Real-Time Sign Language Recognition on Smartphones via MobileNetV2 and Edge Impulse

JINMING XIA

github: https://github.com/JINMING2333/casa0018_coursework
edge impluse link: https://studio.edgeimpulse.com/studio/679174

Introduction

Sign language is one of the most important ways for deaf people to communicate with the outside world. However, most able-bodied individuals cannot understand sign language (Fig.1), which becomes a barrier for hearing-impaired individuals to integrate into society (Wei Hao et al, 2024). Therefore, this project aims to develop a lightweight sign language recognition system based on a smartphone camera, through real-time image recognition and classification, in order to improve the communication convenience between hearing-impaired individuals and able-bodied individuals.

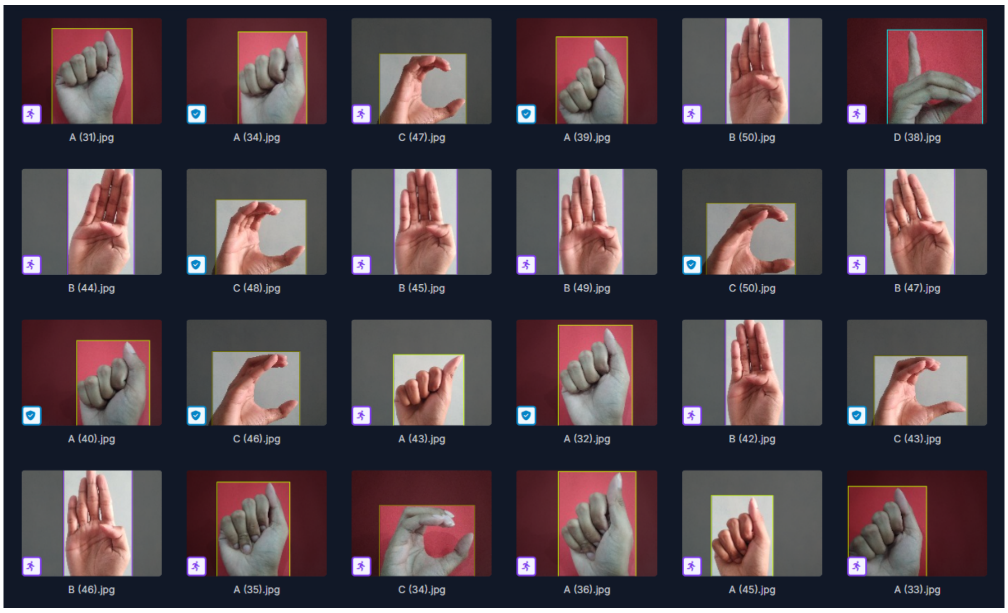


Fig.1 Difficult sign language

Sign language recognition technology can be categorized into vision-based and sensor-based methods, depending on the motion capture equipment used. Early research primarily relied on wearable sensors, but in recent years, with the development of deep learning and computer vision, vision-based sign language recognition has gradually become the mainstream solution (Mukhiddin Toshpulatov et al, 2025). Although Edge Impulse and the TinyML community have demonstrated that deep learning models can run on microcontrollers such as Arduino Nano, vision-based gesture recognition requires higher computational resources and camera quality. Therefore, smartphones, with their strong computing capabilities, high camera quality, and widespread usage, have become an ideal deployment platform.

This project successfully deployed a lightweight sign language recognition model on a mobile device, utilizing transfer learning based on MobileNetV2, and achieved high-accuracy recognition of five basic sign language letters.

Research Question

Is it feasible to develop a deep learning-based sign language recognition model for mobile devices to enable real-time communication?

Application Overview

This project includes the following implementation stages (Fig.2):
• Data Collection and Processing: Initial testing is conducted using open-source datasets, and real-world samples are collected through the smartphone for data augmentation. The data preprocessing module on the Edge Impulse platform is used to quickly extract features, laying the foundation for model training.
• Model Selection and Optimization: The control variable method is employed to systematically test different model architectures and hyperparameter combinations. Training performance and generalization capability are evaluated iteratively to progressively optimize the model.
• Deployment and Application: The final model is deployed to a smartphone by scanning the QR code generated by the Edge Impulse, enabling real-time image acquisition and sign language recognition.

