

December 15, 2025

**SIGN LANGUAGE CNN**

[1. Introduction 2](#_Toc216773201)

[2. Objectives 2](#_Toc216773202)

[3. Methodology 2](#_Toc216773203)

[3.1 Dataset Description: 2](#_Toc216773204)

[4. Model Architecture: 3](#_Toc216773205)

[5. Training: 3](#_Toc216773206)

[6. Evaluation: 4](#_Toc216773207)

[7. Conclusion 5](#_Toc216773208)

# 

# Introduction

Sign language is a vital means of communication for the deaf and hard-of-hearing community. Automatic Sign Language Recognition (SLR) systems aim to bridge the communication gap between sign language users and non-signers by translating hand gestures into readable text or speech. With recent advances in Deep Learning and Computer Vision, Convolutional Neural Networks (CNNs) have shown remarkable performance in image classification tasks, making them well-suited for recognizing hand signs.

This project focuses on building an American Sign Language (ASL) alphabet recognition system using a deep CNN model trained on the Sign Language MNIST dataset.

# Objectives

* Analyze the dataset and prepare images and labels through reshaping, normalization, and encoding for CNN training.
* Build a deep CNN model with convolutional, pooling, and dropout layers to effectively extract hand gesture features.  
  Use image transformations to increase data diversity and improve model generalization.
* Train the model on augmented data and validate it to monitor performance and prevent overfitting.
* Measure model performance using accuracy, confusion matrix, and classification report.
* Use the trained model to classify new hand sign images and predict corresponding ASL letters.

# Methodology

## 3.1 Dataset Description:

The **Sign Language MNIST** dataset consists of grayscale images of hand gestures representing **24 American Sign Language (ASL) alphabet letters**, excluding **J and Z**. Each image has a resolution of **28 × 28 pixels**, making it suitable for efficient training of deep learning models.

**Data Preprocessing**

The following preprocessing steps were applied to prepare the data:

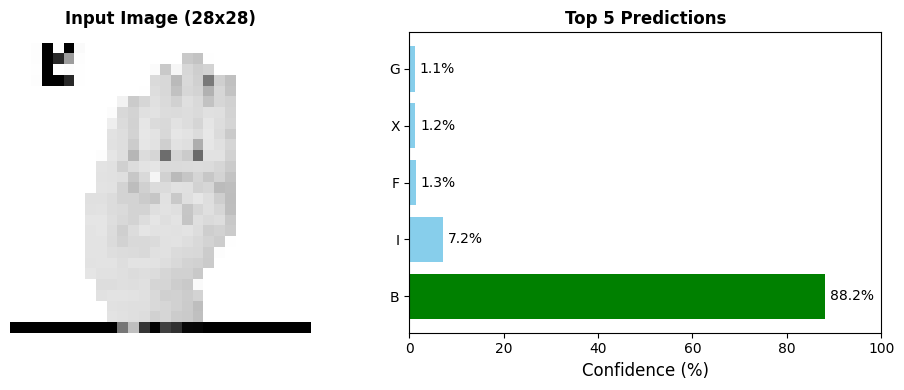
* **Label separation** from image features
* **Image reshaping** to match the model input format
* **Normalization** to scale pixel values between 0 and 1
* **One-hot encoding** of class labels
* **Train–validation split** to evaluate model performance during training

**Data Augmentation**

To improve model generalization and reduce overfitting, data augmentation techniques such as **rotation, zooming, shifting, and shearing** were applied during training.

# Model Architecture:

An improved deep Convolutional Neural Network (CNN) is designed using multiple convolutional blocks with increasing filter sizes of 32, 64, and 128 to progressively extract low-level to high-level features from input images. Each convolution layer is followed by batch normalization, which stabilizes and accelerates the training process. Max pooling layers are used to reduce spatial dimensions while preserving important features, thereby lowering computational complexity. Dropout layers are incorporated to minimize overfitting by randomly deactivating neurons during training. The extracted features are then passed through fully connected dense layers to learn complex patterns required for accurate classification. Finally, a softmax output layer is used to perform multi-class classification by assigning probability scores to each ASL letter. The entire model is implemented using the TensorFlow Keras Sequential API for efficient model construction and training.

