

Sign Language Detection Using Machine Learning

**Program: Computer Engineering and Software Systems Program**

***Course Code: CSE-492***

***Course Name: Computer & Systems Engineering Graduation Project (2)***

***Graduation Project Supervisor***

**DR. Haytham Azmi Mahmoud Elmiligi**

**Ain Shams University**

**Faculty of Engineering**

**Spring Semester – 2023**

**Students Personal Information**

|  |  |  |
| --- | --- | --- |
| **NAME** | **ID** | **MAJOR** |
| Khaled Medhat Mahmoud Khalifa | 18P3557 | CESS |
| Mohamed Magdy Mostafa Sayed Ahmed | 18P5160 | CESS |
| Yomna Hussien Mohamed Abd El Hamid | 18P5794 | CESS |
| Mohamed Sayed Awwad Badawy | 18P7298 | CESS |
| Mahmoud Mourad Youssef | 18P6555 | CESS |

**Plagiarism Statement**

I certify that this report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this report has not previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

**Signature/Student Name:**

**Mohamed Magdy Mostafa Sayed Ahmed**

**Yomna Hussien Mohamed Abd El Hamid**

**Khaled Medhat Mahmoud Khalifa Mahmoud Mourad Youssef**

**Mohamed Sayed Awwad Badawy**

Table of Contents

[INTRODUCTION 2](#_Toc137675372)

[RELATED WORK 3](#_Toc137675373)

[COMPARISON 17](#_Toc137675374)

[I. Recurrent Neural Networks (RNNs) 17](#_Toc137675375)

[II. Long Short-Term Memory (LSTM) 17](#_Toc137675376)

[III. Gated Recurrent Unit (GRU) 17](#_Toc137675377)

[STATIC SIGN LANGUAGE 18](#_Toc137675378)

[I. Dataset 18](#_Toc137675379)

[Dataset Description 18](#_Toc137675380)

[Pre-processing and data augmentation 19](#_Toc137675381)

[II. Methodology 22](#_Toc137675382)

[III. Experimental Evaluation and Results 23](#_Toc137675383)

[Initial Version of Models 23](#_Toc137675384)

[Final Version of Models 25](#_Toc137675385)

[DYNAMIC SIGN LANGUAGE 32](#_Toc137675386)

[I. Dataset 32](#_Toc137675387)

[II. Methodology 32](#_Toc137675388)

[Phase One 33](#_Toc137675389)

[Phase Two 34](#_Toc137675390)

[Phase Three 34](#_Toc137675391)

[III. Experimental Evaluation and Results 34](#_Toc137675392)

[BIOMETRIC SCANNER 39](#_Toc137675393)

[I. Dataset 39](#_Toc137675394)

[II. Methodology 39](#_Toc137675395)

[III. Experimental Evaluation and Results 41](#_Toc137675396)

[CONCLUSION 51](#_Toc137675397)

[REFERENCES 53](#_Toc137675398)

Table of Figures

[Figure 1 Top 25 Words Histogram 18](#_Toc137675291)

[Figure 2 Top 25 Static Words Histogram 19](#_Toc137675292)

[Figure 3 Full Image of Word “Shirt” 20](#_Toc137675293)

[Figure 4 Cropped Image of Word “Shirt” 20](#_Toc137675294)

[Figure 5 Flipped Image 21](#_Toc137675295)

[Figure 6 Initial Models Validation Accuracy 23](#_Toc137675296)

[Figure 7 Initial Models Validation Loss 23](#_Toc137675297)

[Figure 8 Inception-V3 Heatmap 24](#_Toc137675298)

[Figure 9 Initial VGG-16 Heatmap 24](#_Toc137675299)

[Figure 10 Initial MobileNet-V2 Heatmap 24](#_Toc137675300)

[Figure 11 Initial ResNet50-V2 Heatmap 24](#_Toc137675301)

[Figure 12 Initial Models Accuracies 25](#_Toc137675302)

[Figure 13 Initial Models F1-Scores 25](#_Toc137675303)

[Figure 14 Models Validation Accuracy 26](#_Toc137675304)

[Figure 15 Models Validation Loss 26](#_Toc137675305)

[Figure 16 EfficientNet Validation Accuracy 27](#_Toc137675306)

[Figure 17 EfficientNet Validation Loss 27](#_Toc137675307)

[Figure 18 VGG-16 Heatmap 28](#_Toc137675308)

[Figure 19 VGG-19 Heatmap 28](#_Toc137675309)

[Figure 20 ResNet50-V2 Heatmap 28](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675310)

[Figure 21 MobileNet-V2 Heatmap 28](#_Toc137675311)

[Figure 22 ResNet101-V2 Heatmap 29](#_Toc137675312)

[Figure 23 EfficientNet-B7 Heatmap 29](#_Toc137675313)

[Figure 24 EfficientNet-V2L Heatmap 29](#_Toc137675314)

[Figure 25 EfficientNet-V2M 29](#_Toc137675315)

[Figure 26 Models Accuracy Comparison 30](#_Toc137675316)

[Figure 27 Models F1-Score Comparison 30](#_Toc137675317)

[Figure 28 VGG-16 Results 31](#_Toc137675318)

[Figure 29 Hand Landmarks 32](#_Toc137675319)

[Figure 30 Pose Landmarks 33](#_Toc137675320)

[Figure 31 DynamicSL 100 Epochs Error Graph 35](#_Toc137675321)

[Figure 32 DynamicSL 60 Epochs Report 35](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675322)

[Figure 33 DynamicSL 60 Epochs Error Graph 35](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675323)

[Figure 34 DynamicSL 50 Epochs Error Graph 36](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675324)

[Figure 35 DynamicSL 50 Epochs Report 36](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675325)

[Figure 36 DynamicSL 20 Epochs Error Graph 37](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675326)

[Figure 37 DynamicSL 20 Epochs Report 37](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675327)

[Figure 38 DynamicSL Heatmap 38](#_Toc137675328)

[Figure 39 KNN K values 41](#_Toc137675329)

[Figure 40 Necessary Equations 42](#_Toc137675330)

[Figure 41 Recall Comparison 43](#_Toc137675331)

[Figure 42 Precision Comparison 43](#_Toc137675332)

[Figure 43 Accuracy Comparison 44](#_Toc137675333)

[Figure 44 F1 score Comparison 44](#_Toc137675334)

[Figure 45 Specificity Comparison 45](#_Toc137675335)

[Figure 46 KNN Prediction on Unlabeled Image Dataset 46](#_Toc137675336)

[Figure 47 KNN Prediction on Labeled Image and Normalized Dataset 46](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675337)

[Figure 48 KNN Prediction on labeled Image Dataset 46](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675338)

[Figure 49 Models Prediction on Unlabeled Image Dataset 47](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675339)

[Figure 50 Models Prediction on Labeled Image Dataset 48](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675340)

[Figure 51 Models Prediction on Labeled Image and Normalized Dataset 49](https://d.docs.live.net/9939176adc34a852/Documents/Graduation%20Project%20phase%201%20sumamry.docx#_Toc137675341)

**Abstract— Communication is essential for every human being, it allows us to request help, enlighten others with important information, and form bonds by sharing attitudes. According to the World Health Organization, currently more than 1.5 billion, which is nearly about 20% of the global population live with hearing loss. To communicate, people who are deaf and mute must use sign language which can be an obstacle for them in everyday communications.**

**To solve this problem, our proposed system will work as a sign language translator where it will convert real-time American sign language into text. The system will use techniques based on deep learning and computer vision to recognize the input gesture from a live camera feed and convert it into text. The dataset used will be American Sign Language based dataset. This system will help in overcoming the communication barrier that exists between people who don’t understand sign language and people with speech and hearing impairment.**

**The aim of this project is to develop a sign language detection system using machine learning. The proposed solution leverages Convolutional Neural Networks (CNNs) for static sign detection and Long Short-Term Memory (LSTM) networks, combined with point detection using MediaPipe hand and pose detection modules, for dynamic sign detection, as well as a biometric scanner that uses the hand dataset gathered by the team members. The project addresses the challenge of recognizing sign language in real-time and accurately, which is a fundamental need for the deaf and hard-of-hearing community.**

**The proposed solution employs pre-processing and fine-tuning techniques to train the CNN and LSTM models, respectively, using publicly available datasets as well as manually collected ones. The models are then integrated into a real-time application that captures the user's hand movements and translates them into sign language. The project presents a comprehensive evaluation of the system's performance, including accuracy, speed, and robustness.**

**The project overcomes several challenges in sign language recognition, including variations in hand shapes and orientations. The proposed solution achieves high accuracy in detecting both static and dynamic signs in real-time. The project provides a step forward in enhancing communication between hearing and non-hearing communities and contributes to creating a more inclusive society.**

**The solution design takes into account the need for a user-friendly, affordable, and accessible system, and hence the proposed solution can be implemented on a variety of devices such as smartphones, tablets, and laptops. Overall, this project showcases the potential of machine learning in addressing societal challenges and promoting diversity and inclusivity.**

**This research project explores three key aspects of sign language recognition: static and dynamic signs recognition, biometric scanner, and finally mobile and web application. However, before delving into each of these topics, a brief overview of the project's achievements in the first half is presented.**

**To select appropriate technologies, techniques, frameworks, and datasets, the team conducted extensive research by reviewing relevant literature. Based on the results of these studies, the American Sign Language (ASL) dataset was chosen because of its availability on platforms like Kaggle. As the team required a larger dataset, we collected additional data by learning the signs themselves and capturing photos manually.**

