

BAB II

KAJIAN LITERATUR

2.1 Bahasa Isyarat

Bahasa isyarat merupakan sistem bahasa yang menggunakan modalitas visual-gestural untuk menyampaikan makna, berbeda dengan bahasa lisan yang menggunakan modalitas audio-vokal [1]. Bahasa isyarat memiliki struktur linguistik yang lengkap, mencakup fonologi, morfologi, sintaksis, dan semantik, sehingga diakui sebagai bahasa alami (*natural language*) yang setara dengan bahasa lisan [2]. Di berbagai negara, komunitas Tuli mengembangkan bahasa isyarat masing-masing yang dipengaruhi oleh budaya dan konteks sosial setempat



Gambar 2.1 Bahasa Isyarat [2]

Di Indonesia, terdapat dua sistem bahasa isyarat utama yang digunakan, yaitu Sistem Isyarat Bahasa Indonesia (SIBI) dan Bahasa Isyarat Indonesia (BISINDO). Keduanya memiliki perbedaan mendasar dalam hal asal-usul, struktur, dan penggunaan.

2.1.1 Sistem Isyarat Bahasa Indonesia (SIBI)

SIBI merupakan sistem bahasa isyarat yang dikembangkan oleh pemerintah Indonesia dan diresmikan melalui Kamus Sistem Isyarat Bahasa Indonesia yang diterbitkan oleh Departemen Pendidikan dan Kebudayaan pada tahun 1994 [4]. SIBI dikembangkan berdasarkan adaptasi dari *Signed Exact English* (SEE-II) dan mengikuti struktur tata bahasa Indonesia lisan secara ketat [5]. Dalam SIBI, setiap kata dalam bahasa Indonesia memiliki padanan isyarat yang baku, termasuk penggunaan imbuhan seperti awalan, akhiran, dan sisipan yang diisyaratkan secara terpisah [4]. SIBI umumnya diajarkan di Sekolah Luar Biasa (SLB) sebagai bahasa pengantar dalam pendidikan formal bagi siswa Tuli [6].

2.1.2 Bahasa Isyarat Indonesia (BISINDO)

BISINDO merupakan bahasa isyarat yang berkembang secara alami di kalangan komunitas Tuli Indonesia [7]. BISINDO tidak mengikuti struktur tata bahasa Indonesia lisan, melainkan memiliki tata bahasa tersendiri yang terbentuk melalui interaksi alamiah antar-anggota komunitas Tuli [3]. BISINDO digunakan secara luas dalam kehidupan sehari-hari oleh komunitas Tuli di berbagai daerah di Indonesia dan diakui oleh Gerakan untuk Kesejahteraan Tunarungu Indonesia (GERKATIN) sebagai bahasa utama komunitas Tuli Indonesia [8]. Komunitas Tuli di Indonesia diperkirakan mencakup sekitar 3–4% dari populasi Tuli global, namun pemahaman masyarakat umum terhadap BISINDO masih sangat terbatas, sehingga menimbulkan hambatan komunikasi antara komunitas Tuli dan masyarakat dengar [12], [13]. Kondisi ini mendorong perlunya pengembangan teknologi penerjemah BISINDO yang dapat menjembatani kesenjangan komunikasi tersebut.

Sebagai bahasa *visual-gestural*, BISINDO memanfaatkan kombinasi gerakan tangan, ekspresi wajah, posisi tubuh, dan orientasi gerakan untuk menyampaikan makna [9]. Secara linguistik, BISINDO memiliki beberapa karakteristik utama yang membedakannya dari bahasa isyarat lainnya [10]:

1. Gestur dinamis: BISINDO menggunakan gerakan tangan yang dinamis dengan variasi bentuk, arah, dan kecepatan gerakan untuk membedakan makna antar-kosakata [9].
2. Peran ekspresi wajah: Ekspresi wajah dalam BISINDO tidak hanya berfungsi sebagai penanda emosi, tetapi juga sebagai komponen gramatikal yang dapat mengubah makna suatu isyarat [10].
3. Penggunaan ruang isyarat: BISINDO memanfaatkan ruang tiga dimensi di depan tubuh penandatangan (*signing space*) untuk menunjukkan referensi, lokasi, dan hubungan antar-entitas dalam kalimat [3].
4. Variasi regional: BISINDO memiliki variasi dialek antar-daerah di Indonesia, meskipun terdapat pemahaman bersama yang memungkinkan komunikasi lintas-daerah [8].
5. *Simultaneitas*: Berbeda dengan bahasa lisan yang bersifat linear, BISINDO dapat menyampaikan beberapa informasi secara bersamaan melalui penggunaan dua tangan, ekspresi wajah, dan gerakan tubuh secara simultan [10].

Perbedaan mendasar antara SIBI dan BISINDO dapat dilihat dari beberapa aspek berikut:

Tabel 2.1 Perbandingan SIBI dan BISINDO [3], [4], [7]

No	Aspek	SIBI	BISINDO
1.	Asal-usul	Dibuat secara formal oleh pemerintah	Berkembang secara alami di komunitas Tuli
2.	Struktur tata bahasa	Mengikuti tata bahasa Indonesia lisan	Memiliki tata bahasa tersendiri
3.	Penggunaan imbuhan	Mengisyaratkan imbuhan secara terpisah	Tidak menggunakan imbuhan terpisah
4.	Konteks penggunaan	Pendidikan formal (SLB)	Komunikasi sehari-hari komunitas Tuli
5.	Sifat gerakan	Cenderung kaku dan sistematis	Lebih ekspresif dan dinamis
6.	Variasi regional	Seragam (dibakukan)	Memiliki variasi antar-daerah

Meskipun SIBI telah dibakukan dan diajarkan secara formal, pada praktiknya komunitas Tuli di Indonesia lebih banyak menggunakan BISINDO sebagai bahasa komunikasi utama [7]. Hal ini disebabkan oleh sifat BISINDO yang lebih *natural*, ekspresif, dan sesuai dengan kebutuhan komunikasi sehari-hari komunitas Tuli [8]. Oleh karena itu, penelitian ini berfokus pada pengenalan BISINDO sebagai bahasa isyarat yang secara riil digunakan oleh komunitas Tuli Indonesia.

Salah satu tantangan utama dalam pengenalan BISINDO secara otomatis adalah adanya kemiripan gestur pada beberapa kosakata yang memiliki makna berbeda [9]. Kemiripan ini umumnya terjadi pada gestur yang memiliki bentuk tangan (*handshape*) dan posisi yang serupa, namun hanya dibedakan oleh arah gerakan (*movement direction*), dinamika gerakan (*movement dynamics*), atau konteks penggunaan [11]. Sebagai contoh, beberapa isyarat dalam BISINDO menggunakan konfigurasi tangan yang sama tetapi dibedakan oleh arah gerakan ke atas, ke bawah, ke kiri, atau ke kanan. Perbedaan yang halus ini menyebabkan kesulitan dalam mengidentifikasi dan membedakan gestur secara akurat, terutama pada sistem berbasis *computer vision* [9]. Dalam penelitian ini, dataset yang digunakan adalah WL-BISINDO, yaitu dataset video BISINDO yang terdiri dari 1.600 video RGB dengan 32 gestur terisolasi yang dilakukan oleh 5 penutur dari Banten [14], guna merepresentasikan variasi gerakan dalam penggunaan bahasa isyarat.

2.2 Deteksi Objek

Deteksi objek (*object detection*) merupakan salah satu tugas fundamental dalam bidang *computer vision* yang bertujuan untuk mengidentifikasi dan melokalisasi objek tertentu dalam sebuah citra atau video [15]. Berbeda dengan klasifikasi gambar (*image classification*) yang hanya menentukan kategori objek secara keseluruhan, deteksi objek menghasilkan dua keluaran sekaligus, yaitu label kelas objek dan posisi objek dalam gambar yang dinyatakan dalam bentuk *bounding box* [16]. Dengan demikian, deteksi objek mencakup dua tugas utama yang dilakukan secara bersamaan, yaitu lokalisasi objek (*object localization*) dan klasifikasi objek (*object classification*) [15].

Proses deteksi objek secara umum bekerja dengan cara memindai seluruh area gambar untuk menemukan keberadaan objek, kemudian menentukan koordinat kotak pembatas (*bounding box*) yang melingkupi objek tersebut, sekaligus memprediksi kelas objek beserta nilai kepercayaan (*confidence score*) prediksinya [16]. Keluaran dari proses ini biasanya berupa sekumpulan *bounding box* dengan label kelas dan skor kepercayaan yang menunjukkan seberapa yakin model terhadap keberadaan objek di lokasi tersebut [17].

Secara historis, pendekatan deteksi objek dapat diklasifikasikan ke dalam dua kategori utama [17]:

1. Metode berbasis dua tahap (*two-stage detectors*): Pendekatan ini memisahkan proses menjadi dua tahap, yaitu tahap proposal region yang mengidentifikasi kandidat area yang mungkin mengandung objek, diikuti oleh tahap klasifikasi dan regresi *bounding box*. Pada tahap pertama, algoritma seperti *Selective Search* atau *Region Proposal Network* (RPN) menghasilkan sejumlah kandidat wilayah (*region proposals*) yang berpotensi mengandung objek. Kandidat wilayah ini kemudian diproses pada tahap kedua untuk diklasifikasikan dan disempurnakan posisinya. Contoh metode ini meliputi R-CNN, Fast R-CNN, dan Faster R-CNN [17]. Metode dua tahap umumnya memberikan akurasi yang tinggi, namun memiliki kecepatan inferensi yang lebih rendah sehingga kurang cocok untuk aplikasi dengan kebutuhan latensi rendah.
2. Metode berbasis satu tahap (*single-stage detectors*): Pendekatan ini melakukan deteksi dan klasifikasi objek secara langsung dalam satu langkah inferensi tanpa memerlukan tahap proposal region yang terpisah. Model memproses seluruh gambar sekaligus dan secara langsung memprediksi *bounding box* serta label kelas untuk semua objek yang terdeteksi. Contoh metode ini meliputi YOLO (*You Only Look Once*), SSD (*Single Shot MultiBox Detector*), dan RetinaNet [18]. Metode satu tahap

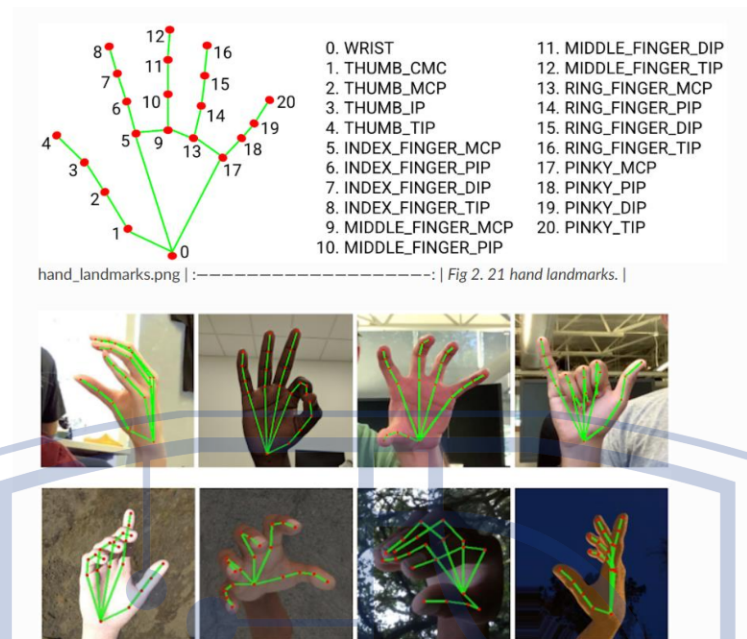
menawarkan kecepatan inferensi yang jauh lebih tinggi sehingga sangat cocok untuk aplikasi dengan kebutuhan latensi rendah seperti pengenalan gestur yang responsif.

