MJI
Media Jurnal Informatika
Vol. 17, No. 2, December 2025
p-issn : 2088-2114, e-issn : 2477-2542

Development Of A CNN Model For Recognizing The Indonesian Sign Language (BISINDO) Alphabet

Sinta Siti Nuriaha,1,*, Ahmad Zamakhsyari Sidiq b,2, Zulkaida Akbar b,3
a Sekolah Tinggi Teknologi Cipasung, Padakembang, Tasikmalaya and 46417, Indonesia
b Sekolah Tinggi Teknologi Cipasung, Padakembang, Tasikmalaya and 46417, Indonesia
c Sekolah Tinggi Teknologi Cipasung, Padakembang, Tasikmalaya and 46417, Indonesia
1 sinta@sttcipasung.ac.id*; 2 ade@sttcipasung.ac.id; 3zulkaida@sttcipasung.ac.id
*sinta@sttcipasung.ac.id

ARTICLE INFO

Article history
Received 2025/08/06
Revised 2025/11/29
Accepted 2025/12/03

Keywords
Indonesian Sign Language (BISINDO)
Convolution Neural NetworkI (CNN)
Deep Learning
Computer Vision
Sign Language Interpreter

ABSTRACT

Deaf people in Indonesia face communication barriers due to the limited understanding of Indonesian Sign Language (BISINDO) among the general public. This results in limited social interaction between deaf people and their surroundings. This study aims to develop a Deep Learning and Computer Vision-based BISINDO alphabet translator model using the Convolutional Neural Network (CNN) method, addressing the limited availability of publicly documented BISINDO datasets for alphabet recognition . The method used involves training the model with a dataset of 3,120 BISINDO alphabet images, covering the letters A to Z. The dataset was divided into 80% for training and 20% for testing. The training process included model architecture design, parameter tuning, selection of the best model based on accuracy, and performance evaluation. The evaluation results showed that the developed CNN model achieved an accuracy of 99.84% in classifying BISINDO letters; however, challenges remain in generalizing the model to variations in lighting, hand orientation, and user differences. Nevertheless, the high accuracy achieved indicates the model's potential to support effective BISINDO translation and improve communication accessibility. This research also opens up opportunities for further development towards comprehensive translation of gestures or sentences in BISINDO.

1. Introduction

According to the 2020 Population Census, Indonesia has a population of approximately 270.2 million (1) or around 3.4% of the world's population. Indonesia ranks fifth in the world in terms of population size, specifically in fourth place after the United States (2). Approximately 8.5% or around 22.97 million people have difficulty taking care of themselves (people with disabilities), which is categorized into several limitations, including physical, sensory, intellectual, and mental impairments (3). Persons with disabilities have the same rights and treatment as citizens and nationals, such as the right to life, education, employment, entrepreneurship and cooperatives, expression, communication, and access to information in accordance with Law No. 8 of 2016. However, these rights cannot be exercised normally, resulting in hindered interaction between ordinary citizens and persons with disabilities.

Sensory limitations, specifically hearing impairment or deafness, are the main focus of this study, as deaf individuals rely on sign language to communicate and express their thoughts in a way that can be understood by the general public (4). Indonesia has two types of sign language, namely the Indonesian Sign Language System (SIBI) and Indonesian Sign Language (BISINDO) (5). SIBI is commonly used in Special Schools (SLB); however, it is considered difficult for many deaf users to understand, as it is adapted from the structure of American Sign Language (ASL) (6). In contrast, BISINDO is widely used among the deaf community in Indonesia and was first officially recognized during the congress of the Movement for the Welfare of Deaf People in Indonesia (GERKATIN). Despite its widespread use, BISINDO remains underrepresented in digital resources, with a scarcity of publicly available and well-annotated datasets, particularly for computer vision–based sign language recognition. This limitation motivates the selection of BISINDO as the focus of this study.

Understanding sign language is considered difficult, as this skill generally requires special guidance from a trainer (4). Because of this, efforts to learn sign language are hampered, resulting in many people not knowing or understanding the use of sign language. The use of Deep Learning and Computer Vision is considered to be a solution, as these technologies have dominated several fields such as business analysis, medical diagnosis, art creation, and image translation (7).