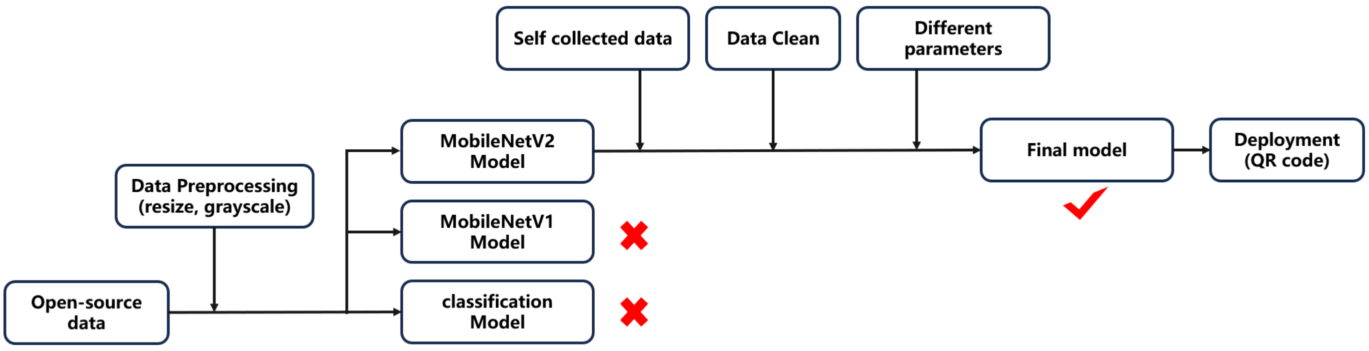
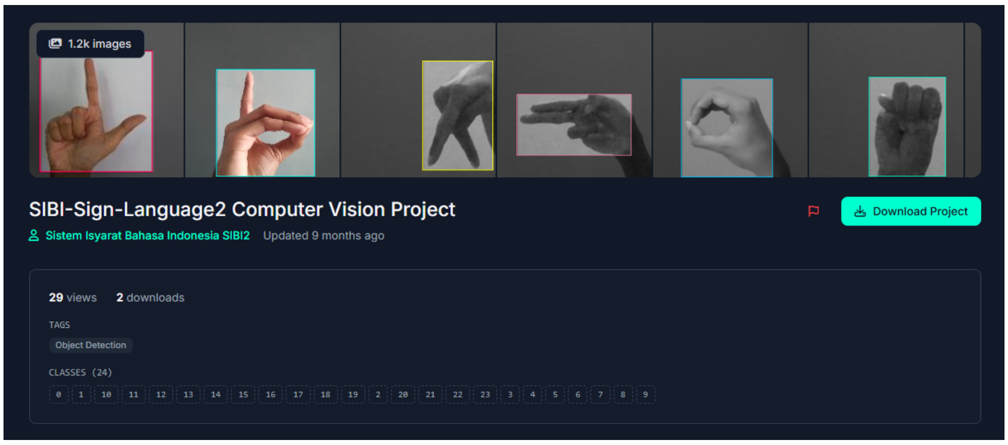


Fig.2 Application_digram

Data

This project combines open-source datasets with self-collected samples. The open-source data (Fig.3) comes from the SIBI_sign_language2 dataset (2024) on the Roboflow Universe platform, focusing on five specific classes: "C," "E," "L," "O," and "V," comprising a total of 334 images (Fig.4). After importing the data into the Edge Impulse, these images are automatically divided into training and testing sets in an approximately 81% to 19% ratio, ensuring the independence of model training and performance evaluation.



SIBI-Sign-Language2 Computer Vision Project
Sistem Isyarat Bahasa Indonesia SIBI2 Updated 9 months ago

29 views 2 downloads

TAGS
Object Detection

CLASSES (24)
0 1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 23 3 4 5 6 7 8 9

Fig.3 The open-source data

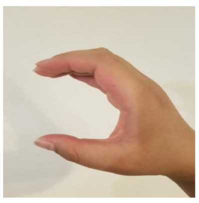
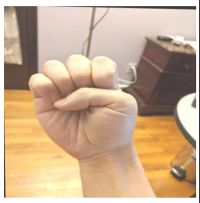
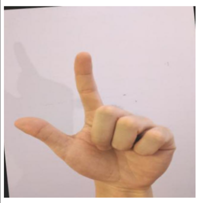
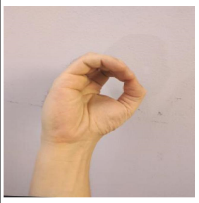
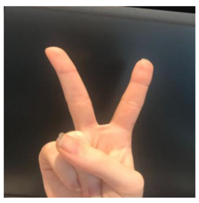
C	E	L	O	V
				

Fig.4 Example images

The self-collected data consists of 52 real-world samples captured by 8 volunteers under different indoor and outdoor lighting conditions using smartphones (Fig.5). This dataset aims to enhance the model's generalization ability in real-world environments.

Sources	Set	C	E	L	O	V	All
Open source	Train	49	54	62	51	53	269
	Test	12	13	14	13	13	65
Self collected	Train	7	10	8	9	8	42
	Test	2	2	2	2	2	10
Total		70	79	86	75	76	386

Fig.5 Data classification

Before model training, all data should be preprocessed using the Input Block and Processing Block of the Edge Impulse. Since the data type is images, the Image mode is selected in the Input Block to uniformly rescale and normalize pixel values, standardizing all image sizes to 96×96 pixels.

Next, I used the Processing Block (DSP Block) for further feature extraction. This block normalizes the image depth and retains spatial features, transforming raw images into feature tensors that are efficient for the model to learn. I built a Baseline Model, keeping other settings unchanged. Through a comparison experiment between RGB and Grayscale color depths, I found that Grayscale significantly improved model performance (Fig.6). Therefore, this project ultimately selected Grayscale as the standard data preprocessing scheme.