**Deep learning was chosen as the primary technique for the project due to the high accuracy of Convolutional Neural Networks (CNNs) in image recognition. Prior to training the model, data pre-processing was conducted using MediaPipe's hand detection module, which automatically detects the hand and crops it for input to the model. This pre-processing was performed for both training and prediction.**

**Finally, for better presentation of the project, the team developed a web app to integrate the algorithm into a more user-friendly interface than a mere notebook. The app provides a platform for the team's developed algorithm to be used within a website or an interface that is more approachable to users.**

**In summary, this research paper provides insights into the initial phase of the team's sign language recognition project, where they selected the appropriate dataset and deep learning technique, pre-processed the data using MediaPipe's hand detection module, and developed a web app to make the algorithm more accessible. These steps set the foundation for the subsequent stages of the project, where dynamic signs recognition, biometric scanner, and the mobile application were developed.**

**Keywords— deep learning, machine learning, sign language, computer vision, hand gesture recognition, hand biometric scanner, convolution neural network (CNN), long short term memory (LSTM)**

# INTRODUCTION

## Motivation

In recent years, video conferencing has revolutionized the way people connect and collaborate, enabling seamless communication across geographical boundaries. However, while video conferencing has become an essential tool for many, it poses challenges for individuals with hearing impairments or those who rely on sign language as their primary means of communication. These individuals often face barriers when trying to participate in virtual meetings, as existing video conferencing software tools lack the capability to effectively interpret and understand sign language.

## Research Problem

To address this crucial accessibility gap, this project introduces a pioneering machine learning model designed specifically for sign language detection from images and video frames. By leveraging the power of advanced computer vision techniques and machine learning algorithms, this model aims to empower individuals with disabilities to actively engage in video conferencing, ensuring their equal participation and inclusion in professional and social settings.

The significance of integrating this machine learning model with existing video conferencing software tools cannot be overstated. By enabling real-time sign language detection, the model offers several transformative benefits for individuals with hearing impairments. Firstly, it facilitates seamless and efficient communication between sign language users and non-sign language users within virtual meetings, transcending the barrier of language. This integration ensures that people with disabilities can actively participate, express their thoughts, and be fully understood by others, fostering equal opportunities, and promoting diversity in the workplace and beyond.

Moreover, the integration of this model with video conferencing software holds immense potential in facilitating access to vital services for individuals with hearing impairments. Services such as remote education, telemedicine, customer support, and legal consultations can become more inclusive and accessible with the provision of real-time sign language interpretation. This breakthrough technology not only enhances the overall user experience but also dismantles the existing barriers that have hindered individuals with disabilities from enjoying the same level of service as their hearing counterparts.

## Project Contribution

|  |  |
| --- | --- |
| **Name** | **Contribution** |
| **Mohamed Magdy** | * Static SL dataset * Static SL algorithm * Static SL experimental evaluation & results * Biometric Scanner dataset * Biometric Scanner algorithm * Biometric experimental evaluation & results |
| **Yomna Hussien** | * Static SL dataset * Static SL algorithm * Static SL experimental evaluation & results * Biometric Scanner dataset * The Application |
| **Khaled Medhat** | * Static SL dataset * Static SL algorithm * Static SL experimental evaluation & results * Dynamic SL dataset * Dynamic SL algorithm * Dynamic SL experimental evaluation & results * Biometric Scanner dataset |
| **Mahmoud Mourad** | * Static SL dataset * Dynamic SL dataset * Dynamic SL experimental evaluation & results * Biometric Scanner dataset |
| **Mohamed Sayed** | * Static SL dataset * Static SL experimental evaluation & results * Biometric Scanner dataset * Biometric Scanner algorithm * Biometric experimental evaluation & results |

# RELATED WORK

Sign Language recognition is not a new computer vision problem. There have been many papers and articles related to this problem over the past twenty years, and many researchers have used different methods from what form of color are the pictures in, how to handle the noise, classifying, and many other techniques that are essential in the SL recognition process.

We’ve looked at many articles and carefully inspected the used methods. Real-time ASL recognition with CNN [1] uses CNNs for classification, which is mostly used for static SL recognition, combined with a SoftMax-based loss function which is accompanied by transfer learning by using pre-trained models with newly trained ones. It also employed Caffe which is a deep learning framework and they used Berkeley Vision and Learning Center’s Google Net pre-trained on 2012 ILSVRC dataset.

The overall approach was to fine-tune the pre-trained model, data is composed of hands in 24 different orientations. The input goes through a web application that is coded in HTML and JavaScript done using an API created by the W3C. Capture rate was a problem because of network request speeds with computation speeds of the neural network whose processing speed limited the capture rate to 1 frame per second. Web application sends images to server one by one, then the server classifies each image and presents probabilities for each letter and then keeps a running cache of classified images; when it feels confident about the sign being made by the user, it records the top 5 most likely letters based on the cache, then it lets the user know to move on to the next letter. Data augmentation zero-centered the data by subtracting the mean image from ILSRVC dataset and making horizontal flips of the image. Problems faced were lighting, background and camera positions, occlusion, sign-boundary detection, and co-articulation.

Another article was Real-time ISL Recognition with Deep Learning [2] which uses a CNN trained model from scratch and a pre-trained VGG19. It used Image Data Generator for data augmentation which generates batches of tensor images to be looped, it uses OpenCV for human-computer interaction, uses Adam Optimizer and categorical cross-entropy loss for CNN, it uses SGD and same loss as CNN, and it has an accuracy of 97% after training for 100 epochs on frontal sign images only.

Another one was Dynamic SL Recognition Based on Improved Residual-LSTM Network [3] which uses YOLO, residual LSTM network and spatial feature extraction. Its method was firstly, a hand positioning module based on framework of YOLO which is pretrained with marked Oxford hand dataset which is used to capture the position information of the hand, then the video frames are trained by convolution layer for feature extraction, then the ROI area of the hand is obtained through the target detection network, then the hand region could be segmented from the background, then the segmented video frames are inputted into spatial feature extraction, then each video feature vector will be provided to the third part for analyzing dynamic information of sign language, lastly, dynamic sign language recognition module which can analyze long-term temporal dynamics and predict the hand gesture label. LSTMs were used here to extract temporal features that CNNs cannot operate on, and it uses SGD with a 16-batch size.

Datasets used were Oxford hand dataset which contains 4170 hand instances and SLR dataset (Chinese) which contains 500 Chinese sign words.

Another was SL Recognition for Static and Dynamic Gestures [4] which has two different methodologies for handling static and dynamic signs, the first one, static, uses CNNs and OpenCV to capture video from the user’s webcam, after capturing the video, it takes a single frame and defines a region of interest (ROI) in that frame. Then, discussing the skin segmentation, the ROI of the frame is transformed into a hand-masked image to provide to the model for predictive purposes; First, blur the image to reduce noise (gaussian blur). After blurring, ROI is converted to HSV color scale in RGB which helps detect better skin than RGB.

As for the dynamic part, they can’t use CNNs because of the need to keep the previous state which uses LSTMs are used. Input continuously delivers a sequence of 8 frames/images extracted from images in the training dataset. Applies an RGB difference filter before serving these 8 frames as input. The RGB differences subtracts the current frame from the previous frame, therefore, only the changed pixels remain in the frame and the remaining still images are deleted. Next layer is MobileNetV2 which accepts only up to 224 x 224 pixels, so they had to resize the frames.

Another paper which was published in Cairo University, 2013 titled Dynamic Hand Gesture Recognition of Arabic Sign Language using Hand Motion Trajectory Features [5] which consists of three modules: hand extraction, feature extraction and gesture recognition module.

Hand extraction extracts the hand area from the input video stream which uses Haar classifier to detect the face from the captured frames, then it replaces the face with a black ellipse to eliminate the confusion between the hand and the face, lastly, the image is converted to YCbCr color space to detect skin color.

Feature extraction module calculates the 14 features for the hand motion trajectory, and the output of this stage is a binary image which shows only the hand blob (calculated features are: center of gravity, area, perimeter, orientation and seven hu moments)

Last module is the object recognition module which uses correlation coefficient to match the features of the input gesture to the stored ones in the database in the learning phase.

As for the dataset, the dynamic gestures from the Arabic sign language dataset were unavailable, so they built their own database in which they collected 20 different signs from eight different signers at different situations with a 780 x 480 resolution.

Another one was Real-time ASL Recognition using Skin Segmentation and Image Category Classification with CNN and Deep Learning [6], the methodology of which was as follows: they used CNN so that time required to extract features from images can be as fast as possible, they then used transfer learning using MATLAB 2017a which comes with a pretrained AlexNet. They generated a feature vector of length 4096, captured images and extracted features using CNN transfer learning which are then passed onto a deep learning classifier to ensure proper classification. Testing accuracy was 94%. In this paper, they used YbCr because it’s more accurate in detecting human skin, and they adapted the aforementioned method of first capturing the background then the human to be able to extract the human skin more easily without the added noise. They applied a denoising step in order to account for external noise and camera noise. Image feature extraction collected 150 images per class and trained CNN to extract a feature vector of length 4096 for four classes, then they used a highly pre-trained built-in NN AlexNet with 25 layers and replaced the last layer with a feature extraction layer.