Similar research using the CNN method has also been conducted in (4) entitled Sign Language Recognition Using The Fusion Of Image and Hand Landmark Through Multi-Headed Convolutional Neural Network. This research demonstrates a cost-effective technique for detecting American Sign Language (ASL). For testing the detection of American Sign Language, an accuracy value of 96.46% was obtained. Another related study that applied the Convolutional Neural Network method by (8) entitled Data Augmentation to Overcome Data Limitations in the Indonesian Sign Language Translation Model (BISINDO) produced a BISINDO alphabet translation model using the implementation of the CNN algorithm with an accuracy of 94.38% in an effort to improve accessibility for deaf BISINDO users.

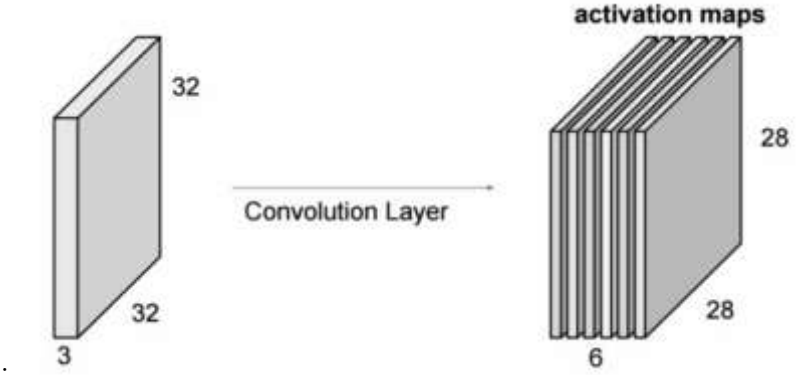
2. Method

The field of computer vision (CV) has been significantly influenced by neural network technology, particularly Convolutional Neural Networks (CNNs) (7). CNNs are specifically designed to process visual data such as images and are capable of learning highly abstract features through hierarchical representation (9)(12)(18). This capability makes CNNs highly effective for image classification, object identification, and pattern recognition tasks (8)(9). Furthermore, the use of weight-sharing mechanisms enables CNNs to achieve better generalization with fewer parameters, thereby reducing the risk of overfitting compared to conventional models (9)(10)(11). In CNN architectures, feature extraction and classification are integrated into a unified learning process, allowing the model to be trained end-to-end (9)(13). In general, a CNN model consists of four main components: (a) convolution layers for feature extraction, (b) pooling layers for spatial downsampling, (c) activation functions for introducing non-linearity, and (d) fully connected layers for classification (9).

Figure 1. Basic CNN Components.

2.1 Convolution Layer

CNN consists of convolutional layers, which filter layer inputs to identify useful attributes. The parameters contained within the convolutional layer are the layer kernel weights. Two-dimensional convolution is used in image processing to implement image filters, for example, to find specific patches in an image or to find several features in an image. In CNNs, convolutional layers filter input tensors in a manner similar to tiles with small windows known as kernels (5). Kernels precisely define what is filtered by the convolution operation and will produce a strong response when finding what was previously sought (5). During CNN training, filter values are automatically adjusted to extract the most useful information. However, to avoid losing all important information after the vector is sent to the fully connected layer, traditional neural networks need to convert the input data into a 1-D vector. In addition, each neuron has different parameters for each pixel, which causes models with large inputs to have many parameters (5). Figure 2 shows the input and output of the six layers of this special filter convolution.



:

Figure 2. Convolution Layer (5).

2.2 Pooling Layer

Pooling Layer is a type of layer commonly used in convolutional neural networks. This layer only takes a lower sample from each feature map created by the convolutional operation; for the most commonly used pooling size of 2, this involves mapping each 2 x 2 section of each feature map to either the maximum value of that section, in the case of maximum pooling, or the average value of that section, in the case of average pooling. For an n x n image, this will map the entire image to an image of size $\frac{n}{2} x \frac{n}{2}$ (14). See figure 3.

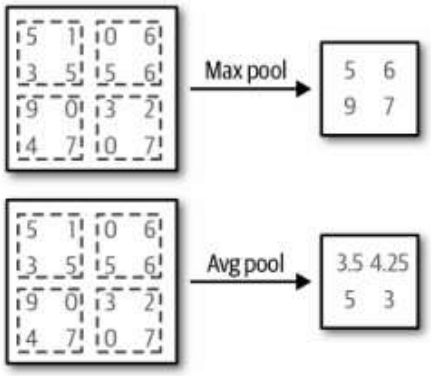


Figure 3. Illustration of maximum and average pooling with 4 x 4 input; each 2 x 2 patch is mapped to the average or maximum value for that patch (14).