Baseline model	Color depth	Epochs	Training accuracy	Validation accuracy	Loss	Testing accuracy
Classification	RGB	50	100	57.4	1.84	35.38
Classification	Grayscale	50	100	61.1	1.69	38.46

Fig.6 Data preprocessing

Model

The final model architecture chosen for this project is on a MobileNetV2 backbone (alpha=0.35), with a custom classification head comprising (Fig.7&8):

Layer	Parameters	Function
Dense layer	64 neurons, ReLU	to further combine and extract high-level features
Dropout layer	0.1 Dropout rate	to prevent overfitting
Flatten layer	-	to flatten the high-dimensional tensor data for final classification
Dense output layer	5 neurons, softmax	Corresponding to the 5 letter categories

Fig.7 Layers

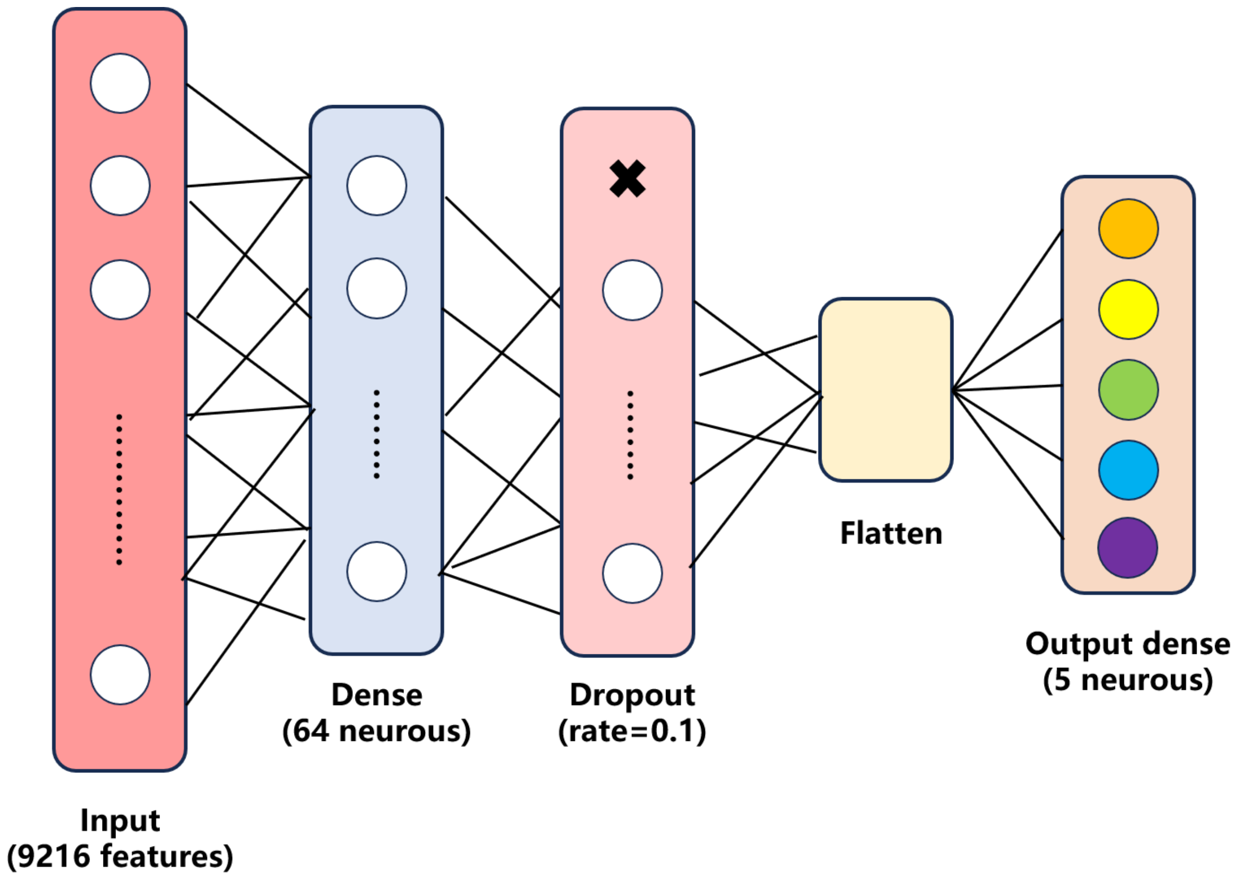


Fig.8 Model architecture

To validate the effectiveness of this architecture, multiple structures were systematically compared within the Learning Block, including shallow classification models (classification) and transfer learning models (MobileNetV1 and MobileNetV2).

The shallow classification models are simple, with low computational overhead, making them suitable for quick experimentation. However, in preliminary testing using 334 open-source images, the classification model achieved 69.7% accuracy on the training set, but performed poorly on the validation (46.3%) and test sets (3.1%), clearly unable to generalize. Even with increased training epochs and adjustments to the number of neurons in the Dense layer, the performance did not significantly improve, and overfitting occurred, indicating that shallow models have very limited generalization ability on small datasets (Fig.9).