Perkembangan arsitektur deteksi objek telah mengalami kemajuan signifikan seiring dengan berkembangnya teknik *deep learning* [17]. Penggunaan *Convolutional Neural Network* (CNN) sebagai *backbone* ekstraktor fitur telah menjadi standar dalam sistem deteksi objek modern, menggantikan pendekatan berbasis fitur manual seperti HOG (*Histogram of Oriented Gradients*) dan Haar Cascade yang digunakan pada generasi sebelumnya [18]. Arsitektur modern seperti *Feature Pyramid Network* (FPN) juga diperkenalkan untuk menangani deteksi objek pada berbagai skala ukuran secara bersamaan, sehingga meningkatkan kemampuan model dalam mendeteksi objek kecil maupun besar dalam satu gambar [17].

Dalam konteks pengenalan bahasa isyarat, deteksi objek berperan penting untuk mengidentifikasi dan melokalisasi tangan, wajah, serta bagian tubuh lainnya yang terlibat dalam gestur isyarat [19]. Kemampuan deteksi objek dalam menentukan posisi spasial komponen-komponen gestur secara akurat menjadi fondasi bagi proses pengenalan dan klasifikasi gestur bahasa isyarat secara otomatis. Penelitian ini menggunakan pendekatan deteksi objek berbasis satu tahap, yaitu YOLOv8, karena kemampuannya melakukan deteksi secara latensi yang rendah dengan akurasi tinggi, yang sangat diperlukan untuk aplikasi penerjemah bahasa isyarat pada perangkat *mobile*.

2.3 MediaPipe

MediaPipe merupakan *framework* sumber terbuka (*open-source*) yang dikembangkan oleh Google untuk membangun *pipeline* pemrosesan data multimedia secara cepat [20]. *Framework* ini menyediakan berbagai solusi *machine learning* yang telah dioptimalkan untuk berjalan pada berbagai platform, termasuk perangkat mobile (Android dan iOS), web, dan desktop [21]. MediaPipe dirancang dengan arsitektur berbasis graf (*graph-based*) yang memungkinkan pengembang untuk membangun dan mengkustomisasi *pipeline* pemrosesan data secara modular dan efisien [20].



Gambar 2.2 Representasi 21 Titik Landmark Tangan MediaPipe [14]

Dalam konteks pengenalan bahasa isyarat, MediaPipe menyediakan beberapa modul penting yang relevan [21]:

1. *MediaPipe Hands*: Modul ini mampu mendeteksi dan melacak 21 *landmark* (titik kunci) pada setiap tangan secara cepat. Setiap *landmark* merepresentasikan sendi atau ujung jari pada tangan, yang direpresentasikan dalam koordinat tiga dimensi (x, y, z) [22]. Proses deteksi tangan pada MediaPipe terdiri dari dua tahap: (a) deteksi telapak tangan menggunakan model *palm detector* berbasis BlazePalm, dan (b) estimasi *landmark* tangan menggunakan model *hand landmark* [22].
2. *MediaPipe Holistic*: Modul ini mengintegrasikan deteksi pose tubuh (33 *landmark*), tangan (21 *landmark* per tangan), dan wajah (468 *landmark*) dalam satu *pipeline* yang terpadu [23]. Modul ini sangat berguna untuk pengenalan bahasa isyarat yang memerlukan informasi dari berbagai bagian tubuh secara simultan.
3. *MediaPipe Pose*: Modul ini mendeteksi 33 *landmark* pada tubuh manusia yang dapat digunakan untuk melacak posisi dan gerakan tubuh secara keseluruhan [20].

Keunggulan MediaPipe dalam konteks penelitian ini meliputi [21]:

- a. Kemampuan pemrosesan cepat dengan latensi rendah, bahkan pada perangkat mobile.
- b. Akurasi tinggi dalam deteksi dan pelacakan *landmark* tangan.
- c. Kompatibilitas lintas-platform yang memudahkan implementasi pada berbagai perangkat.

- d. Efisiensi komputasi yang memungkinkan penggunaan pada perangkat dengan sumber daya terbatas.

Meskipun MediaPipe menyediakan kemampuan deteksi *landmark* yang sangat baik, *framework* ini pada dasarnya hanya mengekstraksi titik-titik kunci (*key points*) tanpa melakukan klasifikasi gestur secara langsung [22]. Oleh karena itu, diperlukan model *deep learning* tambahan seperti LSTM untuk memproses urutan *landmark* yang dihasilkan MediaPipe dan mengklasifikasikannya menjadi gestur bahasa isyarat tertentu. Dalam penelitian ini, MediaPipe digunakan sebagai komponen pendukung dalam proses ekstraksi fitur spasial dari gestur tangan yang terdeteksi.

2.4 Deep Learning

Deep learning merupakan subbidang dari *machine learning* yang menggunakan jaringan saraf tiruan (*artificial neural networks*) dengan banyak lapisan tersembunyi (*hidden layers*) untuk mempelajari representasi data secara hierarkis [24]. Istilah "*deep*" merujuk pada kedalaman arsitektur jaringan yang terdiri dari banyak lapisan, yang memungkinkan model untuk mempelajari fitur-fitur abstrak dari data mentah secara otomatis tanpa memerlukan rekayasa fitur secara manual (*manual feature engineering*) [25].

Konsep dasar *deep learning* berakar pada jaringan saraf tiruan yang pertama kali diperkenalkan oleh McCulloch dan Pitts pada tahun 1943, kemudian berkembang melalui perceptron oleh Rosenblatt pada tahun 1958, dan *backpropagation* oleh Rumelhart, Hinton, dan Williams pada tahun 1986 [26]. Namun, perkembangan signifikan *deep learning* baru terjadi pada dekade 2010-an, didorong oleh tiga faktor utama: (a) ketersediaan dataset berskala besar, (b) peningkatan daya komputasi melalui GPU, dan (c) inovasi dalam arsitektur dan teknik pelatihan jaringan [25].

Arsitektur *deep learning* yang umum digunakan meliputi [24]:

1. Convolutional Neural Network (CNN): Dirancang khusus untuk memproses data grid seperti citra, CNN menggunakan operasi konvolusi untuk mengekstraksi fitur spasial secara hierarkis [27]. CNN sangat efektif untuk tugas-tugas *computer vision* seperti klasifikasi gambar, deteksi objek, dan segmentasi semantik.
2. Recurrent Neural Network (RNN): Dirancang untuk memproses data sekuensial, RNN memiliki koneksi berulang (*recurrent connections*) yang memungkinkan informasi dari langkah waktu sebelumnya memengaruhi pemrosesan pada langkah waktu saat ini [28]. Varian RNN seperti LSTM dan GRU dikembangkan untuk mengatasi masalah *vanishing gradient* pada RNN standar.

3. Generative Adversarial Network (GAN): Terdiri dari dua jaringan yang saling berkompetisi, yaitu *generator* dan *discriminator*, GAN digunakan untuk menghasilkan data sintetis yang realistis [29].
4. Transformer: Arsitektur berbasis mekanisme *attention* yang awalnya dikembangkan untuk pemrosesan bahasa alami dan kemudian diterapkan pada berbagai domain termasuk *computer vision* [30].

Proses pelatihan *deep learning* umumnya menggunakan algoritma *backpropagation* yang dikombinasikan dengan teknik optimasi seperti *Stochastic Gradient Descent* (SGD), Adam, atau RMSprop untuk memperbarui bobot jaringan secara iteratif [25]. Fungsi *loss* digunakan untuk mengukur perbedaan antara prediksi model dan nilai sebenarnya, dan tujuan pelatihan adalah meminimalkan nilai *loss* tersebut.

Dalam konteks penelitian ini, *deep learning* diterapkan melalui dua arsitektur utama: CNN (dalam bentuk YOLOv8) untuk ekstraksi fitur spasial dan deteksi gestur, serta LSTM untuk memodelkan dependensi temporal dalam urutan gerakan BISINDO.

2.5 Computer Vision

Computer vision merupakan bidang ilmu interdisipliner yang bertujuan untuk memberikan kemampuan pada komputer untuk memahami dan menginterpretasikan informasi visual dari dunia nyata, sebagaimana yang dilakukan oleh sistem visual manusia [31]. *Computer vision* mencakup berbagai tugas seperti akuisisi, pemrosesan, analisis, dan pemahaman citra digital untuk mengekstraksi informasi bermakna dari data visual [17].

Secara umum, *pipeline* sistem *computer vision* terdiri dari beberapa tahapan [11]:

1. Akuisisi citra (*Image Acquisition*): Proses penangkapan citra atau video menggunakan perangkat sensor seperti kamera digital, kamera *smartphone*, atau sensor kedalaman (*depth sensor*) [12]. Kualitas citra yang diperoleh pada tahap ini sangat memengaruhi performa keseluruhan sistem.
2. Pra-pemrosesan (*Preprocessing*): Tahap ini mencakup berbagai teknik untuk meningkatkan kualitas citra sebelum diproses lebih lanjut, seperti normalisasi, penyesuaian kontras, pengurangan *noise*, dan *resize* citra [13]. Pra-pemrosesan bertujuan untuk menstandarkan data input dan mengurangi variabilitas yang tidak relevan.
3. Ekstraksi fitur (*Feature Extraction*): Proses mengidentifikasi dan mengekstraksi informasi penting dari citra, seperti tepi, sudut, tekstur, warna, dan bentuk [11]. Dalam pendekatan *deep learning*, ekstraksi fitur dilakukan secara otomatis oleh jaringan saraf konvolusi.

4. Klasifikasi dan interpretasi: Tahap akhir di mana fitur-fitur yang telah diekstraksi digunakan untuk mengklasifikasikan objek, mengenali pola, atau membuat keputusan [22].

Tugas-tugas utama dalam *computer vision* meliputi [13]:

- a. Klasifikasi gambar (*Image Classification*): Menentukan kategori atau kelas dari suatu gambar.
- b. Deteksi objek (*Object Detection*): Mengidentifikasi dan melokalisasi objek dalam gambar.
- c. Segmentasi gambar (*Image Segmentation*): Membagi gambar menjadi beberapa wilayah atau segmen.
- d. Pelacakan objek (*Object Tracking*): Melacak pergerakan objek dalam rangkaian frame video.
- e. Estimasi pose (*Pose Estimation*): Menentukan posisi dan orientasi bagian tubuh manusia.
- f. Pengenalan tindakan (*Action Recognition*): Mengidentifikasi aktivitas atau gerakan yang dilakukan dalam video.

Dalam konteks pengenalan bahasa isyarat, *computer vision* berperan sebagai pendekatan non-invasif yang memungkinkan sistem untuk menangkap dan menganalisis gestur isyarat melalui kamera tanpa memerlukan perangkat sensor tambahan yang dilekatkan pada tubuh pengguna [14]. Pendekatan berbasis *computer vision* memiliki beberapa keunggulan dibandingkan pendekatan berbasis sensor *wearable*, antara lain [19]:

1. Tidak memerlukan perangkat tambahan yang mahal atau tidak nyaman digunakan.
2. Mampu menangkap informasi gestur secara menyeluruh, termasuk ekspresi wajah dan gerakan tubuh.
3. Lebih natural dan tidak mengganggu proses komunikasi.
4. Lebih mudah diakses karena hanya membutuhkan kamera yang tersedia pada *smartphone*.

