SAPIENZA
UNIVERSITÀ DI ROMA

BABY SIGN LANGUAGE RECOGNITION

Rokshana Ahmed
Firdaous Hajjaji
Elena Martellucci

Sapienza University of Rome
Applied Computer Science and Artificial Intelligence

ABSTRACT

This thesis presents the development and evaluation of a sign language recognition program aimed at recognizing baby sign language gestures using video datasets. The study explores the application of different training models, including 2D CNN, 3D CNN, and ResNet18, to achieve accurate recognition. The research delves into the challenges encountered during training, including issues with accuracy, validation performance, dataset characteristics, and preprocessing requirements. By investigating these challenges and employing appropriate solutions, the study contributes to the advancement of sign language recognition systems.

***Index Terms*—** Convolutional Neural Networks (CNNs), Baby Sign Language, ResNet18, Data Augmentation

1. INTRODUCTION

In a world where communication is an essential pillar of human interaction, the inability to convey thoughts, emotions, and ideas can be isolating. For individuals with hearing impairments, this challenge is a daily reality, making the development of effective sign language recognition systems a matter of paramount importance. Sign language, a visual-gestural language used by the Deaf community, is a vital medium for communication and a bridge to inclusion for millions of individuals worldwide. Beyond the traditional forms of sign language, there exists a unique and valuable form known as ”baby sign language.”

1.1. Background and Motivation

Baby sign language, a simplified version of sign language designed for infants, serves as a bridge between proverbial infants and their caregivers, facilitating communication during the crucial early years of development. By teaching babies basic signs for everyday needs and expressions, parents and caregivers empower them to convey their desires and emotions long before they can speak. This form of early communication has been shown to strengthen the parent-child bond and potentially accelerate language development in hearing infants. However, the accurate recognition of these tiny hands and gestures presents a formidable challenge—one that modern technology and computer vision can help address.

1.2. Objective of the Study

The objective of this thesis is to develop and evaluate a dynamic sign language recognition program tailored specifically for baby sign language using video datasets. Even though previous research has focused on recognizing static alphabets and numbers in various sign languages. Static gestures represent individual letters as images. However, effective communication especially between baby and caregiver often requires dynamic gesture recognition, as relying solely on letters can be challenging. Our research explores the application of various training models, including 2D Convolutional Neural Networks (CNNs), 3D CNNs, and the ResNet18 architecture, in the quest for accurate and reliable recognition.

By delving into the intricacies of baby sign language recognition, we aim to enhance the efficacy of communication between caregivers and infants, thereby fostering stronger early development.

2. LITERATURE REVIEW

2.1. Sign Language Recognition and its Applications

Sign language is a visual-gestural language used by the Deaf community to communicate. The development of sign language recognition systems has been a topic of interest in the field of computer vision and natural language processing. These systems have wide-ranging applications, from assisting the Deaf and hard-of-hearing community in communication to enabling sign language translation for wider audiences.

Early efforts in sign language recognition focused on recognizing American Sign Language (ASL) and other standardized sign languages used by Deaf adults and older children. These systems have made significant progress, leveraging technologies like convolutional neural networks (CNNs) to

achieve impressive accuracy rates. However, adapting these systems to recognize the unique nuances of baby sign language poses distinct challenges.

2.2. Baby Sign Language and its Importance

Baby sign language, also referred to as infant sign language or baby signing, is a simplified version of sign language designed specifically for infants and toddlers. Its primary purpose is to facilitate early communication between proverbial infants and their caregivers. By introducing basic signs for common needs and expressions, parents and caregivers, empower infants to convey their desires and emotions even before they can articulate these thoughts through spoken language.[1] This practice is gaining recognition for several reasons:

- **Enhanced Parent-Child Bonding**: Teaching and using baby sign language can foster stronger emotional connections between caregivers and infants. It provides a means for infants to communicate their needs and feelings effectively, reducing frustration and promoting mutual understanding. The resulting improved parent-child bonding can contribute to a more positive and nurturing environment for the infant's early development.

- **Potential Language Development Advantages**: Research studies have indicated the potential benefits of introducing baby sign language to hearing infants. While ongoing research is further exploring this area, some findings suggest that exposure to baby sign language might accelerate language development in infants. This possibility underscores the significance of effective recognition systems for this unique form of communication.

