LONTAR KOMPUTER VOL. 16, NO. 2 AUGUST 2025 **p-ISSN 2088-1541**
DOI : 10.24843/LKJTI.2025.v16.i2.p04 **e-ISSN 2541-5832**
Accredited Sinta 2 by RISTEKDIKTI Decree No. 158/E/KPT/2021

Extensive Deep Learning Models Evaluation For Indonesian Sign Language Recognition

Audrey Tilanov Pramasa[a1], Ni Putu Sutramiani[a2], I Putu Agung Bayupati[a3], I Wayan Agus Surya Darma[a4]

[a]Department of Information Technology, Engineering Faculty, Udayana University
Jimbaran, Bali, Indonesia
[1]audrey.tilanov003@student.unud.ac.id
[2]sutramiani@unud.ac.id
[3]bayupati@unud.ac.id
[4]agussurya@unud.ac.id

Abstract

Sign language is a vital communication method for individuals with hearing loss or deafness, with variations reflecting unique cultural contexts. Real-time recognition of sign language can bridge communication gaps, yet developing tools for Indonesian Sign Language (BISINDO) is challenging due to limited datasets. This research addresses these challenges by enhancing BISINDO detection and real-time recognition, focusing on flexible dataset collection and adaptation to varying lighting conditions. Three convolutional neural networks, namely InceptionV3, MobileNetV2, and ResNet50, are evaluated with optimizers SGD, Adagrad, and Adam to determine the best architecture-optimizer combination. Models were trained on a common dataset and analyzed for optimal performance. Real-time recognition uses MobileNetV2 SSD, integrating data augmentation to improve performance under diverse lighting. The system was deployed on a mobile device for practical use. Results showed the real-time model attained a mean Average Precision (mAP) of 90.34%. This study demonstrates significant advancements in BISINDO recognition and real-time application.

Keywords: *Convolutional Neural Network, Sign Language, Indonesian Sign Language, BISINDO, MobileNetV2 SSD*

1. Introduction

Sign language serves as the main form of communication for individuals who are deaf or hard of hearing, differing significantly from spoken language used by people who can hear and speak clearly. This difference often creates a barrier in communication between the deaf and hearing communities. To address this issue, machine learning can be used to develop a device that facilitate communication, such as Sign Language Recognition (SLR) system. Unlike spoken languages, sign language is a combination of visual elements, including facial expressions, body movements, and hand gestures. Converting sign language into text can serve as an essential tool to bridge the gap between hearing and deaf communities. Research into machine learning algorithms for SLR has been extensive, with techniques like Convolutional Neural Networks (CNN) playing a key role in the classification and extraction of sign language gestures. In the context of Indonesian Sign Language (ISL), specifically BISINDO, the development of an SLR system is challenging due to the limited resources of high-quality BISINDO datasets. Therefore, creating a BISINDO dataset through processes like data augmentation and annotation, along with using models that are suitable for classification and detection while being lightweight, is crucial for achieving accurate, real-time recognition of ISL in the BISINDO format.

While prior studies have contributed significantly to BISINDO recognition using convolutional neural networks, many remain limited in terms of real-time deployment and mobile efficiency. For instance, [1] developed a custom CNN model that achieved high accuracy (98.3%) in recognizing BISINDO gestures under varying lighting and perspectives. However, their work was focused solely on static image classification and did not address inference speed or model deployment on resource-constrained devices. Meanwhile, [2] implemented a CNN model to detect BISINDO and SIBI signs, obtaining 82.3% testing accuracy on BISINDO data. Despite these promising results,

the study used a relatively small dataset (only 12 samples per class for BISINDO), lacked noise reduction techniques, and did not incorporate mobile-oriented architecture or real-time evaluation. In contrast, our research builds upon these foundations by evaluating three powerful CNN architectures. MobileNetV2, ResNet50, and InceptionV3, in combination with optimization strategies such as Adam, AdaGrad, and SGD. Among these, MobileNetV2 is prioritized for its lightweight design, making it highly suitable for deployment on mobile and embedded devices. To further support real-time recognition, MobileNetV2 is integrated with the Single Shot Multibox Detector (SSD), enabling simultaneous gesture classification and localization in a single pass. This integration ensures efficient inference speed without sacrificing accuracy. While ResNet50 and InceptionV3 provide valuable performance benchmarks, the MobileNetV2 SSD pairing serves as the core of our proposed mobile-friendly BISINDO recognition system. This approach fills a critical gap in existing BISINDO research by addressing both classification performance and the practical demands of real-time, on-device deployment.

Deep learning-based object detection is now commonly used in everyday life. These techniques assist in identifying parts of objects within images and videos, supporting a wide range of tasks. Examples include its application in autonomous driving, medical imaging, facial recognition, cultural objects visualization, all of which depend heavily on object detection technology. Recently, numerous studies have applied deep learning in those fields, autonomous driving [3], [4], cultural objects visualization [5], [6], facial recognition [7], [8], sign language recognition [9], [10], [11], and several other fields. There are many topics discussing sign language. However, the application of object classification and real-time detection in indonesian sign language specifically BISINDO is very limited. Sign language is different in every country, with their own uniqueness and being developed from each of their own culture making object classification and detection on BISINDO challenging. The uniqueness of each country's sign language, shaped by cultural influences, means that classification and detection systems must be tailored specifically to that language's characteristics. For BISINDO, the lack of high-quality datasets and models optimized for real-time detection further complicates the development process.

