Indonesian Sign Language (BISINDO) Alphabet Detection Through a Mobile-Based Approach Based on YOLO Algorithm Version 11

Jeremya Dharmawan Raharjo
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
jeremyadraharjo@gmail.com

Rinaldi Munir
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
rinaldi@staff.stei.itb.ac.id

***Abstract*—Individuals who are Deaf or hard of hearing depend significantly on sign language, a visual mode of communication that employs bodily movements, facial emotions, and hand gestures. Advances in deep-learning methodologies offer substantial potential for automating the recognition of Bahasa Isyarat Indonesia (BISINDO), thereby enhancing the language's accessibility to a broader range of users. This paper proposes a BISINDO alphabet recognition system based on You Only Look Once (YOLO), a state-of-the-art real-time object detection model, with the latest YOLOv11 version used to recognize hand gestures corresponding to each alphabet letter. The suggested system uses an effective and portable detection architecture that minimizes latency while preserving high precision in mobile environment. Comparative training of several YOLO variants yielded a peak mean average precision (mAP) of 82.3 % and a minimum latency of 10 frames per second, demonstrating the system's viability for real-time BISINDO recognition on mobile platforms.**

***Keywords*—BISINDO, YOLOv11, Real-Time Object Detection, Alphabet Recognition, Mobile Environment**

I. Introduction

Sign language conveys meaning visually through coordinated hand shapes, bodily movement, body posture, and facial expression. It serves as an effective communication medium for individuals who are deaf or hard of hearing, enabling them to share their ideas and engage with others [1]. The World Health Organization reported in 2024 that approximately 430 million people worldwide—about 34 million of them children—experience hearing loss, and this figure is projected to rise to over 700 million, or roughly one in ten people, by 2050 [2]. The development of precise and extensively used sign-language recognition systems that close communication gaps and promote inclusion has also emerged as a key research priority in the current period due to the rapid advancement of information technology.

Deaf populations in Indonesia employ Bahasa Isyarat Indonesia (BISINDO), a naturally developed two-handed signing system that is now only maintained by one official training provider, the Indonesian Sign Language Center (Pusbisindo). Although BISINDO is expressive and community-driven, its range of vocabulary, nuanced gestures, and regional variation mean that relatively few people, aside from specially trained interpreters, can use it fluently. This shortage of proficient signers limits Deaf Indonesians' access to public events and everyday interaction [3].

Modern computer vision drives applications in medicine, sports, media, and autonomous systems, achieving robust image classification, detection, localization, and identification through deep convolutional neural networks [4]. Building on these advances, this study introduces a real-time BISINDO alphabet-detection system built on the latest YOLOv11 deep learning architecture. The purpose of this study is to achieve real-time BISINDO alphabet recognition on mobile device environment while delivering high precision and low latency.

The remainder of this paper is structured as follows: Section II reviews prior research pertinent to the study; Section III details the proposed method; Section IV presents the experimental results and their analysis; and Section V presents the concluding remarks.

II. Related Work

Treating ASL (American Sign Language) recognition as an object-detection task, Imran et al. (2024) [5] proposed a real-time detection system using the YOLOv9 architecture. Their approach integrates Programmable Gradient Information (PGI) and a Generalized Efficient Layer Aggregation Network (GELAN) to maintain feature integrity. While the YOLOv9e variant achieved a mean average precision of 69.38% at IoU thresholds 0.5–0.95 (mAP@0.5:0.95), this performance came with a significant trade-off. The inference latency was impractically high for real-time use, averaging approximately 1300-2700 ms per frame.

Focusing on a different application of pose estimation, Arlin et al. (2023) [6] developed an application for counting push-up movements. The system employed the MediaPipe library for pose estimation and a K-Nearest Neighbors (KNN) algorithm for classification. Across 176 test cases involving both direct user interaction and video analysis, the system achieved an overall accuracy of 84.7%. Although effective for a repetitive action, the study demonstrates a simpler classification task and does not address the complexities of recognizing diverse sign language gestures.

Peling et al. (2024) [7] developed an ASL detection system for personal computers using MediaPipe for hand pose detection and TensorFlow Lite for classification. The model achieved 81% accuracy on a 10-word dataset and 73% on a 26-letter alphabet dataset. The authors reported technical challenges, including webcam stability issues with larger datasets and the need for more adequate training data to improve model performance, indicating limitations in its robustness and scalability.