In this thesis, we focus on the recognition of baby sign language gestures, aiming to contribute to the development of more effective and accurate recognition systems tailored specifically to the needs of infants and their caregivers.

2.3. Previous Approaches to Sign Language Recognition

Sign language recognition is a dynamic field with researchers focusing on diverse sign languages worldwide, often utilizing datasets or creating custom ones, encompassing static and dynamic signs. Here are a few different articles that encompass the current research areas within sign language recognition.

Agapito et al. (2014) pioneered Italian Sign Language recognition, achieving 91.7% accuracy with CNNs and Microsoft Kinect using the CLAP14 dataset. [2] Chillarige et al. (2019) introduced an Indian Sign Language recognition system with a remarkable 99.40% accuracy for 26 alphabets, employing self-compiled datasets. [3] An i3D inception model [4] achieved 100% training accuracy for ASL recognition with 10 words and signers, but validation accuracy dropped as more signers were added, due to overfitting. A real-time ASL hand gesture recognizer achieved 95.52% (GoogLeNet) and 99.39% (AlexNet) accuracy after fine-tuning pre-trained CNNs [5]. Neel Kamal's [6] real-time Indian Sign Language recognition system achieved 98.81% accuracy using 36 gestures and Microsoft Kinect RGB-D data. A mobile-friendly CNN SqueezeNet model achieved 87.47% training and 83.29% validation accuracy for ASL alphabet recognition. [7] A Kinect-based system predicted numerals and 20 English alphabets in sign language with a 70.59% recognition accuracy using SVM. [8] Bhutanese Sign Language digit recognition reached 97.62% training accuracy with CNNs and a dataset of 20,000 sign images. [9] And lastly, a texture-based features using GLCM achieved 73% accuracy for recognizing 60 baby signs with KNN and Random Forest. [10]

A real-time model for Indian Sign Language achieved 98.81% training accuracy with CNNs and 36 static gestures.

These diverse approaches contribute to the evolving landscape of sign language recognition, employing various techniques to enhance accuracy and effectiveness.

2.4. CNN-Based Models in Computer Vision

Convolutional Neural Networks (CNNs) represent a specialized class of deep learning models meticulously crafted to tackle one of the most intriguing challenges in the field of artificial intelligence: understanding and extracting meaningful information from grid-like structures, with images serving as the quintessential example.

Within computer vision, and as seen in Figure 1, a CNN typically has 3 types of layers: a convolutional layer, a pooling layer, and a fully connected layer along with the input and output layer.

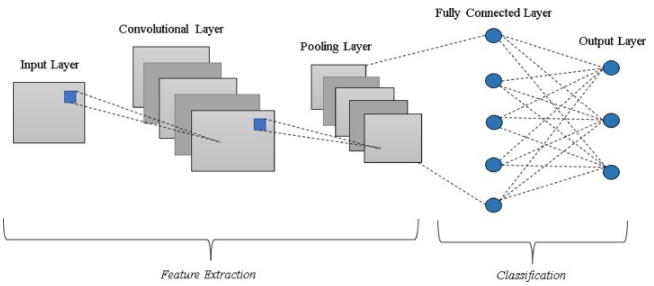

Fig. 1. The architecture of Convolutional Neural Network [11]

Input Layer: The CNN takes an input image represented as a grid of pixels, with color images having three channels (red, green, blue), and grayscale images having one channel.

Convolutional Layer: Using learnable filters, this layer slides over the input image, performing a convolution operation. It creates feature maps by summing weighted pixel val-

ues within receptive fields. Multiple filters capture diverse features.

Activation Function: After convolution, an activation function (e.g., ReLU) introduces nonlinearity, aiding the network in learning complex feature relationships.

Pooling Layers: These layers downsample feature maps, reducing spatial dimensionality while retaining essential information.

Repeat Convolution, Activation, and Pooling: Subsequent layers build abstract features on previous representations.

Flattening: Feature maps from convolutional and pooling layers are flattened into a 1D vector, simplifying spatial information.

Fully Connected Layers: The flattened vector enters fully connected layers, performing standard neural network operations. These layers learn to classify input based on extracted features. The final layer produces output, representing predicted class probabilities or task-specific values.