The research was conducted using three different CNN architectures: InceptionV3, MobileNetV2, and ResNet50 along with three optimizers: SGD (Stochastic Gradient Descent), AdaGrad, and Adam, to evaluate which combination offers the fastest and most accurate classification performance. Each model's architecture has unique strengths. InceptionV3 provides a balance between speed and accuracy, ResNet50 is known for its depth and performance, and MobileNetV2 is highly efficient, particularly for mobile deployment. For real-time detection, MobileNetV2 SSD (Single Shot Detector) was used due to its lightweight and efficient nature, making it ideal for Android devices. This ensures that the model can run smoothly without consuming excessive resources, crucial for mobile applications. Our approach involves creating a dedicated BISINDO dataset specifically for this study and fine-tuning the MobileNetV2 SSD model for detecting Indonesian Sign Language (BISINDO). Through parameter optimization and the application of techniques such as data augmentation and annotation, this research aim to improve both the accuracy and speed of detection. Specifically, the contributions of this research are as follows:

a. **Evaluation of Convolutional Neural Network Architectures**: Three different CNN architectures were evaluated; InceptionV3, MobileNetV2, and ResNet50, to identify the model that provides the highest accuracy and performance for classifying Indonesian Sign Language (BISINDO).
b. **Development of a BISINDO Dataset**: A dedicated dataset for Indonesian Sign Language (BISINDO) was introduced, incorporating extensive data augmentation and annotation. This dataset is tailored specifically for real-time sign language detection.
c. **Real-Time Sign Language Recognition**: A real-time sign language recognition system was introduced utilizing the MobileNetV2 Single Shot Detector (SSD). The model was optimized and fine-tuned to achieve the highest accuracy with minimal loss for BISINDO detection.

2. Research Methods

The research workflow outlines distinct approaches for the classification and real-time processes due to differences in methods and techniques used. The classification process workflow is split into two phases: the general stage, which covers the process from dataset collection to classification, and the specific stage, which details the model training process using various hyperparameters to identify the optimal model-hyperparameter combination. In contrast, the workflow for real-time sign language recognition consists of a single stage, as it involves only one model, the MobileNetV2 Single Shot Multibox Detector (SSD), encompassing the entire process.

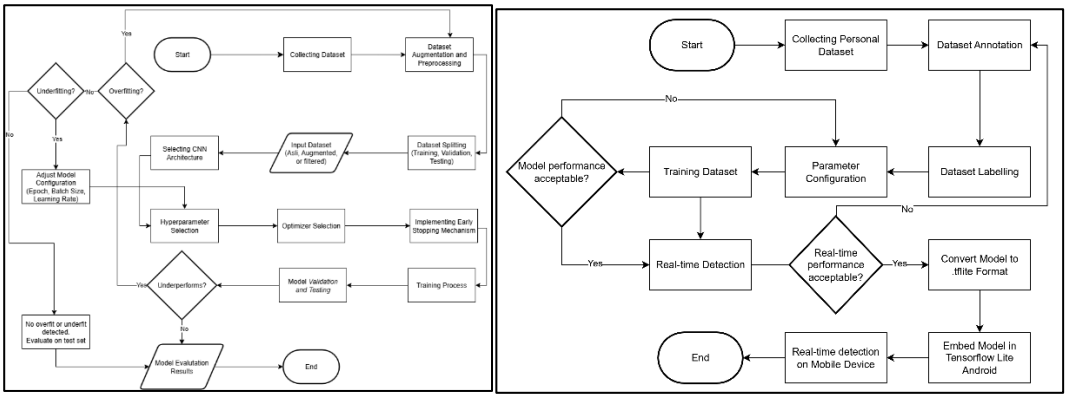


Figure 1. Block Diagram for classification (a) and real-time detection workflow (b)

Figure 1 above shows the general workflow diagram (a) for the classification process, where the specific workflow focuses on explaining the Model Training process. While the real-time research workflow diagram (b) involves the processed of a dataset collection and is distinct from the classification workflow. The collected and augmented data undergoes an annotation process first, enabling the model to focus on classifying only the regions containing alphabet class characteristics.

Although both classification and real-time object detection models utilize convolutional neural networks, their training processes differ significantly due to their distinct objectives and data requirements. In classification, the model is trained to assign a single class label to an entire image, using relatively simple inputs consisting of images and corresponding labels. In contrast, real-time detection models such as MobileNetV2-SSD must simultaneously perform localization and classification, requiring input images annotated with bounding boxes and class labels for each object. it shifts from learning only global image features to learning multi-scale spatial features necessary for detecting objects of varying sizes and positions. As a result, training a real-time detection model demands more structured and diverse data, as well as a fundamentally different learning approach compared to standard image classification.

In classification problems, the dataset is split into 70% for training, 20% for validation, and 10% for testing to ensure reliable model performance reporting and avoid bias. In contrast, for real-time object detection; where data labeling is expensive and real-world feedback is crucial, a more practical 80% training and 20% validation split is used, omitting a separate test set where performance is often validated directly in real-world deployment environments rather than through a held-out test set.

2.1. Indonesian Sign Language (BISINDO)

With the development of Indonesian Sign Language among the deaf community, two standards are currently recognized. The first is SIBI (Indonesian Sign System), created by the government, and the second is BISINDO (Indonesian Sign Language), which is widely used by the deaf community and developed naturally within Indonesia's deaf culture. BISINDO consists of 26 alphabets, corresponding to the Indonesian alphabet. These alphabets serve as the foundation for this study, where a camera will recognize hand shapes resembling BISINDO alphabets in real-time and produce text as output. BISINDO is chosen for this research due to its popularity within the deaf community and to promote the BISINDO standard to the broader Indonesian public [12].

2.2. Hyperparameter Tuning & Optimization

Hyperparameters, such as Batch Size and Learning Rate, along with Optimization methods, significantly influence a model's performance. These parameters can affect outcomes, determining whether the model achieves high or low accuracy, overfits or underfits, or aligns well with the provided dataset.

Hyperparameters control the overall training process and are categorized into two types, model-related and optimization-related. Model hyperparameters define the network's structure, while optimization hyperparameters determine the training outcome, influencing whether the desired results are achieved [13]. Thus, these parameters play a crucial role in producing outputs that align with the dataset and meet the desired objectives.