III. Proposed Methodology

A. *Sign-Language Model Development Workflow*

In Figure 1, we present the entire proposed-model development pipeline. The first step is to collect the dataset, which consists of images of BISINDO alphabet signs. Next, preprocessing in the dataset is performed, by annotating the images with bounding boxes and image augmentation for producing various images from the original dataset.

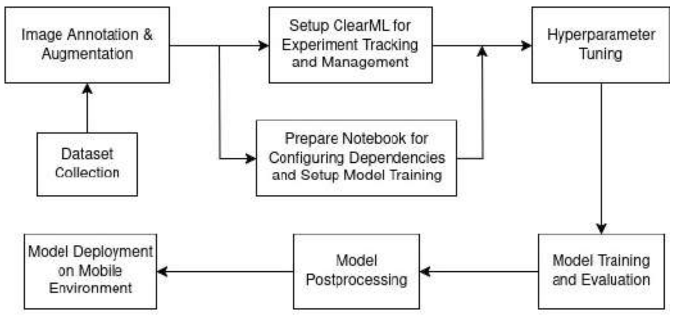
Image Annotation & Augmentation
Setup ClearML for Experiment Tracking and Management
Hyperparameter Tuning
Dataset Collection
Prepare Notebook for Configuring Dependencies and Setup Model Training
Model Deployment on Mobile Environment
Model Postprocessing
Model Training and Evaluation

Figure 1. Proposed model pipeline development

Afterward, we prepare the development notebook by installing the required libraries and packages. For experiment tracking and management, we rely on ClearML, a machine learning operations oriented platform that streamlines model versioning, deployment, and performance monitoring [8]. The hyperparameter tuning is performed when configuration of the training environment is ready. This step is to determine the right hyperparameters for each model, because the model behaves differently with different hyperparameters on the datasets.

Next, we train the model using the YOLOv11 architecture, and evaluate its performance on the validation and test datasets. After that, we perform postprocessing on the trained model by applying quantization and adjustment in the input size. Finally, we deploy the model to a mobile device and evaluate its performance in real-time BISINDO alphabet detection.

B. *YOLO (You Only Look Once) Version 11 Object Detector*

The YOLO (You Only Look Once) framework is a cutting-edge approach to real-time object detection. It recasts detection as a single regression job that simultaneously predicts

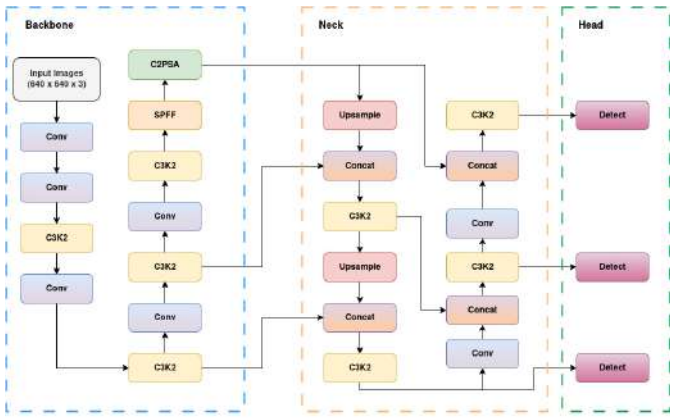

Figure 2. YOLOv11 architecture.

bounding-box coordinates and class probabilities, in contrast to previous classifier-based pipelines. The entire image is processed by a single network in a single pass, allowing for end-to-end performance optimization [9]. Many improvements have been made to YOLO since its inception, the most recent being YOLOv11. Figure 2 illustrates the three primary components of the YOLOv11 model: the head, neck, and backbone. The head generates the final predictions, the neck accumulates and refines these characteristics across scales, and the backbone extracts key feature representations.