Training and Optimization: During training, CNN parameters (e.g., filter weights, biases) are optimized to minimize a loss function. Backpropagation computes gradients for parameter updates using optimization algorithms like gradient descent. This fine-tunes the network for accurate predictions.

2.5. Overview of 2D CNN, 3D CNN, and ResNet18

In this project, we explore the potential of several CNN-based models, including 2D CNN [12], 3D CNN [13], and ResNet18 [14], in the context of baby sign language recognition. These models have been successful in various computer vision tasks and hold promise for enhancing the recognition accuracy of baby sign language gestures.

- **2D CNN (Convolutional Neural Network)**: 2D CNNs are primarily designed for processing two-dimensional data, such as images. They accept input in the form of 2D grid-like structures, where each element represents a pixel or a feature. The architecture of a 2D CNN typically includes convolutional layers, pooling layers, fully connected layers, and activation functions. These networks are commonly used in computer vision tasks like image classification, object detection, and image segmentation, as they excel at preserving spatial information in the data. Notable examples of 2D CNN architectures include LeNet, VGG, and AlexNet.

- **3D CNN (Convolutional Neural Network)**: 3D CNNs are specialized for handling three-dimensional data, making them suitable for tasks involving volumetric information, such as videos or medical volumetric scans. They process input in the form of 3D grids of data, where each element represents a voxel (volumetric pixel) or a feature. The architecture of a 3D CNN extends that of 2D CNNs by incorporating 3D convolutional layers, 3D pooling layers, and 3D fully connected layers. These networks are commonly applied in video analysis, action recognition, medical image analysis, and other tasks that require capturing both spatial and temporal information within the data. Examples of 3D CNNs include C3D and 3D-ResNet.

- **ResNet18 (Residual Network-18)**: ResNet18 is a specific architecture within the ResNet family of neural networks. Its design innovation addresses the vanishing gradient problem in very deep networks by introducing residual connections. ResNet18 comprises 18 layers, including convolutional, pooling, and fully connected layers. The key feature of ResNet architectures is the use of residual blocks, which include shortcut connections that bypass one or more convolutional layers. This innovation allows for the direct flow of gradients, improving gradient flow and training stability. ResNet18 finds applications in various computer vision tasks, including image classification and object detection. Pre-trained ResNet18 models are available, making them valuable for transfer learning and feature extraction. The architecture is also scalable to deeper variants, such as ResNet50 or ResNet101, depending on the complexity of the task at hand.

3. DATA COLLECTION AND PREPROCESSING

3.1. Baby Sign Language Video Dataset

To train and evaluate our sign language recognition program, we assembled a comprehensive dataset of baby sign language videos. This dataset includes recordings of infants and caregivers using baby sign language to communicate various common needs and expressions. The videos capture a wide range of gestures and signing styles, reflecting the diversity of communication in this context. We started by collecting just two expressions; *milk* and *eat*. This was done to not only test whether or not the models worked but also which model had the best accuracy and the lowest validation loss (explained in more detail in section 4. With that out of the way, we also worked with the expressions: *I don't know*, *down*, *drink*, *frustrated*, *I love you*, *mad/grumpy*, *mine*, *mom*, *potty* and *sorry*.

3.2. Challenges in Dataset Collection

There is no extensive baby sign language dataset. So we had to go and personally collect all of them. We couldn't automatically use ASL or BSL datasets, because oftentimes words were signed in their more difficult version (for example there are 2 ways to say I love you). We collected this dataset from places such as the "Baby Sign Language" blog [15], the Youtube channels "Bright Signs Learning" [16], "Elite Betty Baby Signs" [17], "Talk Box Mum"[18] and "ASL 101" [19].

3.3. Preprocessing Techniques Applied

3.3.1. Video Processing

To prepare the dataset, we initially processed the videos using the MoviePy [20] library in Python. The following techniques were applied; **Frame Rate Adjustment** where we adjusted the frame rate of the videos to a target of 20 frames per second (FPS) to standardize the temporal aspect of the dataset. **Resolution Standardization** which consisted of resizing the video frames to a common resolution of 1280 x 720 pixels (width x height) to ensure consistent input dimensions for the recognition models. And lastly, **Video Compression**, where the processed videos were encoded using the libx264 codec for efficient storage and compatibility.