The pooling function produces another output vector (9), as shown in Figure 2. There are several pooling techniques, such as average pooling and max-pooling, among which max-pooling is the most commonly used technique that significantly reduces the size of the map (15)(9). The main advantage of pooling is computation. By downsampling the image to create a quarter of the pixels of the previous layer, pooling reduces the number of weights and the amount of computation required to train the network by a factor of 4 (14).

2.3 Fully Connected Layer

Fully connected layer in a convolutional model. The output of the first phase (including convolution and repeated collection) is fed into a fully connected layer, and the points of the weight vector and input vector are calculated to obtain the final output value (16)(9). Gradient descent is also known as batch mode learning or offline algorithm to reduce the cost function over the entire training data set and update the parameters only after one period, according to traversing the entire dataset. This produces a minimum value, but if the training dataset is larger, the time required to train the network will increase significantly(9).

2.4 Activation Function

Activation functions in neural networks convert inputs into nonlinear outputs, allowing models to handle data complexity. Some commonly used activation functions include (17):

1. Linear function: Produces output equal to input $y = f(x) = x$, used primarily in the output layer.
2. Sigmoid function: Converts input to a range of 0–1 $f(x) = 1/(1 + e^{-x})$, seful for logistic models, but prone to vanishing gradients.
3. Tanh function: Similar to sigmoid but has an output in the range of -1 to 1, with a stronger gradient, although it can still experience vanishing gradient problems.
4. Due to its simplicity and effectiveness, the Rectified Linear Unit (ReLU) is the most popular function. It outputs 0 for negative inputs and a linear output for positive inputs.

2.5 CNN Model Architecture

A conventional CNN architecture was deliberately chosen in this study to maintain architectural simplicity and full control over the feature learning process. This approach enables the model to learn domain-specific visual features of BISINDO hand gestures directly from the data, without relying on pre-trained models that are typically trained on large-scale natural image datasets. Such pre-trained representations may not optimally capture the fine-grained characteristics of sign language gestures, particularly given the limited availability of annotated BISINDO datasets. The model architecture consists of three convolutional layers (Conv2D), each followed by a max pooling layer (MaxPooling2D), which serves to reduce spatial dimensions and retain important features from the input image. Based on the best parameter tuning results, each convolutional layer uses a 5×5 kernel size and a ReLU activation function, which is effective in accelerating convergence and avoiding vanishing gradient problems.

After the feature extraction process, the output from the convolutional layer is flattened and passed to a fully connected layer (Dense) consisting of 512 neurons with a ReLU activation function. To minimize the risk of overfitting, a dropout technique of 0.5 is applied to this layer. The last layer is the output layer with 26 neurons, corresponding to the number of letters A to Z in the BISINDO alphabet, which uses the softmax activation function for multi-class classification.

The model is compiled using the Adam optimizer, categorical cross-entropy loss function, and accuracy evaluation metric. The training process is carried out for 30 epochs with a batch size of 32. Complete details of the CNN architecture structure used can be seen in Table 1.

Table 1. CNN Model Architecture

No.	*Layer Type*	*Parameter*	*Output Shape*	*Description*
1	*Input*	(224, 224, 3)	224 x 224 x 3	RGB input image
2	Conv2D	32 filters, kemel size 5 x 5, activation ReLU,	224 x 224 x 32	Feature extraction
3	MaxPooling2D	Pool size 2 x 2	112 × 112 × 32	Spatial downsampling
4	Conv2D	64 filters, kernel size 5 x 5, activation ReLU	112 × 112 × 64	Feature extraction
5	MaxPooling2D	Pool size 2 x 2	56 × 56 × 64	Spatial downsampling
6	Conv2D	128 filters, kemel size 5 x5, activation ReLU	56 x 56 x 128	Feature extraction
7	MaxPooling2D	Pool size 2 x 2	28 × 28 × 128	Spatial downsampling
8	Flatten	-	100352	Convert feature maps to vector
9	Dense	512 units, activation ReLU	512	Fully connected layer

No.	*Layer Type*	*Parameter*	*Output Shape*	*Description*
10	Dropout	Rate 0.5	512	Overfitting prevention
11	Output (Dense)	26 units, activation softmax	26	BISINDO alphabet classification

The architectural structure of the CNN model remains consistent for each combination of parameters tested. The differences lie only in the kernel size, activation function, and type of optimizer used, as summarized in the Parameter Tuning Table. The following table presents the general architecture of the CNN model used in this study. The best configuration for each combination was then used as the basis for building the final model (pretrained model), which was fully retrained with those parameters to improve final performance and generalization stability.