In terms of module implementation, the YOLOv11 architecture offers a number of improvements over its predecessors [10]. To extend YOLOv8's capabilities, the standard C2f block is replaced with a custom C3K2 CSP bottleneck that breaks the original large convolution into two smaller kernels, reducing compute and memory demands and thus accelerating inference while maintaining expressive power. Moreover, the effective SPPF pooling layer is kept in place to expand the receptive field; however, a recently added C2PSA module takes its place. This module sharpens feature maps for downstream detection by fusing channel-wise and spatial-wise information via a multi-head attention mechanism. Lastly, YOLOv11 uses an adaptive anchor-frame approach that automatically adjusts anchor sizes and aspect ratios based on real-time dataset characteristics. This work used three variants of the YOLOv11 model: nano (n), small (s), and medium (m), to evaluate the trade-offs among performance metrics, latency, and model complexity under different resource situations.

C. *Datasets*

There are 26 classes in the dataset used in this experiment, one for each static letter of the BISINDO alphabet. The data was collected from a variety of sources, including frames in YouTube videos, open-source photos on Roboflow Universe, and Pusbisindo community café sessions. Each image was manually annotated with bounding boxes, and the dataset was further expanded through targeted augmentations to oversample minority classes. The augmentation strategy include rotation, shearing, and noise injection, which improve robustness of the model against variations in the input data, especially handle for noisy input and the smaller object detection [11]. Our dataset comprises 2912 images with equal distribution across the 26 classes, ensuring a balanced representation of each letter. This dataset is split into training, validation, and test sets, with 80% for training, 10% for validation, and 10% for testing.

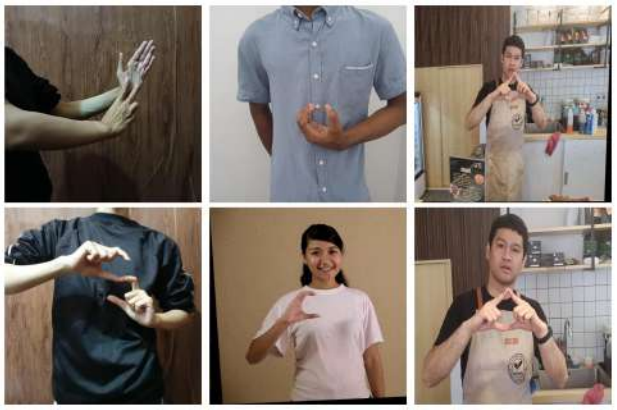
Figure 3. Example of dataset.

D. Hyperparameter Tuning Approach

Training the neural networks is also dependent on unchangeable parameters called hyperparameters. These hyperparameters are set before the main training process and remain constant during the training [12]. Determining appropriate hyperparameters is crucial for achieving optimal performance of the model. Ultralytics YOLO provides hyperparameter tuning by utilizing genetic algorithm, resulting an increase in mAP@0.5 by 11.8 % for 700 iterations in COCO dataset [13]. Tuning the hyperparameters requires a lot of resources and time, therefore, it is important to provide a systematic way to tuning the hyperparameters in this experiment. This experiment will utilize an initial random search instead of genetic algorithm to find the best hyperparameters.

E. Model Quantization

Post-training quantization reduces the memory footprint and speeds up inference without retraining in deep learning by converting models that were initially trained with 32-bit floating-point weights into lower-precision forms. In order to reduce multiply-accumulate operations and cut model size, this compression typically uses 16-bit floating-point (FP16) or 8-bit integer (INT8) formats. This allows the network to operate on hardware with significantly smaller memory and processing power budgets. Such FP16/INT8 post-training quantization, when applied to YOLO models using TensorFlow Lite format [14], reduces latency and energy consumption while maintaining nearly the same accuracy as the original FP32 network, allowing for real-time inference on edge devices [15].

F. Hardware and Computing Resources

All model training and hyper-parameter searches were carried out on Google Colab, which provides a virtualized Intel Xeon CPU, 13 GB RAM, and an NVIDIA Tesla T4 GPU (16 GB VRAM). Therefore, real-time inference was deployed and evaluated on a Vivo V50 Lite device powered by a MediaTek Dimensity 6300 CPU, 16 GB RAM, and a Mali-G57 MC2 GPU. Deploying the model in TensorFlow Lite format provides a lightweight runtime environment that supports quantization, enabling efficient inference on edge and mobile devices [14].