These preprocessing steps aimed to create a uniform video dataset suitable for further analysis.

3.3.2. Frame Extraction and Labeling

In addition to video processing, we extracted individual frames from the videos for further analysis. The following steps were followed: The **Frame Extraction**, from each video in the dataset were saved as image files. We used the OpenCV library to achieve this. **Category Labeling**, Consisted of organizing the frames into subdirectories based on their corresponding sign categories. Each category was labeled, and frames were grouped accordingly.

This frame extraction and labeling process was essential for subsequent feature extraction and model training.

3.4. Data Augmentation Strategies

Data augmentation played a crucial role in improving the dataset's diversity and preventing overfitting during model training. By introducing variations in the dataset, we aimed to enhance the model's ability to recognize signs under different conditions. We specifically used the following functions:

- **Color Jittering**: This function adds slight color variations to the image, making it less sensitive to changes in lighting conditions.
- **Gaussian Noise**: Gaussian noise is added to the image to simulate real-world noise and improve the model's noise tolerance.
- **Horizontal & Vertical Flips**: Flipping the image horizontally and vertically creates mirror images, increasing the dataset's diversity.
- **Background Removal**: A background subtraction technique is applied to separate the object of interest from the background.
- **Random Rotation**: The image is randomly rotated by an angle between -60 and 60 degrees to introduce rotational invariance.
- **Brightness & Contrast Adjustment**: This function changes the image's brightness and contrast, mimicking variations in lighting conditions.

3.5. Hand Cropping for Isolating Hand Gestures in Videos

In the realm of computer vision and gesture recognition, one of the essential preprocessing steps is "hand cropping." This process involves isolating and extracting the region of interest, in this case, human hands, from video frames. Hand cropping is a critical step in recognizing and analyzing hand gestures, which have numerous applications, including sign language recognition, human-computer interaction, and virtual reality experiences.

Hand cropping can be implemented using Python and OpenCV, with the assistance of the **cvzone.HandTracking Module** [21]. This code takes video frames as input, detects hands within them, and then carefully crops and saves the hand regions as individual images.

3.5.1. The Significance of Hand Cropping

- **Enhanced Gesture Recognition:** Hand cropping isolates the most critical information, i.e., the hand gestures, from the video frames. This allows for more accurate and efficient hand gesture recognition.
- **Reduced Noise:** By eliminating unnecessary background elements, hand cropping reduces noise and distractions, resulting in cleaner and more reliable input data for gesture recognition models.
- **Versatile Applications:** Hand cropping is essential in various applications, including sign language recognition, virtual reality experiences, and human-computer interaction systems, where understanding hand movements is crucial.
- **Improved Model Training:** Cropped hand images are ideal for training machine learning models. They provide a consistent and focused dataset, enhancing the model's learning process.

4. TRAINING MODELS; THEIR CHALLENGES AND SOLUTIONS

4.1. Problems with 2D CNN

4.1.1. Low Accuracy and Validation Loss

When designing a 2D Convolutional Neural Network (CNN) choosing the right number of layers is a crucial yet challenging aspect to take into account. Too few layers may result in a model that is too simplistic to capture complex patterns and features in the data, leading to bad performance. On the other

hand, an excessively deep network can suffer from overfitting, vanishing gradients, and increased computational complexity. Finding the right balance requires a deep understanding of the dataset and the problem at hand, often involving a process of trial and error. Consequently, this choice involves careful experimentation and validation to achieve something that both extracts meaningful features and generalizes well to unseen data.

4.1.2. Addressing the Issues

Our attempt to train the 2D CNN model yielded unsatisfactory results with poor accuracies when applied to our training frames. Despite implementing a relatively simple convolutional neural network architecture with two convolutional layers, ReLU activation functions, max-pooling layers for down-sampling, and fully connected layers for classification, the model's performance crashed. Several factors may have actually contributed to this poor performance, such as the model's depth and complexity, the choice of hyperparameters, the quality and diversity of the training dataset, and the presence of any class imbalances or data preprocessing issues. To address these challenges and improve model performance, we tried to experiment with different architectures and at the very end we considered an alternative deep learning approach, focusing on the possibility of using a modified version of a pre-trained model.