2.3. CNN Model

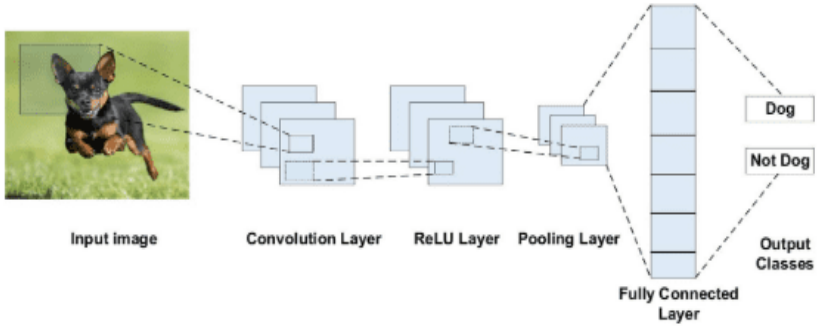


Figure 2. CNN Model Architecture

Convolutional Neural Network (CNN) is a popular deep learning technique commonly used for object detection in images or videos, including applications in computer vision, facial recognition, and biomedical fields. This popularity stems from CNN's ability to automatically extract relevant features from images without human intervention (unsupervised) [14]. As can be seen in Figure 2 above, a CNN consists of layers that form the backbone of its processing, as illustrated in Figure 2. These layers include the convolution layer, which performs convolutions using predefined kernels on the input image, and the pooling layer, which processes features from the convolutional layer to retain information relevant to the input image. The pooling layer also reduces the dimensionality of the features, making them more manageable. The output is then passed to the fully connected layer, which classifies the input image into predefined classes [14].

2.3.1 ResNet50

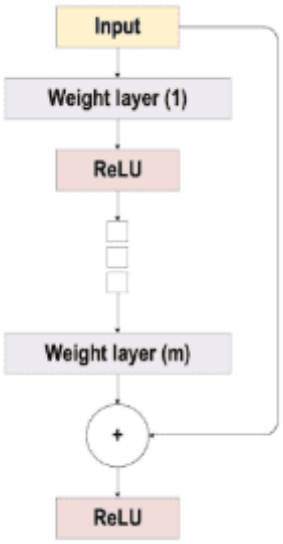


Figure 3. ResNet50 Diagram Block

ResNet50, or Residual Network 50, is a convolutional neural network (CNN) architecture with 50 layers, designed to address the vanishing gradient problem. Comprising 25.5 million parameters, ResNet50 employs skip connections between layers, enabling information to 'bypass' preceding layers, thus preventing the loss of crucial data during training. This feature, illustrated in the ResNet block diagram in Figure 3 above, allows the network to learn more complex features and enhance accuracy. This characteristic is also utilized within InceptionResNet to mitigate the vanishing gradient issue [14].

2.3.2 MobileNetV2

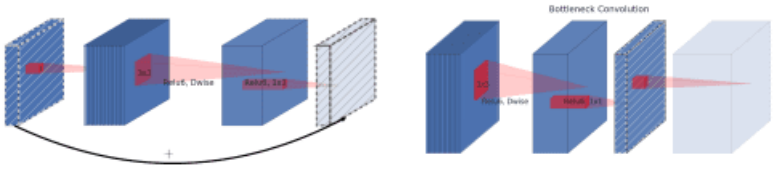


Figure 4. MobileNetV2 Inverted Residual (Left) and Linear Bottlenecks (Right)

MobileNetV2 is an efficient deep learning model designed for image classification, focusing on portability and reduced computational cost. Building on MobileNetV1, it introduces key innovations such as Depthwise Separable Convolutions (DSC), Linear Bottlenecks, and Inverted Residuals. As demonstrated in recent applications such as real-time face mask recognition [8], as illustrated in Figure 4. Furthermore, MobileNetV2 can be effectively combined with Single Shot Detector (SSD) and SSDLite frameworks, which are ideal for mobile deployment. SSD's decoupled convolution process reduces computational costs and model size, maintaining strong object detection performance. The integration of MobileNetV2 with SSD enables significant advancements in mobile computer vision model development

2.3.3 InceptionV3

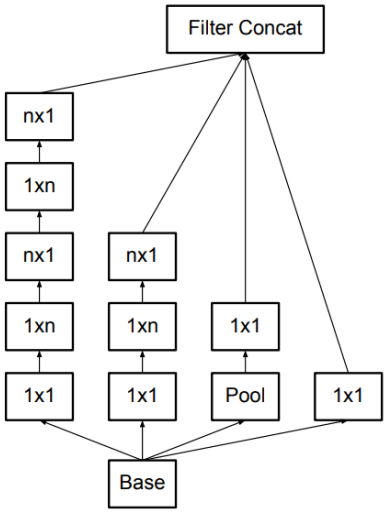


Figure 5. InceptionV3 Factorization of Convolution Process

InceptionV3 is a convolutional neural network architecture designed for computer vision tasks. Its innovation lies in the factorization of convolution processes, as illustrated in Figure 5, where large convolutional layers are broken down into smaller, more efficient operations, reducing computational costs. This approach maintains performance and effectiveness in image recognition. InceptionV3 also enhances and stabilizes training performance by using auxiliary classifiers to aid the training process and applying label smoothing as a regularization technique to improve the model's generalization capabilities. These enhancements enable InceptionV3 to achieve high accuracy in image classification [15].