IV. Experiment and Analysis

Due to limitations in the dataset, specifically the number of images available for each class, as well as constraints on computing resources, the experiments will be conducted at both the module level and the architecture level. This approach aims to ensure a thorough evaluation of the model while maintaining feasible computational costs.

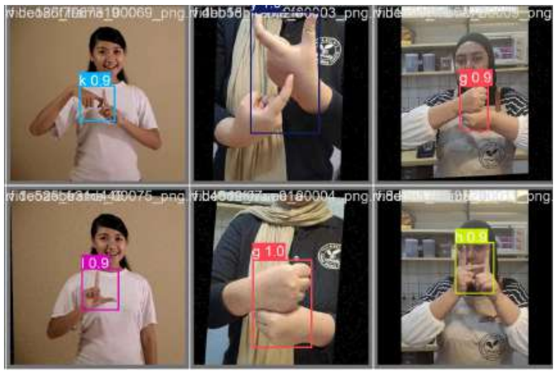

Figure 4. Sample images and detection results from YOLOv11 model inference on validation set.

To evaluate the performance of object detection models, several key metrics are employed to measure prediction quality and generalization. These metrics include Precision, Recall, F1-score, mean Average Precision (mAP), and training loss. In contrast, accuracy is unsuitable metric for object detection task because the true negatives are undefined, making it misleading due to the large number of uncounted background regions [15].

a) Precision: Precision quantifies the proportion of correctly predicted positive samples (true positives) out of all predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

TP means True Positive, and FP means False Positive. A higher precision indicates fewer false positives in predictions.

b) Recall: Recall measures the proportion of actual positive samples that are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

TP means True Positive, and FN means False Negative. A higher recall means the model is able to detect most of the actual objects.

c) F1-score: The F1-score is the harmonic mean of precision and recall, providing a balanced metric between the two:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

This metric is useful when there is an uneven class distribution.

d) Mean Average Precision (mAP): mAP measures the average precision across all classes and serves as a comprehensive indicator of object detection performance. It is calculated based on the precision-recall curve for each class. In this study, mAP is evaluated using two thresholds: IoU@0.5 and IoU@0.5:0.95, where IoU (Intersection over Union) quantifies the overlap between the predicted bounding box and the ground truth bounding box:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{4}$$

A higher IoU indicates a more accurate prediction. The metric mAP@0.5 considers a prediction correct if IoU exceeds 0.5, while mAP@0.5:0.95 averages the mAP scores over multiple IoU thresholds from 0.5 to 0.95 with a step of 0.05:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{5}$$

where AP_i is the Average Precision for class i, and N is the number of object classes.

e) Loss Function:

$$\mathcal{L}_{total} = \mathcal{L}_{cls} + \mathcal{L}_{loc} + \mathcal{L}_{dfl} \tag{6}$$

The overall loss function, composed of three fundamental components, serves both to guide model parameter optimization and to monitor training convergence: $\mathcal{L}_{cls}$, the class probability loss based on cross-entropy that quantifies the divergence between predicted and ground truth class probabilities to improve classification accuracy; $\mathcal{L}_{box}$, the bounding box regression loss that enhances localization precision by minimizing the discrepancy between predicted and actual bounding boxes using Intersection over Union (IoU); and $\mathcal{L}_{dfl}$, distributed focal loss that enables the model to identify and prioritize difficult samples, thus improving detection robustness in challenging scenarios [10].

A. Hyperparameter Tuning Experiments and Results

In order to achieve optimal model performance, several hyperparameters are adjusted first before the main training experiment. The custom hyperparameters to be tuned include initial learning rate (lr0), final learning rate factor (lrf), and batch size. The evaluation metric based on the mAP@0.5:0.95, as a standard evaluation metric for YOLO model performance for COCO dataset, in validation set is used to determine the best custom hyperparameters. To limit computational resources, the custom hyperparameter search is conducted over 15 epochs before proceeding to the full training stage. Each model variant (n, s, m) is trained with different combinations of custom hyperparameters, and the results are summarized in Table I.

