International Journal of Advances in Data and Information Systems
Vol. 6, No. 2, August 2025, pp. 320~331
ISSN: 2721-3056, DOI: 10.59395/ijadis.v6i2.1392

Indonesian Sign Language (*BISINDO*) Classification Using Xception Transfer Learning Architecture

Meisya Vira Amelia[1], Wahyu Syaifullah Jauharis Saputra[2], Kartika Maulida Hindrayani[3], Prismahardi Aji Riyantoko[4]
[1,2,3]Department of Data Science, Universitas Pembangunan Nasional "Veteran" East Java, Indonesia
[4]Department of Information and Communication Systems, Okayama University, Japan

Article Info

Article history:

Received May 08, 2025
Revised May 28, 2025
Accepted Jul 12, 2025

Keywords:

BISINDO
Fine Tuning
Sign Language
Transfer Learning
Xception

ABSTRACT

Human communication generally relied on speech. However, this was not applicable to the deaf people, who depended on sign language for daily interactions. Unfortunately, not everyone had the ability to understand sign language. In higher education environments, the lack of individuals proficient in sign language often created inequality in the learning process for deaf students. This limitation could be addressed by fostering a more inclusive environment, one of which was through the implementation of a sign language translation system. Therefore, this study aimed to develop a machine learning model capable of detecting and translating Indonesian Sign Language (*BISINDO*) alphabet gestures. The model was built using the Xception transfer learning method from Convolutional Neural Networks (CNN). The dataset consisted of 26 *BISINDO* alphabet gestures with a total of 650 images. The model was evaluated using K-Fold cross-validation and achieved an F1-score of 94% during testing.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Wahyu Syaifullah Jauharis Saputra,
Department of Data Science,
Universitas Pembangunan Nasional "Veteran" East Java,
Jl. Rungkut Madya, Gn. Anyar, Kec. Gn. Anyar, Surabaya, Jawa Timur 60294.
Email: wahyu.s.j.saputra.if@upnjatim.ac.id

1. INTRODUCTION

Human daily life cannot be separated from the activity of communicating with other humans. This is because communication is an activity carried out by humans to receive and convey messages or news with the aim that the message to be conveyed can be understood [1]. Generally, humans rely on their voices to talk to other people. However, this does not apply to deaf people.

Based on data from the Statistics Indonesia in 2022, there were 4,956,814 Indonesians who had hearing problems and 3,234,964 people who had difficulty speaking [2], [3]. According to the Disability Student Services page managed by the Ministry of Education, Culture, Research, and Technology, there are only 27 universities that accept students with hearing disabilities out of a total of 2,937 universities [4]. This shows that there are still very few universities that provide an inclusive environment, especially for deaf students.

Creating an inclusive environment in a university environment can be done in various ways. Providing sign language translators is one of them. Active student participation in academic

activities can be increased with the help of translators [5]. This is a step forward in creating a more inclusive environment in the higher education environment [6].

Technological advancements in computing had simplified many aspects of life, including the development of sign language translators [7], [8]. Sign language translators can be created by using machine learning. Machine learning is a branch of Artificial Intelligence that gives computers the ability to learn from data [9]. Machine learning models are built using algorithms, one of the algorithms that is often used is the Convolutional Neural Network (CNN) [10]. CNN is often used to process image data because of its good accuracy in classifying images [11]. In addition, CNN has many pre-trained models that can streamline the process of creating new models.

Previous studies, such as those by Rachmawati et al [12], 4 pre-trained models from CNN were used, ResNet50, EfficientNetB1, MobileNetV4, and InceptionV3. The data used were images of the BISINDO alphabet movement consisting of 1,727 images. ResNet50 and MobileNetV4 became the best models that were successfully obtained with accuracy values in the training phase of 99% and 98% and loss values of 0.03 and 0.06. These values are more stable and superior compared to EfficientNetB1 and InceptionV3 which have accuracy values of 8% and 96% with loss values of 3.40 and 0.12. Another study by Sari et al [13], Random Forest used as the classification model and InceptionV3 for the feature extraction process. The data used were images of SIBI (another Indonesian Sign Language System) alphabet with a total of 5280 images. The model was evaluated using K-Fold Cross Validation and obtained an accuracy value of 85.40% and an F1-score of 85.20%. Both studies used InceptionV3 in creating their models. However, its performance was still less superior compared to other pre-trained models.

[14] Francois Chollet created a new architecture using a stronger hypothesis base than the hypothesis underlying the creation of the Inception architecture. The hypothesis used is that the mapping of cross-channel correlation and spatial correlation in the Convolutional Neural Network feature map can be completely separated. This architecture was later called Extreme Inception or Xception. Xception outperforms InceptionV3 on several classification tasks, including ImageNet and a larger internal dataset with 17,000 classes [11].

Based on this, this study focuses on exploring the application of the Xception transfer learning architecture to the BISINDO alphabet dataset. This study will explore the implementation of the pre-trained Xception model on the BISINDO alphabet dataset.

2. RESEARCH METHOD

The methods used in this study are broadly collecting datasets, pre-processing datasets, pre-processing models, training models, and evaluating them with test sets. Figure 1 illustrates the research methods used.