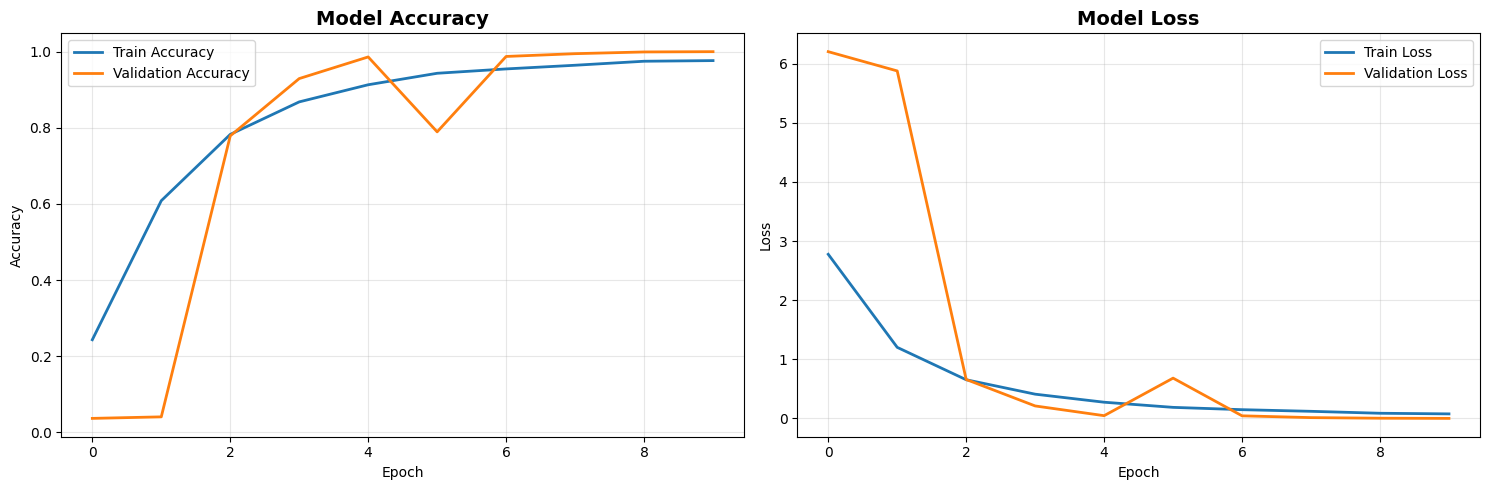


# Training:

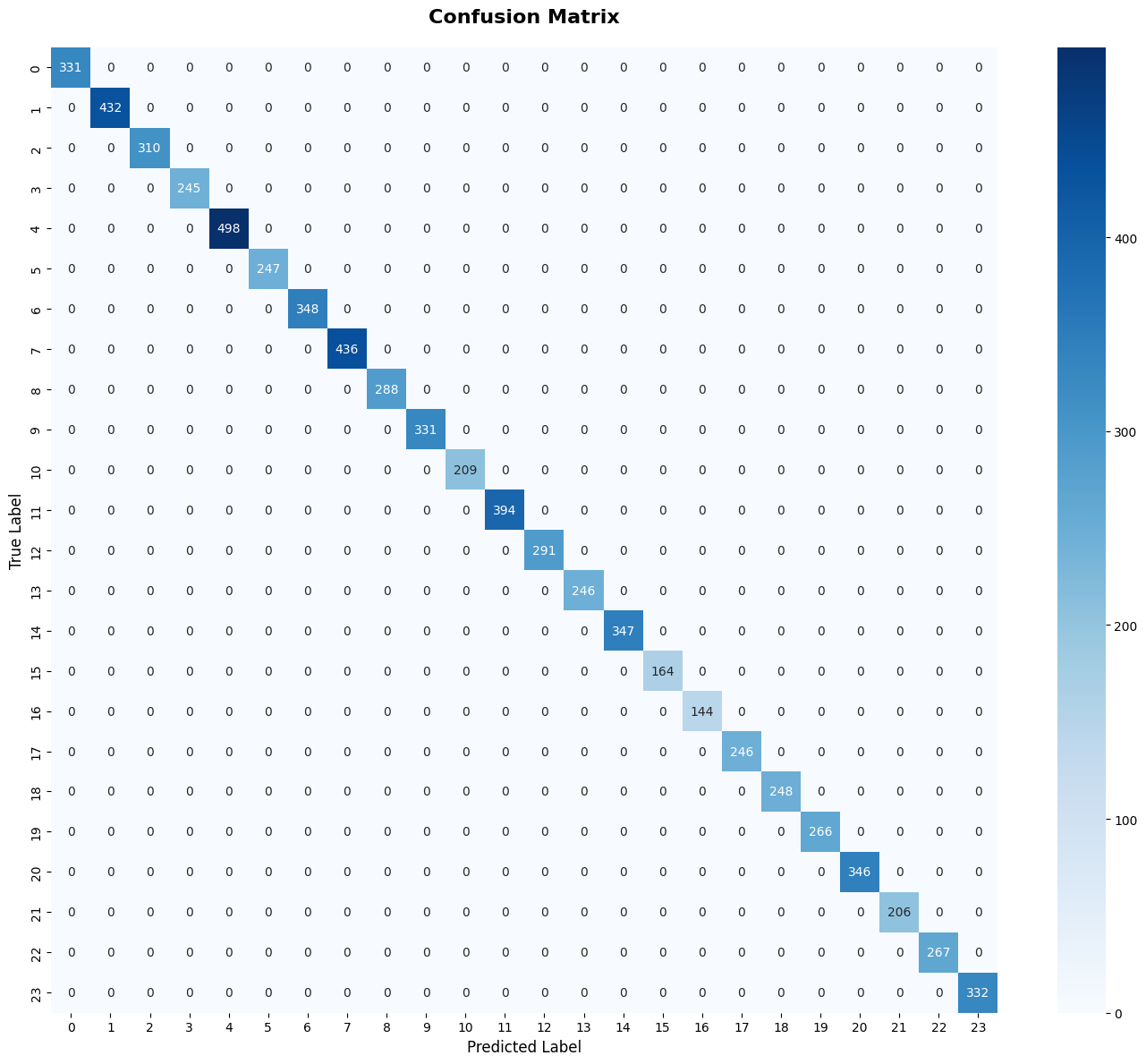
1. **Loss Function (Categorical Crossentropy):** Used for multi-class classification. It measures the difference between predicted class probabilities and true labels, guiding the model to improve predictions.
2. **Optimizer (Adam, LR = 0.001):** Adam efficiently updates model weights using adaptive learning rates, ensuring fast and stable convergence.
3. **Metric (Accuracy):**  Tracks the percentage of correctly classified samples during training and validation.
4. **Batch Size (128):** The model updates its weights after processing 128 samples, balancing training speed and stability.
5. **Epochs (10):** The dataset is passed through the model up to 10 times, allowing sufficient learning without excessive training.
6. **Early Stopping:** Stops training when validation performance no longer improves, helping to prevent overfitting.
7. **Reduce Learning Rate on Plateau:**  Automatically lowers the learning rate when progress slows, improving fine-tuning and convergence.
8. **Model Saving:** The trained model is saved for later use in inference or further training.

# 6. Evaluation:

* **Overall Test Accuracy:** Measures the model’s ability to correctly classify samples in the test dataset



* **Matrix Visualization:** Shows correct and incorrect predictions for each class, helping identify misclassification patterns.
* **Precision, Recall, and F1-Score:** Computed using a classification report to evaluate class-wise performance and balance between false positives and false negatives.



* **Performance Insight:**Together, these metrics provide a reliable assessment of the model’s prediction accuracy and robustness across all classes.

# 7. Conclusion

This project successfully demonstrates the application of Convolutional Neural Networks for American Sign Language alphabet recognition. The model achieves high accuracy on the test dataset and shows strong generalization capability due to effective preprocessing, data augmentation, and architectural design. The implemented prediction pipeline allows real-world testing using custom images, making the system practical and extensible.

Future enhancements may include real-time gesture recognition using video input, inclusion of dynamic gestures (J and Z), and deployment as a web or mobile application to assist communication accessibility.