Table I
EXPERIMENT RESULTS ON HYPERPARAMETER TUNING

Model	lr0	lrf	Batch	mAP@0.5:0.95 (%)
YOLOv11n	0.01*	0.01*	32*	79.8
	0.01	0.01	8	78.2
	0.005	0.1	16	78.7
	0.001	0.01	8	78.2
YOLOv11s	0.01*	0.01*	16*	81.2
	0.01	0.01	8	80.2
	0.005	0.1	16	80.5
	0.001	0.01	8	80.8
YOLOv11m	0.01	0.01	8	79.6
	0.01*	0.01*	16*	80.9
	0.005	0.1	16	80.0
	0.001	0.01	8	79.6

* Hyperparameters selected for the main training experiment.

B. Model Training and Evaluation

Table II
MAIN HYPERPARAMETERS FOR TRAINING YOLOV11 MODELS

Hyperparameter	Value
Maximum Epochs	200
Optimizer	AdamW
Patience	50
IoU Threshold	0.7
Image Size	640
Momentum	0.937

The main training experiment utilize custom hyperparameter that selected from the prior hyperparameter tuning process. The selected best-performing hyperparameters, listed in Table I, were subsequently used in combination with a fixed set of common training hyperparameters [16], as listed in Table II. Other common hyperparameters not listed in Table II are set to their default configurations. This experiment experiment uses the validation set to monitor per-epoch metrics, with final evaluation performed on the test set.

Table III
EVALUATION METRICS FOR MAIN TRAINING EXPERIMENT ON THREE YOLOV11 MODEL VARIANTS

Metrics	YOLOv11n	YOLOv11s	YOLOv11m
Validation-Precision (%)	98.3	98.2	97.0
Validation-Recall (%)	98.4	99.0	99.1
Validation-F1-Score (%)	98.3	98.6	98.0
Validation-mAP@0.5:0.95 (%)	82.7	**83.1**	83.0
Validation-mAP@0.5 (%)	99.0	99.2	99.3
Test-Precision (%)	97.77	98.14	99.2
Test-Recall (%)	98.40	98.58	99.3
Test-F1-Score (%)	98.1	98.4	99.2
Test-mAP@0.5:0.95 (%)	81.7	81.5	**82.3**
Test-mAP@0.5 (%)	99.5	99.5	99.5
Size (MB)	5.3	18.3	38.7
Loss	1.344	1.813	1.505

As shown in Table III, the YOLOv11s variant achieves the highest validation performance with a mAP@0.5:0.95 of 83.1%. YOLOv11m exhibits the best generalization capabilities on the test set, achieving a mAP@0.5:0.95 of 82.3%, suggesting greater detection performance on unseen samples.

C. Model Postprocessing

Postprocessing techniques are applied to the trained models primarily to reduce model size and accelerate inference speed. In this experiment, two main postprocessing methods are used: model quantization and input image resolution resizing. The original input images captured from the camera have a native resolution of 640×480 pixels, which does not match the square input shape expected by the YOLOv11 model. Therefore, before being passed into the model, each image undergoes a preprocessing step that involves center-cropping to a square aspect ratio (e.g., 480×480), followed by resizing to the target input dimensions. Thus, each square-cropped frame is resized to correspond with the YOLOv11 model's *Image Size* parameter (224, 352, or 480) before to inputting into the network.

Table IV
PERFORMANCE METRICS OF YOLOV11 VARIANTS ACROSS RESOLUTIONS AND MODEL FORMATS.

Model	**Metric (%)**	float16			int8		
		224	352	480	224	352	480
YOLOv11n	mAP@0.5	95.6	99.5	99.5	89.8	99.5	99.4
	mAP@0.5:0.95	66.5	79.7	81.2	60.0	77.7	79.5
	Precision	94.3	98.9	98.7	89.5	98.1	98.0
	Recall	86.3	99.0	99.1	80.0	98.7	98.8
	F1	89.4	98.9	98.9	83.2	98.3	98.4
	Size(MB)	5.1	5.2	5.3	2.7	2.7	2.9
YOLOv11s	mAP@0.5	96.9	99.5	99.5	95.2	99.5	99.4
	mAP@0.5:0.95	68.9	79.8	80.5	64.1	76.7	78.3
	Precision	95.0	98.6	98.6	89.2	98.4	98.6
	Recall	91.7	99.2	99.3	91.3	99.1	99.5
	F1	93.0	98.9	99.0	89.6	98.7	99.1
	Size(MB)	18.1	18.1	18.5	9.2	9.2	9.6
YOLOv11m	mAP@0.5	97.2	99.5	99.5	95.3	99.3	99.5
	mAP@0.5:0.95	71.0	80.7	81.2	66.5	77.6	78.9
	Precision	93.8	98.9	99.0	94.0	98.9	98.5
	Recall	93.4	99.1	99.1	88.0	99.1	99.2
	F1	93.2	99.0	99.1	89.9	99.0	98.8
	Size(MB)	38.4	38.6	38.5	19.4	19.6	19.5