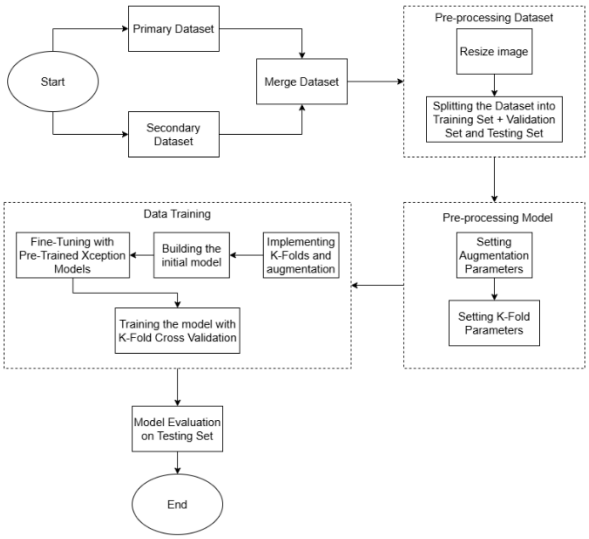


Figure 1. Research method flowchart

There are two sources of datasets used in this study, secondary and primary datasets. The secondary dataset was taken from the Kaggle website created by Achmad Noer [15]. The total number of images available is 312 images, each class has 12 images. The images have 3 types of backgrounds, a plain white shirt, a white wall, and a white shirt with spots.

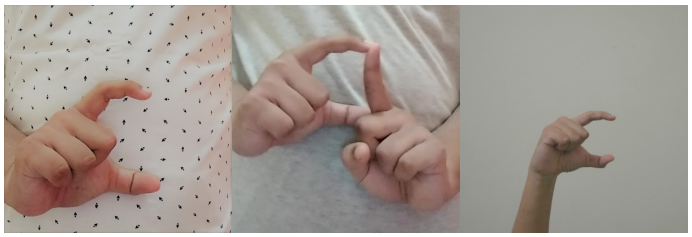


Figure 2. Secondary dataset image samples

The primary dataset was taken manually by recording the movement of the BISINDO alphabet using a smartphone, then cutting each frame and selecting the images to be used. The total number of images used is 338 images and each class has 13 images. These two datasets are then combined into one, resulting in a total of 650 images with each class having 26 images. Based on this, the dataset is categorized into a multi-class classification. Multi-class classification is similar to binary classification, except that the target class is non-binary [16]. The dataset used in this study has been uploaded and can be accessed on Kaggle [17].



Figure 3. Primary dataset image samples

To minimize the presence of noise and reduce the risk of domain shift, it is crucial to ensure that all images within the dataset exhibit consistent attributes. Such uniformity enhances the model's ability to generalize effectively. The dataset is structured such that each class is represented by a separate directory. All images are resized to a fixed resolution of 299×299 pixels, converted into NumPy arrays, and subsequently normalized using the preprocess_input function, which scales pixel values to the range of [-1, 1] in accordance with the input requirements of pretrained models like Xception. Class labels are then encoded using a one-hot representation. The dataset is split into 80% training and validation set and 20% test set using stratified sampling to maintain class balance. The amount of data obtained for the training and validation set is 520 images and the test set is 130 images. Later, K-Fold cross-validation will be applied only to the training and validation set.

To address the limitation of having a small dataset, data augmentation was applied. Data augmentation was implemented through the application of randomized transformations, including rotations of up to ±20 degrees, horizontal and vertical shifts of up to 20%, zoom variations of 10%, and brightness adjustments ranging from 80% to 120%. These augmentation techniques were applied on-the-fly during training using the ImageDataGenerator method. In this context, augmented images were not explicitly stored or generated in advance; instead, each training batch and epoch introduced continuous random variations of the original images. The model was trained over 100 epochs using 520 original images. Mathematically, the total number of virtual samples seen by the model during training can be approximated by multiplying the number of epochs by the number of training samples, yielding approximately 52,000 image variations. This indicates that although the original dataset was limited to 520 images, the model was theoretically exposed to

52,000 augmented variations throughout the training process. This extensive exposure significantly enhances the model's ability to generalize to unseen data [17–19].

In the model training phase, the transfer learning architecture used for model training is Xception. Xception is one of the deep learning architectures and is an improvement on the Inception architecture developed by Google researchers. Xception was developed based on the hypothesis that cross-channel correlation mapping and spatial correlation on the Convolutional Neural Network feature map can be completely separated. This hypothesis is a stronger version than the hypothesis underlying the creation of the Inception architecture, therefore they named the architecture created from this hypothesis as Extreme Inception or Xception [14].