2.6 *Tools*

This research was developed using the Python programming language, which was run in the Anaconda Navigator work environment. Anaconda was chosen because it provides stable environment management and integration with various libraries needed for image processing and machine learning model development. To build and train the Convolutional Neural Network (CNN) model, the TensorFlow and Keras frameworks were used, which enable modular and efficient deep learning architecture design. The image processing, including reading images from a webcam, resizing, format conversion, and augmentation, was performed using the OpenCV library and the ImageDataGenerator module from Keras. Visualization of the training results, such as accuracy and loss curves, was presented using the Matplotlib and Seaborn libraries. Model performance evaluation is carried out with the help of the Scikit-learn (sklearn) library, which provides metrics such as classification reports and confusion matrices.

3. Results and Discussion

This section presents the results of experiments obtained from the training and testing process of the BISINDO alphabet classification model using Convolutional Neural Network (CNN). The discussion is carried out systematically based on the stages of the process, starting from dataset processing, model architecture, parameter tuning, to performance evaluation. The results are presented in the form of tables, graphs, and other visualizations to facilitate interpretation and analysis of model performance. The series of processes can be seen in Figure 4 as the flow of the research stages.

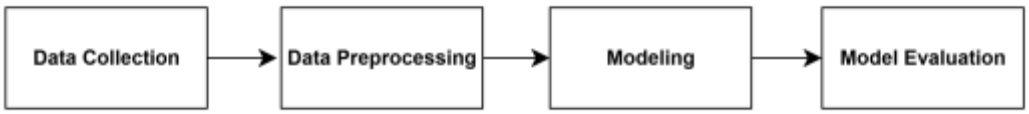


Figure 4. Research Stages

3.1. Data Collection

The alphabet dataset used in this study was collected independently by the author using a webcam as an image acquisition tool. The data collection process was carried out automatically using a Python script that was specifically designed to capture and store images in real time. Each image was categorized and stored according to the letters A to Z, which were determined based on user input before the recording process took place. However, since the dataset was collected from a single individual, it may introduce dataset bias related to hand shape, size, and skin tone, which can limit the model's ability to generalize to diverse users.

With this approach, the author successfully collected a dataset of 3,120 images, covering visual representations for each letter of the Indonesian Sign Language (BISINDO) alphabet. This process ensured that each letter class had sufficient variation in terms of hand gestures, shooting angles, and lighting, which was useful for improving the model's generalization ability.

3.2. Data Preprocessing

To ensure that the data had a uniform format and was ready for model training, several preprocessing steps were applied. All images were resized to 224×224 pixels and normalized to a pixel value range of 0 to 1. Data augmentation techniques such as rotation, flipping, shifting, shearing, and zooming were applied to increase image variation and reduce overfitting. The dataset was then divided into two subsets, with 80% used for training and 20% for testing, resulting in 2,496 training images and 624 testing images.

Data augmentation was applied only to the training set to improve the model's generalization ability, while the test data was only normalized to preserve its original distribution. The augmentation techniques included random rotations of up to ±20°, horizontal and vertical shifts of up to 20% of the image dimensions, shear transformations of up to 0.2, zoom variations within a range of 0.2, and horizontal flipping. These parameters were selected to simulate realistic variations in hand orientation, position, and scale commonly observed in sign language gestures, while avoiding excessive distortion that could alter the semantic meaning of BISINDO hand signs.

3.3. Modeling

3.3.1. Model Design

The CNN model used was designed with several convolution and pooling layers to extract visual features from hand images. After the feature extraction process, the results were processed by several dense and dropout layers for classification into 26 BISINDO alphabet classes. The dropout layer was used to reduce the risk of overfitting, especially since the dataset was limited.

3.3.2. Model Architecture

To obtain the best CNN model performance, we explored 15 hyperparameter combinations covering variations in kernel size (3×3 and 5×5), activation functions (ReLU, tanh, sigmoid), and optimizer types (Adam, SGD, RMSprop). Each configuration was trained for 25 epochs, and the training and validation accuracy results are shown in Table 2.