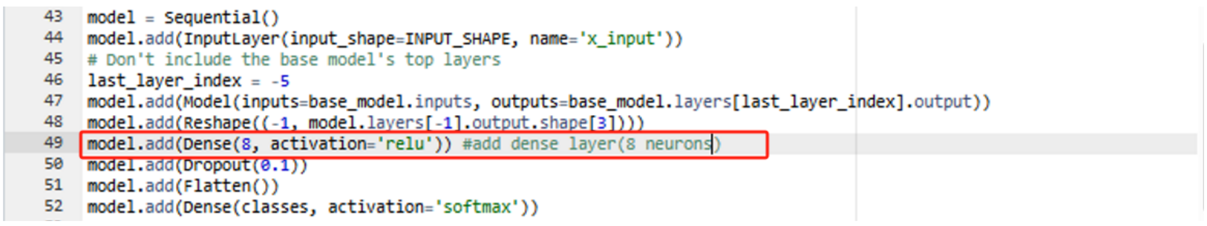
Model	Layer	Epochs	Training accuracy	Validation accuracy	Loss	Testing accuracy
Classification	-	10	69.77	46.3	2.15	3.08
Classification	add a dense layer (64 neurons)	50	100	57.4	1.84	35.38

Fig.9 Eliminate shallow models

In contrast, the transfer learning model is based on pre-trained models and supports data augmentation, making it more suitable for small-sample training. Experimental comparisons showed that MobileNetV2 consistently outperforms MobileNetV1 in both generalization and classification accuracy (Fig.10). Even after manually supplementing MobileNetV1 with a final Dense layer (to match the MobileNetV2 structure) in Expert Mode (Fig.11), its performance remained inferior. Therefore, I can definitively exclude MobileNetV1 and confirm that MobileNetV2 is a better choice for subsequent optimization and deployment.

Model	Val Accuracy	Test Accuracy	Loss	F1 Score	Inferencing time	Peak RAM usage
MobileNetV1 96x96 0.25 (no final dense layer, 0.1 dropout)	35.20%	9.23%	1.46	0.31	5ms	115.9K
MobileNetV1 96x96 0.2 (no final dense layer, 0.1 dropout)	25.90%	1.54%	1.73	0.23	3ms	95.6K
MobileNetV1 96x96 0.1 (no final dense layer, 0.1 dropout)	25.90%	3.08%	1.6	0.22	2ms	60.1k
MobileNetV2 96x96 0.35 (final layer: 8 neurons, 0.1 dropout)	72.20%	41.54%	1	0.71	6ms	214.6K
MobileNetV2 96x96 0.1 (final layer: 8 neurons, 0.1 dropout)	66.70%	44.62%	0.98	0.67	4ms	169.1K
MobileNetV2 96x96 0.05 (final layer: 8 neurons, 0.1 dropout)	48.10%	29.23%	1.44	0.45	3ms	159.2k
MobileNetV1 96x96 0.25 (final layer: 8 neurons, 0.1 dropout)	40.70%	-	1.54	0.36	3ms	116.0k

Fig.10 Comparison



```
model = Sequential()
model.add(InputLayer(input_shape=INPUT_SHAPE, name='x_input'))
# Don't include the base model's top layers
last_layer_index = -5
model.add(Model(inputs=base_model.inputs, outputs=base_model.layers[last_layer_index].output))
model.add(Reshape((-1, model.layers[-1].output.shape[3])))
model.add(Dense(8, activation='relu')) #add dense layer(8 neurons)
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(classes, activation='softmax'))
```

Fig.11 Add a final dense layer

Experiments

After selecting MobileNetV2, I further explored the impact of different alpha values (0.1 and 0.35) on model performance. Alpha is a parameter in MobileNet that controls the network width. The smaller the value, the lighter the model and the faster the inference speed, but it is typically accompanied by a decrease in accuracy. While initial performances of alpha=0.1 and alpha=0.35 were similar, extended training to 50 epochs revealed that the alpha=0.35 model consistently outperformed the alpha=0.1 model in both generalization and accuracy (Fig.12), with only a minor increase in resource consumption. Therefore, MobileNetV2 with alpha=0.35 was selected for further optimization.

Model	Epochs	Val Accuracy	Test Accuracy	Loss	F1 Score	Inferencing time	Peak RAM usage
MobileNetV2 96x96 0.35	50	79.60%	76.92%	0.67	0.79	5ms	214.6K
MobileNetV2 96x96 0.1	50	66.70%	44.44%	0.9	0.66	4ms	169.1K