Table IV shows that applying quantization (from float32 to int8) effectively reduces the model size of YOLOv11 variants by approximately 50%, which significantly benefits deployment scenarios with limited storage or computational resources. The performance that indicated by evaluation metrics shows only minor degradation at higher image resolutions (352 and 480 pixels) despite the notable reduction in model size. At the lower resolution of 224 pixels, the YOLOv11 models experience a noticeable drop in performance, particularly in mAP@0.5:0.95, which is not intended for use in the real-time experiment.

D. Real Time Experiment

To evaluate the practical deployment capabilities of our BISINDO alphabet detection system, we conducted comprehensive real-time performance experiments on mobile devices utilizing GPU acceleration through TensorFlow Lite [17]. The experiments focused on assessing the trade-offs between model complexity, quantization techniques, and inference performance across different input resolutions, leveraging the GPU capabilities mentioned in the previous section to optimize inference in mobile environment.

Table V
LOAD AND INFERENCE TIME ACROSS QUANTIZATION FORMATS AND INPUT SIZES

Model	**Quantization**	**Input Size**	**Load Time (s)**	**Frame per Second (FPS)**
YOLOv11n	float16	352	8.1	10.2
		480	13.5	5.2
	int8	352	6.9	10.2
		480	12.7	5.2
YOLOv11s	float16	352	6.7	5.1
		480	17.6	3.2
	int8	352	6.5	5.0
		480	19.57	3.5
YOLOv11m	float16	352	11.5	3.5
		480	11.9	2.5
	int8	352	12.3	3.5
		480	9.7	2.5

The performance measurements displayed in Table V offer critical insights into the viability of real-time implementation. All values reported are averaged over multiple trials to ensure consistency. Notably, YOLOv11n with an input size of 352×352 achieves the highest inference speed, maintaining 10 frames per second (FPS) across both float16 and int8 quantization formats. This outcome identifies YOLOv11n-352 as the most suitable option for real-time applications when throughput is the foremost priority.

While quantization exerts a negligible influence on inference and loading durations, it presents a performance trade-off that merits consideration. Empirical findings from prior trials demonstrate that although quantization reduces model size, it may also marginally impair detection performance, as evidenced by the mean Average Precision (mAP). Nevertheless, for this specific deployment scenario, the trade-off remains acceptable. The real-time interface, as shown in Figure 5, displays detection results by overlaying bounding boxes and composing recognized letters into words, while also providing configurable settings such as confidence threshold and camera configuration adjustment.