Another paper was Real-time SL Recognition Using Computer Vision [7] which uses CNN (static signs), and HSV instead of RGB because RGB colors are all co-related to the color luminance, while HSV on the other hand is used to separate image luminance from color information. Then a mask is created and passes through processing steps which are defined in details in the paper. They used a dataset that consisted of 240 images of 10 images for each alphabetical sign (two signs J and Z which require motion are not part of the dataset due to the lack of LSTM usage and complete dependency on CNN).

Intelligent Sign Language Recognition Using Image Processing [8], they use the webcam to capture images then change this RGB image into grayscale so they can be able to generate coordinates of the fingertips of this captured image which is used in comparing the user entered image with the stored ones after that the sign will be converted into corresponding text and audio.

Indian Sign Language Detection and Recognition Using Deep Learning [9], with a dataset of 11 Indian words only, their F1-Score successfully achieved 0.97. It was done by a combination of two layers of LSTM and GRU. They divide the resulting videos into frames and in order to extract the features from the frames InceptionResNetV2 was used in the proposed architecture then these features are passed to a recurrent neural network to predict the correct word.

Indian Sign language (ISL) Recognition Using Machine Learning Algorithm [10], first, they convert RGB images to binary images then cropping of image is to be done so that unwanted part of images can be removed after that they use edge detection method that can detect the boundary of cropped images which is further used for feature extraction method. By using Support Vector Machine (SVM), The results on these experiments have a 100% accuracy rate.

Saudi Sign Language based on Convolutional Neural Networks [11], using a couple of convolution layers, max pooling, and dropout, they achieved 99.47% for the testing data, they have done it by using 40 Saudi signs with about 700 images for each sign. Also, they have two types of applications Mobile application which was built using Flutter with Dart language to program the interface, backend was built using TensorFlow Lite and a Desktop application was built using TKinter.

Training CNNs for 3D Sign Language Recognition with Color Texture Coded Joint Angular Displacement Maps [12], they created a 3D mocap SL dataset consisting of 200 Indian SL signs this resulted in a total of 20,000 3D sign videos. All 3D sign skeletons were represented using 57 human upper body joints, they derived color texture JADMs for each video where they compute JADMs from the 3D data then they Encode the JADMs into RGB images and finally they pass these images to some convolution layers to be trained.

Sign Language Recognition Based on Computer Vision [13], by using a downloaded ASL sign language dataset from Kaggle 12,500 and a well-designed desktop application where the users can either select sign language recognition, translation capabilities and capture images via OpenCV or select sign language generation by recognizing where the user will be able to add a name of the sign and record a video of it. Feature extraction is carried out by the improved Eiffel tower, and the Inception-V3 architecture is adopted. This feature is passed to an LSTM to train this dataset.

In Recognition of Sign Language Using Image Processing [14], they focused on translating the sign alphabet by computing the histogram of the input image and checking for similarity with the histograms of pre-saved images by using the Bhattacharyya, first they captured the image by the camera where they used OpenCV for image processing, then they change it from BGR color space to HSV color space because the HSV color space represents the RGB color space in a cylindrical co-ordinate form, also HSV separates luma, or the image intensity, from chroma or the color information which is not achieved in the RGB color space, secondly the histograms of all the images are computed, and by analyzing the histogram of a particular image, we can get an idea about brightness, contrast, intensity distribution and various other parameters of the image, then the Bhattacharyya Distance is used to calculate the similarity between the histograms and smallest value it gets is the right sign, finally the right alphabet sign is displayed as output, but unfortunately there were some constraints such as the input image should have a black background, accessories should not be present on the hand depicting the letter, and the finger spelling should not contain any kind of movement. Hence, the letters J and Z cannot be detected by this system.

In Recognition of American Sign Language Using Image Processing and Machine Learning [15], they used CNN architecture, consisting of multiple convolutional and dense layers, the architecture included 3 groups of 2 convolutional layers followed by a maxpool layer and a dropout layer, and two groups of fully connected layer followed by a dropout layer and one final output layer. The images were captured via the code of opening a webcam through OpenCV and frames will be captured every second which will be stored in another directory where all the input images are stored in another directory and then comparison of the captured image and the pre-stored images are made using SIFT algorithm. SIFT approach takes a picture and transforms it into a “big collection of local feature vectors”, each of the feature vectors never changes to any of scaling, rotation, or translation of the image. The gesture will be identified of the input of hand movement and on the completion of the entire process the application will be then translated into its recognized character or alphabet from the gesture, 1- dimensional array of 26 characters corresponding to alphabets has been passed where the image number stored in the database is provided in the array then the recognized text is converted to speech and an audio output is executed. For the implementation first there is the image acquisition model, then the pre-processing model where improvements are done to the image data to reduce unwanted deviation or enhances image features for further processing, then the cropping to remove the unwanted parts of an image to improve framing, accentuate subject matter or change aspect ratio, then the resizing where images are resized to suit the space allocated or available, then the feature learning which is comprised of one or more convolutional layers and followed by one or more fully connected layers as in a standard multilayer neural network. It implicitly extracts relevant features from a Feed-forward network that can extract topological properties from an image, CNNs are trained with a version of the backpropagation algorithm, then there is pooling layer to reduce the spatial size of the representation to reduce the number of parameters, with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along with both the width and the height, discarding 75% of the activations spatially, using the MAX operation, then the ReLU layer which increases the nonlinear properties, then the fully connected layer where neurons in a fully connected layer have full connections to all activations in the previous layer. The activations are computed with matrix multiplication.

In Towards Continuous Sign Language Recognition with Deep Learning [16], they used NGT1 corpus where participants were recorded storytelling, The mean length of a sign is 6.75 frames where one frame length is approximately 40 milliseconds, The average amount of examples per sign is approximately 11 videos and it wasn’t enough, additional data was generated using extracted features from the original data. For every video example of the real data, 200 more examples were synthesized by adding perturbation along both x and y axes to the extracted features. For feature extraction, a standard camera is used, and features are extracted with the help of the deep learning techniques provided by the openpose library. The main assumption for the segmentation is that the hands move slower during the signing than during the motion epenthesis. Motion epenthesis are identified by looking at the distance traveled by each hand an interval, the centroids of all the hand points are calculated and accumulated for the period of 5 frames (T1-T5) the minimum bounding box is calculated. At the end, the longest side of the minimum bounding box (either H1 or H2) is taken to decide whether the segment is motion epenthesis or a part of the sign. In Classification, and with the video segmented, isolated sign language recognition is done by training deep learning models using TensorFlow and openpose libraries, The architecture is composed of three stacked LSTM layers with the first two layers producing a sequence of vectors with 32 dimensions and the last LSTM layer producing a single vector, composed of 32 dimensions. At the output of the network, the dense layer outputs the likelihood of every sign, The first layer accepts a sequence of inputs of length equal to the number of extracted features per one frame. The maximum number of chunks is set to be the longest sequence of frames for a sign and all other sequences are padded at the end with zeros. The network is trained offline with the objective function set to categorical cross entropy and the optimizer set to resilient backpropagation with the adaptive learning rate, which is a good choice for the recurrent neural networks. The dataset is split into training 80%, validation, and testing sets 10%. The segmentation accuracy is then tested with f-measure, precision, and recall while for classification test, the training can produce an effective model for the recognition of the signs. However, the training is not stable, the accuracy fluctuates between the epochs and occasionally drops down to the random choice accuracy level. When the model is trained with facial features, the performance degrades, because the input feature vector is increased in size, while when the number of features is reduced from full facial to reduced facial information, the accuracy increases, but does not surpass the accuracy of the model without the facial features. Generally, the more classes the model is trained to distinguish, the more challenging the recognition task.

In American Sign Language Recognition Using Deep Learning and Computer Vision [17], they used custom made American language videos for the dataset where each sign is performed five times by a single signer in varying lighting conditions and speed of signing, the videos were recorded on an iPhone 6 camera on 60fps and at 720p resolution, each video was broken down by frame to images and trimmed to 300 frames and then augmented to increase the data set for each sign to 2400 image the data set was then divided into training set (1800) and test set (the rest). CNN (Convolutional Neural Network) model named Inception is used to extract spatial features from the video stream for Sign Language Recognition, and they used a LSTM (Long Short-Term Memory) and, a RNN (Recurrent Neural Network) model to extract temporal features from the video sequences via two methods which are using the outputs from the Softmax and the Pool layer of the CNN respectively. For Gesture detection the used transfer learning to retrain the existing inception model to work on the dataset, inception performs all the convolutions in parallel and concatenates the resulting feature maps before going to the next layer (it repeats the operations to create a deeper network). For Gesture classification, the outputs of the Softmax Layer and the Max Pooling layer and feed it to the RNN architecture**,** the gesture segments identified and processed by the CNN are classified by the LSTM into one of the gesture classes using sequence data.Since the input segments must be fixed size, the length of all the frame sequences is trimmed. It was observed during testing that Accuracy of model drop when different skin tones, different signers face (so videos must be trimmed to the neck, and different clothing.

