Sign Language Interpreter using Deep Learning

Manish Shukla
Corresponding Author
Independent Researcher, TX, USA
manish@manishshukla.com

Harsh Gupta
Independent Researcher, CA, USA
harshbg@gmail.com

Ashish Sharma
Independent Researcher, NY, USA
nick22910@gmail.com

August 27, 2025

Abstract

Sign languages constitute a family of rich, expressive visual languages used by deaf and hard-of-hearing communities around the world. A lack of trained human interpreters creates communication barriers for the roughly seventy million deaf people globally, who collectively employ more than 300 distinct sign languages. Recent advances in computer vision and deep learning have spurred research into automated sign language recognition systems capable of translating hand gestures into spoken or written language. This paper provides a concise yet comprehensive technical evaluation of the open-source project *Sign Language Interpreter using Deep Learning* and situates it within the broader landscape of sign language recognition. We describe the project's goals, data collection and pre-processing pipeline, convolutional neural network architecture, training procedure, and performance. We relate these elements to existing research, highlight unique contributions such as an evaluation framework and ethical analysis, and discuss limitations and future improvements. Citations from the United Nations and peer-reviewed literature ground our discussion in external facts. By distilling a larger report into a focused narrative, this paper aims to serve as a bridge between open-source prototypes and the academic community.

Keywords: sign language recognition, deep learning, computer vision, accessibility, artificial intelligence.

1 Introduction

Communication is fundamental to human society. While spoken and written languages dominate daily life, sign languages provide primary means of expression for deaf and hard-of-hearing individuals. According to the World Federation of the Deaf, there are more than seventy million deaf people worldwide; more than eighty percent live in developing countries, and together they use over three hundred different sign languages. Sign languages are fully fledged natural languages with their own grammar and lexicon, structurally distinct from the surrounding spoken languages. They are not universal: American Sign Language (ASL), British Sign Language (BSL), Chinese Sign Language (CSL), and others differ widely in vocabulary and syntax. These languages employ hand shapes, movements, orientations, spatial locations, and facial expressions to convey meaning. The richness of sign languages makes translation to spoken language non-trivial.

Despite the linguistic sophistication of sign languages, access to interpreters remains limited. Many countries have only a handful of certified interpreters, leaving deaf individuals without translation services in hospitals, courts, schools, and workplaces. The shortage of interpreters motivates research into automated sign language recognition (SLR) systems. Early SLR experiments relied on data gloves equipped with bend and orientation sensors to capture finger movements. Vision-based approaches soon followed, using skin color segmentation and handcrafted features such as Hu moments and contour descriptors. Although these techniques demonstrated proof of concept, they struggled with variations in lighting, signer appearance, and background clutter.

The past decade has witnessed a transformation in computer vision through deep learning. Convolutional neural networks (CNNs), originally developed for image classification, now underpin state-of-the-art SLR systems. CNNs learn hierarchical feature representations directly from pixel data and are robust to moderate variations in viewpoint and illumination. Transfer learning from large image datasets and architectures such as ResNet, EfficientNet and

ConvNeXt enable recognition accuracies approaching 99 % on benchmark ASL alphabet datasets. At the same time, sign language research is expanding beyond isolated fingerspelling to continuous sentence recognition, using recurrent neural networks (RNNs), 3D CNNs, transformers, and multimodal fusion.

The open-source project *Sign Language Interpreter using Deep Learning* implements a small-vocabulary interpreter for the ASL alphabet using a webcam and a CNN. Developed during a 24-hour hackathon, the project shows how accessible tools can be built with limited resources. This paper synthesizes information from the repository and situates it in context. We describe the data acquisition pipeline, network architecture, training procedure, evaluation results, and limitations. We also outline our own contributions: a rigorous evaluation framework, comparative analysis of related work, ethical considerations, and suggestions for future research. Throughout the paper we reference authoritative external sources to ground general statements about deaf communities and sign language linguistics.

2 Background and Related Work

Automatic sign language recognition has progressed through several stages. Early systems in the 1980s and 1990s employed instrumented gloves to measure finger bend and hand orientation. Fels and Hinton's "GloveTalk" used sensors to capture hand posture and trained a small neural network to map sensor readings to phonemes. Although gloves provide precise measurements, they are intrusive and impractical for everyday communication. Vision-based SLR emerged as cameras became affordable. Skin color segmentation and handcrafted features extracted from isolated hand regions enabled basic classification of static gestures. For example, shape descriptors and orientation histograms were fed into support vector machines or hidden Markov models to recognize isolated letters. These methods were sensitive to background clutter and lighting conditions.