Fig.12 Comparison

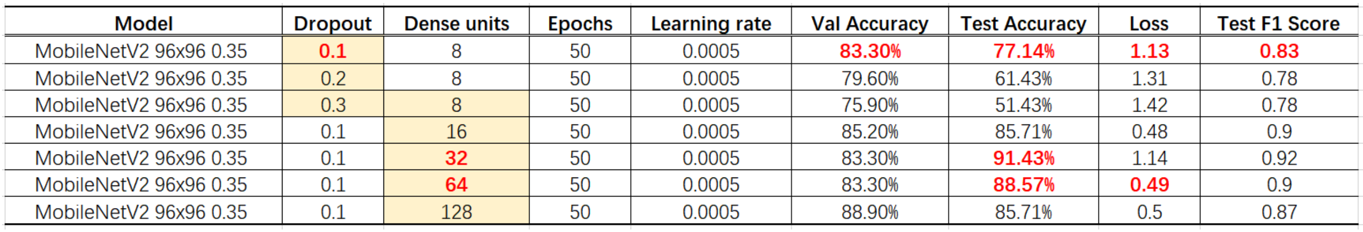
Next, to bridge the gap between the laboratory data and real-world data (Fig.13), we incorporated part of the self-collected data into the training set, while keeping the remaining data in the test set, balancing model training with generalization performance evaluation. After manually filtering and removing ambiguous samples, the accuracy of the test set improved from 73.3% to 77.1%, and the F1 score increased from 0.79 to 0.83. Although the training loss slightly increased, the overall generalization performance and the clarity of the decision boundary were significantly enhanced, validating the effectiveness of the data cleaning strategy.

Model	Data augmentation	Val Accuracy	Test Accuracy	Loss	Test F1 Score
MobileNetV2 96x96 0.35	unadd_realdata	79.60%	76.92%	0.67	0.84
MobileNetV2 96x96 0.35	add_realdata_test	75.90%	56.41%	0.71	0.65
MobileNetV2 96x96 0.35	add_realdata_train&test	79.60%	73.33%	0.65	0.79
MobileNetV2 96x96 0.35	train_test_deletewrong	83.30%	77.14%	1.13	0.83

Fig.13 Data augmentation

Additionally, due to the slight oscillation in validation loss during late training stages indicated minor overfitting. To mitigate this, different Dropout rates (0.1, 0.2, 0.3) were evaluated. Dropout is a commonly used regularization method that prevents the model from over-relying on specific feature combinations by randomly "dropping out" some neurons in the neural network, thus improving generalization ability. The results showed that a Dropout rate of 0.1 performed the best (77.1% test accuracy, 0.83 F1-score), while higher Dropout rates actually reduced the model's performance. This could be because excessive neuron deactivation limited the model's expressive power (Fig.14).

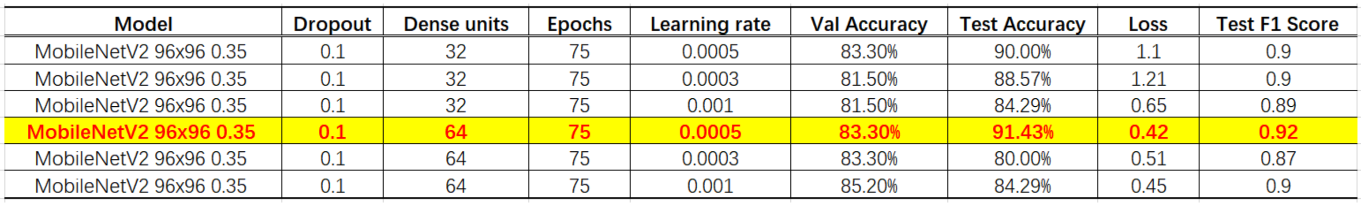
So, I tried increasing the number of neurons in the Dense layer (8, 16, 32, 64, 128) to enhance the model's expressive capability. The experimental results indicated that a Dense layer with 32 neurons achieved the highest accuracy, while 64 neurons showed better performance in terms of loss and generalization stability (Fig.14).



Model	Dropout	Dense units	Epochs	Learning rate	Val Accuracy	Test Accuracy	Loss	Test F1 Score
MobileNetV2 96x96 0.35	**0.1**	8	50	0.0005	**83.30%**	**77.14%**	**1.13**	**0.83**
MobileNetV2 96x96 0.35	0.2	8	50	0.0005	79.60%	61.43%	1.31	0.78
MobileNetV2 96x96 0.35	0.3	8	50	0.0005	75.90%	51.43%	1.42	0.78
MobileNetV2 96x96 0.35	0.1	16	50	0.0005	85.20%	85.71%	0.48	0.9
MobileNetV2 96x96 0.35	0.1	**32**	50	0.0005	83.30%	**91.43%**	1.14	0.92
MobileNetV2 96x96 0.35	0.1	**64**	50	0.0005	83.30%	**88.57%**	**0.49**	0.9
MobileNetV2 96x96 0.35	0.1	128	50	0.0005	88.90%	85.71%	0.5	0.87