In Deep Convolutional Neural Networks for Sign Language Recognition [18], their objective was to simulate algorithms that can optimally execute on a mobile platform and the main module is to extract information frames to reduce input video data per frame. The dataset was created from Indian sign language performed by 5 native ISL users in 5 different viewing angles at a rate of 30fps. Training is initiated with three different batch sizes where in Batch-I, 200 signs performed by 1 user in 5 different viewing angles for 2 seconds at 30fps, total of 60000 sign images, and Batch-II is done using 2 sets, total of 120000 sign images. The model is constructed with an input layer, four convolutional layers, five rectified linear units (ReLU), two stochastic pooling layers, one dense and one SoftMax output layer. The convolutional windows are of size 16 16, 9 9, 5 5 and 5 5 from layer 1 to 4. The feature representation is done by considering two layers of stochastic pooling. The classification stage is implemented with dense/fully connected layers followed by an activation function. SoftMax regression is adopted in classification. The architecture of the CNN model consists of four convolutional layers. While the first two layers extract the low-level features (like lines, corners, and edges) and the last two layers learn high level features. Over a region the max value of a feature is obtained using stochastic pooling technique by calculating the probability values for each region to reduce the data variance. The network is trained to learn the features of each sign by means of supervised learning. In Batch-I, CNN trains with only one set of data. During the training different feature maps were observed at different layers where low level features like lines, edges and corners are learned from Convolutional layer 1 and 2. High level features learned from Convolutional layer 3 and 4. In Batch-II, CNN trains with two sets of data,training is performed for two sets of data on an HPC machine in 100 epochs. Testing is done with the same data of training and a third dataset, by increasing the number of data sets for training it is observed that a good amount of recognition is achieved compared to Batch-I training, as the number of training data sets increased accuracy in recalling the sign is substantially increased. In Batch-III, CNN trains with three sets of data,further improvement in recognition rates is achieved by increasing the training to CNN where a total of five datasets were created, out of which three sets were used in training and two sets for testing. An average confusion matrix is generated based on the recognition rates and number of matches for three training batches.

 The Sign Language Recognition Prototype is a real-time vision-based system [19] whose purpose is to recognize the Sign Language The purpose of the prototype was to test the validity of a vision-based system for sign language recognition and at the same time, test and select hand features that could be used with machine learning algorithms allowing their application in any real-time sign language recognition systems. For that, the user must be positioned in front of the camera, doing the sign language gestures that will be interpreted by the system and their classification will be displayed on the right side of the interface. The implemented solution uses only one camera, a Kinect camera, and is based on a set of assumptions, hereby defined:

1. The user must be within a defined perimeter area, in front of the camera.
2. The user must be within a defined distance range, due to camera limitations. The

system-defined values are 0.7m for the near plane and 3m for the far plane.

1. Hand pose is defined with a bare hand and not occluded by other objects.
2. The system must be used indoor, since the selected camera does not work well under sunlight conditions.

The proposed system architecture consists of two modules, namely: data acquisition, pre-processing, and feature extraction; and sign language gesture classification.

In the first module, the hand is detected, tracked, and segmented from the video images. From the obtained segmented hand, features are extracted for gesture classification. In the gesture classification module, the obtained feature vector (instance vector) is normalized and classified with a previous trained Support Vector Machine (SVM), which is a pattern recognition technique in the area of supervised machine learning, which works very well with high-dimensional data.

By another method [20] following the data acquisition, a labeled map is created which is a representation of all the objects within the model, i.e., it contains the label of each sign (alphabet) along with their id. The label map contains 26 labels, each one representing an alphabet. Each label has been assigned a unique id ranging from 1 to 26. This will be used as a reference to look up the class name. TF records of the training data and the testing data are then created using generate\_tfrecord which is used to train the TensorFlow object detection API. TF record is the binary storage format of TensorFlow. Binary files usage for storage of the data significantly impacts the performance of the import pipeline consequently, the training time of the model. It takes less space on a disk, copies fast, and can efficiently be read from the disk.

The open-source framework, TensorFlow object detection API makes it easy to develop, train and deploy an object detection model. They have their framework called the TensorFlow detection model zoo which offers various models for detection that have been pre-trained on the COCO 2017 dataset. The pre-trained TensorFlow model that is being used is SSD MobileNetV2 320 × 320. The SSD MobileNetV2 Object detection model is combined with the FPN-lite feature extractor, shared box predictor, and focal loss with training images scaled to 320 × 320. Pipeline configuration, i.e., the configuration of the pre-trained model is set up and then updated for transfer learning to train it by the created dataset. For configuration, dependencies like TensorFlow, config\_util, pipeline\_pb2, and text\_format have been imported. The major update that has been done is to change the number of classes, which is initially 90 to 26, the number of signs (alphabets) that the model will be trained on. After setting up and updating the configuration, the model was trained in 10000 steps. The hyper-parameter used during the training was to set up the number of steps in which the model will be trained, which was set up to 10000 steps. During the training, the model has some losses such as classification loss, regularization loss, and localization loss. The localization loss is mismatched between the predicted bounding box correction and the true values.

For more closer look on the Sign Language Interpretation System works in two stages [21] The first is the preprocessing phase i.e., image processing phase, where the hand shape and other distinguishable features are extracted from the image using background subtraction, blob analysis, filtering and noise removal, grayscale conversion, brightness and contrast normalization, scaling and several other image processing techniques. The second stage involves the classification of an image into given many different possible gestures using Haar Cascade Classifier, where this classifier is trained on a given training set that contains samples of the different gestures. This training sample images are taken from several different angles and captured in different lighting conditions. Training dataset consists of positive, negative as well as test sample databases. Positive samples are those image samples which contain perfect hand gestures whereas in negative sample images the required gesture is absent or only background details are available, no hand movement is present.

These datasets are mostly used in the training part of the classification phase. The test sample dataset can be used in the testing part of the classification phase. After the training of setup is done, the system is now ready to interpret input images from the videos. A database of HAAR cascade classifiers which denotes different signs is then observed. The classifier which produces the highest probability is then chosen as the most possible interpretation of the sign. Classification or ANN phase which follows the text to speech conversion. This phase is known as speech synthesis phase. A. Preprocessing Phase This phase involves extracting frames from video streams and performing image processing steps to extract features from the image by performing background subtraction, Blob analysis, noise reduction, gray scale conversion, brightness normalization and scaling operation one by one.

1. Background Subtraction: This phase involves removing unwanted background details from captured image frames from video streams. and extracting only hand signs to perform image processing steps.
2. Blob Analysis: A blob is a region having the same properties and pixel values which are constant or vary within a prescribed range. This step discovers the region of interest for further processing by finding all connective parts of the frame and choosing the biggest (largest area) amongst them (since the hand is the largest area suspected of being a hand). Blob analysis is applicable in the field of object recognition or object tracking.
3. Noise Reduction: Noise reduction is meant to filter the discontinuity and noise by using a smooth Gaussian filter. This filter removes the noise by smoothing operation. The Gaussian kernel size used for this filter is 3
4. Grayscale Conversion: This step converts color image into grayscale image which helps in further calculations on pixel operations and interrelating signs. Memory space in terms of bits required to store grayscale images is lesser than the bits required storing color image.
5. Brightness and Contrast Normalization: Images acquired in low illumination have close contrast values hence there is a need to adjust pixel intensity values. Histogram equalization is performed in order to adjust and normalize brightness and contrast of the processing frame.
6. Image Scaling: Image scaling is done to reduce the computational effort needed for image processing. Every image will be scaled to 45\*45 sizes for further processing. B. Classification phase This phase involves application of haar cascade algorithm to correctly classify the extracted feature. Input to the segmentation block is processed resized images. Output of this phase is correctly classified word/sentence in textual format.

The classification phase is further divided into training and testing stages.

1. Training stage: HAAR Cascade Classifier is trained using 500 positive, 500 negative and 50 test image samples of each gesture. These images are stored in their respective folders. These images, especially positive samples are collected from different people with different hand shapes, size and color and different lighting conditions in various angles. Accuracy of recognition can be improved by locating areas of interest in each sample image. This can be accomplished by drawing a box around the region of interest i.e. hand shape. The coordinates of the region of interest are then analyzed to measure the contrast between each of these images. This stage will enable us to build the required cascade and find thresholds after analyzing each coordinate of hand sign.

Classifier uses HAAR like features like edge, line, and center surround features to be trained using simple HAAR function. To achieve perfection in results it is recommended to use maximum no of samples. Training a classifier in order to interpret the different signs based on the features learnt by preprocessing takes longer time. Training procedure executes only once, where HAAR Cascade Classifier is trained for a particular sign. After training is over the system is ready to interpret signs in the video using a web camera.

1. Testing Stage: Once training is over, the classifier is now well trained to distinguish between different signs. Testing is performed on the live video through a web camera. The output of this phase is in text form.

In the proposed system for sign language recognition [22] sign language is a language which mainly uses actions or gestures to convey meaning, as opposed to acoustically conveyed sound patterns. There are significant differences between signed and spoken languages, because of the constraints offered by visual gestures. Yet the two are fundamentally similar as both have their own syntax and semantics.

Groups of hearing and speech impaired people have used signs to communicate for many years and so sign language is developed among them. American Sign Language substantially facilitates communication in the hearing-impaired community. However, there are only ~250,000-500,000 speakers which limits the number of people that they can communicate with. To diminish this obstacle and to enable better communication, we would like to propose an ASL recognition system that uses Convolutional Neural Networks to translate a user’s ASL signs into text in real time.