4.1.3. Validation Performance Issues and Causes of Validation Loss Increase

The performance of a machine learning model is often measured using metrics like validation loss. While training it is not uncommon to encounter issues where the validation loss starts increasing instead of steadily decreasing. This phenomenon, known as "validation loss increase," is indicative of several underlying causes.

One key cause of validation loss increase is overfitting. Overfitting occurs when the model learns to fit the training data too closely, capturing noise and irrelevant patterns, rather than generalizing. As a result, when the model encounters data from the validation set, which it hasn't seen during training, it performs poorly because it has essentially memorized the training data.

Another cause can be a high learning rate or poor learning rate scheduling. A learning rate that is too large can cause the model to overshoot the optimal parameters during training, leading to divergent behavior and increased validation loss. Conversely, a learning rate that is too small may result in slow convergence or getting stuck in local minima.

Additionally, insufficient regularization techniques or the absence of techniques like dropout, weight decay, or early stopping can result in overfitting and contribute to validation loss increase. Furthermore, there could be issues related to data preprocessing, such as inconsistent data scaling or missing data that can also lead to unexpected increases in validation loss.

Addressing these validation loss increase issues involves fine-tuning hyperparameters, employing regularization methods, ensuring a representative and diverse training dataset, and monitoring the learning rate. These steps aim to obtain a balance between model complexity and generalization, ultimately leading to better model performance and lower validation loss.

4.1.4. Overcoming Validation Challenges

We initally divided our dataset into training and validation sets manually to assess our model's performance. However, we encountered the significant issue mentioned above: the validation loss was consistently poor, indicating that there was probably a problem with our data split. To overcome this challenge, we chose a more robust approach by implementing k-fold cross-validation, opting for 2 and 3 kfold cross-validation, given our relatively small dataset and poor computational resources. K-fold cross validation technique allowed us to divide the dataset into 'k' subsets, letting us train and validate our model 'k' times, rotating the subsets each time. This way, we obtained a more comprehensive evaluation of our model's performance, mitigating the issues associated with an inadequate initial data split and leading to more reliable results.

4.2. Dataset Characteristics

4.2.1. Frame Count and Size

As previously mentioned, acquiring our dataset proved to be a significant challenge. We found it necessary to create a custom dataset tailored to our specific objectives, as the existing datasets were primarily made for the alphabet of ASL, BSL etc. Our dataset primarily comprised videos and our chosen pre-trained model was Resnet18. In order to leverage the power of this model in a video context, given its specialty in images, we decided to directly extract frames and ensure uniformity by standardizing both the frame size and frame count across all videos.

To achieve this consistency, we resized all frames to dimensions of 1280 x 720 and established a consistent frame rate of 30 frames per second (fps) following the extraction process.

4.2.2. Dataset Size and Diversity

In the beginning, our dataset was rather modest in size, as we initially conducted experiments with just two words ("eat", "milk") to assess the functionality of our model. We gathered approximately 8 videos, each lasting 2 seconds, with a primary focus on capturing the gestures. Our goal was to diver-

sify as much as possible, encompassing various individuals and backgrounds. At that point, we had around 300 frames for each word, totaling 600 frames. However, during the initial training attempts, our model consistently exhibited signs of overfitting which prompted us to explore data augmentation techniques to expand our dataset, ultimately growing it to a total of 2,500 frames. With this enhanced dataset, our training outcomes substantially improved.

Encouraged by this progress, we decided to incorporate an additional 10 words into our dataset. Once again, we applied data augmentation techniques, resulting in the expansion of our dataset to a more substantial 10,000 frames.

4.2.3. Importance of Data Augmentation

Data augmentation played a pivotal role in enhancing the quality of our dataset. Not only did it expand our initially limited dataset in terms of quantity, but it also introduced some variations. Noise injections mimicked real-world imperfections, while flips and rotations introduced spatial variations that mirrored the unpredictability of gesture presentations. These augmentations proved to be a valuable asset in training our model to recognize gestures even when presented with slightly distorted frames, ultimately contributing to its robust performance.

5. EXPERIMENTAL RESULTS AND DISCUSSION

5.1. Performance Metrics

At the end of training, we evaluated the models' performance using several metrics, including both training and validation accuracy.