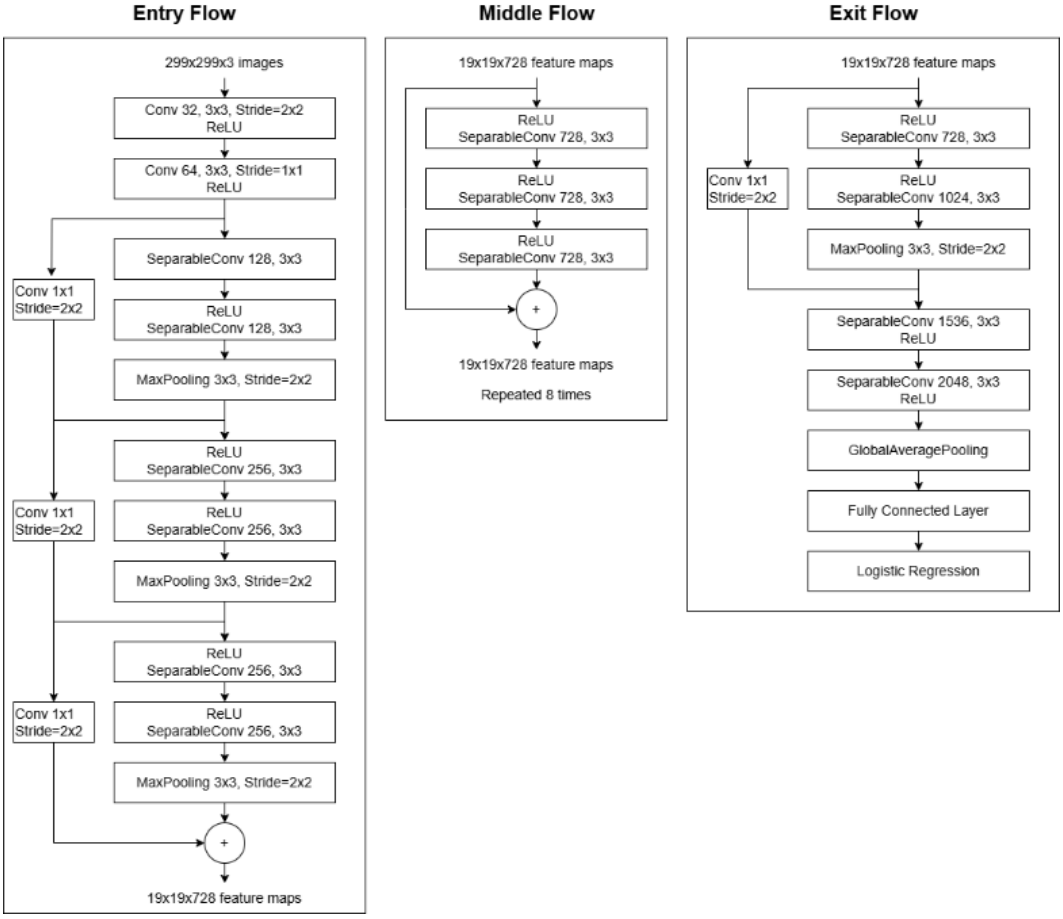


Figure 4. The Xception Architecture

The training process is conducted over a total of 100 epochs, comprising an initial training phase of 50 epochs followed by an additional 50 epochs dedicated to fine-tuning. To assess the effect of different fine-tuning strategies, two model configurations are evaluated. The primary distinction between them lies in the number of layers unfrozen during fine-tuning. The first model fine-tunes the final 56 layers, while the second model begins fine-tuning from the last 26 layers. Prior to entering the initial training phase, both models undergo a preparatory stage in which only the last four layers are trained, allowing the classifier to adapt to the new dataset without altering the pretrained feature extractor. Consequently, during the fine-tuning phase, the first model updates the weights of layers 71 through 126, whereas the second model updates layers 101 through 126. All earlier layers remain frozen to retain the general feature representations learned during pretraining. Throughout all training stages, a consistent batch size of 32 is maintained.

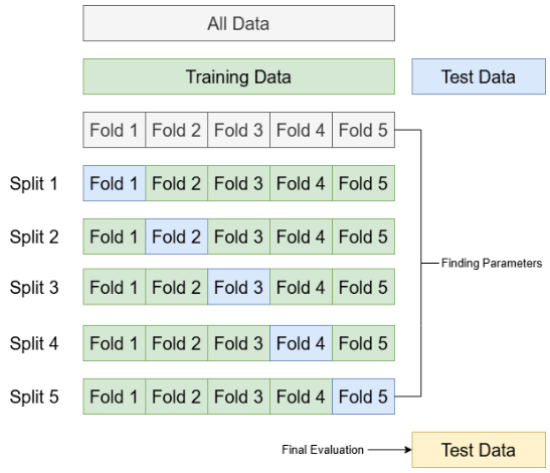


Figure 5. K-Fold Cross Validation Workflow

Model training was performed using the K-Fold Cross Validation method. K-Fold Cross Validation was chosen as an evaluation technique in this study because of the small number of datasets and to reduce the risk of bias in the data during model training [21]. K-Fold Cross Validation works by breaking the data in the training and validation sets into 'k' parts of the data set with the same amount of data divided. In this study, the k value set is 5, meaning there will be 5 folds. K-Fold Cross Validation workflow is shown in Figure 5.

