms/step - loss: 0.0421 - accuracy: 0.9897 - val_loss: 0.0054 - val_accuracy: 0.9990

Epoch 105/156

468/468 - 16s 35ms/step - loss: 0.0326 - accuracy: 0.9898 - val_loss: 0.0022 - val_accuracy: 0.9990

Epoch 106/156

468/468 - 16s 35ms/step - loss: 0.0357 - accuracy: 0.9887 - val_loss: 0.0037 - val_accuracy: 0.9995

Epoch 107/156

468/468 - 16s 35ms/step - loss: 0.0384 - accuracy: 0.9886 - val_loss: 0.0031 - val_accuracy: 0.9993

Epoch 108/156

468/468 - 16s 35ms/step - loss: 0.0324 - accuracy: 0.9902 - val_loss: 0.0029 - val_accuracy: 0.9990

Epoch 109/156

468/468 - 16s 35ms/step - loss: 0.0394 - accuracy: 0.9890 - val_loss: 0.0026 - val_accuracy: 0.9985

Epoch 110/156

468/468 - 16s 35ms/step - loss: 0.0296 - accuracy: 0.9911 - val_loss: 0.0054 - val_accuracy: 0.9986

Epoch 111/156

468/468 - 17s 35ms/step - loss: 0.0379 - accuracy: 0.9897 - val_loss: 0.0022 - val_accuracy: 0.9991

Epoch 112/156

468/468 - 16s 35ms/step - loss: 0.0399 - accuracy: 0.9900 - val_loss: 0.0091 - val_accuracy: 0.9983

Epoch 113/156

468/468 - 16s 35ms/step - loss: 0.0408 - accuracy: 0.9882 - val_loss: 0.0056 - val_accuracy: 0.9985

Epoch 114/156

468/468 - 16s 35ms/step - loss: 0.0379 - accuracy: 0.9900 - val_loss: 0.0040 - val_accuracy: 0.9988

Epoch 115/156

468/468 - 16s 35ms/step - loss: 0.0313 - accuracy: 0.9910 - val_loss: 0.0015 - val_accuracy: 0.9997

Epoch 116/156

468/468 - 16s 35ms/step - loss: 0.0388 - accuracy: 0.9876 - val_loss: 0.0160 - val_accuracy: 0.9971

Epoch 117/156

468/468 - 16s 35ms/step - loss: 0.0374 - accuracy: 0.9897 - val_loss: 0.0046 - val_accuracy: 0.9986

Epoch 118/156

468/468 - 16s 35ms/step - loss: 0.0351 - accuracy: 0.9892 - val_loss: 0.0073 - val_accuracy: 0.9978

Epoch 119/156

468/468 - 16s 35ms/step - loss: 0.0435 - accuracy: 0.9877 - val_loss: 0.0053 - val_accuracy: 0.9986

Epoch 120/156

468/468 - 16s 35ms/step - loss: 0.0426 - accuracy: 0.9879 - val_loss: 0.0042 - val_accuracy: 0.9983

Epoch 121/156

468/468 - 16s 35ms/step - loss: 0.0372 - accuracy: 0.9889 - val_loss: 0.0029 - val_accuracy: 0.9993

Epoch 122/156

468/468 - 16s 35ms/step - loss: 0.0365 - accuracy: 0.9898 - val_loss: 0.0027 - val_accuracy: 0.9988

Epoch 123/156

468/468 - 16s 35ms/step - loss: 0.0398 - accuracy: 0.9904 - val_loss: 0.0025 - val_accuracy: 0.9986

Epoch 124/156

468/468 - 16s 35ms/step - loss: 0.0357 - accuracy: 0.9888 - val_loss: 0.0020 - val_accuracy: 0.9991

Epoch 125/156

468/468 - 16s 35ms/step - loss: 0.0335 - accuracy: 0.9903 - val_loss: 0.0036 - val_accuracy: 0.9988

Epoch 126/156

468/468 - 18s 38ms/step - loss: 0.0360 - accuracy: 0.9896 - val_loss: 0.0048 - val_accuracy: 0.9988

Epoch 127/156

468/468 - 16s 35ms/step - loss: 0.0344 - accuracy: 0.9902 - val_loss: 0.0021 - val_accuracy: 0.9993

Epoch 128/156

468/468 - 16s 35ms/step - loss: 0.0314 - accuracy: 0.9900 - val_loss: 0.0030 - val_accuracy: 0.9990

Epoch 129/156

468/468 - 16s 35ms/step - loss: 0.0391 - accuracy: 0.9892 - val_loss: 0.0055 - val_accuracy: 0.9985

Epoch 130/156

468/468 - 16s 34ms/step - loss: 0.0327 - accuracy: 0.9904 - val_loss: 0.0016 - val_accuracy: 0.9991

Epoch 131/156

468/468 - 16s 35ms/step - loss: 0.0426 - accuracy: 0.9887 - val_loss: 0.0022 - val_accuracy: 0.9993

Epoch 132/156

468/468 - 16s 35ms/step - loss: 0.0406 - accuracy: 0.9890 - val_loss: 0.0019 - val_accuracy: 0.9993

Epoch 133/156

468/468 - 16s 34ms/step - loss: 0.0301 - accuracy: 0.9900 - val_loss: 9.4606e-04 - val_accuracy: 0.9997

Epoch 134/156

468/468 - 16s 35ms/step - loss: 0.0408 - accuracy: 0.9887 - val_loss: 0.0013 - val_accuracy: 0.9997