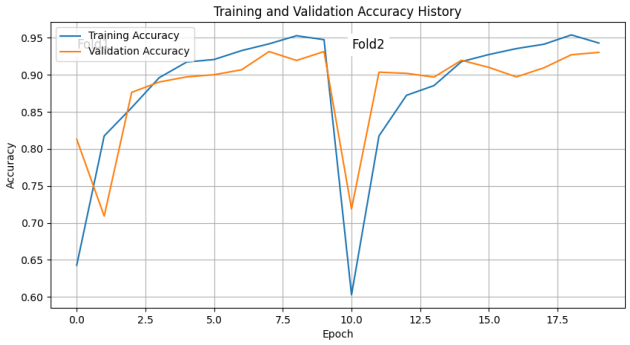

Fig. 2. Results when using: 2 Folds, 10 epoch, batch size 16

With figure 2 we started by using a small batch size but we did only keep it to 2 folds. This was done to then test it and see whether it worked and determine its effectiveness. During the testing, we found a few words in which the model struggled to recognize them (e.g. *frustrated* and *mad/grumpy*).To try to fix the problems with the previous model we tried to increase the number of folds and epochs but we had to decrease the batch size to 32, otherwise it would have taken too long to train. This can be seen in figure 3.

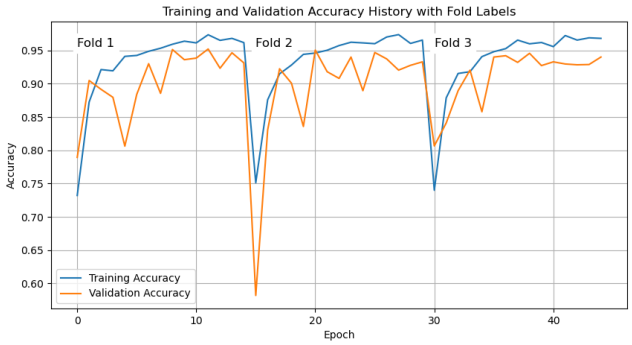

Fig. 3. Results when using: 3 Folds, 15 epoch, batch size 32

Figure 3 shows us as the epochs increased, so did the accuracy (form 94% to 96%) and having more folds gives you a higher chance of getting a better model. But when actually testing the data we realized that two expression specifically were not being recognized. These being *eat* and *I don't know*. When looking back at the dataset we realized the that hand-tracking module from CVZone [21] had removed quite a number of frames because of blurring, background clutter or occlusion. An explanation on how it was fixed is in the following section 5.2. After fixing it we reran the data with immense success, as seen in figure 4.

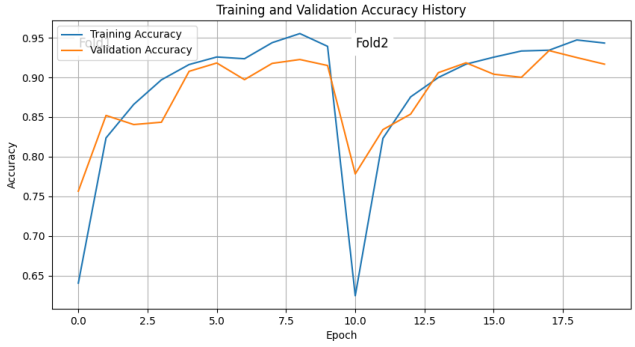

Fig. 4. Results when using: 2 Folds, 10 epoch, batch size 16 with dataset after removing frames that were deemed "bad" and the addition of new frames from other videos to make sure that each class (i.e. word) had approximately the same number of videos.

Figure 4 only used 2 folds and an epoch of 10 due to timing issues and available computing CPU, but it still resulted in the best results when running the test even though it ended with a lower training and validation accuracy compared to the previous two models.

5.2. Impact of Preprocessing Techniques

As seen in the difference between figure 2 and figure 4, we noticed that due to the hand-tracking module from CVZone [21] removing quite a number of frames because of blurring, background clutter or occlusion. Due to this we had to go back and add new frames from other videos to make sure that each class (i.e. word) had approximately the same number of videos. This means that next time within preprocessing section, after applying the hand-tracking module, it is advised to check the size of the classes to make sure that the number of frames are more or less equal.