Penelitian ini memanfaatkan pendekatan *computer vision* untuk memproses input visual dari kamera *smartphone*, di mana gestur BISINDO ditangkap dalam bentuk rangkaian frame video dan diproses menggunakan model *deep learning* untuk menghasilkan terjemahan dalam bentuk teks.

2.6 Augmentasi Data

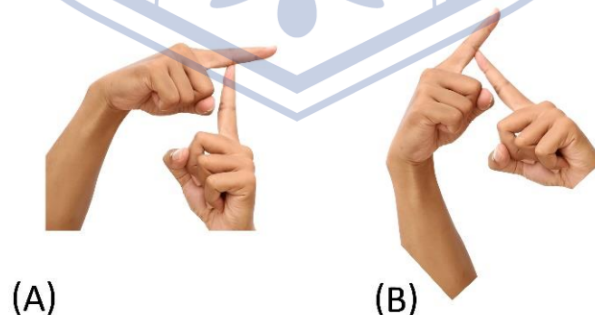
Augmentasi data merupakan teknik yang umum digunakan untuk mengurangi *overfitting* dengan menghasilkan data baru melalui transformasi terhadap data asli sehingga meningkatkan variasi data yang awalnya terbatas dan membantu model untuk tidak sekadar menghafal pola pada data pelatihan, melainkan mampu melakukan generalisasi terhadap data baru [15]. Teknik augmentasi data terdapat beberapa metode yang secara umum diklasifikasikan ke dalam dua pendekatan utama, yaitu modifikasi warna citra dan transformasi geometrik. Modifikasi warna citra meliputi penyesuaian kontras, histogram *equalization*, peningkatan ketajaman (*sharpening*), dan *white balance*, yang bertujuan untuk mengubah karakteristik warna tanpa mengubah bentuk objek pada citra. Sementara itu, transformasi geometrik mencakup teknik seperti rotasi, refleksi, *shear*, dan dilatasi, yang bertujuan untuk mengubah struktur spasial citra guna menghasilkan representasi baru dari data yang sama [16]. Teknik augmentasi data yang banyak digunakan adalah rotasi dan pencerminan.

1. Rotasi

Rotasi adalah transformasi yang memutar citra dengan proses memindahkan nilai pikselnya sebesar θ° yang dihitung dari pusat rotasi menggunakan Persamaan 23 sebagai berikut:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos \theta^\circ & \sin \theta^\circ \\ \sin \theta^\circ & \cos \theta^\circ \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \dots\dots\dots (1)$$

Di mana koordinat (x_1, y_1) adalah posisi awal sebelum rotasi, dan (x_2, y_2) adalah koordinat titik setelah dilakukan rotasi sebesar θ° dengan pusat rotasi di titik $(0, 0)$. Visualisasi dari efek rotasi dapat dilihat pada Gambar 2.3 yang menunjukkan perubahan posisi pada citra sebelum dan sesudah transformasi dilakukan.



Gambar 2.3 (a) Gambar Asli, (b) Gambar setelah diputar sebesar 45° [16]

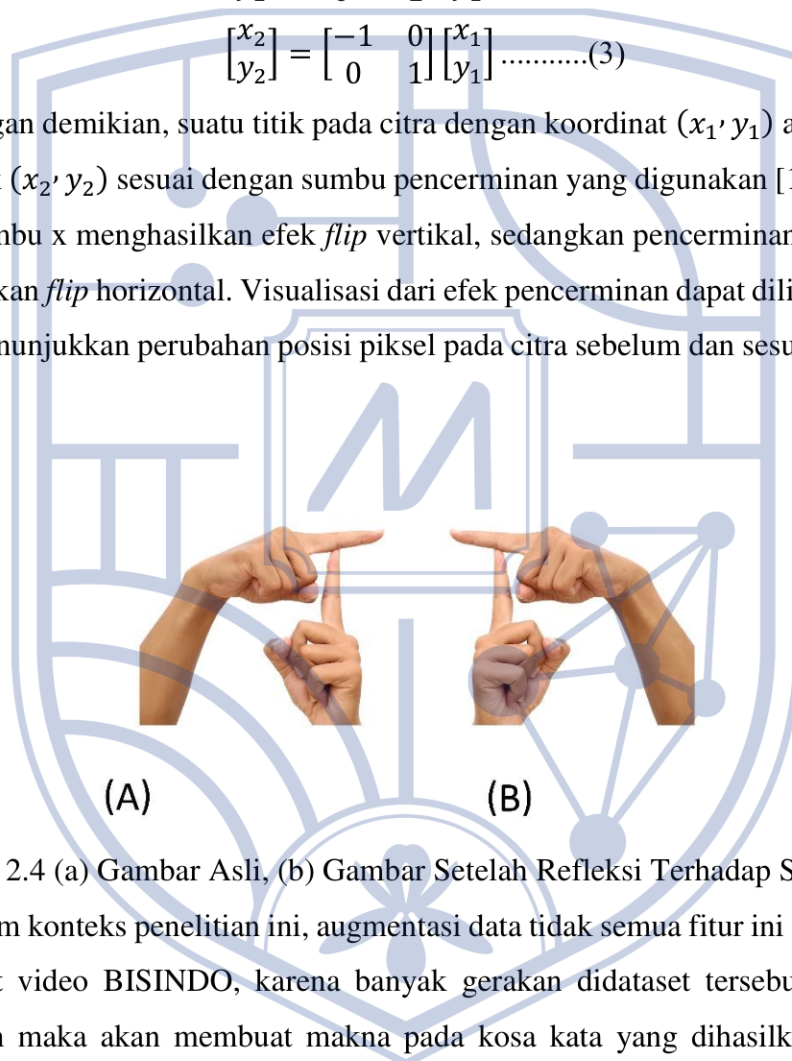
2. Pencerminkan

Pencerminkan adalah salah satu bentuk transformasi geometrik pada citra digital yang dilakukan dengan membalik posisi piksel terhadap sumbu tertentu, yaitu sumbu horizontal (sumbu x) atau sumbu vertikal (sumbu y). Proses perubahan posisi nilai piksel tersebut dapat dihitung menggunakan Persamaan 24 dan Persamaan 25 sebagai berikut:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \dots\dots\dots(2)$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \dots\dots\dots(3)$$

Dengan demikian, suatu titik pada citra dengan koordinat (x_1, y_1) akan dicerminkan menjadi titik (x_2, y_2) sesuai dengan sumbu pencerminan yang digunakan [16]. Pencerminkan terhadap sumbu x menghasilkan efek *flip* vertikal, sedangkan pencerminan terhadap sumbu y menghasilkan *flip* horizontal. Visualisasi dari efek pencerminan dapat dilihat pada Gambar 2.4 yang menunjukkan perubahan posisi piksel pada citra sebelum dan sesudah transformasi dilakukan.



Gambar 2.4 (a) Gambar Asli, (b) Gambar Setelah Refleksi Terhadap Sumbu y [16]

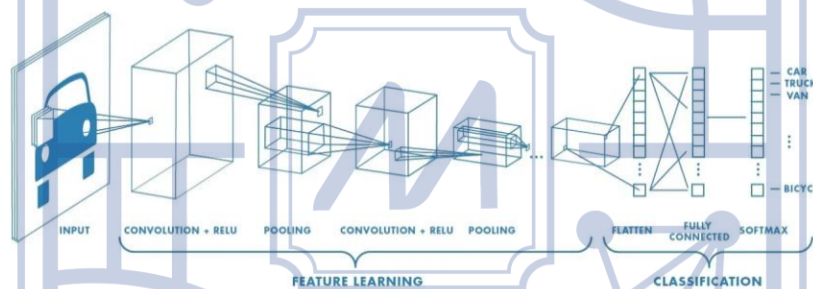
Dalam konteks penelitian ini, augmentasi data tidak semua fitur ini dapat diterapkan pada dataset video BISINDO, karena banyak gerakan didataset tersebut jika dilakukan pencerminan maka akan membuat makna pada kosa kata yang dihasilkan akan berbeda makna.

2.7 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) pertama kali diperkenalkan oleh Kunihiro Fukushima pada tahun 1980 melalui model yang disebut *Neocognitron*, yang terinspirasi oleh mekanisme penglihatan pada makhluk hidup. *Neocognitron* memiliki kemampuan untuk mengenali pola visual dan dianggap sebagai cikal bakal arsitektur CNN *modern* [35]. Pada tahun 1998, Yann LeCun dan rekan-rekannya mengembangkan LeNet-5, sebuah CNN

yang dirancang untuk mengenali digit tulisan tangan, yang menunjukkan efektivitas CNN dalam tugas pengenalan gambar [36]. Sejak itu, CNN telah mengalami perkembangan pesat, terutama dengan ketersediaan dataset besar dan peningkatan daya komputasi, menjadikannya metode utama dalam berbagai aplikasi visi komputer.

Convolutional Neural Network (CNN) merupakan salah satu pengembangan dari jaringan saraf tiruan yang terinspirasi dari jaringan saraf manusia dan biasa digunakan pada data gambar untuk mendeteksi dan mengenali suatu objek pada sebuah gambar. *Convolutional Neural Network* (CNN) adalah perkembangan metode dari *Multi Layer Perception* (MLP). Namun, CNN memiliki jumlah dimensi yang lebih banyak dibandingkan dengan MLP. CNN memiliki masukan (*input*) array mulai dari dua dimensi hingga lebih [17]. Struktur utama CNN terdiri dari beberapa jenis lapisan, yang dapat terlihat pada gambar berikut.