Table 2**.** Parameter Tuning Result

Model	***Kernel Size***	***Activation***	***Optimizer***	***Accuracy***	**Validasi *Accuracy***
1	(3, 3)	relu	Adam	0.9197	0.9919
2	(3, 3)	relu	SGD	0.2502	0.5897
3	(3, 3)	relu	RMSprop	0.9173	0.9780
4	(3, 3)	tanh	Adam	0.8419	0.9807
5	(3, 3)	tanh	SGD	0.4015	0.7451
6	(3, 3)	tanh	RMSprop	0.8283	0.9743
7	(3, 3)	sigmoid	Adam	0.0379	0.0384
8	(3, 3)	sigmoid	SGD	0.0367	0.0384
9	(3, 3)	sigmoid	RMSprop	0.0419	0.0384
10	(5, 5)	relu	Adam	0.9117	0.9983
11	(5, 5)	relu	SGD	0.2566	0.2692
12	(5, 5)	relu	RMSprop	0.9161	0.9837
13	(5, 5)	tanh	Adam	0.7812	0.9839
14	(5, 5)	tanh	SGD	0.3952	0.7291

Model	*Kernel Size*	*Activation*	*Optimizer*	*Accuracy*	Validasi *Accuracy*
15	(5, 5)	tanh	RMSprop	0.0363	0.0384

Based on the hyperparameter tuning experiments, the best-performing configuration was Model 10, which employed a 5×5 kernel, ReLU activation function, and Adam optimizer, achieving a validation accuracy of 99.83%. A comparable configuration using a 3×3 kernel (Model 1) also demonstrated strong performance with a validation accuracy of 99.19%. In contrast, models utilizing the sigmoid activation function (Models 7–9 and 15) failed to learn effectively, likely due to vanishing gradient issues, resulting in accuracies of approximately 3–4%. The SGD optimizer generally showed lower performance (Models 2 and 11), except when combined with the tanh activation function, which produced moderate results. While the tanh activation demonstrated stable behavior, it tended to yield lower accuracy than ReLU; for instance, Model 13 achieved high validation accuracy (98.39%) but relatively low training accuracy, indicating potential underfitting.

Although multiple experimental trials were conducted during the tuning phase, this study reports only the most representative and best-performing configurations for clarity. Despite the inclusion of approximately six background variations and the application of data augmentation techniques, the dataset diversity remains limited and does not fully reflect complex real-world conditions. Furthermore, as the images were collected from a single individual, the model may be biased toward specific hand characteristics, such as shape and skin tone. Therefore, further evaluation using more diverse participants and real-world data is necessary to assess performance stability and generalization capability.

3.3.3. Training Final Model

After obtaining the best hyperparameter combination through the tuning process, the author built the final model using the optimal configuration, namely a 5×5 kernel, ReLU activation function, and Adam optimizer. At this stage, the model was retrained with 50 epochs and a batch size of 32 to maximize performance. The model architecture consists of three stacked convolutional layers, each followed by max pooling, then continued with a flattening process. The dense layer contains 512 neuron units, accompanied by a dropout of 0.5 to prevent overfitting, and ends with an output layer that functions as softmax activation, for the classification of 26 BISINDO alphabet classes.

The training results show that the model achieved a training accuracy of 94%, indicating its ability to learn discriminative patterns from the data. Interestingly, the evaluation accuracy reached a higher value of 99.84%. This inversion can be attributed to the use of data augmentation during training, which introduces more challenging and diverse samples, making the training process more difficult than evaluation. In contrast, the evaluation data consists of normalized but non-augmented images, which are visually simpler and closer to the learned representations. Additionally, the training and evaluation datasets were strictly separated to prevent data leakage. Figure 5 illustrates the accuracy and loss curves for both training and validation, showing stable convergence without signs of overfitting.