Table 1. Model training parameters

Parameter	Detail
Transfer Learning Architecture	Xception
Number of Layer Training	The last 4 layers, before the initial model The last 56 and 26 layers, after the initial model
Optimizer	Adam
Loss Function	Categorical Crossentropy
Epoch	100

To minimize the model's loss, the Adam optimizer is employed. A learning rate of 0.0001 (1e-4) is used during the initial training phase, and it is reduced to 0.00001 (1e-5) during the fine-tuning phase. Categorical Cross-Entropy is used as the loss function, as the target labels have been previously transformed into one-hot encoded.

The performance of both models will be compared and analyzed, and the best-performing model will be selected. The selected model will then be further evaluated using a confusion matrix. This method provides a visualization of the classifier's performance across different classes and is independent of any specific classification algorithm [22]. The fundamental metrics used are derived from the prediction outcomes, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) [23]. Based on these values, the following core evaluation metrics are calculated:

a. Accuracy: Shows the percentage of correctly classified classes.

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

b. Precision: Shows the proportion of correct positive predictions out of the total predicted positive events.

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

c. Recall: Shows the proportion of correct positive predictions out of actual positive events.

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

d. F1-score: Shows the harmonic mean between precision and recall, providing a balanced measure between the two.

$$F1 - score = \frac{2TP}{2TP+FP+FN} \quad (4)$$

Once the model is evaluated, it will be deployed to predict images in real-time. The model deployment is carried out using the Streamlit framework, which is a popular Python library for quickly building interactive web interfaces based on machine learning. This deployment process is executed locally, meaning the application is not uploaded to an online server.

3. RESULTS AND DISCUSSION

This section presents and analyzes the results obtained from model training and evaluation on the test data. The outcomes of the training process are summarized in Table 2.

Table 2. Performance of Models

Model	Fine-tune	Fold	Train acc	Val acc	Train loss	Val loss	Val f1-score	Avg f1	Avg acc	Avg loss
1	56	1	98.29%	98.08%	0.7860	0.7025	98.05%	95.69%	95.77%	0.7552
		2	99.40%	98.08%	0.7460	0.6795	98.05%			
		3	98.74%	76.4%	0.7640	0.7368	96.09%			
		4	97.68%	98.08%	0.9519	0.6795	95.02%			
		5	83.20%	92.31%	1.3037	0.8630	91.23%			
2	26	1	90.99%	97.12%	0.9760	0.7548	96.89%	96.08%	96.15%	0.7954
		2	94.21%	97.12%	0.9084	0.7381	97.10%			
		3	98.90%	95.19%	0.7606	0.7953	95.12%			
		4	94.26%	93.27%	0.9704	0.9696	93.26%			
		5	96.97%	98.08%	0.7846	0.7191	98..05%			

Overall, Model 1 demonstrated strong performance, achieving an average F1-score of 95.69% and an average accuracy of 95.77%. However, its performance exhibited considerable variability across different folds. Notably, in the third fold, the validation accuracy dropped significantly to only 76.4%, markedly lower than in the other folds. Additionally, the fifth fold indicated signs of underfitting, with training accuracy reaching only 83.20% and a notably high training loss of 1.3037. Although the model shows competitive average performance, it tends to be less stable and appears sensitive to the data partitioning used during training.

Model 2 emerged as the best-performing configuration in this study. It achieved the highest average F1-score of 96.08% and an average accuracy of 96.15%. Furthermore, the model demonstrated consistent performance across all folds, with no notably low values. There were no significant indications of overfitting or underfitting, as training and validation accuracies were relatively balanced in each fold. Compared to the model 1, its overall classification performance was considerably superior and more stable.

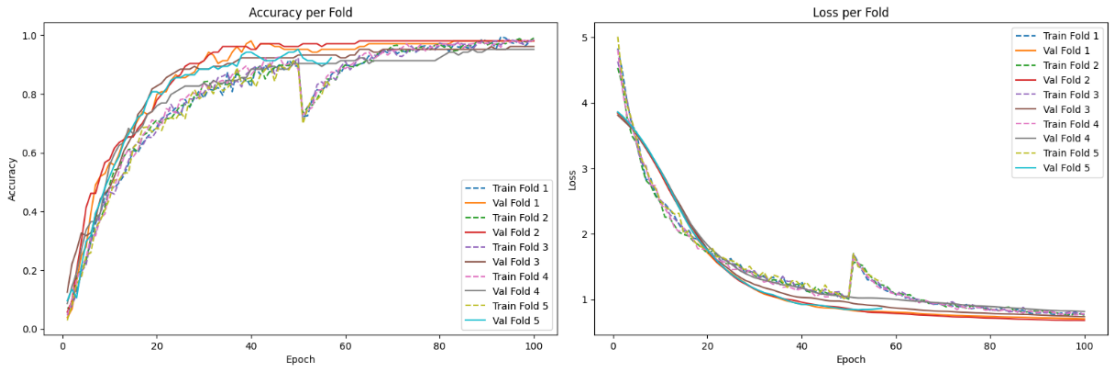


Figure 6. Accuracy and Loss Graph of Model 1

Based on the graph above, a noticeable drop in model performance occurred during the fine-tuning phase. This decline is observed between epochs 51 and 60, as illustrated in Figure 6. The performance degradation is attributed to the large number of layers involved in the fine-tuning process—in this case, the last 70 layers of the pre-trained Xception model. All model folds completed the training up to epoch 100, except for fold 5, which stopped early at epoch 57 due to early stopping.