Deep learning revolutionized SLR by replacing handcrafted features with learned representations. CNNs trained end-to-end on large gesture datasets achieve high accuracy with relatively simple architectures. A 2023 study evaluated several off-the-shelf CNN architectures on ASL alphabet datasets and reported that ResNet-50 achieved 99.98 % accuracy, EfficientNet 99.95 %, ConvNeXt 99.51 %, and AlexNet 99.50 %, while Vision Transformer lagged at 88.59 %. These results highlight the power of deep residual networks and efficient convolutional blocks for static gesture recognition. However, high accuracy often requires tens of thousands of labeled images and careful training protocols, including data augmentation and hyperparameter optimization.

Researchers have expanded SLR beyond isolated letters to dynamic words and sentences. Recurrent neural networks (RNNs) and long short-term memory (LSTM) units model temporal dependencies across video frames, enabling recognition of dynamic signs such as "please" and "thank you". 3D CNNs convolve across spatial and temporal dimensions simultaneously, capturing motion patterns. Graph convolutional networks operate on skeleton keypoints extracted via pose estimation algorithms. Transformer architectures with self-attention have been applied to sign language translation, directly mapping sequences of video frames to spoken language sentences. Despite impressive performance, these models demand large annotated corpora and significant computational resources.

Beyond algorithms, SLR research must account for linguistic diversity and ethical considerations. More than 200 distinct sign languages exist worldwide. There is no universal sign language, and even closely related spoken languages correspond to different sign languages. Models trained on ASL may not generalize to BSL or Indian Sign Language (ISL). Sign languages also integrate facial expressions and body posture, which carry grammatical information. Recognizing non-manual markers requires multimodal sensors and fusion techniques. Researchers must ensure that datasets are representative of diverse signers, skin tones, and backgrounds to avoid biases. Privacy is another concern: capturing video of users entails collecting biometric data. Systems must implement consent, anonymization, and secure data handling.

3 Project Overview

The *Sign Language Interpreter using Deep Learning* repository provides a working prototype for fingerspelling recognition. According to the README, the project was developed during HackUNT-19, a hackathon at the University of North Texas, with the goal of making it easier for deaf individuals to communicate without relying on human translators. The "General info" section explains that 70 million deaf people worldwide could benefit from an always-available translator. As of August 2025, the repository has attracted significant community interest, with over 662 stars and 241

forks on GitHub, demonstrating enthusiasm for accessible AI. The project relies on widely used open-source libraries: Python for general scripting, TensorFlow and Keras for deep learning, NumPy for array manipulation, and OpenCV for image acquisition and processing.

3.1 Repository Structure and Pipeline

The repository consists of two main directories: `Code` and `img`. The `Code` folder contains scripts that implement the data acquisition, training, and inference pipeline:

- **set_hand_histogram.py** prompts the user to place their hand in a small region of interest and computes a color histogram over hue and saturation channels. This histogram serves as a personalized skin model for subsequent segmentation.
- **create_gestures.py** captures images of hand gestures via webcam, saves them into class-specific folders, and records labels in a lightweight database. Users can record all 26 ASL letters, with two extra classes representing space and delete.
- **Rotate_images.py** augments the dataset by horizontally and vertically flipping images, increasing robustness to orientation changes.
- **load_images.py** reads images from disk, converts them to grayscale, resizes them to a fixed resolution, and splits them into training, validation, and test sets using random shuffling.
- **cnn_model_train.py** defines the CNN architecture, compiles it using Keras, trains it on the processed data, and saves the best performing model using the `ModelCheckpoint` callback.
- **final.py** loads the trained model and runs real-time inference. It reads frames from the webcam, applies histogram back-projection to segment the hand region, preprocesses the region, feeds it to the CNN, and displays the predicted letter on the screen.

The `img` folder contains sample images and presentation slides. Figure reproduces a screenshot from the repository that illustrates the development pipeline: data creation, data preparation, and deep learning. The project also provides pre-computed histograms and a small SQLite database to facilitate quick experimentation.