In deep convolution neural networks for sign language [23] Convolutional Neural Networks (CNNs) are machine learning algorithms that have seen a great success as they handle a variety of tasks related to processing videos and images. Like other machine learning algorithms, CNNs seek to optimize some objective function, specifically the loss function. CNNs have seen a rapid improvement in image classification with many proposed models like Google Net, Alex Net giving an accuracy almost near to human perception. The main cause of the recent improvement in CNNs has been due to the ImageNet Large Scale Visual Recognition Competition (ILSVRC). For image processing we propose to use OpenCV library along with TensorFlow and Keras which will be used for training the classifier. For other mathematical calculations we may use the NumPy Array in Python. The various approaches we considered are explained in the subsequent paragraphs. Neural Networks are inspired by the biological arrangement of processing elements called neurons in the brain. These neurons enable parallel processing of computational tasks. This enables Neural networks to solve complex problems of pattern recognition better than procedural algorithms. CNNs are neural networks in which the response of the neuron can be calculated by a convolution operation. The initial layer of CNN can be used for matching images with respect to a fixed template. The subsequent layer can then be used for detecting variations of the identified image for improved accuracy and for generating patterns of a pattern.

In deep neural framework [24] In this work, our proposed architecture adopts a feature ex traction module composed of a deep CNN followed by temporal fusion layers, and a sequence learning module using RNNs with bidirectional long short-term memory (Bi-LSTM) architecture. We use the end-to-end recognition system to generate alignment proposals between video segments and gestural labels. Given the large number of gestural segments with supervisory labels, we train the feature extraction module and then fine-tune the whole system iteratively. In the remainder of this section, we will first present our model formulation and then introduce its iterative training strategy. Model design: The proposed deep neural architecture consists of a deep CNN followed by temporal operations for representation learning, and Bi-LSTMs for sequence learning. For experiments with modalities from dominant hands as the inputs, we build the deep convolutional network based on the VGG-S model (from layer conv1 to fc6), which is memory-efficient and shows competitive classification performance on ILSVRC-2012 dataset. The input images, which are the region of dominant hands cropped from original frames, are resized to 101 × 101 in dimension, and they are then transformed to 1024-dimensional feature vectors through the fully connected layer fc6. The stacked temporal convolution and pooling layers are utilized to generate spatiotemporal representation for each segment. Note that it is hard to learn the extremely long dynamic dependencies with no temporal pooling, while a coarse temporal stride will lead to loss of temporal details. We select the temporal stride δ to ensure sufficient overlapping between neigh boring segments, as well as pool the representation sequence to a moderate length. For videos in RWTH-PHOENIX-Weather database, we set L= 16 frames, δ = 4 frames, and we set L = 25 frames, δ = 9 frames in experiments on SIGNUM corpus. In the feature extraction module, rectifier and max pooling are adopted for all the non-linearity and pooling operations. We use Bi-LSTMs with 2 × 512 dimensional hidden states and peephole connections to learn the temporal dependencies. The hidden states are then fed into the SoftMax classifier, with the dimension equal to the vocabulary size. We are also investigating the performance of our training framework with full video frames as the inputs. We use Google Net and VGG-S net as the deep convolutional network in our feature extractor, and we adopt two stacked Bi-LSTMs to build the sequence learning module. Due to the limitations on GPU memory to fit in the whole system, we fix the parameters of CNN at the end-to-end stage and only tune the sequence learning module. The video frames are resized to 224 × 224 as the inputs of CNN, transformed to feature vectors after the average pooling layer, and then fed into the temporal fusion layers. The employed Google Net is initialized with the weights pretrained on ILSVRC2014 dataset, and we initialize the feature extractor by fitting it to the alignment proposal generated by the model end-to-end trained on dominant hand images. Multimodal fusion: To incorporate the appearance and motion information, we also take color image and optical flow for dominant hand regions as the inputs of our deep neural architecture. We adopt sum fusion approach at the conv5 layer for fusing the two stream networks. It computes element-wise sum of the two feature maps at the same spatial location and channel for the fusion. Our intention here is to put appearance and motion cues at the same spatial position in correspondence, without introducing extra filters in order to join the feature maps together. The sum fusion approach also shows a decent performance on the task of action recognition in video compared to other spatial fusion methods. Our end-to-end architecture for SL recognition from dominant hands is depicted in Fig. 2. Note that parameters for different modalities are not shared before the sum fusion. In experiments on multiple modalities of full frames, we adopt fusion of color and optical flow at two layers (after inception\_3b andinception\_4c in Google Net. the fusion structures we build for experiments on recognition from multiple modalities of full frames. We also adopt the auxiliary classifiers as in Google Net by adding to temporal fusion layers after inception\_4a and inception\_4d during the phase of feature extractor finetuning.

Unlike traditional neural networks that treat inputs as independent entities, RNNs are designed to handle sequential data by incorporating feedback loops that allow information to be passed from one step to the next. This allows the network to maintain information over time, making it well-suited for tasks such as natural language processing and speech recognition.

The basic architecture of an RNN involves a hidden state that is updated at each time step based on the current input and the previous hidden state. This hidden state acts as the "memory" of the network, allowing it to maintain information over time. The hidden state is updated using a set of weights that are learned during training.

Training an RNN involves adjusting the weights that govern the behavior of the network so that it can accurately predict the output for a given input sequence. This is done using an algorithm called back propagation through time (BPTT), which is a variation of the standard back propagation algorithm that considers the sequential nature of the data.

One challenge associated with training RNNs is the "vanishing gradient" problem, where the gradient used to update the weights becomes very small for long sequences, making it difficult for the network to learn long-term dependencies. This can be addressed using various techniques, such as gradient clipping and initializing the weights with specific values.

There are several variants of RNNs that have been developed to address the vanishing gradient problem, including the long short-term memory (LSTM) and gated recurrent unit (GRU) networks.

RNNs have a wide range of applications across various fields. In natural language processing, RNNs are used for tasks such as sentiment analysis and machine translation. In speech recognition, RNNs are used for speech-to-text transcription. In time-series prediction, RNNs can be used to forecast stock prices or weather patterns, among other things.[32]

n traditional neural networks, each input is processed independently, and the network doesn't have any way of keeping track of previous inputs. However, for sequential data (like text or time-series data), the order of the input’s matters, and previous inputs can have a significant impact on future predictions. LSTMs are designed to handle this type of data by incorporating a memory cell that can store information over time and various gates that can selectively let information into or out of the cell.

The memory cell in an LSTM is like a conveyor belt that information can flow through over time. The cell can selectively store information from previous inputs and outputs based on the current input and previous decisions made by the network. This allows LSTMs to selectively remember important information and forget irrelevant information, which is particularly useful for handling long-term dependencies in sequential data.

The gates in an LSTM are used to control the flow of information into and out of the memory cell. The input gate determines how much of the new input to let into the cell, the forget gate determines how much of the previous cell state to forget, and the output gate determines how much of the cell state to output to the next layer. These gates allow LSTMs to selectively remember or forget information from previous inputs and outputs based on the current input[33]

The article “understanding LSTMs Networks” by Christopher Olah[34] also covers some variations of LSTMs, such as bidirectional LSTMs and stacked LSTMs, and discusses some applications of LSTMs in areas like natural language processing, speech recognition, and handwriting recognition.

In addition to the basic LSTM architecture, the article covers some variations of LSTMs that can be used for different types of data or applications. For example, bidirectional LSTMs can process sequential data in both forward and backward directions to better capture the context of each input. Stacked LSTMs can be used to create deeper networks by stacking multiple LSTM layers on top of each other. Finally, the article discusses some applications of LSTMs in areas like natural language processing, speech recognition, and handwriting recognition, where they have been shown to be effective at handling sequential data.

There was also a paper under the title "Long-term Recurrent Convolutional Networks for

Visual Recognition and Description"[35] that proposes a new neural network architecture, called Long-term Recurrent Convolutional Networks (LRCN), that combines the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks for visual recognition and description tasks.

CNNs are powerful tools for image feature extraction and have been widely used for visual recognition tasks. However, CNNs are not designed to model temporal dependencies in sequences of images or videos. LSTMs, on the other hand, are specialized for modeling sequential data and can capture long-term dependencies in the data. By combining these two types of networks, the LRCN model can capture both spatial and temporal information in the image data.

The LRCN model consists of two main components: a CNN and an LSTM. CNN is used to extract features from each frame of the input sequence. The LSTM then takes as input the sequence of feature vectors produced by the CNN and models the temporal dependencies in the sequence. The LSTM is trained to predict the label of the sequence (e.g., the category of the video) or generate a textual description of the sequence.

Another paper that was titled "Show and Tell: A Neural Image Caption Generator"[36] proposes a new model for generating natural language descriptions of images. The model uses a Convolutional Neural Network (CNN) to extract features from the image and a Long Short-Term Memory (LSTM) network to generate a natural language sentence that describes the image.

The model, called the "Neural Image Caption (NIC) generator," is trained on a large dataset of image-caption pairs. During training, the CNN is used to extract a fixed-length feature vector from the image, which is then fed into the LSTM. The LSTM is trained to predict the next word in the caption, given the previous words and the feature vector from the CNN. This process is repeated until the end of the caption is reached.

During testing, the model is given an input image and uses CNN to extract a feature vector. The LSTM then generates a natural language sentence that describes the image based on the extracted features.

The authors evaluate the NIC generator on two datasets: the Flickr8K dataset and the Flickr30K dataset. They compare the performance of the NIC generator to several other state-of-the-art models and show that the NIC generator outperforms these models on both datasets. They also conduct a human evaluation and show that the captions generated by the NIC generator are preferred by human evaluators over those generated by other models.