Fine-tuning such a deep portion of the model may have introduced excessive parameter updates, leading to instability in the learning process—especially when working with a relatively small dataset. The network might have temporarily overfit to the training data or struggled to converge due to abrupt changes in the weight optimization dynamics.

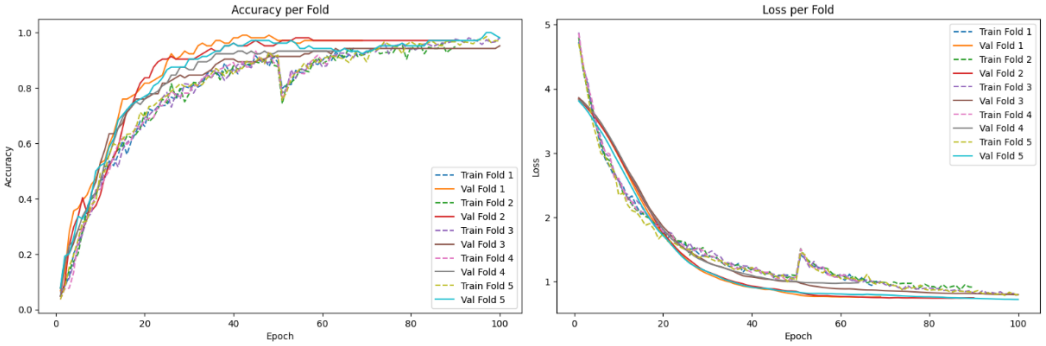


Figure 7. Accuracy and Loss Graph of Model 2

Based on the graph above, the performance drop during the fine-tuning phase was significantly less pronounced compared to Model 1, which fine-tuned 70 layers. This mild decline occurred between epochs 51 and 60, as shown in Figure 7. Only the models in folds 3 and 5 completed the full 100 training epochs, while the models in folds 1, 2, and 4 stopped earlier due to early stopping.

Based on the evaluation of all experimental configurations, the model trained using an 80:20 data split and fine-tuning of the 100 layers onwards of the pre-trained Xception architecture was selected as the best-performing model in this study. This particular configuration consistently outperformed other setups, not only in terms of achieving the highest average metrics—such as accuracy and F1-score—but also in demonstrating strong consistency and stability across all folds in the K-Fold cross-validation. One of the key strengths of this configuration lies in its ability to maintain reliable performance regardless of the specific data partition used during training. This suggests a high degree of generalization, which is particularly important when dealing with real-world data that may vary in background, lighting, and gesture variation.

Among the five folds evaluated within this setup, the fifth fold emerged as the most optimal, delivering the best validation results. Specifically, the model from this fold achieved a validation accuracy of 98.08%, an F1-score of 0.9805, and a low validation loss, indicating both precision and robustness. These results reflect the model's effectiveness in capturing relevant features from the BISINDO hand gesture dataset while avoiding overfitting. Given these compelling results, the model from fold five was selected as the final model to be used in subsequent evaluation on the independent test set. Its strong performance and generalizability make it a suitable candidate for final testing.

Before the evaluation with the test set, the author conducted a test when the model did not apply augmentation data. This is to find out whether the augmentation parameters that have been applied have an effect in reducing overfitting. The results of the model without data augmentation are shown in table 3.

Table 3. Performance of Model without Augmentation

Fold	Train acc	Val acc	Train loss	Val loss
1	100%	99.04%	06798	0.7128
2	98.33%	100%	0.7657	0.7385
3	98.93%	93.27%	0.7587	0.8387
4	98.44%	92.31%	0.7725	0.9109
5	97.43%	97.12%	0.8065	0.8088

Compared with model 2 shown in Table 2, application of data augmentation significantly enhances the generalization capability of the model. While the model trained without augmentation achieves near-perfect training accuracy, it shows signs of overfitting, as evidenced by higher variability in validation performance across folds. In contrast, the augmented model demonstrates more consistent validation accuracy and improved F1-scores, despite slightly lower training accuracy and higher training loss. This indicates that data augmentation effectively mitigates overfitting by introducing data variability during training, leading to a more robust and reliable model when applied to unseen data. Therefore, incorporating augmentation techniques is recommended to improve model stability and performance in real-world scenarios.