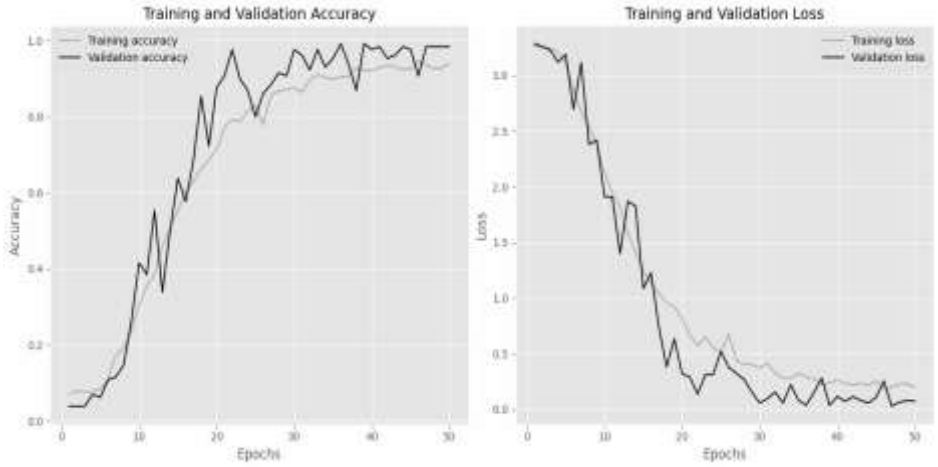


Figure 5. Final Model Training Results Plot

Based on the visualization of the accuracy and loss curves in Figure 3, the model shows a stable convergence trend between the training and validation data. There are no indications of significant overfitting, so it can be concluded that the model has reliable performance and is suitable for use in further testing stages.

3.4. Model Evaluation

At this stage, the previously constructed BISINDO alphabet classification model was evaluated using a test dataset that was completely separated from the training data to ensure an unbiased assessment of the model's generalization capability. The evaluation aimed to measure how well the model could recognize BISINDO letters from images it had never seen before. The evaluation results show that the model achieved an overall accuracy of 99.84%, indicating strong performance in classifying sign language alphabet gestures. The evaluation accuracy visualization is presented in Figure 6, which demonstrates stable and consistent performance across classes.

```
Found 624 images belonging to 26 classes.
20/20 [==============================] - 13s 629ms/step - loss: 0.0305 - accuracy: 0.9984
Evaluation completed. Accuracy: 0.9984
```

Figure 6. Accuracy of Model Evaluation Results

To further analyze classification performance at the class level, a classification report was generated, including precision, recall, and F1-score metrics for each BISINDO alphabet letter (A–Z), as shown in Figure 7.

Classification Report:

	precision	recall	f1-score	support
A	1.00	1.00	1.00	24
B	1.00	1.00	1.00	24
C	1.00	1.00	1.00	24
D	1.00	1.00	1.00	24
E	1.00	1.00	1.00	24
F	1.00	1.00	1.00	24
G	1.00	1.00	1.00	24
H	1.00	1.00	1.00	24
I	1.00	1.00	1.00	24
J	1.00	1.00	1.00	24
K	1.00	1.00	1.00	24
L	1.00	1.00	1.00	24
M	0.96	1.00	0.98	24
N	1.00	0.96	0.98	24
O	1.00	1.00	1.00	24
P	1.00	1.00	1.00	24
Q	1.00	1.00	1.00	24
R	1.00	1.00	1.00	24
S	1.00	1.00	1.00	24
T	1.00	1.00	1.00	24
U	1.00	1.00	1.00	24
V	1.00	1.00	1.00	24
W	1.00	1.00	1.00	24
X	1.00	1.00	1.00	24
Y	1.00	1.00	1.00	24
Z	1.00	1.00	1.00	24
accuracy			1.00	624
macro avg	1.00	1.00	1.00	624
weighted avg	1.00	1.00	1.00	624

Figure 7. Report Classification

The results indicate that the model achieves highly balanced performance, with most classes obtaining near-perfect precision, recall, and F1-scores. Slight performance reductions are observed in a small number of letters, such as M, N, and O, with precision or recall values ranging between 0.96 and 0.98. These results suggest minor misclassifications in visually similar gestures; however, the macro-average and weighted-average metrics remain close to 1.00, indicating robust and evenly distributed classification performance across all classes.

3.5. Results

After undergoing training and evaluation, the model was then retested by performing prediction experiments on the prepared alphabet dataset. The purpose of this stage was to determine the extent of the model's ability to classify sign language alphabet images into the appropriate labels. Prediction experiments were conducted using images that were not included in the training or validation data to ensure that the model was truly capable of generalization. The results of these predictions are shown in Figure 7, which displays the input images along with the prediction labels generated by the model.