2.3.4 Single Shot Multibox Detector (SSD)

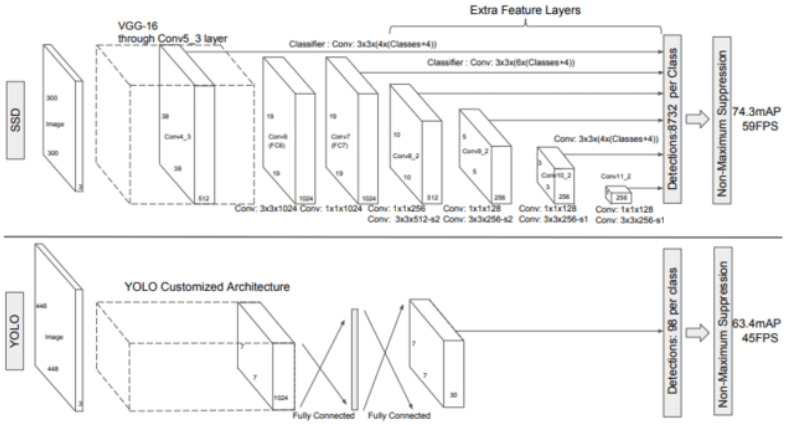


Figure 6. Comparison of SSD and YOLO Architecture Model

The Single Shot MultiBox Detector (SSD) is a robust object detection model that leverages VGG-16 as its base architecture. By incorporating additional feature layers and processing images at multiple scales and aspect ratios, SSD achieves a high number of detections per class, significantly enhancing its detection capabilities. This approach allows SSD to outperform models like YOLO and Faster R-CNN in terms of both speed and accuracy. Notably, SSD's architecture facilitates real-time applications by efficiently detecting objects at varying scales [16].

2.4. Evaluation Method

Evaluation assesses a model's problem-solving capabilities using metrics such as accuracy, categorical cross-entropy loss, and validation coefficients. These metrics help evaluate classification and image/gesture recognition tasks, and also identify overfitting or underfitting. In this study, the evaluation focuses on Accuracy, Precision, and F1-score. Accuracy measures overall correctness, Precision assesses true positives against false positives, and the F1-Score balances both Precision and Recall.

Additionally, a Confusion Matrix is used to visualize prediction accuracy by showing true and false results per class. While the traditional confusion matrix is commonly presented in a binary classification setting (2×2 matrix), this study involves **multiclass classification with 26 classes**, requiring an **extended 26×26 confusion matrix**. In this matrix, each row represents the actual class, and each column represents the predicted class. Diagonal elements indicate correct predictions, while off-diagonal elements represent misclassifications.

To compute Precision, Recall, and F1-Score in a multiclass context, these metrics are calculated per class and then averaged using different strategies:

a. **Macro averaging** treats all classes equally by averaging metrics independently across all classes.
b. **Micro averaging** aggregates the total true positives, false positives, and false negatives across all classes before calculating the metrics.
c. **Weighted averaging** averages the metric for each class while taking into account the number of true instances per class, which is useful when class distributions are imbalanced.

These approaches provide a comprehensive evaluation of the model's performance across all 26 classes rather than simplifying it to a binary scenario [17].

Below are the standard definitions:

a. **TP (True Positive)**: The model correctly predicts a given class.
b. **TN (True Negative)**: The model correctly predicts all other classes as not being the target class.
c. **FP (False Positive)**: The model incorrectly predicts a class when it's not present.
d. **FN (False Negative)**: The model fails to predict the class when it is present.

Accuracy measures the proportion of total correct predictions:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (1)$$

Precision measures the model's accuracy in identifying true positives among all predicted positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

The F1-Score is the harmonic mean of precision and recall.

$$\text{F1} - \text{Score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \tag{3}$$

Recall quantifies the model's ability to identify all actual positive cases.

$$\text{F1} - \text{Score} = 2 \cdot \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

Categorical Cross Entropy Loss measures CNN model performance, also known as Log Loss, used for multi-class classification, replacing Square Error Loss. It uses a softmax activation in the output layer for probability distribution. This loss function assesses model training accuracy by comparing predicted probabilities to the dataset [13].

$$-\log\left(\frac{e^{sp}}{\Sigma_j^c e^{sj}}\right) \tag{5}$$

Real-time recognition evaluation uses localization loss and confidence loss from training the MobileNetV2 SSD model. The calculation is automatically generated during training. These metrics indicate if the model has adequately learned the dataset.

$$L(x, c, l, g) = \frac{1}{N}(L_{con\,f}(x, c) + \alpha L_{loc}(x, l, g)) \tag{6}$$

TensorFlow Lite is a framework for running machine learning models on mobile devices. Models must be converted to *.tflite* format with embedded label maps. The provided GitHub link: **https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android_play_services**, enables real-time object detection using custom *.tflite* models.

2.5. Real-Time Recognition

The real-time model used is MobileNetV2 combined with the Single Shot Multibox Detector (SSD), resulting in the MobileNetV2 SSD model. This model is then converted for deployment on a mobile system built using the TensorFlow Lite framework for Android. There are experimental scenarios for the classification process, involved a combination of models, hyperparameters, and optimizers.

Table 1. Experimental Scenarios for Classification Process

No	Model	Optimizer	Early Stopping Patience
1	ResNet50	AdaGrad	7
2	ResNet50	Adam	7
3	ResNet50	SGD	7
4	InceptionV3	AdaGrad	7
5	InceptionV3	Adam	7
6	InceptionV3	SGD	7
7	MobileNetV2	AdaGrad	7
8	MobileNetV2	Adam	7
9	MobileNetV2	SGD	7

Table 1 above presents nine combinations resulting from pairing different models with optimizers. Each of these combinations will be evaluated, and the model achieving the highest accuracy will be selected as the best-performing model. The early stopping mechanism was set with a patience of 7 epochs across all model variations. This value was selected based on a balance between preventing overfitting and allowing the model sufficient time to continue learning. A patience value that is too low may cause training to stop prematurely due to minor fluctuations in validation performance, especially in complex models or noisy datasets. On the other hand, a very high

patience may delay convergence unnecessarily, leading to overfitting or wasted training time. Through initial experimentation, a patience value of 7 was found to provide the model enough room to improve while still reacting quickly to performance stagnation on the validation set.