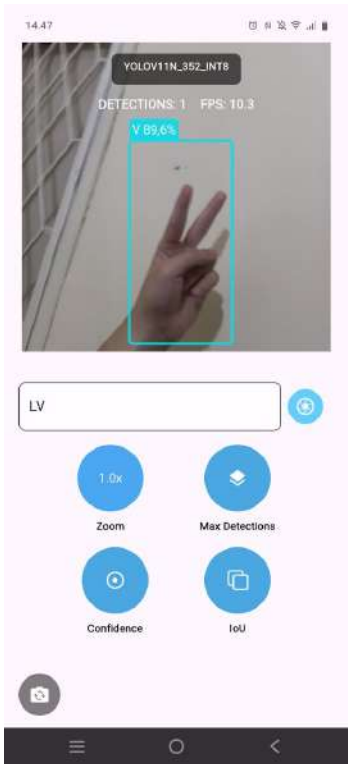

Figure 5. Mobile Interface for BISINDO Alphabet Real-time Detection

V. Conclusion

This paper proposes and implements a mobile-based approach to BISINDO alphabetic sign-language recognition based on the YOLOv11 (You Only Look Once) object-detection algorithm, detailing a comprehensive methodology that combines optimized object-detection networks to deliver reliable real-time performance on GPU-powered handheld devices. The trained YOLOv11 models exhibit robust object-detection performance. Specifically, the medium-sized variant, YOLOv11m, attains a mean average precision of $\mathrm{mAP0.5{:}0.95} = 82.3\%$ and a precision of 99.2%, while the more compact YOLOv11s and YOLOv11n variants each maintain $\mathrm{mAP0.5{:}0.95} > 80\%$. In addition, post-training quantisation and the adoption of reduced input resolutions (352×352 and 480×480 pixels) lower both model size and inference latency with negligible performance loss. These optimisations ensure that alphabetic BISINDO recognition remains reliable and responsive on mobile environment, sustaining throughput in the range of 2–10 fps.

References

[1] N. Tarigopula, S. Tornay, S. Muralidhar, and M. Magimai.-Doss, “Towards accessible sign language assessment and learning,” in *Proceedings of the 24th ACM International Conference on Multimodal Interaction (ICMI ’22)*, ser. ICMI ’22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, p. 1. [Online]. Available: https://doi.org/10.1145/3536221.3556623

[2] World Health Organization, “Deafness and hearing loss,” https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss, 2024, accessed: July 1, 2025.

[3] R. Sutjiadi, “Android-based application for real-time indonesian sign language recognition using convolutional neural network,” *TEM Journal*, vol. 12, pp. 1541–1549, 08 2023.

[4] A. Karn, “Artificial intelligence in computer vision,” *International Journal of Engineering Applied Sciences and Technology*, vol. 6, pp. 249–254, 07 2021.

[5] A. Imran, M. Shashishekhara Hulikal, and H. A. A. Gardi, “Real time american sign language detection using yolo-v9,” 2024.

[6] R. Arlin and R. Munir, “The development of push up counter android application with computer vision,” 2023.

[7] I. B. A. Peling, I. M. P. A. Ariawan, and G. B. Subiksa, “Deteksi bahasa isyarat menggunakan tensorflow lite dan american sign language (asl),” *Jurnal Krisnadana*, vol. 3, pp. 198–205, 2024.

[8] ClearML, “Clearml research report: Mlops in 2023,” https://6165398.fs1.hubspotusercontent-na1.net/hubfs/6165398/ClearML%20Research%20Report_%20MLOps%20in%202023.pdf, 2023, accessed: July 7, 2025.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” arXiv:1506.02640 [cs.CV], 2016.

[10] Z. He, K. Wang, T. Fang, L. Su, R. Chen, and X. Fei, “Comprehensive performance evaluation of yolov11, yolov10, yolov9, yolov8 and yolov5 on object detection of power equipment,” arXiv:2407.15904 [cs.CV], 2024.

[11] M. Kisantal, Z. Wojna, J. Murawski, J. Naruniec, and K. Cho, “Augmentation for small object detection,” arXiv:1902.07296 [cs.CV], 2019.

[12] T. Yu and H. Zhu, “Hyper-parameter optimization: A review of algorithms and applications,” *arXiv:2003.05689*, 2020.

[13] R. Sharma and J. Hough, “Hyperparameter optimization on yolov8 model for training costs reduction,” pp. 25–30, Jan. 2025, accessed via ResearchGate.

[14] I. L. Orăşan, C. Seiculescu, and C. D. Caleanu, “Benchmarking tensorflow lite quantization algorithms for deep neural networks,” in *Proceedings of the 2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2022, pp. 221–226.

[15] A. Saxena, A. K. Bishwas, A. A. Mishra, and R. Armstrong, “Comprehensive study on performance evaluation and optimization of model compression: Bridging traditional deep learning and large language models,” arXiv:2407.15904 [cs.LG], 2024.

[16] S. Ruder, “An overview of gradient descent optimization algorithms,” arXiv:1609.04747 [cs.LG], 2017.

[17] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann, “On-device neural net inference with mobile gpus,” in *Proceedings of the Efficient Deep Learning for Computer Vision Workshop (ECV), CVPR 2019*, 2019, workshop on Computer Vision and Pattern Recognition (CVPR).