Another paper was titled "Learning Spatiotemporal Features with 3D Convolutional Networks[37]proposes a new deep learning architecture for video analysis tasks. The architecture, called the 3D Convolutional Network (3D CNN), extends the traditional 2D CNN to work with spatiotemporal data, such as video.

The 3D CNN uses three-dimensional convolutional kernels to learn spatiotemporal features directly from raw video data. The architecture consists of multiple layers of 3D convolutional kernels, followed by pooling and normalization layers, and ends with one or more fully connected layers for classification or regression.

The authors evaluate the 3D CNN on several video analysis tasks, including action recognition and human detection, and show that it outperforms several other state-of-the-art methods. They also conduct a thorough ablation study to analyze the contributions of different components of the architecture.

One of the key advantages of the 3D CNN is that it can learn spatiotemporal features directly from raw video data, without requiring hand-crafted features or optical flow estimation. This makes the 3D CNN particularly well-suited for tasks where the relevant spatiotemporal features may not be immediately apparent or are difficult to define.

Overall, the LRCN model proposed in the first paper is a powerful tool for visual recognition and description tasks that involve analyzing sequences of images or videos. By combining the strengths of CNNs and LSTMs, the model can capture both spatial and temporal information in the image data, leading to improved performance on a range of tasks. The NIC generator proposed in the second paper is a powerful tool for generating natural language descriptions of images. By combining the strengths of CNNs and LSTMs, the model can capture both the visual features of the image and the linguistic structure of the caption, leading to improved performance on this challenging task. The 3D CNN proposed in the third paper is a powerful tool for video analysis tasks, particularly those involving spatiotemporal data.

By extending the traditional 2D CNN to work with 3D data, the architecture can learn more powerful spatiotemporal features, leading to improved performance on a range of video analysis tasks.

The GRU is a type of RNN that uses gating mechanisms to address the vanishing gradient problem. Specifically, it has an update gate and a reset gate that control the flow of information between time steps, allowing the network to selectively remember or forget information.

The paper[38]evaluates the performance of GRUs on several different sequence modeling tasks, including language modeling (predicting the next word in a sentence), sentiment analysis (predicting the sentiment of a sentence), and speech recognition (transcribing spoken words into text).

The experiments show that GRUs generally outperform vanilla RNNs, which do not use gating mechanisms. The authors also find that GRUs are often comparable to LSTMs in terms of performance, although LSTMs are generally better at tasks that require more complex memory mechanisms.

The ablation experiments help to shed light on the importance of the different components of the GRU architecture. For example, the reset gate is shown to be important for modeling long-term dependencies, while the update gate is more important for modeling short-term dependencies.

The authors note that while GRUs are effective for many sequence modeling tasks, they may not be the best choice for all tasks. For example, tasks that require more complex memory mechanisms, such as tasks involving long-term memory retrieval, may benefit from the use of LSTMs or other architectures.

# COMPARISON

After reading many papers, we conducted a comparison in order for us to understand the pros and cons of each architecture and to better help us in making decisions.

## Recurrent Neural Networks (RNNs)

* Simplest type of neural network for processing sequential data.
* Can suffer from the vanishing gradient problem, where the gradients used to update the parameters of the network can become very small over time, making it difficult for the network to learn long-term dependencies.
* Limited in their ability to capture long-term dependencies.

## Long Short-Term Memory (LSTM)

* A type of RNN that is designed to overcome the vanishing gradient problem and effectively model long-term dependencies in sequences.
* Uses memory cells and gating mechanisms (input, forget, and output) gates to address the vanishing gradient problem.
* Can selectively remember or forget information using the gating mechanisms.
* Effective in modeling long-term dependencies and has been successfully applied in various applications.

## Gated Recurrent Unit (GRU)

* A type of RNN that is like LSTM but with fewer parameters.
* Uses gating mechanisms to address the vanishing gradient problem and effectively model long-term dependencies in sequences.
* Has two gates (reset gate and update gate) that determine which information to forget and which information to add to the memory.
* Effective in modeling long-term dependencies and has been successfully applied in various applications.
* Requires less computation compared to LSTM, making it more suitable for resource-limited applications.

# STATIC SIGN LANGUAGE

## Dataset

### Dataset Description

The base dataset that we used is World Level American Sign Language (WLASL) [25] dataset which is uploaded on Kaggle. It is widely used in many research papers cited below, but it has a couple of problems, some of which are:

* It contains both static and dynamic signs which wouldn’t work in our case since we only applied CNNs, and thus the temporal factor was neglected.
* Some signs’ data are either missing or bad (inaccurate or blurry)
* Contains both videos and images so we had to manually collect the data to filter out only the good parts

So, the first thing we did was create a histogram to show the top 25 signs that have the largest number of clips available in the dataset and we showed the ones that have videos in the uploaded dataset, and the ones that are available as links in a JSON file (these contained both static and dynamic signs)

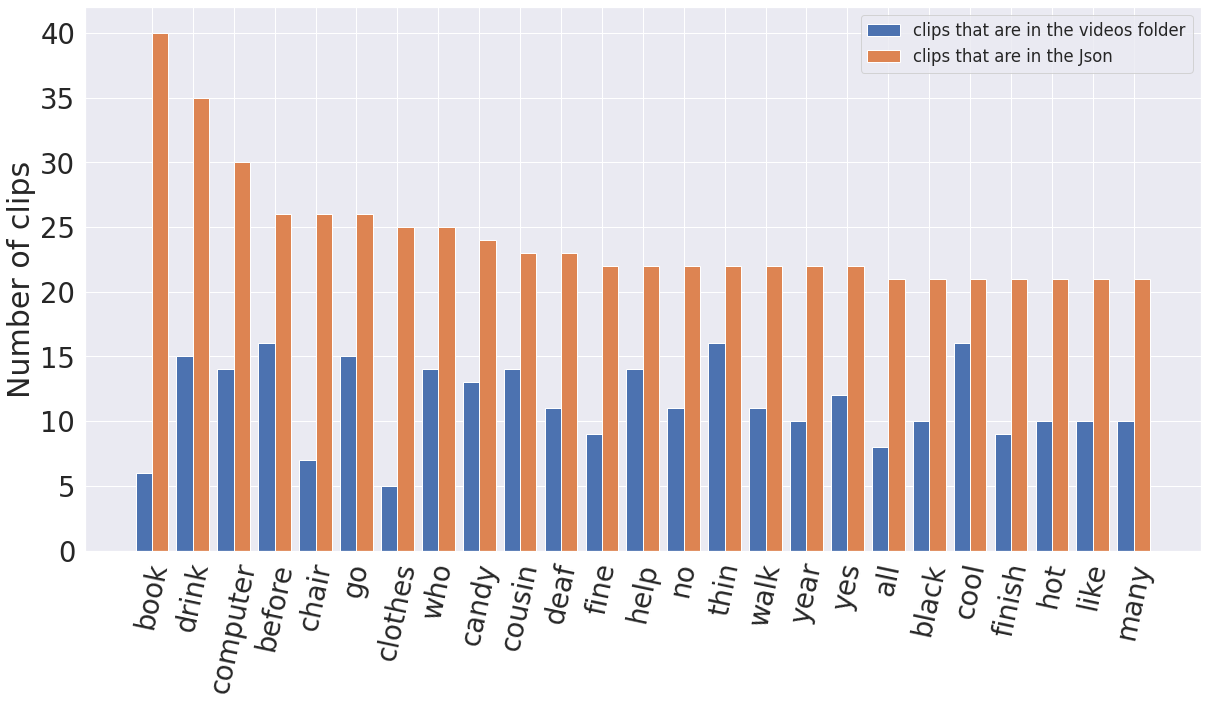


Figure 1 Top 25 Words Histogram

Then we wanted to get the top 10 static words so we can gather as much data as possible to increase the data in both the training and validation folders. So, we manually scanned the entire dataset and found all the static words, appended them into a list, and extracted the top 25 words with the largest amount of data available in the dataset.

Chart, bar chart, histogram

Description automatically generated

Figure 2 Top 25 Static Words Histogram

The project aimed to develop a system for sign language recognition. Initially, the World Level American Sign Language (WLASL) dataset was used, but it presented challenges due to a mixture of static and dynamic signs and missing or inaccurate data a collection of 40 images per sign all the same size, then we split the data 75% train and 25% validation (30 and 10 images respectively). Further iterations involved collecting data from YouTube and ASL Sign Bank, refining the dataset to focus on hand gestures. MediaPipe Hand-Detection module was employed for automated cropping. The final dataset consisted of 15 words with 60 training images and 20 validation images, all at a resolution of 350 x 350 pixels.

### Pre-processing and data augmentation

Initially, the focus was on 12 static signs (bed, shirt, wrong, cow, full, water, show, sick, police, sandwich, father, have). But due to an unequal number of videos for each sign, additional videos were collected from different sources to ensure a total of 40 videos per sign. The videos were performed by different signers under varying conditions to enhance the accuracy and robustness of the model. Corrupt sample files were replaced with new samples from diverse sources. Clean squared frames representing each sign were extracted from the sample videos, standardized to a resolution of 1080x1080, and organized into folders as the initial dataset the seven words used at that time were “Bed”, “Father”, “Full”, “Police”, “Shirt”, “Water”, and “Wrong”. Each word having 20 training images, 3 to 5 validation images, and 5 test images for each word for a total of 35 test images uniformly distributed. We trained a couple of models on this iteration of the dataset.