3. Result and Discussion

3.1 Dataset

This research utilized a combined dataset comprising images from online sources (Kaggle) and personal collections, focusing on BISINDO sign language. The Kaggle dataset contributed 312 images, with 12 images per class across 26 alphabetical classes. To expand the dataset's diversity and size, these were supplemented with 5590 personally captured smartphone images, each at a resolution of 2544x3392 pixels, with 215 images per class. This resulted in a total dataset of 5902 images spanning the 26 classes.

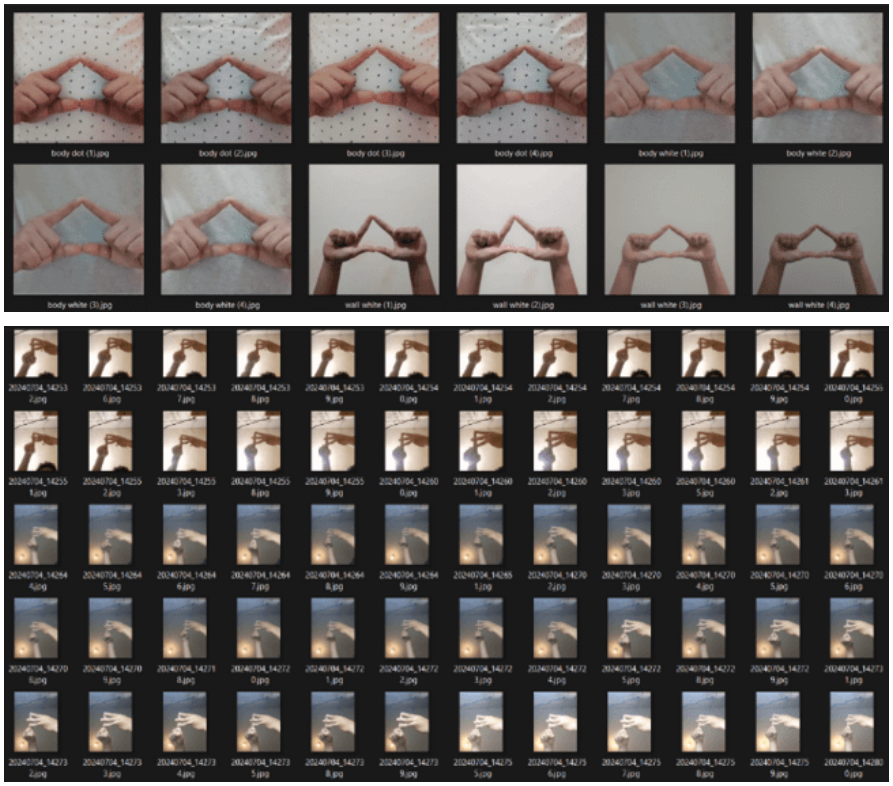


Figure 7. Dataset Sourced from Kaggle (Top) and Personally Compiled Dataset (Bottom)

The personal dataset was meticulously captured with varying ISO and shutter speed settings to accommodate diverse lighting conditions, from 50W lamp assistance to low-light, high-light, and natural lighting. Each configuration yielded at least 30 images. Each class of the personal dataset was then split into 166 training images and 49 testing images to ensure robust model training and evaluation.

Table 2. Dataset Collection Configuration

No	Shutter Speed	ISO	Total Images	Lamp Status
1	1/90s	125	30	Yes
2	1/125s	320	30	Yes
3	1/180s	100	35	Yes
4	1/125s	125	30	Yes
5	1/125s	400	30	Yes
6	1/50s	1600	30	No
7	1/60s	1600	30	No

The combined dataset exhibited diverse characteristics, including varying backgrounds and lighting conditions. To enhance the dataset's size and variability, data augmentation techniques were employed on the Kaggle dataset, increasing the initial 12 images per class to 45.

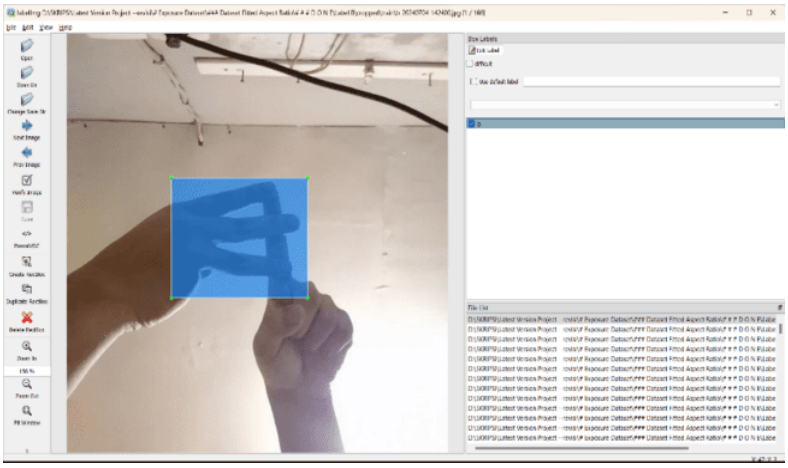


Figure 8. Data Annotation in Preprocessing Step

Upon collection, the personally captured images, initially featuring a 3:4 aspect ratio, were preprocessed to a 600x600 pixel resolution. This adjustment was necessary to align with the input requirements of the MobileNetV2 Coco 300x300 pretrained model, facilitating subsequent annotation and model training.

To increase the size and variability of the dataset, data augmentation techniques were applied, expanding the initial 12 images per class to 45, resulting in a total of 1,170 augmented images. For the image classification task, the combined dataset was partitioned into 70% for training, 20% for validation, and 10% for testing. This three-way split ensures that the model is trained and fine-tuned on separate subsets, with a completely independent test set used for final evaluation providing a robust measure of the model's generalization on unseen data.

In contrast, for the real-time object detection task using TensorFlow Lite, the dataset was divided into 80% for training and 20% for validation, without a dedicated test set. This simplified split is commonly used in real-time scenarios, where annotated data (such as bounding boxes) is costly and performance is often assessed directly in deployment environments rather than through a held-out test set. This tailored partitioning approach aligns with the specific requirements and evaluation methods of each task.