Gambar 2.5 Arsitektur CNN [12]

Convolutional Neural Network (CNN) bekerja melalui serangkaian lapisan yang dirancang untuk mengekstraksi fitur dari data *input*, seperti gambar, dan melakukan klasifikasi. Proses ini melibatkan beberapa tahapan utama:

1. Lapisan Konvolusi (*Convolutional Layer*)

Lapisan konvolusi berfungsi untuk mengekstraksi fitur dari data *input* dengan menggunakan operasi konvolusi. Lapisan ini terdiri dari sejumlah filter atau kernel yang belajar secara otomatis selama proses pelatihan. Kernel ini diterapkan pada gambar *input* untuk mendeteksi berbagai fitur, seperti tepi, sudut, atau tekstur, dengan cara menggeser filter melintasi gambar dan melakukan perhitungan *dot product* antara nilai piksel dan bobot kernel. Hasil dari operasi konvolusi adalah peta fitur (*feature map*), yang merepresentasikan informasi spasial dari fitur yang terdeteksi dalam gambar. Bobot pada lapisan ini menentukan kernel yang digunakan, dan bobot tersebut diperbarui selama proses pelatihan untuk meningkatkan akurasi ekstraksi fitur [38]. Ukuran *feature map* tidak selalu sama pada tiap convol, perbedaan tersebut bertujuan untuk mendapatkan pola yang bervariasi. Misalkan

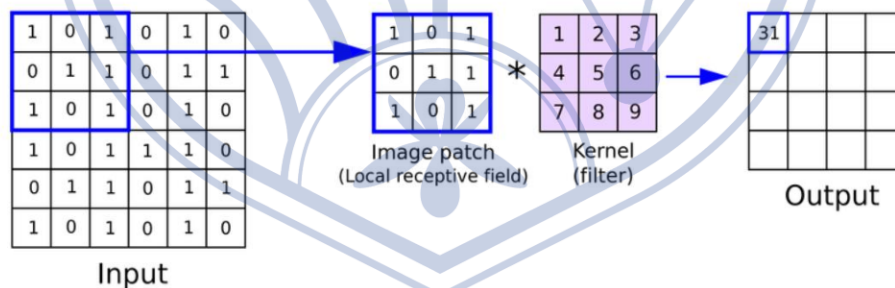
pada ukuran 7x7, kemudian selanjutnya yaitu berukuran 3x3, hal ini menunjukkan bahwa proses konvolusi dapat mengenali pola secara lebih detail dan terperinci. Ukuran juga ditentukan dari nilai *stride* dan *padding* yang merupakan parameter yang digunakan untuk menentukan nilai tambahan 0 dan pergeseran nilai filter pada citra *input*. *Padding* merupakan penambahan nilai nol di tepi gambar *input* untuk mempertahankan ukuran *output* atau mengontrol ukuran *feature map*. Sementara itu, *stride* merupakan jumlah langkah filter bergeser setiap kali melakukan konvolusi, yang menentukan seberapa banyak *input* yang dilewati. Secara matematis, ukuran matriks *feature map* dapat ditentukan dengan rumus berikut [38]:

$$u_H = \frac{w - f + 2p}{s} + 1 \dots \dots \dots (4)$$

Di mana u adalah ukuran matriks H, w adalah ukuran matriks *input*, f adalah ukuran matriks filter, p adalah *padding* dan s adalah *stride*. Hasil matriks H menggunakan Persamaan 2 sebagai berikut:

$$\sum_m \sum_n I_{(i+m,j+n)} * K_{(m,n)} \dots \dots \dots (5)$$

Di mana c adalah matriks *input* dan f adalah konvolusi filter. Tujuan utama lapisan konvolusi adalah untuk memperoleh representasi fitur yang lebih abstrak seiring bertambahnya kedalaman jaringan, sambil tetap mempertahankan informasi spasial yang penting dalam data *input* [38].



Gambar 2.6 Proses pada *Convolutional Layer* [4]

2. Batch Normalization

Pada lapisan *batch normalization*, dilakukan proses normalisasi terhadap data dalam setiap *mini-batch* dengan tujuan utama untuk mengurangi risiko *overfitting*. *Overfitting* terjadi ketika model terlalu menyesuaikan diri dengan data pelatihan sehingga kinerjanya menurun saat diuji dengan data baru. Selain mencegah *overfitting*, *batch normalization* juga mempercepat dan menstabilkan proses pembelajaran pada jaringan *neural*. *Normalization*

menggunakan nilai rata-rata (μ) dan varian (σ^2) pada matriks *input*. Selain itu, digunakan pula dua parameter pembelajaran, yaitu γ (*gamma*) dan β (*beta*), serta sebuah konstanta kecil ε (*epsilon*) untuk menjaga kestabilan numerik selama proses normalisasi. Proses ini dirumuskan dalam Persamaan 3 [38].

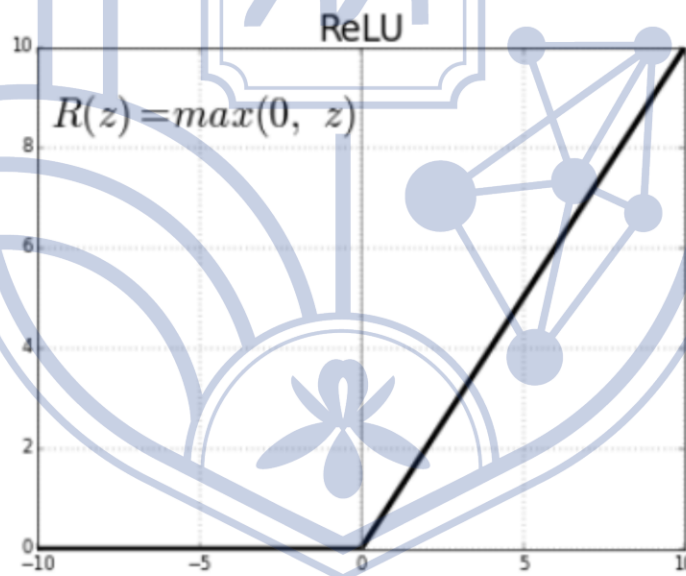
$$BN = (\gamma \frac{I - \mu}{\sqrt{\sigma^2 + \varepsilon}}) + \beta \dots \dots \dots (6)$$

3. Fungsi Aktivasi ReLU

Rectified Linear Unit (ReLU) adalah fungsi aktivasi yang umum digunakan dalam *Convolutional Neural Network* (CNN) karena kesederhanaannya dan efisiensi komputasinya. Fungsi ini bekerja dengan mengubah nilai negatif menjadi nol, sementara nilai positif tetap dipertahankan sebagaimana adanya, dirumuskan sebagai [39]:

$$f(x) = \max(0, x) \dots \dots \dots (7)$$

Dalam arsitektur CNN, fungsi aktivasi ReLU biasanya diterapkan setelah setiap operasi konvolusi dan sebelum operasi *pooling*, serta di lapisan *fully connected*. Penerapan ReLU memungkinkan jaringan untuk mempelajari representasi *non-linear* yang kompleks dari data *input*, yang sangat penting untuk tugas-tugas pengenalan pola visual yang rumit.



Gambar 2.7 Aktivitas ReLU [14]

4. Lapisan Pooling (*Pooling Layer*)

Setelah operasi konvolusi, lapisan *pooling* digunakan untuk mengurangi dimensi dari peta fitur (*feature map*), yang bertujuan untuk mengurangi kompleksitas komputasi dan jumlah parameter dalam jaringan. Dengan melakukan reduksi dimensi ini, lapisan *pooling* membantu meningkatkan efisiensi proses pelatihan serta mengurangi kemungkinan

overfitting. Hasil *feature map* keluaran pada *layer* ini dihitung dengan melihat ukuran filter dan *stride* yang ditentukan sehingga didapatkan Persamaan 5 sebagai berikut [38]:

$$u_K = \frac{w - f}{s} + 1 \dots \dots \dots (8)$$

Teknik *pooling* yang umum digunakan adalah *max pooling* dan *average pooling*. *Max pooling* memilih nilai maksimum dalam jendela tertentu pada peta fitur, sedangkan *average pooling* mengambil rata-rata nilai dalam jendela tersebut. *Pooling* memungkinkan CNN untuk mempertahankan fitur dominan dari gambar tanpa kehilangan informasi yang signifikan [18].



Gambar 2.8 Ilustrasi Perhitungan *Pooling Layer*

5. Lapisan Fully Connected (*Fully Connected Layer*)

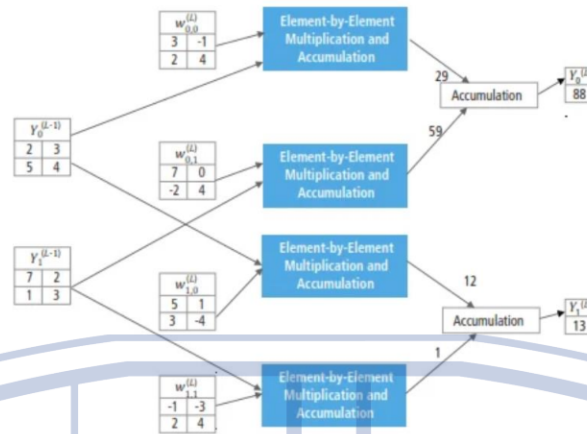
Peta fitur yang telah melalui proses konvolusi dan *pooling* kemudian diratakan menjadi vektor satu dimensi (*flattening*) dan diteruskan ke lapisan *fully connected*. Lapisan ini berfungsi untuk melakukan transformasi pada fitur yang telah diekstraksi agar dapat diklasifikasikan secara linear. Dalam lapisan ini, setiap *neuron* memiliki koneksi penuh ke semua *neuron* di lapisan sebelumnya, dan *output*-nya dihitung melalui operasi perkalian matriks yang diikuti dengan penambahan bias *offset*. Aktivasi dalam lapisan *fully connected* sering kali menggunakan fungsi aktivasi seperti *Softmax* atau ReLU, yang membantu menentukan probabilitas dari setiap kelas atau label *output* [38]. Vektor satu dimensi tersebut akan terhubung dengan jaringan saraf tiruan, seperti yang diilustrasikan pada Gambar 2.5 dengan menggunakan rumus sebagai berikut [18]:

$$f_{cj} = b_j + \sum_i x_i W_{i,j}, j = 1, 2, 3, \dots, t \dots \dots \dots (9)$$

Di mana:

- b = nilai vektor bias
- W = bobot
- t = jumlah kelas

d. x = vektor *input*.



Gambar 2.9 Ilustrasi Lapisan *Fully Connected*

6. Softmax

Softmax adalah fungsi aktivasi yang digunakan dalam proses klasifikasi linear dengan memperhitungkan probabilitas suatu data termasuk ke dalam masing-masing kelas. Nilai keluaran untuk setiap kelas berada dalam rentang antara 0 hingga 1, dan total dari semua nilai keluaran kelas bernilai 1. Probabilitas keluaran fungsi *softmax* dihitung dengan rumus berikut [38]:

$$softmax_j = \frac{e^{f_{c_j}}}{\sum_t e^{f_{c_j}}}, j = 1, 2, 3, \dots, t \dots \dots \dots (10)$$

Di mana f_c adalah nilai dari lapisan *fully connected*.

2.8 YOLO (You Only Look Once)

YOLO (You Only Look Once) merupakan algoritma deteksi objek berbasis deep learning yang dirancang untuk melakukan deteksi objek secara cepat dengan akurasi tinggi [40]. Berbeda dengan pendekatan deteksi objek konvensional seperti R-CNN yang memisahkan tahap region proposal dan klasifikasi, YOLO melakukan deteksi objek dalam satu tahap inferensi (single-stage detector) yang memungkinkan prediksi bounding box dan klasifikasi objek dilakukan secara bersamaan [18]. Pendekatan ini menjadikan YOLO sangat efisien untuk aplikasi yang memerlukan pemrosesan cepat, termasuk pada perangkat dengan sumber daya komputasi terbatas seperti smartphone.