Figure 8. Predictions from the Best Alphabet Model

Although the model demonstrates strong overall performance in recognizing most BISINDO alphabet letters, minor misclassifications were still observed in several classes. These errors are reflected in the class-wise prediction variations shown in Table 3, indicating a degree of limitation in distinguishing visually similar hand gestures. Such misclassifications are likely caused by visual similarities between certain alphabet signs, as well as variations in hand orientation and lighting conditions that were not fully represented in the training data. Consequently, while the model exhibits excellent overall classification capability, these results reveal structural challenges in recognizing visually similar alphabet gestures, which should be addressed in future model refinement. To support this analysis, the following section presents prediction results from several experimental trials.

Table 3 Results of Prediction Experiments Against the Model

Original Label	**Experiments**				
	1	**2**	**3**	**4**	**5**

	Pred	**Acc**	**Pred**	**Acc**	**Pred**	**Acc**	**Pred**	**Acc**	**Pred**	**Acc**
A	A	100.00	A	100.00	A	100.00	A	100.00	A	100.00
B	B	100.00	B	100.00	B	100.00	B	100.00	B	99.79
C	C	100	C	100.00	C	100.00	C	100.00	C	99.98
D	D	99.94	D	100.00	D	100.00	D	100.00	D	100.00
E	E	100.00	E	99.92	E	100.00	E	99.98	E	100.00
F	F	100.00	F	100.00	F	99.92	F	100.00	F	99.94
G	G	99.97	G	100.00	G	100.00	G	99.98	G	100.00
H	H	99.58	H	99.85	H	99.882	H	100.00	H	100.00
I	I	98.96	I	92.72	I	100.00	I	99.94	I	99.95
J	J	99.98	J	100.00	J	100.00	J	100.00	J	100.00
K	K	98.68	K	99.68	K	99.33	K	99.35	K	98.77
L	L	90.18	L	99.75	L	99.49	L	99.43	L	99.28
M	M	99.21	M	98.51	M	99.08	M	98.93	M	99.27
N	N	97.52	N	93.18	N	97.90	N	99.62	N	99.30
O	O	99.96	O	99.96	O	99.97	O	99.97	O	99.93
P	P	99.95	P	82.31	P	87.25	P	94.31	P	96.36
Q	Q	100.00	Q	100.00	Q	100.00	Q	100.00	Q	100.00
R	R	99.88	R	100.00	R	100.00	R	99.55	R	99.90
S	S	99.51	S	94.61	S	98.50	S	99.91	S	99.93
T	T	99.98	T	100.00	T	100.00	T	100.00	T	100.00
U	U	100.00	U	100.00	U	99.91	U	99.98	U	100.00
V	V	99.93	V	99.97	V	100.00	V	98.56	V	99.56
W	W	100.00	W	100.00	W	100.00	W	100.00	W	100.00
X	X	100.00	X	100.00	X	100.00	X	100.00	X	100.00
Y	Y	100.00	Y	100.00	Y	99.98	Y	100.00	Y	100.00
Z	Z	100.00	Z	100.00	Z	100.00	Z	100.00	Z	100.00

4. Conclusion

Based on the results of the research that has been conducted, the objectives formulated in the introduction were optimally achieved. The implementation of the Indonesian Sign Language (BISINDO) character detection model using the Convolutional Neural Network (CNN) method produced a high accuracy of 99.84%, which shows that this approach is effective in recognizing static letter gestures. This success proves that the method used is capable of answering the problem formulation and making a significant contribution to the development of sign language recognition technology.

However, this study has several limitations. The developed model was only tested on static gestures in the form of the letters A to Z, so it does not yet cover dynamic gestures that represent words or sentences, as used in everyday communication by deaf people. In addition, data variation is still limited, both in terms of lighting, background, and the number of individuals who are the source of the data, so the model has the potential to experience bias in more complex real-world conditions.

As a follow-up, development can be directed towards expanding the scope of a more varied dataset and applying a more complex or attention-based model architecture to handle dynamic signs. On the other hand, the application of transfer learning from pretrained models can also be an alternative to improve performance. In terms of implementation, this model has the potential to be integrated into real-world platforms, such as camera-based mobile applications to assist in direct communication for people with disabilities, or web-based systems capable of real-time detection using webcams. Thus, this research not only contributes theoretically to the fields of deep learning and computer vision, but also opens up broad opportunities for the development of inclusive assistive technologies in the future. Future work may extend this study by evaluating the model on dynamic sign language gestures using video-based data, as well as by incorporating multi-user datasets with diverse hand shapes, skin tones, and gesture styles to improve realism and generalizability.