Data augmentation was applied to both the classification and real-time detection datasets to enhance variability and prevent overfitting, with particular emphasis on real-time object detection tasks. Models like MobileNetV2-SSD require a larger and more diverse dataset due to the added complexity of localizing and classifying multiple objects within an image, unlike classification tasks that predict a single label. Although MobileNetV2-SSD is designed for lightweight and fast inference on mobile devices, its architecture relies heavily on datasets that capture a wide range of variations in object size, orientation, lighting, occlusion, and background context. Without sufficient or diverse data, the model risks poor generalization, missed detections, or inaccurate bounding boxes during real-world deployment. Therefore, to ensure reliable performance in real-time scenarios, a larger volume of annotated images is essential, further justifying the use of aggressive data augmentation.

Table 3. Dataset Information for Classification Process

Dataset Name	26 Classes	Total
Real Dataset	12 Images per class	312 Images
Augmented Dataset	45 Images per class	1.170 Images
Private Dataset	166 Images per class	4.316 Images

Table 4. Dataset Information for Real-Time Detection Process

Dataset Name	26 Classes	Total
Real Dataset	215 Images per class	5.590 Images
Augmented Dataset	215 Images per class	5.590 Images
Private Dataset	1074 Images per class	27.924 Images

The augmentation applied in this study was performed on all dataset. The technique used was a light augmentation strategy focused on adjusting image brightness, implemented via ImageDataGenerator with a brightness_range of [0.7, 1], rescale=1./255, and fill_mode='reflect'. This approach was chosen to simulate realistic lighting variations while maintaining the original structure and features of the input image [18].
Heavier augmentation techniques such as rotation, zoom, horizontal flipping, or warping were intentionally not applied in this case. This decision was to preserve class-defining features. Since the dataset contains images where object orientation and positioning are critical for correct classification (e.g., gestures or fine-grained object details), aggressive transformations could distort key features, potentially confusing the model. Thus, the augmentation strategy was deliberately kept minimal to improve generalization without overcomplicating the feature space or reducing model stability.

3.2 Model Training and Real-Time Detection

Model configuration, managed via the pipeline.config file, allows customization of MobileNetV2 SSD, including dataset classes, image dimensions, hyperparameters, CNN layers, and fine-tuning checkpoints. Training commences after configuration, dataset preparation, and TFRecord creation. As seen in Figure 9 below, real-time detection is performed using a laptop webcam, with accuracy impacted by image size discrepancies between the dataset and capture device. OpenCV is used to load model checkpoints and integrate them for frame-by-frame webcam detection.

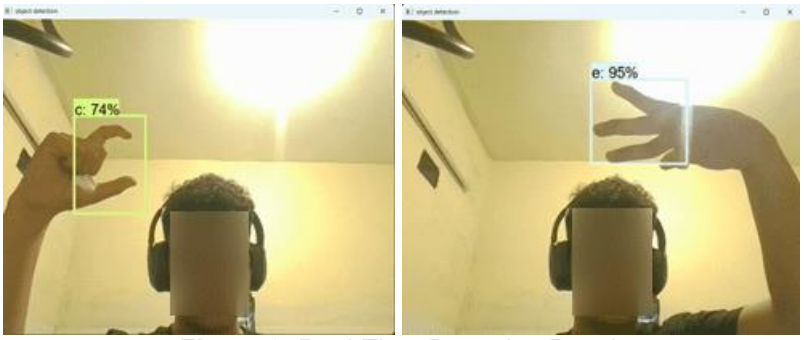


Figure 9. Real-Time Detection Result

After training, the MobileNetV2 SSD model's checkpoints are converted to .pb (saved_model) format, then to .tflite for mobile use. Metadata is repopulated to remap classes for .tflite compatibility. The .tflite model is then integrated into the Android app by adding a constant reference and placing it in the assets folder. Real-time detection is achieved using the device's camera, with results displayed on screen and a detection history logged. This process optimizes the trained model for mobile deployment, enabling real-time object detection on Android devices, as can be seen in Figure 10 below.

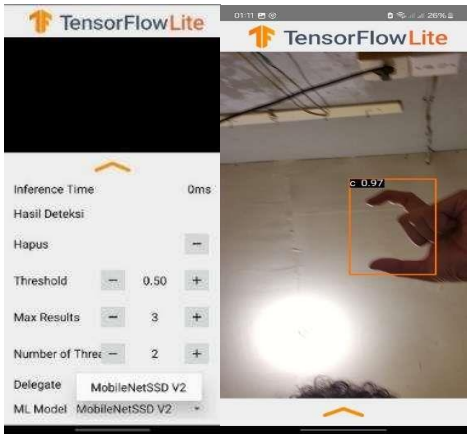


Figure 10. Real-Time Detection Using TensorFlow Lite (Left) with a C Labeled Image (Right)

Figure 10 shows the real-time detection result where the detected object is the alphabet class C with an accuracy of 97%. This indicates that the model has been successfully loaded and has

successfully detected the Indonesian Sign Language alphabet class based on adequate lighting conditions with a relatively neutral background, in accordance with the initial goal.

3.3 Result Analysis

Detection results were analyzed to evaluate accuracy and mean Average Precision (mAP) using TensorFlow Lite Android. The analysis for real-time recognition was divided into overall model performance and light level (lux) impact, and the analysis was divided based on each model combination and its hyperparameters. Overall mAP was calculated by averaging per-class detection accuracy. Tests were conducted under controlled lighting similar to the dataset.