2.8.1 Sejarah dan Perkembangan YOLO

YOLO pertama kali diperkenalkan oleh Joseph Redmon et al. pada tahun 2016 dengan pendekatan revolusioner yang memandang deteksi objek sebagai masalah regresi

tunggal, dari piksel gambar langsung ke koordinat bounding box dan probabilitas kelas [40]. Sejak saat itu, YOLO telah mengalami berbagai evolusi signifikan melalui beberapa versi:

YOLOv1 (2016) merupakan versi pertama yang memperkenalkan konsep single-stage detection dengan membagi gambar menjadi grid $S \times S$ dan memprediksi bounding box serta probabilitas kelas untuk setiap sel grid [40], meskipun masih memiliki keterbatasan dalam mendeteksi objek kecil dan objek yang berdekatan. YOLOv2/YOLO9000 (2017) memperkenalkan anchor boxes, batch normalization, dan arsitektur Darknet-19 sehingga mampu mendeteksi hingga 9000 kategori objek sekaligus mempertahankan kecepatan inferensi [41]. YOLOv3 (2018) mengadopsi arsitektur Darknet-53 yang lebih dalam dan memperkenalkan prediksi multi-skala menggunakan Feature Pyramid Network (FPN) sehingga menunjukkan peningkatan signifikan dalam mendeteksi objek dengan berbagai ukuran [42]. YOLOv4 (2020) mengintegrasikan berbagai teknik modern seperti CSPDarknet53 sebagai backbone, PANet sebagai neck, Mosaic data augmentation, dan CIOU loss, sehingga mencapai state-of-the-art performance pada dataset COCO dengan kecepatan 65 FPS [43]. YOLOv5 (2020) dikembangkan oleh Ultralytics dengan fokus pada kemudahan penggunaan, modularitas, dan efisiensi deployment, serta memperkenalkan berbagai varian ukuran model yang memungkinkan trade-off antara akurasi dan kecepatan sesuai kebutuhan aplikasi [16]. YOLOv7 (2022) memperkenalkan arsitektur E-ELAN (Extended Efficient Layer Aggregation Network) dan teknik pelatihan auxiliary head yang meningkatkan kemampuan pembelajaran representasi fitur sehingga mencapai kecepatan dan akurasi lebih tinggi dibandingkan versi sebelumnya [44]. YOLOv8 (2023) merupakan versi terbaru yang dikembangkan oleh Ultralytics dengan arsitektur yang dioptimalkan untuk efisiensi dan fleksibilitas, memperkenalkan pendekatan anchor-free detection, decoupled head, dan modul C2f yang lebih efisien dibandingkan CSP pada YOLOv5 [10].

Tabel 2.2 menunjukkan perbandingan evolusi arsitektur YOLO dari versi ke versi.

Tabel 2.2 Perbandingan Evolusi Arsitektur YOLO [40], [41], [10]

No	Versi	Thn	Backbone	Neck	Head	Ancho r	Inovasi Utama
1.	YOLOv1	2016	Custom CNN	-	Single	None	Single-stage detection
2.	YOLOv2	2017	Darknet-19	-	Single	Anchor -based	Anchor boxes, Batch Norm

3.	YOLOv3	2018	Darknet-53	FPN	Single	Anchor -based	Multi-scale prediction
4.	YOLOv4	2020	CSPDarknet 53	PANet	Single	Anchor -based	Mosaic augmentation, CIoU
5.	YOLOv5	2020	CSPDarknet	PANet	Single	Anchor -based	Multiple model sizes
6.	YOLOv7	2022	E-ELAN	E-ELAN- PANet	Single	Anchor -based	Auxiliary head training
7.	YOLOv8	2023	C2f- CSPDarknet	PAN + SPPF	Decouple d	Anchor -free	C2f module, anchor-free

Perkembangan dari YOLOv1 hingga YOLOv8 menunjukkan tren peningkatan berkelanjutan dalam hal akurasi, kecepatan, dan efisiensi komputasi, menjadikan YOLO sebagai salah satu algoritma deteksi objek yang paling banyak digunakan dalam penelitian dan aplikasi industri.

2.8.1.1 Varian Model YOLOv8

YOLOv8 tersedia dalam lima varian ukuran yang memungkinkan *trade-off* antara akurasi dan efisiensi komputasi sesuai dengan kebutuhan aplikasi dan ketersediaan sumber daya perangkat [10]. Tabel 2.3 menunjukkan perbandingan varian model YOLOv8.

Tabel 2.3 Perbandingan Varian Model YOLOv8 [10], [11]

No	Varian	Parameters	FLOPs	mAP@0.5:0.95	Inference Speed	Use Case
1.	YOLOv8n (Nano)	3.2M	8.7G	~37.3%	80 FPS	Edge devices, IoT
2.	YOLOv8s (Small)	11.2M	28.6G	~44.9%	60 FPS	Mobile applications
3.	YOLOv8m (Medium)	25.9M	78.9G	~50.2%	40 FPS	Balanced performance

4.	YOLOv8l (Large)	43.7M	165.2G	~52.9%	25 FPS	<i>High accuracy required</i>
5.	YOLOv8x (Extra-Large)	68.2M	257.8G	~53.9%	15 FPS	<i>Maximum accuracy</i>

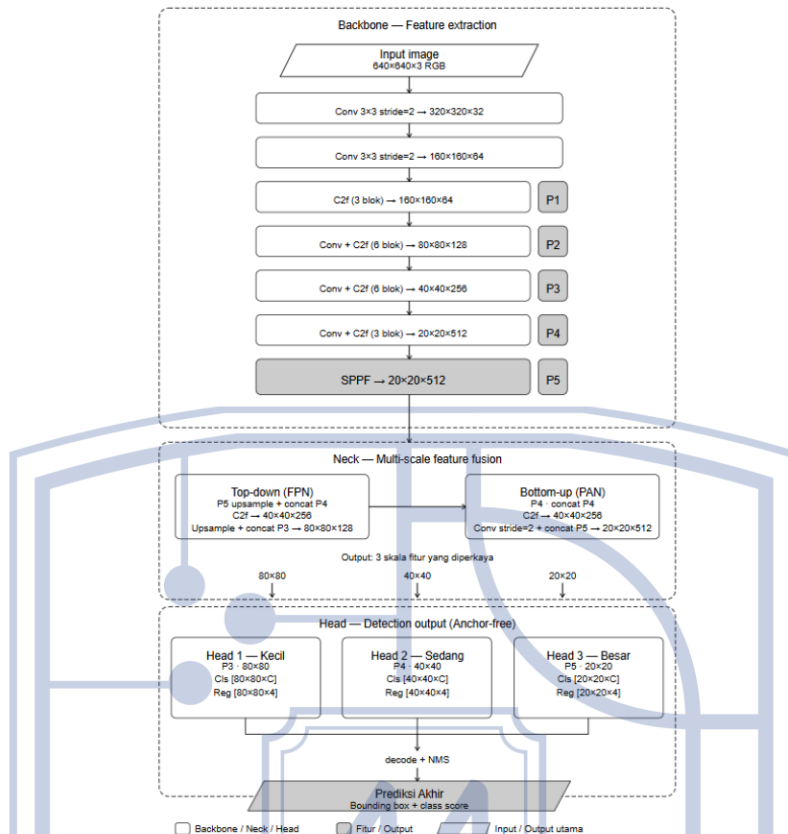
Keterangan:

- a) Parameters: Jumlah parameter model (dalam juta)
- b) FLOPs: *Floating Point Operations* (dalam giga)
- c) mAP@0.5:0.95: *Mean Average Precision* pada COCO dataset
- d) *Inference Speed*: Kecepatan inferensi pada GPU NVIDIA V100

Dalam penelitian ini, digunakan YOLOv8s (Small) karena memberikan keseimbangan optimal antara akurasi dan kecepatan untuk aplikasi mobile penerjemah BISINDO, dengan ukuran model yang cukup ringan (11.2M parameters) namun tetap memberikan akurasi yang memadai (mAP ~44,9%).

2.8.2 Arsitektur YOLOv8

YOLOv8 merupakan versi terbaru dari keluarga YOLO yang dikembangkan oleh Ultralytics pada tahun 2023 sebagai penerus YOLOv5 dengan berbagai peningkatan arsitektur dan performa [10]. Arsitektur YOLOv8 dirancang dengan tiga komponen utama: *Backbone*, *Neck*, dan *Head*, yang bekerja secara terintegrasi untuk melakukan ekstraksi fitur, agregasi multi-skala, dan prediksi akhir. Gambar 2.11 menunjukkan arsitektur lengkap YOLOv8.



Gambar 2.10 Arsitektur Lengkap YOLOv8

Sumber: Diadaptasi dari [23]

2.8.2.1 Komponen *Backbone*

Backbone berfungsi sebagai *feature extractor* yang mengekstraksi representasi fitur hierarkis dari citra input [11]. YOLOv8 menggunakan modifikasi dari CSPDarknet dengan modul C2f (*Cross Stage Partial with 2 Convolutions and Faster*) yang menggantikan modul CSP pada YOLOv5.

Proses pada *Backbone*:

1. Input Layer – Menerima citra RGB berukuran 640×640×3 piksel.
2. Stem Layer – Terdiri dari dua lapisan konvolusi dengan stride 2 untuk *downsampling* awal, menghasilkan *feature map* 160×160×64.
3. C2f Modules – Modul C2f diterapkan pada berbagai *stage* dengan jumlah *block* yang bervariasi:
 - a. Stage 1: 3 C2f blocks → 160×160×64
 - b. Stage 2: 6 C2f blocks → 80×80×128
 - c. Stage 3: 6 C2f blocks → 40×40×256
 - d. Stage 4: 3 C2f blocks → 20×20×512

Ukuran feature map pada setiap tahap downsampling di backbone dapat dihitung menggunakan rumus umum ukuran output konvolusi berikut:

$$H_{out} = \left\lfloor \frac{W - K + 2p}{s} \right\rfloor + 1 \dots\dots\dots(11)$$

Di mana:

- a. H_{out} Hout = ukuran dimensi output (tinggi atau lebar) feature map
- b. W W = ukuran dimensi input
- c. K K = ukuran kernel (filter)
- d. p p = padding
- e. s s = stride

Rumus ini digunakan untuk memverifikasi transisi ukuran feature map pada setiap stage backbone, misalnya dari input $640 \times 640 \times 3$ menjadi $160 \times 160 \times 64$ setelah Stem Layer, dan seterusnya hingga mencapai $20 \times 20 \times 512$ pada P5.

Operasi konvolusi pada setiap posisi dihitung menggunakan Persamaan berikut, sebagaimana telah dijelaskan pada Sub-bab 2.7.2:

$$H_{i,j} = \sum_m \sum_n I_{(i+m, j+n)} \times K_{(m,n)} \dots\dots\dots(12)$$

Di mana $H_{i,j}$ $H_{i,j}$ adalah nilai piksel output pada posisi (i, j) (i, j) , I adalah matriks input, dan K K adalah kernel konvolusi berukuran $(m \times n)$ $(m \times n)$. Notasi ini digunakan secara konsisten pada pembahasan implementasi konvolusi di Bab III.

Modul C2f memiliki keunggulan dibandingkan CSP konvensional karena menggunakan lebih sedikit parameter sambil mempertahankan *gradient flow* yang lebih baik melalui *shortcut connections* [41]. Arsitektur C2f dapat dirumuskan sebagai:

$$C2f(x) = \text{Concat}(\text{Conv}(x), \text{Bottleneck}_1(x), \dots, \text{Bottleneck}_n(x))$$

Di mana:

- a. x = input feature map
- b. Bottleneck_i = bottleneck block ke-i dengan residual connection
- c. n = jumlah bottleneck blocks

Berbeda dengan CNN konvensional yang umumnya menggunakan ReLU (Persamaan 7), YOLOv8 menggunakan fungsi aktivasi SiLU (*Sigmoid Linear Unit*) pada seluruh lapisan konvolusinya, yang dirumuskan sebagai:

$$SiLU(x) = x \cdot \sigma(x) \dots\dots\dots(13)$$

Di mana $\sigma(x)$ adalah fungsi sigmoid. SiLU dipilih oleh Ultralytics karena sifatnya yang *smooth* dan *non-monotonic*, yang terbukti memberikan performa lebih baik dibandingkan ReLU pada jaringan yang dalam, khususnya dalam mengurangi masalah *dying neuron* [cite].