References

[1] N. Haliza, E. Kuntarto, and A. Kusmana, "Language acquisition in children with special needs (deaf) in understanding language," Jurnal Genre (Bahasa, Sastra, dan Pembelajarannya), vol. 2, no. 1, pp. 5–11, Jul. 2020, doi: 10.26555/jg.v2i1.2051.

[2] J I. Sari, Fivrenodi, E. Altiarika, and Sarwindah, "Sign Language Development System for Communicating with Persons with Disabilities (Deaf)," Journal of Information Technology and society, vol. 1, no. 1, pp. 20–25, Jun. 2023, doi: 10.35438/jits.v1i1.21.

[3] T. Handhika, R. I. M. Zen, Murni, D. P. Lestari, and I. Sari, "Gesture recognition for Indonesian Sign Language (BISINDO)," J Phys Conf Ser, vol. 1028, p. 012173, Jun. 2018, doi: 10.1088/1742-6596/1028/1/012173.

[4] R. K. Pathan, M. Biswas, S. Yasmin, M. U. Khandaker, M. Salman, and A. A. F. Youssef, "Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network," Sci Rep, vol. 13, no. 1, Dec. 2023, doi: 10.1038/s41598-023-43852-x.

[5] Iffat Zafar, Richard Burton, Leonardo Araujo, and Giounona Tzanidou, Hands-On Convolutional Neural Networks with TensorFlow: Solve computer vision problems with modeling in TensorFlow and Python. Packt Publishing Ltd, 2018.

[6] R. Z. Fadillah, A. Irawan, M. Susanty, and I. Artikel, "Data Augmentation to Overcome Data Limitations in the Indonesian Sign Language Translation Model (BISINDO)," JURNAL INFORMATIKA, vol. 8, no. 2, 2021, [Online]. Available: http://ejournal.bsi.ac.id/ejurnal/index.php/ji

[7] M. Hassaballah and A. I. Awad, Deep Learning in Computer Vision, 1st ed. CRC Press, 2020.

[8] C. Nebauer, "Evaluation of convolutional neural networks for visual recognition," IEEE Trans Neural Netw, vol. 9, no. 4, pp. 685–696, Jul. 1998, doi: 10.1109/72.701181.

[9] S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, "Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach," Procedia Comput Sci, vol. 132, pp. 679–688, 2018, doi: 10.1016/j.procs.2018.05.069.

[10] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]," IEEE Comput Intell Mag, vol. 5, no. 4, pp. 13–18, Nov. 2010, doi: 10.1109/MCI.2010.938364.

[11] E. A. Smirnov, D. M. Timoshenko, and S. N. Andrianov, "Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks," AASRI Procedia, vol. 6, pp. 89–94, 2014, doi: 10.1016/j.aasri.2014.05.013.

[12] A. ANHAR and R. A. PUTRA, "Design and Implementation of a Self-Checkout System in Retail Stores using Convolutional Neural Networks (CNN)," ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik

Telekomunikasi, & Teknik Elektronika, vol. 11, no. 2, p. 466, Apr. 2023, doi: 10.26760/elkomika.v11i2.466.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[14] Seth Weidman, Deep Learning from Scratch Building with Python from First Principles. y O'Reilly Media, Inc., 2019.

[15] H. Liu, B. Li, X. Lv, and Y. Huang, "Image Retrieval Using Fused Deep Convolutional Features," Procedia Comput Sci, vol. 107, pp. 749–754, 2017, doi: 10.1016/j.procs.2017.03.159.

[16] Y. Zhou, H. Wang, F. Xu, and Y.-Q. Jin, "Polarimetric SAR Image Classification Using Deep Convolutional Neural Networks," IEEE Geoscience and Remote Sensing Letters, vol. 13, no. 12, pp. 1935–1939, Dec. 2016, doi: 10.1109/LGRS.2016.2618840.

[17] Giuseppe Ciaburro and Balaji Venkateswaran, Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles. 1BDLU1VCMJTIJOH-UE, 2017.

[18] J. Fieres, J. Schemmel, and K. Meier, "Training convolutional networks of threshold neurons suited for low-power hardware implementation," in The 2006 IEEE International Joint Conference on Neural Network Proceedings, IEEE, 2006, pp. 21–28. doi: 10.1109/IJCNN.2006.246654..