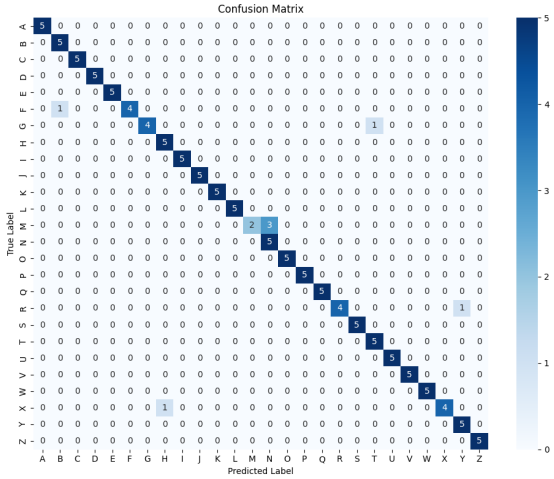


Figure 8. Confusion Matrix

Based on the confusion matrix results shown in Figure 8, a total of 7 images from 5 different letters were misclassified. The misclassifications are as follows:

- The letter F was misclassified as B (1 image),
- The letter G was misclassified as T (1 image),
- The letter M was misclassified as N (3 images),
- The letter R was misclassified as Y (1 image), and
- The letter X was misclassified as H (1 image).

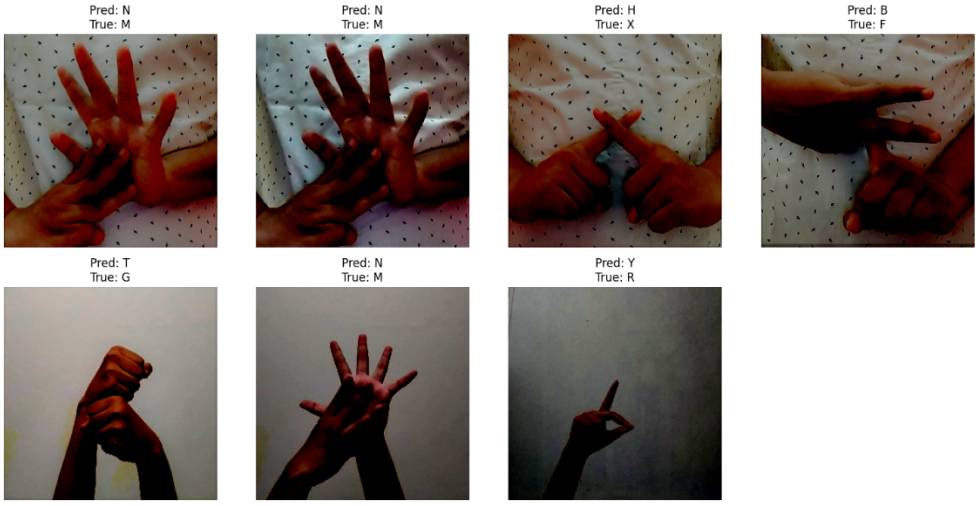


Figure 9. Misclassified Images

Figure 9 offers several insights into the possible factors contributing to the model's misclassifications. One of the most prominent cases involves the letter M, which was frequently confused with the letter N. The hand gestures representing these two letters are inherently similar in

shape and orientation, which may have caused ambiguity for the model during inference. Out of five test images labeled as M, only two were correctly classified, indicating that the model has difficulty when distinguishing between signs that have only subtle visual differences. This challenge underscores a broader limitation in gesture-based recognition models, where fine-grained variations in hand posture, finger positioning, or orientation may not be sufficiently captured, especially when training data is limited.

Another contributing factor to the misclassification appears to be suboptimal lighting conditions. All of the incorrectly classified images exhibited lower brightness levels compared to the original, unaugmented training samples. These images were part of the augmented dataset, designed to increase robustness by simulating real-world variations. However, in this case, the model's reduced ability to correctly interpret the gesture under dim lighting suggests that it may not have fully learned lighting-invariant features. Low illumination likely obscured critical details in the hand region, making it more difficult for the model to detect edges and contours that are essential for accurate gesture recognition.

In addition to gesture similarity and lighting, background context may have also played a role. Upon closer inspection of Figure 9, it was observed that only two types of backgrounds were present in the misclassified images: one involving a subject wearing a black-and-white polka dot shirt, and another with a plain white background. The polka dot pattern, in particular, may have introduced visual noise or interference, potentially distracting the model's attention away from the hand region. Such background complexity can mislead the feature extraction process, especially if the model was not sufficiently trained to isolate hand features from patterned or cluttered scenes.

These findings collectively indicate that while the model performs well under standard conditions, its robustness may be challenged by subtle gesture variations, poor lighting, and visually complex backgrounds. This highlights the need for more diverse training data and possibly the integration of preprocessing techniques or attention mechanisms to better focus on relevant gesture features. The next part of the figure also includes an example of a correctly classified image, serving as a contrast to highlight the types of input conditions in which the model performs effectively.