SPPF (*Spatial Pyramid Pooling Fast*) – Diterapkan pada akhir *backbone* untuk mengagregasi fitur spasial dari berbagai *receptive field sizes*. SPPF menggunakan beberapa operasi *max pooling* berukuran 5×5 secara bertingkat yang kemudian dikonkatenasi:

$$SPPF(x) = \text{Concat}(x, \text{MaxPool}_5(x), \text{MaxPool}_9(x), \text{MaxPool}_{13}(x)), \dots (11)$$

Di mana:

- a. $\text{MaxPool}_k = \text{max pooling dengan kernel size } k \times k$
- b. Output SPPF memiliki 4× channel dari input x

SPPF memberikan kemampuan model untuk menangkap konteks spasial pada berbagai skala, meningkatkan deteksi objek dengan ukuran yang bervariasi [11].

2.8.2.2 Komponen Neck

Neck bertugas untuk menggabungkan fitur dari berbagai level resolusi yang dihasilkan *backbone* melalui mekanisme *Feature Pyramid Network* (FPN) dan *Path Aggregation Network* (PAN) [41]. YOLOv8 menggunakan kombinasi FPN dan PAN yang dioptimalkan dengan modul C2f untuk efisiensi komputasi.

Proses pada *Neck*:

A. Top-Down Pathway (FPN)

FPN bekerja dengan melakukan upsampling dari feature map resolusi rendah ke tinggi, kemudian menggabungkannya dengan feature map dari *backbone* melalui operasi *concatenation*:

$$P_l = \text{Upsample}(P_{l+1}) \oplus \text{Conv}_{1 \times 1}(C_l), \dots (13)$$

Di mana:

- a. $P_l = \text{feature pyramid level } l$.
- b. $C_l = \text{feature map dari backbone di level } l$.
- c. $\oplus = \text{operasi concatenation}$.
- d. $\text{Upsample} = \text{nearest neighbor upsampling}$.

B. Bottom-Up Pathway (PAN)

PAN memperkaya informasi lokasi spasial dengan mengalirkan informasi dari *feature map* resolusi tinggi ke rendah melalui *downsampling*:

$$N_l = \text{Conv}_{\text{stride}=2}(N_{l-1}) \oplus P_l, \dots (14)$$

Di mana:

- a. N_l = output PAN di level l .
- b. $\text{Conv}_{\text{stride}=2}$ = konvolusi dengan stride 2 untuk *downsampling*.

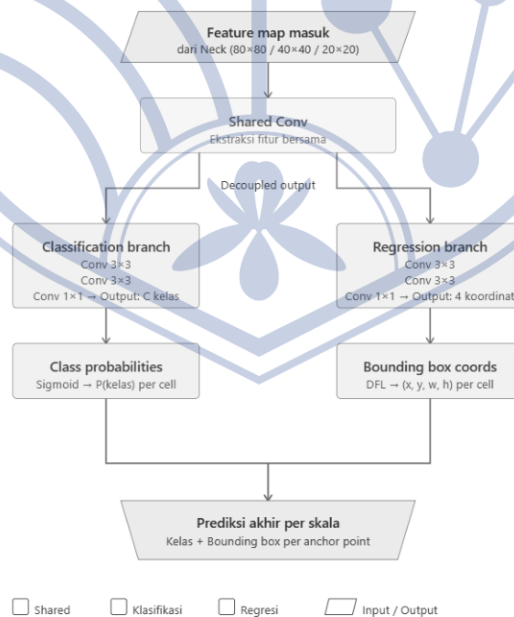
C. Fast PAN (FP-PAN)

YOLOv8 mengimplementasikan Fast PAN yang menghilangkan beberapa lapisan konvolusi redundan pada jalur *upsampling*, mengurangi beban komputasi tanpa mengorbankan akurasi secara signifikan [12]. Integrasi FPN dan PAN menghasilkan tiga *level feature map* yang digunakan untuk deteksi *multi-skala*:

- a. P3 (80×80) – untuk deteksi objek kecil
- b. P4 (40×40) – untuk deteksi objek sedang
- c. P5 (20×20) – untuk deteksi objek besar

2.8.2.3 Komponen Head (Decoupled Head)

Salah satu inovasi utama YOLOv8 adalah penggunaan *decoupled head* yang memisahkan proses klasifikasi dan *regresi bounding box* menjadi dua *branch* independen [10]. Berbeda dengan *coupled head* pada YOLOv5 yang menggunakan satu *branch* untuk kedua tugas, *decoupled head* memungkinkan model untuk belajar representasi yang lebih spesifik untuk masing-masing tugas. Berikut merupakan detail Decoupled Head yang menjadi salah satu perubahan kunci YOLOv8 dibanding pendahulunya.



Gambar 2.11 *Detail Decoupled Head*

Sumber: Diadaptasi dari [10], [11], [23]

Classification Branch – Memprediksi probabilitas kelas untuk setiap lokasi pada *feature map*:

$$P_{cls}(x, y, c) = \sigma(\text{Conv}_{cls}(\text{Conv}_{shared}(F))) \dots (15)$$

Di mana:

- a. P_{cls} = probabilitas kelas
- b. σ = fungsi *sigmoid*
- c. F = input *feature map*
- d. c = indeks kelas

Regression Branch – Memprediksi koordinat *bounding box* ($x, y, width, height$):

$$B_{reg}(x, y) = \text{Conv}_{reg}(\text{Conv}_{shared}(F)) \dots (16)$$

Keunggulan *decoupled head* adalah:

- a. Memungkinkan pembelajaran representasi yang lebih spesifik untuk klasifikasi dan regresi
 - b. Mengurangi konflik gradien antara dua tugas yang berbeda
 - c. Meningkatkan akurasi deteksi, terutama pada dataset dengan banyak kelas
- [10]

2.8.2.4 Pendekatan Anchor-Free Detection

Berbeda dengan YOLOv5 yang menggunakan *anchor boxes*, YOLOv8 menerapkan *anchor-free detection* yang memprediksi *bounding box* secara langsung dari titik tengah objek [11]. Pendekatan ini menghilangkan kebutuhan untuk mendefinisikan *anchor boxes* secara manual dan membuat model lebih fleksibel terhadap variasi ukuran dan rasio aspek objek.

Proses Prediksi *Anchor-Free*:

1. Untuk setiap lokasi (x, y) pada *feature map*, model memprediksi:
 - a. Distance ke keempat sisi *bounding box*: (l, t, r, b)
 - b. Probabilitas kelas: P_{cls}
2. Koordinat *bounding box* dihitung sebagai:

$$x_1 = x - l; x_2 = x + r$$

$$y_1 = y - t; y_2 = y + b$$

Di mana (x_1, y_1, x_2, y_2) adalah koordinat *bounding box* dalam format *xyxy*.

3. Untuk setiap prediksi, dihitung *objectness score* yang menggabungkan *classification confidence*:

$$\text{Score} = \sqrt{P_{cls} \times \text{IoU}_{pred}} \dots (17)$$

Anchor-free detection memberikan beberapa keuntungan:

- Menghilangkan hyperparameter terkait *anchor boxes*.
- Lebih mudah generalisasi ke objek dengan bentuk *arbitrary*.
- Mengurangi kompleksitas *post-processing*.
- Meningkatkan kecepatan inferensi.

2.8.3 Fungsi Loss YOLOv8

Untuk melatih model YOLOv8 agar menghasilkan prediksi yang akurat, digunakan kombinasi tiga fungsi *loss* yang masing-masing menangani aspek berbeda dari tugas deteksi objek [11].

2.8.3.1 Complete IoU (CIoU) Loss

CIoU Loss digunakan untuk mengevaluasi kualitas prediksi bounding box terhadap *ground truth* dengan mempertimbangkan tiga faktor: *overlap* area, jarak pusat, dan konsistensi rasio aspek [43]:

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v \dots (18)$$

Di mana:

- IoU = *Intersection over Union* antara prediksi dan *ground truth*.
- $\rho^2(b, b^{gt})$ = jarak Euclidean kuadrat antara pusat *bounding box* prediksi b dan *ground truth* b^{gt}
- c = panjang diagonal dari kotak terkecil yang mengelilingi kedua *bounding box*
- v = mengukur konsistensi rasio aspek

$$v = \frac{4}{\pi^2} \left(\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h} \right)^2 \dots (19)$$

- α = faktor penyeimbang positif

$$\alpha = \frac{v}{(1-IoU)+v} \dots (19)$$

CIoU Loss memberikan penalti yang lebih komprehensif dibandingkan *IoU* standar, menghasilkan prediksi *bounding box* yang lebih akurat terutama untuk objek dengan rasio aspek yang beragam.

2.8.3.2 Binary Cross-Entropy Loss untuk Klasifikasi

Untuk klasifikasi objek, YOLOv8 menggunakan *Binary Cross-Entropy (BCE) Loss* yang dihitung untuk setiap kelas secara independen [11]:

$$L_{cls} = - \sum_{i=1}^C [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \dots (21)$$

Di mana:

- a. C = jumlah kelas
- b. y_i = label ground truth untuk kelas ke- i (0 atau 1)
- c. $\hat{y}_i$ = probabilitas prediksi untuk kelas ke- i

Penggunaan BCE Loss memungkinkan model untuk melakukan *multi-label classification*, di mana satu objek dapat memiliki beberapa label secara bersamaan.

2.8.3.3 Distribution Focal Loss (DFL)

YOLOv8 memperkenalkan *Distribution Focal Loss* untuk regresi *bounding box* yang memodelkan distribusi probabilitas dari koordinat *bounding box* [10]:

$$L_{DFL} = - \sum_{i=0}^n ((y_{i+1} - y) \log(S_i) + (y - y_i) \log(S_{i+1})) \dots \dots \dots (22)$$

Di mana:

- a. y = nilai *ground truth*.
- b. y_i, y_{i+1} = batas diskretisasi terdekat.
- c. S_i, S_{i+1} = probabilitas *softmax*.

DFL memungkinkan model untuk mempelajari ketidakpastian dalam prediksi koordinat, menghasilkan *bounding box* yang lebih presisi.