However, the performance of the models trained on this dataset was poor, necessitating the need to modify both the models and the dataset.



Figure 3 Full Image of Word “Shirt”

The Hand Detection module from Media Pipe was integrated into the system. This module was used to detect the hand and draw a bounding box around it. The extracted bounding box was then used as a separate frame. To standardize the frame size, a 350 x 350 box was created, with the remaining areas filled with white. Depending on the orientation of the hand, either left and right or top and bottom areas were filled. The larger dimension of the X and Y coordinates determined the orientation. This process was applied to all words in separate folders, and the resulting pictures were saved in a designated folder.

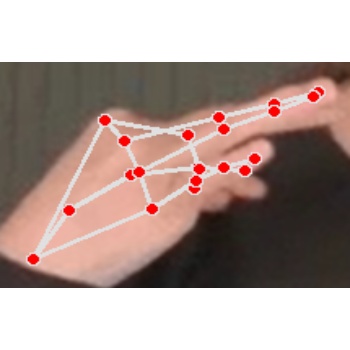


Figure 4 Cropped Image of Word “Shirt”

Then after training, the model managed to predict some words with a lot more accuracy than others. So, we thought of three solutions:

* Change the model architecture
* Get more data
* Change the pre-processing algorithm

The model initially showed varying levels of accuracy in predicting different words. To address this, three potential solutions were considered: changing the model architecture, acquiring more data, and modifying the pre-processing algorithm. Different model architectures were tested, but the results were either similar or worse than the existing model. Modifying the pre-processing algorithm was also explored, but a scalable and efficient alternative could not be identified. This led to the decision to gather more data. Despite searching for additional images, it was still insufficient. As a result, the team collected their own data by applying the pre-processing algorithm of cropping the bounding box and converting it into a 350 x 350 image. The images were saved with unique names, such as "shirt\_2," to indicate the corresponding word.

The data gathering process resulted in a total of 40 images for each word. To enhance the model's robustness and accommodate both left and right-handed signs, the images were vertically flipped, resulting in a final count of 80 images per word. These images were then split into 60 training images (30 normal and 30 flipped) and 20 validation images (10 normal and 10 flipped), all sized at 350 x 350 pixels. The final dataset comprised 15 words, including "Drink," "Food," "Full," "Have," "Hello," "I," "I love you," "Police," "Prefer," "Shirt," "Telephone," "Water," "Wrong," "Yes," and "You." Each word had 60 training images and 20 validation images, totaling 900 training images and 300 validation images, all with a resolution of 350 x 350 pixels.

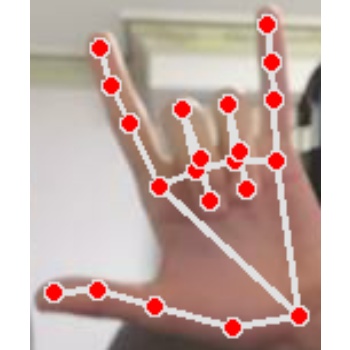
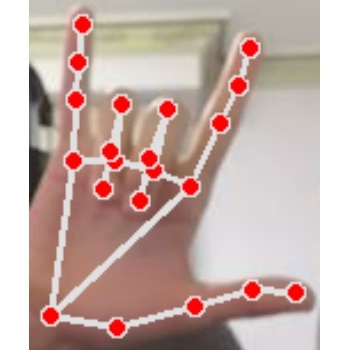


Figure 5 Flipped Image

## Methodology

The team considered multiple aspects of the project, including the data collection process, color format selection, automatic hand detection, and model choice. Initially, using full 1080x1080 images that included both the hand, the background, and the person, yielded poor accuracy. To address this, we focused on normalizing the data by detecting and isolating only the hands. MediaPipe's Hand Detection Module was incorporated, which detects hands and draws bounding boxes around them in a rendered frame. To create a scalable dataset, an automatic cropping process was implemented:

1. Loop on all images in all folders then Automatically detect the hand using the MP Hand Detector
2. Crop the bounding box of the hand which displays the hand and the inner connections.
3. Fill a 350x350 box with the resized hand.
4. Fill the empty spots with white (if the hand is vertical, we fill left and right, if it’s horizontal, we fill top and bottom).
5. Save the 350x350 image in a new directory for each word.
6. Flip the images vertically and split into training and validation.
7. Flip the images vertically
8. Split into training and validation

Then for the prediction, the process was as follows:

1. Read video frame
2. Automatically detect the hand using the MP Hand Detector
3. Crop the bounding box of the hand which displays the hand and the inner connections
4. Fill a 350x350 box with the resized hand.
5. Fill the empty spots with white (if the hand is vertical, we fill left and right, if it’s horizontal, we fill top and bottom)
6. Predict using the trained model with the 350x350 window as input
7. Output label gets displayed on top of the user’s hand

Multiple models, including VGG-16, MobileNet-V2, and ResNet-V2-50, were tested and evaluated in a real-time environment.

Ultimately, VGG-16 was chosen as the final model, performing better than VGG-19. A challenge arose during transitional periods between signs, where the model would predict signs with high confidence, even during transitions. To address this, various solutions were explored. Initially, a cooldown period between signs was considered but deemed undesirable. Graphing predicted values as percentages and setting a threshold was also attempted, but the confidence remained consistently high. To mitigate this issue, the framerate was lowered, either by reducing the capture framerate or skipping prediction frames, effectively treating them as transition frames. The model was connected to the front-end using Flask and rendered on an HTML page for user interaction.

## Experimental Evaluation and Results

### Initial Version of Models

After careful examination of some research papers, we decided to first try the following models: Inception-V3, VGG-16, MobileNet-V2, and ResNet50-V2.

As we previously stated in the dataset section, the initial results were very poor because of the unfiltered nature of the dataset, the unoptimized hyperparameters, and the layers of the models themselves.

The dataset that we trained on at that time consisted of seven words: “Bed”, “Father”, “Full”, “Police”, “Shirt”, “Water”, and “Wrong” using the Hand Detection from MediaPipe.

A picture containing line, diagram, plot

Description automatically generated

|  |
| --- |
| Figure Initial Models Validation Accuracy |
| Figure Initial Models Validation Loss |

|  |  |
| --- | --- |
| Figure Inception-V3 Heatmap | Figure Initial VGG-16 Heatmap |
| Figure Initial MobileNet-V2 Heatmap | Figure Initial ResNet50-V2 Heatmap |

The heatmaps show that the accuracies on the initial dataset are only remotely good on the VGG-16, MobileNet-V2 and ResNet50-V2. The other model didn’t have good accuracy as well, also the Inception -V2 produced worse results when used in a real-time environment, so we decided to keep experimenting on the VGG-16, MobileNet-V2, and the ResNet50-V2 models.

|  |  |
| --- | --- |
| Figure Initial Models Accuracies | Figure Initial Models F1-Scores |

The final comparison shows that the MobileNetV2 is the most accurate model due to both high accuracy and F1-Score which (in theory) should have been the most suitable one in our experiments, but when we tested in a real time environment, the VGG-16 proved to have slightly better results in comparison to MobileNetV2.

### Final Version of Models

The second batch of models we tried consisted of: VGG-16, VGG-19, ResNet50-V2, MobileNetV2, ResNet101-V2, EfficientNet-B7, EfficientNet-V2L, and EfficientNet-V2M

Through each iteration of training the models and adjusting the dataset, we kept monitoring the validation accuracy and loss throughout each epoch. Training each model for 8 epochs on the final dataset produced these results but we trained all of the Efficient Net models on 25 epochs because 8 epochs wasn’t that enough.

A picture containing line, diagram, screenshot, plot

Description automatically generated

|  |
| --- |
| Figure Models Validation Accuracy |
| Figure Models Validation Loss |

Since EfficientNet-B7, EfficientNet-V2M, and EfficientNet-V2L produced very poor results, we decided to increase the training epochs to 25 and observed the following:

A picture containing text, font, line, handwriting

Description automatically generated

|  |
| --- |
| Figure EfficientNet Validation Accuracy |
| Figure EfficientNet Validation Loss |

When inspecting these two line graphs, we can clearly see that the validation loss of the ResNet50-V2 and the MobileNet-V2 are the highest of the bunch, while VGG-16, VGG-19, and ResNet101-V2 are all almost equal.

The one downside of the ResNet101-V2 is that its’s a lot heavier than the other two models, both when training and when predicting, so it was not suitable in a real-time environment.

The dataset we trained these models on was the final dataset that consisted of 15 words: “Drink”, “Food”, “Full”, “Have”, “Hello”, “I”, “I love you”, “Police”, “Prefer”, “Shirt”, “Telephone”, “Water”, “Wrong”, “Yes”, and “You”.

The heatmaps produced by the models gave a clearer insight as to how each model behaves and how accurate each model is.

|  |  |
| --- | --- |
| Figure VGG-16 Heatmap | Figure VGG-19 Heatmap |
| A screenshot of a video game  Description automatically generated with medium confidence  Figure 20 ResNet50-V2 Heatmap | Figure MobileNet-V2 Heatmap |
| Figure ResNet101-V2 Heatmap | Figure EfficientNet-B7 Heatmap |
| Figure EfficientNet-V2L Heatmap | Figure EfficientNet-V2M |

From these heatmaps, we can observe that the almost all EfficientNet models produce very poor results, and the VGG-16, VGG-19, ResNet50V2, and the MobileNetV2 all have really high accuracies with the VGG-16 being the most accurate of the bunch, so that’s what we went with when considering the final model.