6. CONCLUSION AND FUTURE WORK

In summary, our project on baby sign language has shown promising results despite its relatively small scale. It's important to recognize that the potential impact of such a gesture recognition program is huge and could be significantly enhanced with access to a larger dataset and more computational resources. With a bigger and more diverse range of signs, we could create a more robust and versatile system that would benefit not only parents and caregivers but also individuals with communication challenges. One effective way to expand it would be encouraging people to contribute by sharing videos of these gestures, giving a contribution to improve the accuracy and inclusivity of our recognition program. By harnessing the power of community involvement and access to more resources, we can strive to make this technology an even more valuable tool for those seeking to bridge communication gaps with non-verbal individuals.

7. REFERENCES

[1] Kylie Rymanowicz and Frank Cox, "Baby sign language: A helpful communication tool," Feb 2023.

[2] Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, and Benjamin Schrauwen, "Sign language recognition using convolutional neural networks," in *Computer Vision - ECCV 2014 Workshops*, Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, Eds., Cham, 2015, pp. 572–578, Springer International Publishing.

[3] Mehreen Hurroo and Mohammad Elham, "Sign language recognition system using convolutional neural network and computer vision," *International Journal of Engineering Research amp; Technology*, Dec 2020.

[4] Suharjito Suharjito, Herman Gunawan, Narada Thiracitta, and Ariadi Nugroho, "Sign language recognition using modified convolutional neural network model," 09 2018, pp. 1–5.

[5] Manuel Eugenio Morocho-Cayamcela and Wansu Lim, "Fine-tuning a pre-trained convolutional neural network model to translate american sign language in real-time," 01 2019, pp. 100–104.

[6] Neel Kamal Bhagat, Vishnusai Y, and Rathna G N, "Indian sign language gesture recognition using image processing and deep learning," 2019, IEEE.

[7] Nikhil Kasukurthi, Brij Rokad, Shiv Bidani, and Dr. Aju Dennisan, "American sign language alphabet recognition using deep learning," 2019.

[8] Sulochana Nadgeri and Arun Kumar, "Deep learning based framework for dynamic baby sign language recognition system," *Indian Journal of Computer Science and Engineering*, vol. 13, pp. 550–563, 04 2022.

[9] Karma Wangchuk, Panomkhawn Riyamongkol, and Rattapoom Waranusast, "Real-time bhutanese sign language digits recognition system using convolutional neural network," *ICT Express*, vol. 7, no. 2, pp. 215–220, 2021.

[10] Sulochana Nadgeri and Arun Kumar, "An image texture based approach in understanding and classifying baby sign language," in *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, 2019, vol. 1, pp. 854–858.

[11] H. Mary Shyni and E. Chitra, "A comparative study of x-ray and ct images in covid-19 detection using image processing and deep learning techniques," *Computer Methods and Programs in Biomedicine Update*, vol. 2, pp. 100054, 2022.

[12] Anjaly S Menon, C J Sruthi, and A Lijiya, "A 2d fast deep neural network for static indian sign language recognition," in *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*, 2022, vol. 1, pp. 311–316.

[13] Jie Huang, Wengang Zhou, Houqiang Li, and Weiping Li, "Sign language recognition using 3d convolutional neural networks," in *2015 IEEE International Conference on Multimedia and Expo (ICME)*, 2015, pp. 1–6.

[14] Shiqi Wang, Kankan Wang, Tingping Yang, Yiming Li, and Di Fan, "Improved 3d-resnet sign language recognition algorithm with enhanced hand features," *Scientific Reports*, vol. 12, no. 1, pp. 17812, 2022.

[15] Baby Sign Language, "Baby Sign Language," .

[16] Bright Signs learning ABC, "Bright signs learning ABC on YouTube," .

[17] Ettie Betty, "ettiebettybabysigns5953 on YouTube," https://www.youtube.com/@ettiebettybabysigns5953.

[18] TalkBoxMom, "TalkBoxMom on YouTube," .

[19] William Vicars, “Asl university,” About American Sign Language (ASL) University.

[20] Zulko, “Moviepy,” User Guide - MoviePy 1.0.2 documentation.

[21] Cvzone, “Cvzone/cvzone: This is a computer vision package that makes its easy to run image processing and ai functions. at the core it uses opencv and mediapipe libraries.,” .