2.8.3.4 Total Loss Function

Total loss untuk YOLOv8 adalah kombinasi *weighted* dari ketiga komponen *loss* di atas:

$$L_{total} = \lambda_{box} L_{CIoU} + \lambda_{cls} L_{cls} + \lambda_{dfl} L_{DFL} \dots \dots \dots (23)$$

Di mana λ_{box} , λ_{cls} , dan λ_{dfl} adalah *hyperparameter* yang mengatur kontribusi masing-masing *loss*. Dalam implementasi default YOLOv8, nilai-nilai ini disetel sebagai:

- a. $\lambda_{box} = 7.5$
- b. $\lambda_{cls} = 0.5$
- c. $\lambda_{dfl} = 1.5$

2.8.4 Metrik Evaluasi YOLOv8

Evaluasi performa model YOLOv8 tidak hanya mengandalkan metrik klasifikasi konvensional, tetapi juga metrik yang mempertimbangkan aspek spasial dan temporal deteksi objek [41]. Sebelum membahas mAP, perlu dijelaskan terlebih dahulu konsep dasar Intersection over Union (IoU), yang menjadi fondasi bagi hampir seluruh metrik evaluasi deteksi objek termasuk CIoU Loss (Sub-bab 2.8.3.1) dan mAP. IoU didefinisikan sebagai:

$$IoU = \frac{\text{Luas Irisan}}{\text{Luas Gabungan}} \dots \dots \dots (24)$$

Di mana Luas Irisan adalah area tumpang tindih antara bounding box prediksi dan ground truth, sedangkan Luas Gabungan adalah total area yang dicakup oleh kedua bounding box tersebut. Nilai IoU berkisar antara 0 (tidak ada tumpang tindih) hingga 1 (tumpang tindih sempurna), dan menjadi dasar penentuan apakah suatu prediksi dianggap benar (*true positive*) berdasarkan threshold tertentu, sebagaimana dijelaskan pada perhitungan mAP@0.5 dan mAP@0.5:0.95. Metrik utama yang digunakan adalah:

1. Mean Average Precision (mAP)

mAP merupakan metrik standar untuk evaluasi deteksi objek yang mengukur rata-rata presisi pada berbagai *threshold IoU* [11]. Terdapat dua varian utama:

- a) mAP@0.5 – Rata-rata presisi dengan *threshold IoU* = 0.5, artinya prediksi dianggap benar jika *overlap* dengan *ground truth* minimal 50%.
- b) mAP@0.5:0.95 – Rata-rata mAP pada *IoU threshold* dari 0.5 hingga 0.95 dengan interval 0.05, memberikan evaluasi yang lebih ketat terhadap kualitas lokalisasi.

Perhitungan mAP dilakukan dengan langkah-langkah berikut:

A. Untuk setiap kelas, hitung *Average Precision* (AP) berdasarkan kurva *Precision-Recall*:

$$AP = \int_0^1 P(R) dR \dots \dots \dots (25)$$

Di mana:

- a. $P = Precision$
- b. $R = Recall$

B. mAP dihitung sebagai rata-rata AP dari semua kelas:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \dots \dots \dots (26)$$

Di mana N adalah jumlah kelas.

2. Precision dan Recall

Precision mengukur ketepatan prediksi positif:

$$Precision = \frac{TP}{TP + FP} \dots \dots \dots (27)$$

Recall mengukur kemampuan model mendeteksi seluruh objek yang ada:

$$Recall = \frac{TP}{TP + FN} \dots \dots \dots (28)$$

Di mana:

- a. $TP = True Positive$ (deteksi benar)

- b. $FP = \text{False Positive}$ (deteksi salah)
- c. $FN = \text{False Negative}$ (objek tidak terdeteksi)

3. F1-Score

F1-Score adalah *harmonic mean* dari *Precision* dan *Recall*:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \dots\dots\dots (29)$$

4. Frames Per Second (FPS)

FPS mengukur kecepatan inferensi model, yaitu jumlah frame yang dapat diproses per detik. Metrik ini sangat penting untuk aplikasi dengan kebutuhan respons cepat:

$$\text{FPS} = \frac{1}{\text{Inference Time (seconds)}} \dots\dots\dots (30)$$

YOLOv8 dirancang untuk mencapai keseimbangan optimal antara akurasi (mAP) dan kecepatan (FPS), menjadikannya ideal untuk aplikasi deteksi objek dengan latensi rendah pada perangkat *mobile*.

2.8.5 Implementasi YOLOv8 dalam Deteksi Gestur BISINDO

Dalam konteks penelitian ini, YOLOv8 digunakan sebagai komponen deteksi spasial untuk mengidentifikasi dan melokalisasi gestur tangan BISINDO pada setiap *frame* video [11]. Proses integrasi YOLOv8 dalam sistem penerjemah BISINDO dilakukan melalui tahapan berikut:

1. Preprocessing Input – Citra dari kamera smartphone di-resize menjadi 640×640 piksel, dilakukan normalisasi nilai piksel ke rentang [0, 1], serta augmentasi data seperti *rotation*, *flip*, *brightness adjustment*.
2. Deteksi Gestur dengan YOLOv8 – YOLOv8 memproses *frame* input dan menghasilkan *bounding box* serta *class probabilities* untuk gestur BISINDO. *Non-Maximum Suppression* (NMS) diterapkan untuk menghilangkan deteksi duplikat dengan *IoU threshold* 0.45 dan *confidence threshold* 0.25.
3. Ekstraksi Fitur Temporal – Fitur spasial dari YOLOv8 (berupa *feature map* atau koordinat *landmark*) diekstraksi untuk setiap *frame*. Urutan fitur dari beberapa frame berurutan dijadikan input untuk model LSTM.
4. Klasifikasi Gestur dengan LSTM – LSTM memodelkan dependensi temporal antar-frame untuk mengklasifikasikan gestur BISINDO. Output LSTM berupa probabilitas kelas gestur yang diprediksi.

5. *Post-processing* dan *Output* – Gestur dengan probabilitas tertinggi dipilih sebagai hasil akhir. Hasil ditampilkan dalam bentuk teks pada antarmuka aplikasi *mobile*.

Kombinasi YOLOv8 untuk deteksi spasial dan LSTM untuk pemodelan temporal memungkinkan sistem untuk mengatasi tantangan dalam membedakan gestur BISINDO yang memiliki kemiripan visual namun berbeda makna berdasarkan pola urutan gerakan [21].

2.8.6 Keunggulan YOLOv8 untuk Aplikasi Mobile

Pemilihan YOLOv8 sebagai algoritma deteksi objek dalam penelitian ini didasarkan pada beberapa keunggulan yang relevan untuk aplikasi mobile penerjemah BISINDO [10], [11]:

1. Kecepatan Inferensi Tinggi – YOLOv8 mampu melakukan deteksi cepat dengan kecepatan hingga 60 FPS pada varian *Small*, memungkinkan pengalaman pengguna yang responsif.
2. Akurasi Kompetitif – Dengan $mAP@0.5:0.95$ sekitar 44.9% pada varian *Small*, YOLOv8 memberikan akurasi yang memadai untuk deteksi gestur tangan.
3. Efisiensi Model – Jumlah parameter yang relatif kecil (11.2M untuk YOLOv8s) memungkinkan *deployment* pada perangkat mobile dengan memori terbatas.
4. Fleksibilitas Arsitektur – Ketersediaan berbagai varian model memungkinkan penyesuaian sesuai dengan kebutuhan trade-off antara akurasi dan kecepatan.
5. Kemudahan Konversi ke TFLite – YOLOv8 dapat dengan mudah dikonversi ke format TensorFlow Lite (TFLite) untuk *deployment* pada platform Android.
6. Dukungan *Anchor-Free Detection* – Menghilangkan kebutuhan tuning anchor boxes, memudahkan adaptasi untuk deteksi gestur dengan variasi bentuk dan ukuran.
7. *Decoupled Head* – Memungkinkan pembelajaran yang lebih baik untuk klasifikasi dan lokalisasi secara terpisah, meningkatkan akurasi deteksi gestur.

Dengan keunggulan-keunggulan tersebut, YOLOv8 menjadi pilihan yang tepat sebagai komponen deteksi spasial dalam sistem penerjemah BISINDO berbasis *mobile* yang dikembangkan dalam penelitian ini.

2.8.7 Pemilihan Laju Pengambilan Frame (FPS) dalam Sistem Pengenalan Gestur

Salah satu parameter penting dalam perancangan sistem pengenalan gestur berbasis video adalah laju pengambilan *frame* (*frame rate*/FPS) pada tahap ekstraksi. Penelitian mengenai kejelasan (*intelligibility*) video bahasa isyarat pada perangkat *mobile*, yang dilakukan dalam kerangka proyek MobileASL di University of Washington, menemukan

bahwa percakapan bahasa isyarat mobile yang tetap dapat dipahami dapat terjadi pada laju serendah 10 FPS [59], [60]. Studi lanjutan yang lebih rigor mengevaluasi persepsi pengguna pada frame rate 1, 5, 10, dan 15 FPS, serta menemukan adanya *intelligibility ceiling effect*, yaitu kondisi ketika peningkatan frame rate di atas 10 FPS tidak lagi meningkatkan kejelasan gestur yang dipersepsikan oleh pengamat [61]. Temuan ini menunjukkan bahwa 10 FPS bukan sekadar nilai default, melainkan titik jenuh (*ceiling*) yang telah divalidasi secara empiris khusus untuk konten bahasa isyarat, sehingga pengambilan sampel pada laju tersebut telah memadai untuk merepresentasikan informasi gestur secara utuh.

Dari sisi akurasi model pengenalan gestur berbasis pembelajaran mendalam, penelitian pada sistem pengenalan bahasa isyarat di lingkungan mobile mengevaluasi performa model pada frame rate 5, 10, 15, 20, dan 25 FPS. Hasil penelitian tersebut menunjukkan bahwa frame rate yang terlalu rendah (5 FPS) secara signifikan menurunkan akurasi karena jumlah frame yang terlalu sedikit untuk merekam dinamika gestur secara memadai, sementara peningkatan frame rate melampaui rentang 10–15 FPS memberikan kenaikan akurasi yang semakin mengecil (*diminishing returns*) dibandingkan dengan tambahan beban komputasi yang ditimbulkan [62]. Berdasarkan kedua kelompok literatur tersebut — kejelasan persepsi bahasa isyarat dan akurasi model pengenalan gestur — laju 10 FPS dipilih sebagai laju ekstraksi standar dalam penelitian ini karena merepresentasikan titik keseimbangan optimal antara kecukupan informasi temporal, akurasi model, dan efisiensi komputasi pada perangkat mobile kelas menengah, sebagaimana diuraikan lebih lanjut pada pembahasan teknis di Sub-bab 3.2.1.

2.9 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) merupakan salah satu varian dari *Recurrent Neural Network* (RNN) yang dikembangkan oleh Hochreiter dan Schmidhuber pada tahun 1997 untuk mengatasi permasalahan *vanishing gradient* dan *exploding gradient* yang sering terjadi pada RNN standar [43]. Masalah *vanishing gradient* menyebabkan RNN konvensional kesulitan dalam mempelajari dependensi jangka panjang (*long-term dependencies*) pada data sekuensial, karena gradien yang digunakan dalam proses *backpropagation through time* (BPTT) cenderung menghilang atau meledak seiring bertambahnya panjang sekuens [14].

LSTM mengatasi permasalahan ini melalui mekanisme *gating* yang mengontrol aliran informasi di dalam jaringan. Arsitektur LSTM terdiri dari sebuah sel memori (*memory cell*) dan tiga buah *gate* (gerbang) utama yang mengatur informasi apa yang disimpan, dibuang, atau diteruskan [43]:

1. Forget Gate (Gerbang Lupa)

Forget gate berfungsi untuk menentukan informasi mana dari *cell state* sebelumnya yang harus dibuang atau dilupakan. *Gate* ini menerima input berupa *hidden state* sebelumnya (h_{t-1}) dan input saat ini (x_t), kemudian menghasilkan nilai antara 0 dan 1 untuk setiap elemen dalam *cell state* melalui fungsi aktivasi sigmoid [43]:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \dots \dots \dots (31)$$

Di mana:

- a. f_t = output *forget gate*
- b. σ = fungsi aktivasi sigmoid
- c. W_f = matriks bobot *forget gate*
- d. h_{t-1} = *hidden state* pada langkah waktu sebelumnya
- e. x_t = input pada langkah waktu saat ini
- f. b_f = bias *forget gate*

Nilai 0 berarti "lupakan sepenuhnya" dan nilai 1 berarti "simpan sepenuhnya".