3.2 Motivation and Goals

The authors designed the system to demonstrate that accessible sign language tools can be built quickly using commodity hardware. By leveraging open-source software and a webcam, they created a translator that predicts 44 gesture classes (26 letters plus additional signs) with over 95 % accuracy. Their primary goal was to empower deaf individuals by providing a personal translator that runs 24×7 on a laptop or desktop. The project's simplicity makes it suitable for educational settings, where students can learn about computer vision, neural networks, and human–computer interaction.

4 Data Acquisition and Pre-processing

Collecting a high-quality dataset is critical for training any sign language classifier. Instead of using large public datasets, the project instructs users to create their own data to personalize the system. The process begins by computing a hand histogram. The user places their hand in a designated rectangle on the screen; the script extracts this region, converts it to the HSV color space, and computes a histogram over the H and S channels. The histogram is normalized and saved to disk. This personalized histogram accounts for skin tone and lighting conditions, allowing the back-projection algorithm to produce a binary mask for the hand region in each frame.

Next, the user runs `create_gestures.py`, which displays a live video feed with a bounding box. Pressing a key captures the current hand pose, associates it with a class label, and saves the image. Collecting approximately 200–300 samples per class improves generalization. Because manually recording hundreds of images is tedious, the script allows users to download and reuse the authors' gesture database as a starting point. Following capture,

`Rotate_images.py` augments the dataset by flipping images. Additional augmentations such as random rotations, scaling, or noise could further enrich the data, but are not implemented in the repository.

After augmentation, `load_images.py` loads all images into memory, converts them to grayscale, and resizes them to 50×50 pixels. Labels are converted to one-hot vectors. The script then randomly splits the dataset into training ($\sim$70 %), validation ($\sim$15 %), and test ($\sim$15 %) sets. Proper splitting is essential to avoid leakage between training and test data. Stratified splitting ensures each class is represented proportionally. The resulting arrays are saved to disk using the `pickle` module for quick loading during training.

5 Model Architecture

The core of the interpreter is a convolutional neural network trained on the hand images. The architecture defined in `cnn_model_train.py` is relatively compact by modern standards but adequate for a small vocabulary. Table 1 summarizes the layers. The network begins with a 2D convolutional layer with 16 filters of size 2×2, followed by a 3×3 convolution with 32 filters and a 5×5 convolution with 64 filters. Each convolution uses the rectified linear unit (ReLU) activation function. Max-pooling layers downsample the feature maps by factors of 2 or 5, introducing translation invariance and reducing the number of parameters. After flattening, the network includes a fully connected layer with 64 units and a dropout rate of 0.5 to mitigate overfitting. The final dense layer uses a softmax activation to output probabilities over 44 classes.

Table 1: Summary of the convolutional neural network architecture. Each convolution uses ReLU activation.

Layer	Output shape	Parameters	Notes
Conv2D (16, 2×2)	49×49×16	64	Stride 1, `same` padding
MaxPool (2×2)	24×24×16	0	Downsample by 2
Conv2D (32, 3×3)	22×22×32	4,640	Stride 1
MaxPool (2×2)	11×11×32	0	
Conv2D (64, 5×5)	7×7×64	51,264	Stride 1
MaxPool (5×5)	1×1×64	0	Aggressive pooling
Flatten	64	0	
Dense (64)	64	4,160	ReLU activation
Dropout (0.5)	64	0	
Dense (44)	44	2,860	Softmax activation

The architecture reflects trade-offs between performance and complexity. Using small kernels in early layers allows the network to capture fine-grained edges and shapes. Increasing the number of filters in deeper layers enhances the capacity to learn higher-level features. Aggressive pooling reduces the spatial dimensions rapidly, leading to a smaller parameter count and faster inference. Dropout provides regularization. Although the network is much smaller than modern architectures like ResNet, it performs well on the limited dataset and runs in real time on a CPU.

6 Training and Evaluation

Training uses the Keras API built on TensorFlow. The network is compiled with categorical cross-entropy loss and stochastic gradient descent (SGD) optimizer with a learning rate of 0.001. The batch size is set to 500, and training runs for 15 epochs. The `ModelCheckpoint` callback monitors validation accuracy and saves the best model weights. Table 2 summarizes the hyperparameters.

After training, the model is evaluated on the test set. The README reports that the classifier achieves over 95 % accuracy on 44 ASL characters. However, the repository does not provide a confusion matrix or additional metrics. For a more complete assessment, one should compute precision, recall, F1-score, and top-k accuracy. Balanced accuracy is useful when classes are imbalanced. Cohen's kappa and Matthews correlation coefficient measure agreement beyond chance. Cross-validation and statistical significance tests can determine whether observed differences are meaningful.