Fig.14 Comparison

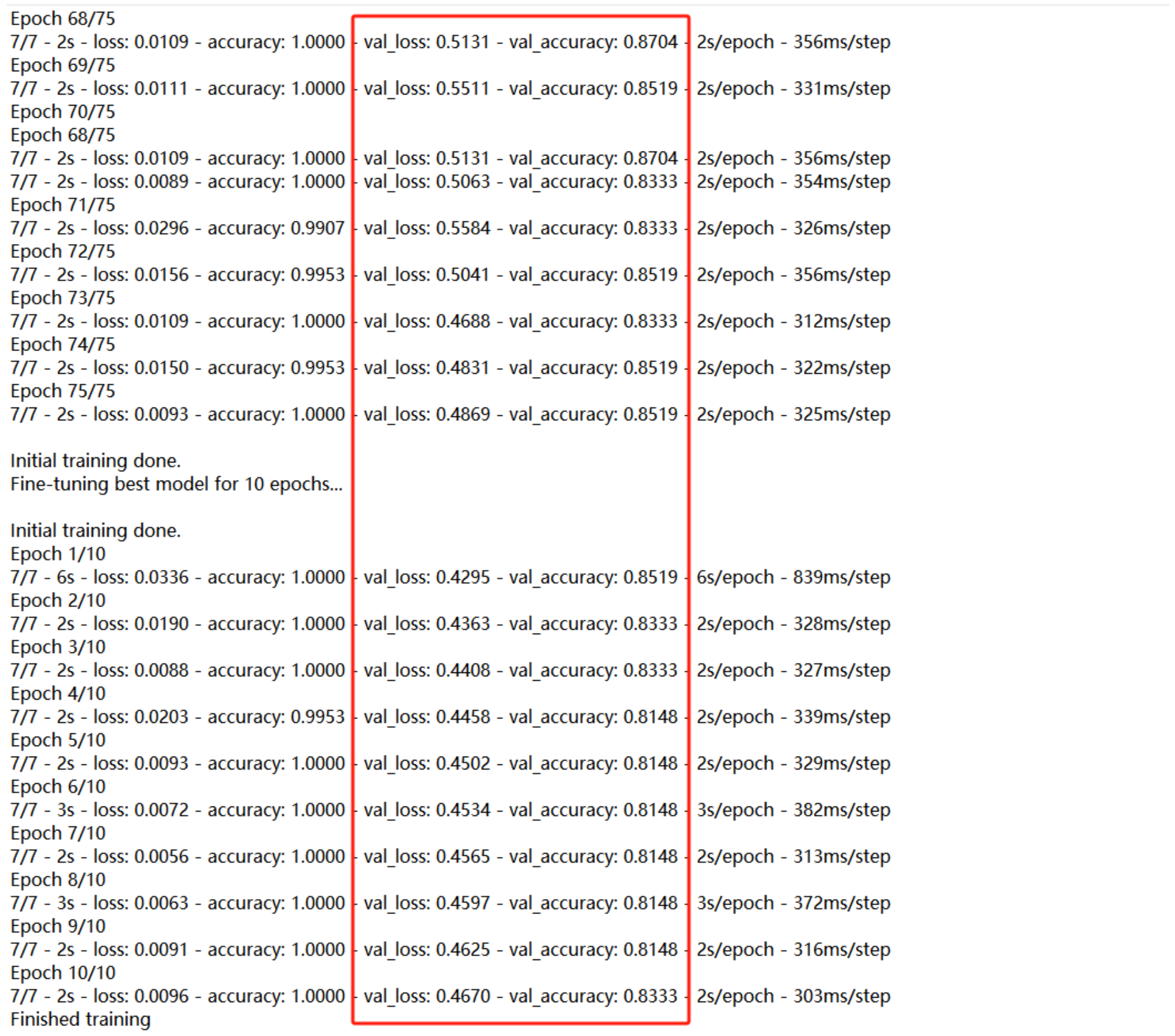
By further tuning different epoch numbers (50, 75) and learning rates (0.0003, 0.0005, 0.001) in an attempt to further converge the model and improve stability, the results showed that with Dense=32, longer training periods led to an increase in loss (e.g., loss rising to 1.21). However, with Dense=64, increasing the training epochs to 75 effectively reduced the loss and improved both validation and test performance. In terms of learning rates, 0.0003 was stable but with slightly lower accuracy, while 0.001 converged quickly but showed significant fluctuations. Considering all factors, the learning rate of 0.0005 was chosen as the best balance (Fig.15).



Model	Dropout	Dense units	Epochs	Learning rate	Val Accuracy	Test Accuracy	Loss	Test F1 Score
MobileNetV2 96x96 0.35	0.1	32	75	0.0005	83.30%	90.00%	1.1	0.9
MobileNetV2 96x96 0.35	0.1	32	75	0.0003	81.50%	88.57%	1.21	0.9
MobileNetV2 96x96 0.35	0.1	32	75	0.001	81.50%	84.29%	0.65	0.89
MobileNetV2 96x96 0.35	**0.1**	**64**	**75**	**0.0005**	**83.30%**	**91.43%**	**0.42**	**0.92**
MobileNetV2 96x96 0.35	0.1	64	75	0.0003	83.30%	80.00%	0.51	0.87
MobileNetV2 96x96 0.35	0.1	64	75	0.001	85.20%	84.29%	0.45	0.9

Fig.15 The best model

In summary, the combination of MobileNetV2 (alpha=0.35), Dropout rate of 0.1, 64 neurons in the Dense layer, 75 epochs, and a learning rate of 0.0005 achieved the best overall performance. The final model achieved 100% training accuracy, with the validation accuracy remaining stable between 81% and 87%, and the validation loss stabilized between 0.45 and 0.55, demonstrating excellent generalization performance, making it suitable for real-world deployment (Fig.16).