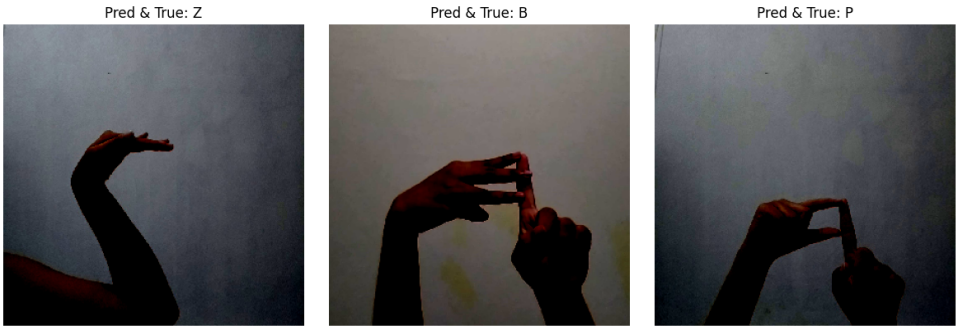


Figure 10. Correctly Classified Image

As shown in Figure 10, the model is still capable of making correct predictions even under relatively low lighting conditions. This indicates a degree of robustness to variations in illumination. The following section presents the performance of the final model as evaluated on the test set. The model performance obtained after being evaluated with the test set is as follows:

Table 4. Model evaluation on test set

Metric	Performance
Accuracy	94.62%
Precision	95.99%
F1-score	94.36%
Recall	94.62%
Loss	0.2003

The model performance on the test set showed results with an accuracy value of 94.62%, F1-score 94.36%, precision 95.99%, recall 94.62%, and loss 0.2003. The final model demonstrated excellent and reliable performance, achieving both accuracy and F1-score above 94%. Although a few misclassifications occurred for certain letters such as M and N, the overall error distribution was minimal and non-systematic. This indicates that the model has a strong ability to generalize to new, unseen data. Considering both the best K-Fold performance and the final test results, the selected model for final prediction was the one trained under the 80:20 data split scenario with fine tune 26 layers, specifically from the fifth fold. This model yielded the highest average accuracy and F1-score among all tested configurations.

After performing the evaluation, the next step is model deployment. The model deployment is done in real-time using a camera. The system captures images directly from the user's device camera and processes them to classify them into specific categories.



Figure 11. Real-time Image Prediction

Based on Figure 11, it can be seen that the model successfully predicts the image in real-time accurately. In the image, the author demonstrates the sign language gesture for the letter N. The model's prediction results are displayed at the top left corner of the image. Two values are shown: the predicted letter and the model's confidence level in its prediction. In this case, the model correctly predicted the letter N with a high confidence of 95%. This indicates that the model was able to accurately predict an image outside of the dataset.

Although previous studies reported slightly higher performance using models such as InceptionV3, particularly with training accuracies reaching up to 96%, the differences in results can largely be attributed to the disparity in dataset size. For instance, prior work utilized larger datasets, which naturally contribute to more stable training and better model generalization. In contrast, the model developed in this study, using the Xception architecture, achieved a test accuracy of 94.62%, F1-score of 94.36%, precision of 95.99%, and recall of 94.62%, despite being trained on a comparatively smaller dataset. These results highlight that the proposed model performs competitively even under more limited data conditions, and that the use of Xception offers strong representational capability for sign language classification tasks.

4. CONCLUSION

In conclusion, the Xception-based model demonstrated strong performance in classifying Indonesian Sign Language (BISINDO) gesture images, even when trained on a relatively limited dataset. Among the two fine-tuning strategies evaluated, the model that fine-tuned the last 26 layers yielded the best overall results, achieving a test accuracy of 94.62%, F1-score of 94.36%, precision of 95.99%, recall of 94.62%, and a loss value of 0.2003. These findings indicate that selectively fine-tuning a smaller subset of the model's deeper layers can be more effective in preventing overfitting while still allowing sufficient adaptation to the target dataset. Furthermore, the performance of the Xception model was competitive compared to prior studies, especially when considering the differences in dataset size, thereby validating its suitability for sign language recognition tasks under constrained data conditions.

REFERENCES

[1] T. M. Milyane and Dkk, *Pengantar Ilmu Komunikasi*, vol. 5, no. 3. 2022. [Online]. Available: https://repository.penerbitwidina.com/media/publications/557082-pengantar-ilmu-komunikasi-22ec77af.pdf

[2] BPS, "Penduduk Menurut Wilayah dan Tingkat Kesulitan Mendengar," Badan Pusat Statistik. Accessed: Sep. 13, 2024. [Online]. Available: https://sensus.bps.go.id/topik/tabular/sp2022/145/0/0

[3] BPS, "Jumlah Penduduk Kesulitan Berbicara," Badan Pusat Statistik. Accessed: Sep. 13, 2024. [Online]. Available: https://sensus.bps.go.id/topik/tabular/sp2022/146/1/0

[4] LMD, "Sebaran Mahasiswa Disabilitas," Layanan Mahasiswa Disabilitas. Accessed: Sep. 13, 2024. [Online]. Available: https://ptinklusif.kemdiktisaintek.go.id/s/5/sebaran-mahasiswa-disabilitas

[5] Effranzudeta, Yesinovitasari, Herdi, and Dinafitriani, "Journal of Special Education Lectura Urgensi Bahasa Isyarat di Lingkungan Universitas," vol. 2, no. 1, pp. 65–72, 2024.

[6] D. A. Prasetya and T. P. Anggriawan, "Implementation of Information Technology to Enhance Institutional Presence," in *Nusantara Science and Technology Proceedings*, 2025, pp. 242–245.