Epoch 135/156

468/468 - 16s 35ms/step - loss: 0.0400 - accuracy: 0.9899 - val_loss: 0.0016 - val_accuracy: 0.9997

Epoch 136/156

468/468 - 16s 35ms/step - loss: 0.0359 - accuracy: 0.9895 - val_loss: 0.0045 - val_accuracy: 0.9986

Epoch 137/156

468/468 - 16s 34ms/step - loss: 0.0466 - accuracy: 0.9876 - val_loss: 0.0027 - val_accuracy: 0.9988

Epoch 138/156

468/468 - 16s 35ms/step - loss: 0.0365 - accuracy: 0.9895 - val_loss: 0.0023 - val_accuracy: 0.9988

Epoch 139/156

468/468 - 16s 35ms/step - loss: 0.0365 - accuracy: 0.9898 - val_loss: 0.0033 - val_accuracy: 0.9993

Epoch 140/156

468/468 - 16s 35ms/step - loss: 0.0397 - accuracy: 0.9893 - val_loss: 0.0053 - val_accuracy: 0.9990

Epoch 141/156

468/468 - 16s 34ms/step - loss: 0.0367 - accuracy: 0.9896 - val_loss: 7.9632e-04 - val_accuracy: 0.9995

Epoch 142/156

468/468 - 16s 35ms/step - loss: 0.0427 - accuracy: 0.9893 - val_loss: 0.0014 - val_accuracy: 0.9995

Epoch 143/156

468/468 - 16s 35ms/step - loss: 0.0351 - accuracy: 0.9910 - val_loss: 0.0020 - val_accuracy: 0.9991

Epoch 144/156

468/468 - 16s 34ms/step - loss: 0.0311 - accuracy: 0.9920 - val_loss: 0.0102 - val_accuracy: 0.9978

Epoch 145/156

468/468 - 16s 35ms/step - loss: 0.0339 - accuracy: 0.9902 - val_loss: 0.0023 - val_accuracy: 0.9993

Epoch 146/156

468/468 - 16s 34ms/step - loss: 0.0359 - accuracy: 0.9902 - val_loss: 0.0031 - val_accuracy: 0.9991

Epoch 147/156

468/468 - 16s 35ms/step - loss: 0.0361 - accuracy: 0.9897 - val_loss: 0.0011 - val_accuracy: 0.9995

Epoch 148/156

468/468 - 16s 35ms/step - loss: 0.0268 - accuracy: 0.9915 - val_loss: 0.0017 - val_accuracy: 0.9995

Epoch 149/156

468/468 - 16s 34ms/step - loss: 0.0373 - accuracy: 0.9893 - val_loss: 9.4123e-04 - val_accuracy: 0.9995

Epoch 150/156

468/468 - 16s 34ms/step - loss: 0.0384 - accuracy: 0.9894 - val_loss: 0.0036 - val_accuracy: 0.9995

Epoch 151/156

468/468 - 16s 34ms/step - loss: 0.0382 - accuracy: 0.9897 - val_loss: 0.0036 - val_accuracy: 0.9993

Epoch 152/156

468/468 - 16s 35ms/step - loss: 0.0418 - accuracy: 0.9886 - val_loss: 0.0018 - val_accuracy: 0.9993

Epoch 153/156

468/468 - 16s 35ms/step - loss: 0.0520 - accuracy: 0.9876 - val_loss: 0.0030 - val_accuracy: 0.9988

Epoch 154/156

468/468 - 16s 34ms/step - loss: 0.0367 - accuracy: 0.9901 - val_loss: 0.0059 -
val_accuracy: 0.9990

Epoch 155/156

468/468 - 16s 34ms/step - loss: 0.0393 - accuracy: 0.9899 - val_loss: 0.0083 -
val_accuracy: 0.9976

Epoch 156/156

468/468 - 16s 35ms/step - loss: 0.0357 - accuracy: 0.9902 - val_loss: 0.0014 -
val_accuracy: 0.9993

Lampiran 2 Top 10 TensorFlow operations on GPU

No	Time (%)	Cumulative time (%)	Category	Operation	TensorCore eligibility	Op is using TensorCore
1	21.4%	21.4%	Conv2DBackpropFilter	gradient_tape/sequential/conv2d_1/Conv2D/Conv2DBackpropFilter	✓	✗
2	18.6%	40%	_FusedConv2D	sequential/conv2d_1/Relu	✗	✗
3	18.6%	58.6%	Conv2DBackpropInput	gradient_tape/sequential/conv2d_1/Conv2D/Conv2DBackpropInput	✓	✗
4	3.5%	62.1%	Conv2DBackpropFilter	gradient_tape/sequential/conv2d/Conv2D/Conv2DBackpropFilter	✓	✗
5	3.5%	65.6%	ReluGrad	gradient_tape/sequential/conv2d/ReluGrad	✗	✗
6	3.3%	68.8%	Conv2D	sequential/conv2d/Conv2D	✓	✗
7	2.8%	71.7%	BiasAdd	sequential/conv2d/BiasAdd	✗	✗
8	2.7%	74.4%	Conv2DBackpropInput	gradient_tape/sequential/conv2d_2/Conv2D/Conv2DBackpropInput	✓	✗
9	2.7%	77%	Conv2DBackpropFilter	gradient_tape/sequential/conv2d_2/Conv2D/Conv2DBackpropFilter	✓	✗
10	2.7%	79.7%	_FusedConv2D	sequential/conv2d_2/Relu	✗	✗