2. Input Gate (Gerbang Masukan)

Input gate berfungsi untuk menentukan informasi baru mana yang akan disimpan ke dalam *cell state*. Proses ini melibatkan dua komponen [13]:

- a. Sigmoid layer yang menentukan nilai mana yang akan diperbarui:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \dots \dots \dots (32)$$

- b. Tanh layer yang membuat vektor kandidat nilai baru yang dapat ditambahkan ke *cell state*:

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \dots \dots \dots (33)$$

Selanjutnya, *cell state* diperbarui dengan menggabungkan informasi dari *forget gate* dan *input gate*:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \dots \dots \dots (34)$$

Di mana:

- a. C_t = *cell state* pada langkah waktu saat ini
- b. C_{t-1} = *cell state* pada langkah waktu sebelumnya

3. Output Gate (Gerbang Keluaran)

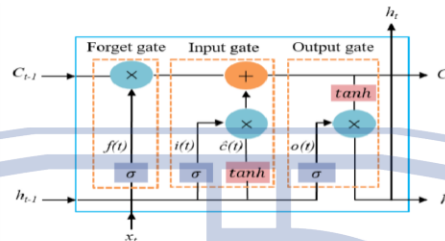
Output gate berfungsi untuk menentukan bagian mana dari *cell state* yang akan menjadi output pada langkah waktu saat ini [43]:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \dots \dots \dots (35)$$

$$h_t = o_t * \tanh(C_t) \dots \dots \dots (36)$$

Di mana:

- a. o_t = output dari *output gate*
- b. h_t = *hidden state* pada langkah waktu saat ini yang juga berfungsi sebagai output LSTM



Gambar 2.12 Arsitektur LSTM [14]

Mekanisme *gating* pada LSTM memungkinkan jaringan untuk secara selektif menyimpan, memperbarui, dan mengakses informasi yang relevan dari langkah-langkah waktu sebelumnya, sehingga mampu menangkap dependensi jangka panjang dalam data sekuensial [14]. Kemampuan ini membuat LSTM sangat cocok untuk berbagai tugas yang melibatkan data berurutan, seperti pemrosesan bahasa alami, pengenalan suara, prediksi deret waktu, dan pengenalan gestur [15].

Dalam konteks pengenalan bahasa isyarat, LSTM memainkan peran penting dalam memodelkan aspek temporal dari gestur. Bahasa isyarat, termasuk BISINDO, merupakan bahasa yang bersifat dinamis, di mana makna suatu gestur tidak hanya ditentukan oleh posisi tangan pada satu titik waktu tertentu, tetapi juga oleh urutan gerakan dari waktu ke waktu [46]. Beberapa gestur BISINDO yang memiliki konfigurasi tangan serupa dapat dibedakan berdasarkan arah, kecepatan, atau pola gerakan yang hanya dapat ditangkap melalui analisis temporal [47].

Dalam penelitian ini, LSTM digunakan sebagai komponen temporal yang menerima fitur-fitur spasial yang diekstraksi dari setiap frame video (melalui YOLOv8 atau MediaPipe) dan memprosesnya secara berurutan untuk mengklasifikasikan gestur BISINDO. Dengan kemampuannya dalam menangkap dependensi jangka panjang, LSTM diharapkan mampu membedakan gestur-gestur BISINDO yang memiliki kemiripan visual namun berbeda makna, berdasarkan pola urutan gerakan yang berbeda [21].

Beberapa keunggulan LSTM dalam pengenalan gestur bahasa isyarat meliputi [20], :

- a. Mampu menangkap hubungan temporal antar-frame dalam rangkaian video gestur.
- b. Mengatasi masalah *vanishing gradient* yang membatasi kemampuan RNN standar dalam mempelajari pola gerakan yang panjang.

- c. Mampu menyimpan informasi kontekstual dari gerakan sebelumnya yang relevan untuk mengklasifikasikan gestur saat ini.
- d. Fleksibel dalam menangani sekuens input dengan panjang yang bervariasi.

Model LSTM dalam penelitian ini dikonversi ke format TFLite (*TensorFlow Lite*) agar dapat berjalan secara efisien pada perangkat mobile Android dengan sumber daya komputasi yang terbatas.

2.10 User Acceptance Testing (UAT)

User Acceptance Testing (UAT) merupakan tahap pengujian akhir dalam siklus pengembangan perangkat lunak yang dilakukan untuk memvalidasi apakah sistem yang dikembangkan telah memenuhi kebutuhan pengguna dan siap untuk diimplementasikan [48]. UAT berbeda dengan pengujian teknis seperti unit testing atau integration testing, karena fokus utamanya adalah pada perspektif pengguna akhir (end-user) terhadap fungsionalitas, kemudahan penggunaan (usability), dan kesesuaian sistem dengan kebutuhan bisnis atau operasional [49].

Tujuan utama dari UAT adalah untuk memastikan bahwa sistem yang dikembangkan dapat diterima oleh pengguna dan mampu menjalankan fungsinya dalam kondisi penggunaan nyata [50]. Dalam konteks pengembangan aplikasi mobile, UAT menjadi sangat penting karena berkaitan langsung dengan pengalaman pengguna (*user experience*) yang menjadi faktor kunci keberhasilan aplikasi [51].

2.10.1 Tahapan User Acceptance Testing

Proses UAT umumnya melibatkan beberapa tahapan yang sistematis untuk memastikan pengujian dilakukan secara komprehensif [49]. Tahapan-tahapan tersebut dapat dilihat pada Tabel 2.4 berikut:

Tabel 2.4 Tahapan User Acceptance Testing [49], [50]

No	Tahapan	Deskripsi	Aktivitas Utama
1	Perencanaan UAT (<i>UAT Planning</i>)	Tahap persiapan yang mencakup identifikasi kriteria penerimaan dan pemilihan responden	a. Menentukan kriteria penerimaan b. Memilih responden yang representative c. Menyusun jadwal pengujian

2	Perancangan <i>Test Case</i>	Merancang skenario pengujian berdasarkan penggunaan nyata sistem	a. Membuat skenario penggunaan b. Menentukan expected result c. Menyusun kuesioner evaluasi
3	Pelaksanaan Pengujian	Responden melakukan pengujian sistem sesuai test case yang dirancang	a. Eksekusi test case b. Observasi langsung c. Dokumentasi masalah
4	Pengumpulan Feedback	Mengumpulkan masukan dan penilaian dari responden	a. Distribusi kuesioner b. Wawancara pengguna c. Pencatatan observasi
5	Analisis dan Evaluasi	Menganalisis data untuk mengidentifikasi area perbaikan	a. Analisis hasil kuesioner b. Identifikasi masalah c. Rekomendasi perbaikan

2.10.2 Metode Pengukuran dalam UAT

Metode pengukuran dalam UAT dapat menggunakan berbagai pendekatan kuantitatif dan kualitatif untuk mengevaluasi penerimaan pengguna terhadap sistem [52]. Tabel 2.5 menunjukkan berbagai metode yang umum digunakan:

Tabel 2.5 Metode Pengukuran *User Acceptance Testing* [51], [52], [53].

No.	Metode	Deskripsi	Skala Pengukuran	Interpretasi
1.	Skala Likert	Metode pengukuran tingkat persetujuan atau kepuasan responden	1-5 atau 1-7	1 = Sangat Tidak Setuju <5 = Sangat Setuju
2.	System Usability Scale (SUS)	Kuesioner standar 10 pertanyaan untuk mengukur usability	Skor 0-100	>80 = Excellent 68-80 = Good <51 = Poor

3.	Task Success Rate	Persentase tugas yang berhasil diselesaikan	0-100%	Semakin tinggi semakin baik
4.	Time on Task	Waktu yang diperlukan untuk menyelesaikan tugas	Waktu (detik/menit)	Semakin cepat semakin efisien
5.	Error Rate	Jumlah kesalahan yang terjadi selama pengujian	Jumlah error	Semakin rendah semakin baik
6.	Satisfaction Rating	Tingkat kepuasan pengguna secara keseluruhan	1-5 atau 1-10	Semakin tinggi semakin puas

2.10.3 Aspek Evaluasi dalam UAT

Dalam pelaksanaan UAT, terdapat beberapa aspek utama yang perlu dievaluasi untuk memastikan sistem memenuhi kebutuhan pengguna [50]. Aspek-aspek tersebut dapat dilihat pada Tabel 2.6:

Tabel 2.6 Aspek Evaluasi User Acceptance Testing [50], [51], [55]

No.	Aspek	Indikator Penilaian	Metode Pengukuran
1.	Functionality (Fungsionalitas)	a. Kelengkapan fitur b. Kesesuaian dengan kebutuhan c. Akurasi hasil	a. Task Success Rate b. Error Rate c. Skala Likert
2.	Usability (Kemudahan Penggunaan)	a. Kemudahan dipelajari b. Efisiensi penggunaan c. Kemudahan navigasi	a. Time on Task b. Skala Likert
3.	Reliability (Keandalan)	a. Konsistensi hasil b. Stabilitas sistem c. Penanganan error	a. Error Rate b. Skala Likert
4.	Performance (Kinerja)	a. Kecepatan respon b. Waktu pemrosesan c. Konsumsi resource	a. Time on Task b. Observasi langsung
5.	User Satisfaction (Kepuasan Pengguna)	a. Kepuasan keseluruhan	a. Satisfaction Rating

		b. Keinginan menggunakan kembali c. Rekomendasi kepada orang lain	b. Skala Likert
--	--	--	-----------------

2.10.4 Penerapan UAT dalam Penelitian

Dalam konteks penelitian ini, UAT dilakukan untuk mengevaluasi aplikasi mobile penerjemah BISINDO ke teks dengan melibatkan responden dari SLB-B Karya Murni di Medan. Pengujian ini bertujuan untuk:

1. Validasi Akurasi Pengenalan Gestur: Memvalidasi apakah gestur BISINDO yang dikenali oleh sistem sesuai dengan makna yang dimaksud oleh pengguna dari komunitas Tuli [56].
2. Evaluasi Usability: Mengukur tingkat kemudahan penggunaan aplikasi dari perspektif pengguna menggunakan metode SUS dan skala Likert [53].
3. Pengukuran Kepuasan Pengguna: Mengevaluasi tingkat kepuasan pengguna terhadap fitur dan fungsionalitas aplikasi secara keseluruhan [51].
4. Identifikasi Permasalahan: Mengidentifikasi permasalahan atau kekurangan yang perlu diperbaiki sebelum implementasi lebih luas melalui feedback kualitatif dan kuantitatif [50].

Kriteria penerimaan sistem dalam penelitian ini ditetapkan berdasarkan standar usability yang umum digunakan, yaitu minimal skor SUS 68 (kategori "Good") dan tingkat akurasi pengenalan gestur minimal 80% untuk dapat dinyatakan layak diimplementasikan [53], [54].

Hasil dari UAT akan digunakan sebagai dasar untuk melakukan perbaikan dan penyempurnaan aplikasi, serta untuk menilai apakah aplikasi telah siap untuk digunakan oleh komunitas Tuli sebagai alat bantu komunikasi dengan masyarakat umum [55]. Dengan melibatkan pengguna sesungguhnya dalam proses pengujian, UAT memberikan validasi empiris bahwa sistem yang dikembangkan tidak hanya berfungsi secara teknis, tetapi juga memberikan nilai guna (utility) dan dapat diterima oleh pengguna akhir [49].