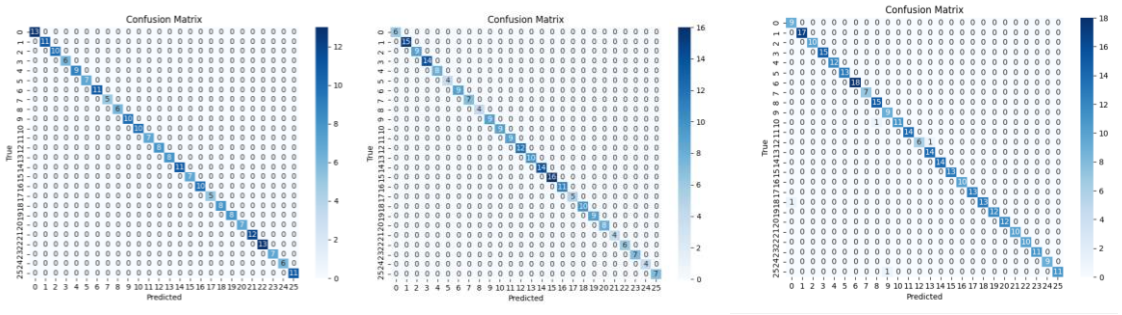
Table 5. mAP Analysis Result of the Dataset

No	Alphabet Classes	Average Precision (AP)
1	A	92%
2	B	82%
3	C	95%
4	D	86%
5	E	84%
6	F	77%
7	G	86%
8	H	88%
9	I	96%
10	J	88%
11	K	92%
12	L	94%
13	M	94%
14	N	92%
15	O	96%
16	P	88%
17	Q	92%
18	R	96%
19	S	96%
20	T	88%
21	U	94%
22	V	92%
23	W	93%
24	X	89%
25	Y	94%
26	Z	85%
mAP		**90.34%**

Lux level analysis measured optimal detection under specific lighting conditions, using a Samsung Galaxy S24's light sensor and the Lux Light Meter Pro app. Measurements were taken 30-40cm from the light source.

Table 6. Lux Level Analysis

No	Lux Level	mAP
1	~900	91%
2	~750	84%
3	~600	75%
4	~400	68%



(a) (b) (c)

Figure 11. Confusion Matrix of The Best Performing Model. Inception SGD (a), MobileNetV2 Adam (b), and ResNet50 SGD (c)

To provide a clearer picture of each model's classification performance, Figure 11 presents the confusion matrices of the best-performing configurations for InceptionV3, ResNet50, and MobileNetV2, respectively. These matrices visualize the prediction accuracy across all 26 classes and highlight how well each model generalized to the validation data. As seen in the confusion matrices, the top-performing models exhibit strong diagonal alignment, indicating high true positive rates and minimal misclassification.

Following this, we present the overall classification accuracy achieved by each model-optimizer combination.

Overall Accuracy of each model: InceptionV3, MobileNetV2 and ResNet50, combined with three optimization algorithms: AdaGrad, Adam, and SGD. Where the InceptionV3 with SGD optimizer achieved the highest accuracy and MobileNetV2 with AdaGrad achieved the lowest accuracy.

Table 7. Classification Accuracy of Each Model Combination

	AdaGrad		Adam		SGD	
	Accuracy	**Val Loss**	**Accuracy**	**Val Loss**	**Accuracy**	**Val Loss**
InceptionV3	98.4%	0.1268	99.2%	0.0181	100%	0.0362
ResNet50	98.8%	0.1528	98.7%	0.0523	99.3%	0.0185
MobileNetV2	96.4%	0.2760	99.7%	0.0091	98.6%	0.0584

The results presented in Table 7 were obtained using a set of optimized hyperparameters that were consistently applied across all three models in this study. These hyperparameters were selected based on empirical experimentation to balance training efficiency and model generalization. The final configuration included an input image size of 64×64×3, 26 output classes, a batch size of 16, and a training duration of 35 epochs. The models were compiled using the selected optimizer, Sparse Categorical Crossentropy loss, and evaluated with accuracy and balanced sparse categorical accuracy metrics. This configuration was chosen after observing stable convergence, strong validation performance, and minimal signs of overfitting. Therefore, the results shown represent the models' best performance under these optimized hyperparameter settings.

Although the model achieved 100% accuracy in one evaluation scenario, this result should be interpreted with caution. Perfect accuracy often raises concerns in machine learning, as it may indicate potential issues such as overfitting, data leakage, or a lack of diversity in the test set. In this case, the dataset consisted of well-preprocessed and augmented images with a relatively small and controlled test set, which may have led to overly optimistic performance. Additionally, the confusion matrix showed perfect diagonal alignment, suggesting that the classes were well-separated, but it remains possible that this is due to data simplicity or a lack of challenging samples. To validate the robustness of this performance, further evaluation on larger, more diverse, or unseen real-world datasets would be necessary. Therefore, while the 100% accuracy result is reported, it is treated cautiously and not taken as a definitive measure of the model's generalization capability.

Despite InceptionV3 achieving the highest classification accuracy, it is not the most suitable model for real-time or mobile deployment due to its significantly larger size and slower inference speed.

A comparative evaluation of model performance showed that MobileNetV2 consistently achieved the fastest inference times, the highest FPS, and the smallest model size, making it the most efficient and practical choice for real-time applications. To enable both classification and object localization, MobileNetV2 was integrated with the Single Shot Multibox Detector (SSD) framework. This combination provides a strong balance between accuracy, speed, and deployability, allowing the model to perform real-time gesture detection efficiently on resource-constrained devices.

Table 8. Comparative Evaluation of Model Efficiency for Real-Time Detection

	Optimizer	Size (MB)	Inference Time (ms)	FPS
MobileNetV2	adagrad	20.45	84.97	11.77
	adam	30.25	84.31	11.86
	sgd	10.35	86.10	11.62
ResNet50	adagrad	184.72	107.54	9.30
	adam	276.60	105.28	9.50
	sgd	92.51	104.96	9.53
InceptionV3	adagrad	171.57	110.20	9.07
	adam	256.69	117.29	8.53
	sgd	86.04	114.04	8.77

Table 8 summarizes the performance of each model-optimizer combination in terms of classification accuracy, model size, inference time, and FPS. While accuracy was comparable across models, MobileNetV2 demonstrated the lowest inference latency and highest FPS, supporting its selection for real-time deployment.