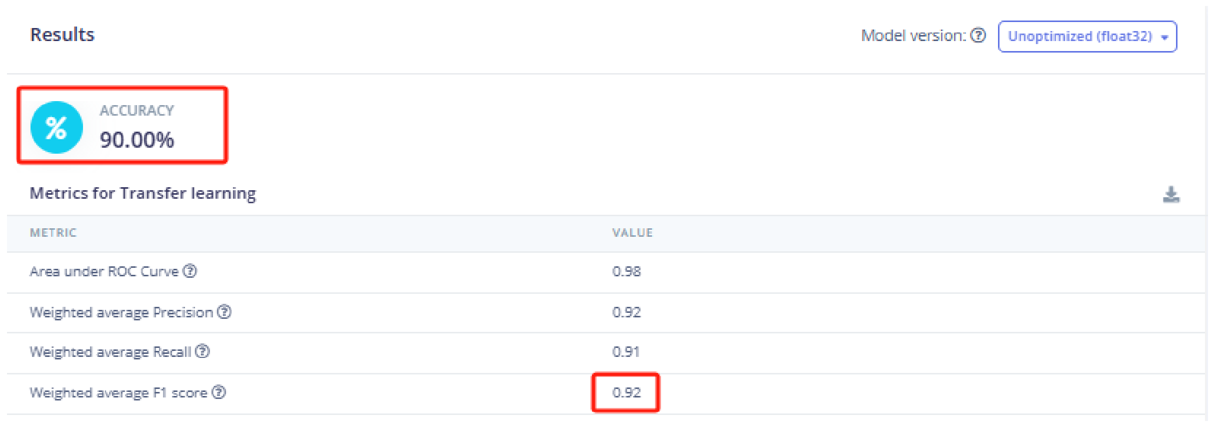


```
Epoch 68/75
7/7 - 2s - loss: 0.0109 - accuracy: 1.0000 - val_loss: 0.5131 - val_accuracy: 0.8704 - 2s/epoch - 356ms/step
Epoch 69/75
7/7 - 2s - loss: 0.0111 - accuracy: 1.0000 - val_loss: 0.5511 - val_accuracy: 0.8519 - 2s/epoch - 331ms/step
Epoch 70/75
Epoch 68/75
7/7 - 2s - loss: 0.0109 - accuracy: 1.0000 - val_loss: 0.5131 - val_accuracy: 0.8704 - 2s/epoch - 356ms/step
7/7 - 2s - loss: 0.0089 - accuracy: 1.0000 - val_loss: 0.5063 - val_accuracy: 0.8333 - 2s/epoch - 354ms/step
Epoch 71/75
7/7 - 2s - loss: 0.0296 - accuracy: 0.9907 - val_loss: 0.5584 - val_accuracy: 0.8333 - 2s/epoch - 326ms/step
Epoch 72/75
7/7 - 2s - loss: 0.0156 - accuracy: 0.9953 - val_loss: 0.5041 - val_accuracy: 0.8519 - 2s/epoch - 356ms/step
Epoch 73/75
7/7 - 2s - loss: 0.0109 - accuracy: 1.0000 - val_loss: 0.4688 - val_accuracy: 0.8333 - 2s/epoch - 312ms/step
Epoch 74/75
7/7 - 2s - loss: 0.0150 - accuracy: 0.9953 - val_loss: 0.4831 - val_accuracy: 0.8519 - 2s/epoch - 322ms/step
Epoch 75/75
7/7 - 2s - loss: 0.0093 - accuracy: 1.0000 - val_loss: 0.4869 - val_accuracy: 0.8519 - 2s/epoch - 325ms/step

Initial training done.
Fine-tuning best model for 10 epochs...

Initial training done.
Epoch 1/10
7/7 - 6s - loss: 0.0336 - accuracy: 1.0000 - val_loss: 0.4295 - val_accuracy: 0.8519 - 6s/epoch - 839ms/step
Epoch 2/10
7/7 - 2s - loss: 0.0190 - accuracy: 1.0000 - val_loss: 0.4363 - val_accuracy: 0.8333 - 2s/epoch - 328ms/step
Epoch 3/10
7/7 - 2s - loss: 0.0088 - accuracy: 1.0000 - val_loss: 0.4408 - val_accuracy: 0.8333 - 2s/epoch - 327ms/step
Epoch 4/10
7/7 - 2s - loss: 0.0203 - accuracy: 0.9953 - val_loss: 0.4458 - val_accuracy: 0.8148 - 2s/epoch - 339ms/step
Epoch 5/10
7/7 - 2s - loss: 0.0093 - accuracy: 1.0000 - val_loss: 0.4502 - val_accuracy: 0.8148 - 2s/epoch - 329ms/step
Epoch 6/10
7/7 - 3s - loss: 0.0072 - accuracy: 1.0000 - val_loss: 0.4534 - val_accuracy: 0.8148 - 3s/epoch - 382ms/step
Epoch 7/10
7/7 - 2s - loss: 0.0056 - accuracy: 1.0000 - val_loss: 0.4565 - val_accuracy: 0.8148 - 2s/epoch - 313ms/step
Epoch 8/10
7/7 - 3s - loss: 0.0063 - accuracy: 1.0000 - val_loss: 0.4597 - val_accuracy: 0.8148 - 3s/epoch - 372ms/step
Epoch 9/10
7/7 - 2s - loss: 0.0091 - accuracy: 1.0000 - val_loss: 0.4625 - val_accuracy: 0.8148 - 2s/epoch - 316ms/step
Epoch 10/10
7/7 - 2s - loss: 0.0096 - accuracy: 1.0000 - val_loss: 0.4670 - val_accuracy: 0.8333 - 2s/epoch - 303ms/step
Finished training
```