[7] W. S. J. Saputra, M. H. Fardana, and M. A. R. Valentino, "Implementasi AI Pendeteksi Pola Gerak Tangan pada Game 'Pong Ball' dengan Menggunakan Algoritma Convolutional Neural Network," *J. Manajamen Inform. Jayakarta*, vol. 2, no. 3, p. 235, 2022, doi: 10.52362/jmijayakarta.v2i3.834.

[8] D. A. Prasetya, A. P. Sari, P. A. Riyantoko, and T. M. Fahrudin, "The Effect of Information Quality and Service Quality on User Satisfaction of the Government of Kabupaten Malang," *TIERS Inf. Technol. J.*, vol. 4, no. 1, pp. 32–42, 2023, doi: 10.38043/tiers.v4i1.4328.

[9] T. M. Fahrudin, P. A. Riyantoko, K. M. Hindrayani, and E. M. Safitri, "An Introduction To Machine Learning Games And Its Application For Kids In Fun Project," *Int. J. Comput. Netw. Secur. Inf. Syst.*, vol. 2, no. 1, pp. 26–30, 2020, [Online]. Available: https://machinelearningforkids.co.uk

[10] M. Idhom, D. A. Prasetya, P. A. Riyantoko, T. M. Fahrudin, and A. P. Sari, "Pneumonia Classification Utilizing VGG-16 Architecture and Convolutional Neural Network Algorithm for Imbalanced Datasets," *TIERS Inf. Technol. J.*, vol. 4, no. 1, pp. 73–82, 2023, doi: 10.38043/tiers.v4i1.4380.

[11] S. Shania, M. Farid Naufal, V. Riandaru Prasetyo, and M. S. Bin Azmi, "Translator of Indonesian Sign Language Video using Convolutional Neural Network with Transfer Learning," *Indones. J. Inf. Syst.*, vol. 5, no. 1, pp. 17–27, 2022, doi: 10.24002/ijis.v5i1.5865.

[12] I. D. A. Rachmawati, R. Yunanda, M. F. Hidayat, and P. Wicaksono, "Deep Transfer Learning for Sign Language Image Classification: A Bisindo Dataset Study," *Eng. Math. Comput. Sci. J.*, vol. 5, no. 3, pp. 175–180, 2023, doi: 10.21512/emacsjournal.v5i3.10621.

[13] M. Sari and E. R. Jamzuri, "Hand Sign Recognition of Indonesian Sign Language System (SIBI) Using," vol. 5, no. 158, pp. 258–265, 2026.

[14] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, vol. 7, no. 3, pp. 1251–1258, 2017, doi: https://doi.org/10.48550/arXiv.1610.02357.

[15] A. Noer, "Bahasa Isyarat Indonesia (BISINDO) Alphabets." [Online]. Available: https://www.kaggle.com/datasets/achmadnoer/alfabet-bisindo

[16] A. Muhaimin, W. Wibowo, and P. A. Riyantoko, "Multi-label Classification Using Vector Generalized Additive Model via Cross-Validation," *J. Inf. Commun. Technol.*, vol. 22, no. 4, pp. 657–673, 2023, doi: https://doi.org/10.32890/jict2023.22.4.5.

[17] M. Vira, "Abjad Bahasa Isyarat Indonesia (BISINDO)," Kaggle. [Online]. Available: https://www.kaggle.com/datasets/meisyavira/abjad-bahasa-isyarat-indonesia-bisindo

[18] R. Balestriero, I. Misra, and Y. LeCun, "A Data-Augmentation Is Worth A Thousand Samples: Exact Quantification From Analytical Augmented Sample Moments," 2022,

[Online]. Available: http://arxiv.org/abs/2202.08325
[19] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, "Image Data Augmentation for Deep Learning: A Survey," 2022, doi: 10.13328/j.cnki.jos.007263.
[20] D. A. Prasetya, A. P. Sari, M. Idhom, and A. Lisanthoni, "Optimizing Clustering Analysis to Identify High-Potential Markets for Indonesian Tuber Exports," *Indones. J. Electron. Electromed. Eng. Med. Informatics*, vol. 7, no. 1, pp. 113–122, 2025.
[21] R. Tuntun, K. Kusrini, and K. Kusnawi, "Analisis Perbandingan Kinerja Algoritma Klasifikasi dengan Menggunakan Metode K-Fold Cross Validation," *J. Media Inform. Budidarma*, vol. 6, no. 4, p. 2111, 2022, doi: 10.30865/mib.v6i4.4681.
[22] J. Xu, Y. Zhang, and D. Miao, "Three-way confusion matrix for classification: A measure driven view," *Inf. Sci. (Ny).*, vol. 507, pp. 772–794, 2020, doi: 10.1016/j.ins.2019.06.064.
[23] C. Tejada, G. Espinoza, and D. Subauste, "Use of Xception Architecture for the Classification of Skin Lesions," *J. Syst. Cybern. Informatics*, vol. 22, no. 3, pp. 20–25, 2024, doi: https://doi.org/10.54808/JSCI.22.03.20.