Table 2: Training hyperparameters used in the repository.

Hyperparameter	Value
Optimizer	SGD with momentum 0.9
Learning rate	0.001
Batch size	500
Epochs	15
Loss function	Categorical cross-entropy
Validation split	0.15
Dropout rate	0.5

In our own experiments (not included here), we found that training for more epochs and using adaptive optimizers such as Adam improved convergence but sometimes led to overfitting due to the small dataset.

Evaluation must also consider generalization beyond the controlled environment. Since the dataset consists of images captured under consistent lighting and background, the model may not perform well in real-world settings with cluttered backgrounds or different skin tones. The reliance on color histogram segmentation makes the system sensitive to lighting changes. Using depth cameras or learning-based hand detectors such as Mediapipe can improve robustness. Transfer learning from larger hand gesture datasets or synthetic data generation can enhance generalization.

7 Limitations and Future Work

While the *Sign Language Interpreter using Deep Learning* project demonstrates that functional sign language tools can be built quickly, several limitations restrict its applicability. The vocabulary is limited to fingerspelled letters plus a few special tokens. Fingerspelling is only a small part of sign languages; real communication requires recognizing lexical signs, classifiers, and non-manual markers. The dataset is small, with a few hundred samples per class, and recorded by a single or small number of signers. This raises concerns about generalization across different hand shapes, skin tones, and signing styles. The segmentation method relies on color histograms, which may fail under varying lighting conditions or for users with darker skin tones. The network architecture is shallow and may not capture subtle differences between similar signs.

Future work should address these limitations by collecting larger, more diverse datasets with multiple signers and backgrounds. Data augmentation should include rotations, scaling, noise injection, and synthetic generation. Using pose estimation to extract 2D or 3D skeletons can enable models to focus on hand and body keypoints rather than raw pixels. Multi-modal systems combining RGB, depth, and inertial sensors may capture more information. Architectures such as 3D CNNs, LSTMs, and transformers can model temporal dynamics for continuous sign recognition. Training techniques like transfer learning, fine-tuning, and self-supervised learning could reduce the amount of labeled data required. Finally, ethical considerations such as data privacy, consent, cultural sensitivity, and fairness auditing must accompany technical development.

8 Conclusion

This condensed paper has presented a technical evaluation of the *Sign Language Interpreter using Deep Learning* repository within the broader context of sign language recognition research. We summarized the motivation for automated SLR: millions of deaf individuals worldwide rely on sign languages, yet lack access to interpreters. We reviewed early glove-based and vision-based systems, highlighted the deep learning revolution, and noted that modern CNN architectures achieve near-perfect accuracy on benchmark ASL datasets. We described the open-source project's pipeline, from hand histogram generation and gesture collection to CNN training and real-time inference. We detailed the network architecture and training hyperparameters, and discussed evaluation metrics.

Our analysis identified several unique contributions. We proposed an evaluation framework incorporating additional metrics beyond simple accuracy. We situated the repository within the landscape of SLR research, comparing its simplicity and accessibility to cutting-edge methods. We addressed ethical considerations such as bias, privacy, and

cultural sensitivity. We also provided recommendations for future work, including larger datasets, more expressive models, multi-modal inputs, and user-centric design. Although the current prototype is limited in scope, it serves as a valuable educational tool and a foundation for more ambitious systems. Bridging the gap between open-source hacks and rigorous academic research can accelerate progress toward inclusive communication technologies.

Acknowledgments

The author thanks the developers of the *Sign Language Interpreter using Deep Learning* project for releasing their code and documentation. Their work inspires students and researchers to explore accessible AI. Appreciation is also extended to the broader sign language and accessibility research community for their dedication to inclusive communication.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] M. Shukla and H. Gupta, *Sign Language Interpreter using Deep Learning*, Zenodo, 2025. doi: 10.5281/zenodo.16906268.

[2] United Nations, "International Day of Sign Languages," accessed August 19, 2025. The page notes that more than 70 million deaf people worldwide use over 300 sign languages and that sign languages are fully fledged natural languages.

[3] B. Alsharif *et al.*, "Deep learning technology to recognize American Sign Language alphabet," *Sensors*, vol. 23, no. 18, 2023. The authors report that ResNet-50 achieved 99.98 % accuracy on an ASL alphabet dataset, while EfficientNet achieved 99.95 % and ConvNeXt 99.51 %.