Fig.16 Training log

Results and Observations

The final deployed model achieved a test accuracy of 90.0% and a weighted F1 score of 0.92 (Fig.17), demonstrating strong overall classification performance. The confusion matrix (Fig.18) shows that each category has a high degree of differentiation, but there are still some misclassifications, particularly for the "V" category, which shares visual features similar to those of gestures like "L," resulting in a relatively higher misclassification rate. Additionally, in the actual testing on the smartphone, the "C" and "O" categories showed significantly higher misjudgment rates due to motion blur during gesture transitions, compared to the ideal scenario.



Results

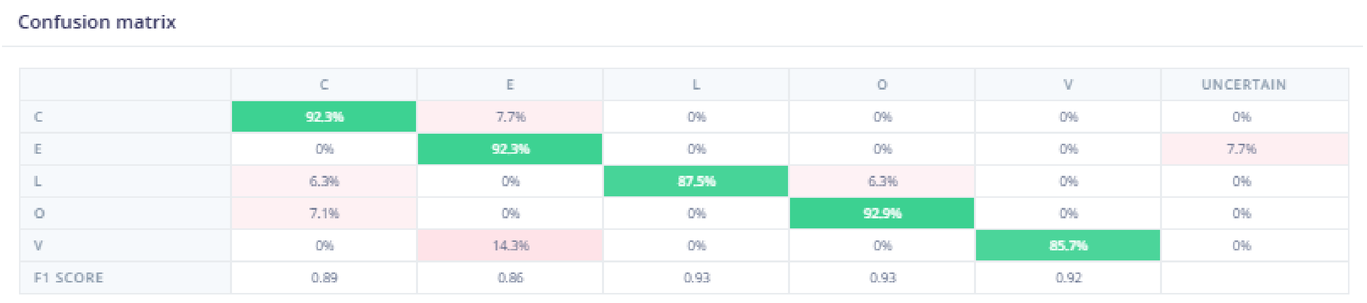
Model version: Unoptimized (float32)

ACCURACY
90.00%

Metrics for Transfer learning

METRIC	VALUE
Area under ROC Curve	0.98
Weighted average Precision	0.92
Weighted average Recall	0.91
Weighted average F1 score	0.92

Fig.17 model testing



Confusion matrix

	C	E	L	O	V	UNCERTAIN
C	92.3%	7.7%	0%	0%	0%	0%
E	0%	92.3%	0%	0%	0%	7.7%
L	6.3%	0%	87.5%	6.3%	0%	0%
O	7.1%	0%	0%	92.9%	0%	0%
V	0%	14.3%	0%	0%	85.7%	0%
F1 SCORE	0.89	0.86	0.93	0.93	0.92	

Fig.18 The confusion matrix

Despite these challenges, the data augmentation and data cleaning strategies significantly improved the model's generalization performance. Additionally, increasing the number of neurons in the Dense layer (to 64 neurons) effectively enhanced the model's feature representation capability, reducing classification errors. Sufficient training epochs (75 epochs) and an appropriate learning rate (0.0005) ensured the model trained stably and converged efficiently. The use of a suitable dropout rate (0.1) helped avoid overfitting.
If more time and resources are available, I plan to implement the following improvements:
• Dataset Expansion: Add more samples under various lighting conditions, backgrounds, and user-specific differences to enhance model robustness.
• Hard Sample Mining: Targeted training on samples near the decision boundary to improve classification margins and reduce misclassification.
• Temporal Modeling: Explore applying temporal models to video data to capture the dynamic features of gestures over time, mitigating the impact of motion blur and improving the stability and accuracy of the model in real-world applications.

Bibliography

1. Hao, W., Hou, C., Zhang, Z., Zhai, X., Wang, L. and Guanghao Lv (2024). A sensing data and deep learning-based sign language recognition approach. Computers & electrical engineering, 118, pp.109339–109339. doi:https://doi.org/10.1016/j.compeleceng.2024.109339.

2. Toshpulatov, M., Lee, W., Jun, J. and Lee, S. (2025). Deep learning pathways for automatic sign language processing. Pattern Recognition, 164, p.111475. doi:https://doi.org/10.1016/j.patcog.2025.111475.

3. Sistem Isyarat Bahasa Indonesia SIBI2 (2024). Sibi Sign Language2 Dataset. Roboflow. Available at: https://universe.roboflow.com/sistem-isyarat-bahasa-indonesia-sibi2/sibi-sign-language2 [Accessed 29 Apr. 2025].

Declaration of Authorship

I, AUTHORS NAME HERE, confirm that the work presented in this assessment is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

JINMING XIA

ASSESSMENT DATE 29/04/2025

Word count:1549