4. Conclusion

Previous research indicates that combinations of InceptionV3, ResNet50, and MobileNetV2 with SGD, Adagrad, and Adam optimizers demonstrate significant potential and excellent results, particularly with the SGD optimizer, in real-time classification and detection of BISINDO sign language. The highest accuracy achieved with the SGD optimizer reached 100%, while the lowest accuracy was 96.4% with MobileNetV2 using the Adagrad optimizer. For future research addressing similar topics, whether BISINDO sign language or sign language in general, the model experiments conducted in this study can be used as a reference. Through these experiments, further research can more deeply explore the models and hyperparameters most suitable for related case studies, such as using newer models and combining it with hyperparameters optimization of other methods, and utilizing larger datasets to improve efficiency.

References

[1] S. Dwijayanti, S. Inas Taqiyyah, H. Hikmarika, and B. Yudho Suprapto, “Indonesia Sign Language Recognition using Convolutional Neural Network.” [Online]. Available: www.ijacsa.thesai.org

[2] A. N. Sihananto *et al.*, “INDONESIAN SIGN LANGUAGE IMAGE DETECTION USING CONVOLUTIONAL NEURAL NETWORK (CNN) METHOD.”

[3] H. Fujiyoshi, T. Hirakawa, and T. Yamashita, “Deep learning-based image recognition for autonomous driving,” Dec. 01, 2019, *Elsevier B.V.* doi: 10.1016/j.iatssr.2019.11.008.

[4] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, “Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues,” *Array*, vol. 10, p. 100057, Jul. 2021, doi: 10.1016/j.array.2021.100057.

[5] I. W. A. S. Darma, N. Suciati, and D. Siahaan, “CARVING-DETC: A network scaling and NMS ensemble for Balinese carving motif detection method,” *Visual Informatics*, vol. 7, no. 3, pp. 1–10, Sep. 2023, doi: 10.1016/j.visinf.2023.05.004.

[6] D. Siahaan, N. P. Sutramiani, N. Suciati, I. N. Duija, and I. W. A. S. Darma, “DeepLontar dataset for handwritten Balinese character detection and syllable recognition on Lontar manuscript,” *Sci Data*, vol. 9, no. 1, Dec. 2022, doi: 10.1038/s41597-022-01867-5.

[7] L. Boussaad and A. Boucetta, “Deep-learning based descriptors in application to aging problem in face recognition,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2975–2981, Jun. 2022, doi: 10.1016/j.jksuci.2020.10.002.

[8] M. H. Rahman, M. K. A. Jannat, M. S. Islam, G. Grossi, S. Bursic, and M. Aktaruzzaman, “Real-time face mask position recognition system based on MobileNet model,” *Smart Health*, vol. 28, Jun. 2023, doi: 10.1016/j.smhl.2023.100382.

[9] S. Katoch, V. Singh, and U. S. Tiwary, “Indian Sign Language recognition system using SURF with SVM and CNN,” *Array*, vol. 14, Jul. 2022, doi: 10.1016/j.array.2022.100141.

[10] C. K. M. Lee, K. K. H. Ng, C. H. Chen, H. C. W. Lau, S. Y. Chung, and T. Tsoi, “American sign language recognition and training method with recurrent neural network,” *Expert Syst Appl*, vol. 167, Apr. 2021, doi: 10.1016/j.eswa.2020.114403.

[11] K. Wangchuk, P. Riyamongkol, and R. Waranusast, “Real-time Bhutanese Sign Language digits recognition system using Convolutional Neural Network,” *ICT Express*, vol. 7, no. 2, pp. 215–220, Jun. 2021, doi: 10.1016/j.icte.2020.08.002.

[12] M. Dewi, T. Wahyuningrum, and N. Adi Prasetyo, “Journal of Informatics, Information System, Software Engineering and Applications Pengenalan Kata Bahasa Isyarat Indonesia (BISINDO) Menggunakan Augmented Reality (AR),” vol. 3, no. 2, pp. 53–060, 2021, doi: 10.20895/INISTA.V3I2.

[13] P. Sharma and R. S. Anand, “A comprehensive evaluation of deep models and optimizers for Indian sign language recognition,” *Graphics and Visual Computing*, vol. 5, p. 200032, Dec. 2021, doi: 10.1016/j.gvc.2021.200032.

[14] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00444-8.

[15] S. H. Khan, A. Sohail, A. Khan, and Y. S. Lee, “COVID-19 Detection in Chest X-ray Images Using a New Channel Boosted CNN,” *Diagnostics*, vol. 12, no. 2, Feb. 2022, doi: 10.3390/diagnostics12020267.

[16] D. and E. D. and S. C. and R. S. and F. C.-Y. and B. A. C. Liu Wei and Anguelov, “SSD: Single Shot MultiBox Detector,” in *Computer Vision – ECCV 2016*, J. and S. N. and W. M. Leibe Bastian and Matas, Ed., Cham: Springer International Publishing, 2016, pp. 21–37.

[17] S. Farhadpour, T. A. Warner, and A. E. Maxwell, “Selecting and Interpreting Multiclass Loss and Accuracy Assessment Metrics for Classifications with Class Imbalance: Guidance and Best Practices,” *Remote Sens (Basel)*, vol. 16, no. 3, Feb. 2024, doi: 10.3390/rs16030533.

[18] S. Park, J. Kim, S. Wang, and J. Kim, “Effectiveness of Image Augmentation Techniques on Non-Protective Personal Equipment Detection Using YOLOv8,” *Applied Sciences (Switzerland)*, vol. 15, no. 5, Mar. 2025, doi: 10.3390/app15052631.