For final evaluation, we decided to plot the accuracies and F1-Scores of each model.

|  |
| --- |
| Figure Models Accuracy Comparison |
| Figure Models F1-Score Comparison |

These two bar charts prove that the VGG-16 is the best model to use for our purposes with at least 10% difference between it and any other trained model.

To dive more deeply into the VGG-16, we calculated the precision, recall, and F1-Score for each of the 15 words.

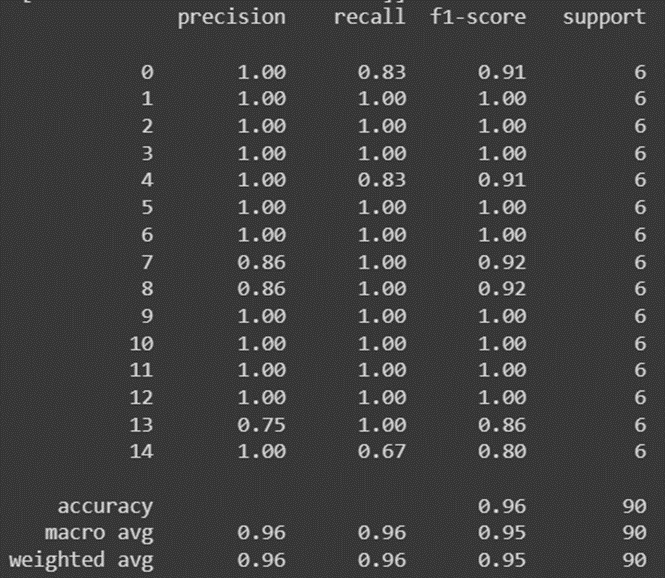


Figure 28 VGG-16 Results

We can see that almost all words are guessed perfectly except for number 7, 8, and 13 which are “Police”, “Prefer”, and “Yes” respectively.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Italian SL  [31] | Portuguese SL [30] | British SL  [28] | Indian SL  [29] | American SL  [17] | Our project |
| Model | CNN | SVM | SVM | Haar cascade classifier | K-NN and SVM | CNN |
| Color format | RGB | RGB | RGB | - | RGB | RGB |
| Dataset size | 11,008 | 4668 | 13,066 | 5250 | 7900 | 900 |
| Dataset type | Letters | Letters | Letters | Words | Letters | Words |
| Accuracy | VGG: 99% | 99.6% | 99% | 92.68% | SVM:98.7% | 96% |
| CNN: 97% | KNN:97.4% |

# DYNAMIC SIGN LANGUAGE

In this study, we explored the suitability of Convolutional Neural Networks (CNNs) for predicting static signs and investigated the feasibility of using different techniques, namely Gated Recurrent Unit (GRU), Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM), to direct the model towards predicting dynamic signs. To achieve this, we needed to modify the implementation of our model.

Subsequently, we considered two options for implementing the new techniques: integrating them with the existing CNN architecture or constructing an entirely new architecture from scratch. However, due to the computationally expensive nature of CNNs, the addition of another neural network on top of it would significantly increase inference time, particularly if performed on a CPU. As a result, careful consideration was given to the computational costs and hardware requirements of each approach.

## Dataset

The initial dataset consisted of 3 words which were Hello, I love you, and Thanks. Each one contains 40 instances; each instance consists of 30 frames. We then added more words and increased our dataset to contain 11 words which are Dance, Hello, How, I love you, Meet, Morning, Please, Thanks, Thirsty, Use, and You. And since each word contains 40 instances, our final dataset consisted of 440 instances which we split 60 percent training, 20 percent testing, and 20 percent validation.

## Methodology

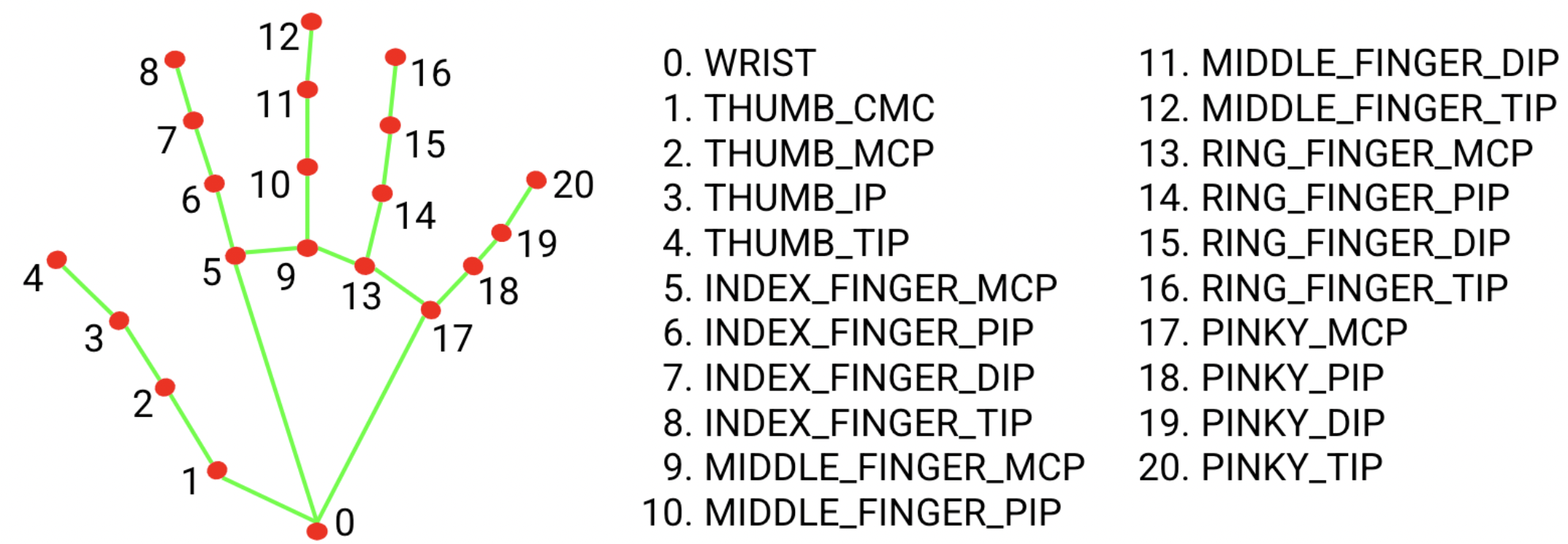
To solve the inference cost, we used MediaPipe's hand detection and pose detection modules which include a multitude of landmarks, each having their own X, Y, and Z coordinates.

Figure 29 Hand Landmarks

As displayed in Figure 29, there are a total of 21 landmarks in MediaPipe's hand detection module, each one having its own X, Y, and Z coordinates and each has its own name so we can reference it in the code. These are available for both the left and right hands.

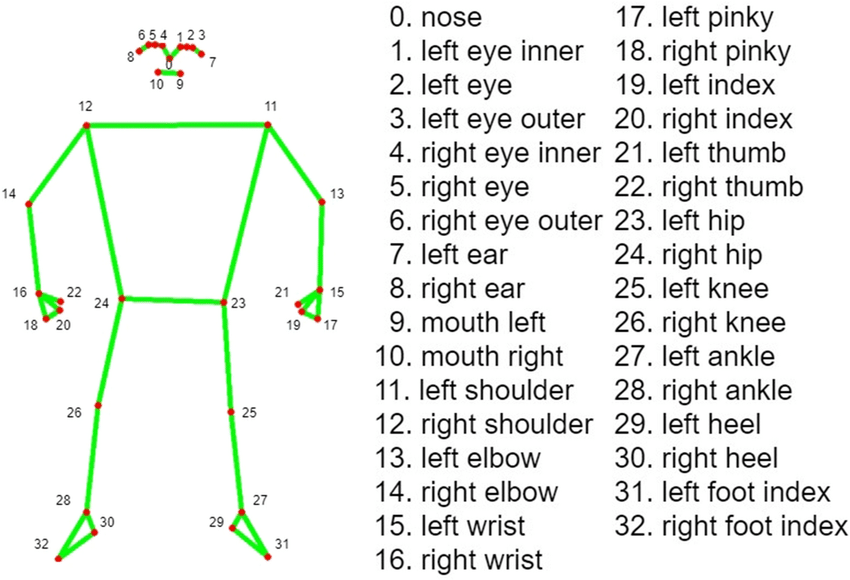


Figure 30 Pose Landmarks

And as shown in Figure 30, there's also a pose detection module which acts the same as the hand detection module, but it displays other points in the human body such as the shoulders, the elbow, knees, eyes, and other parts that can be used.

There's also a face detection module which creates a facemesh around the user's face and has many landmarks that we can take advantage of to do our processing for whatever application we would like to implement.

So, for the actual methodology, we went through three phases, gradually improving as we move from one phase to the other.

### Phase One

For the initial implementation, we extracted four different landmarks from the user: pose landmarks, face landmarks, left-hand landmarks, and right-hand landmarks. Each hand contains 21 landmarks, while the pose contains 33 landmarks, and the face mesh contains 468 landmarks, since each landmark has its own X, Y, and Z coordinates, the total number of coordinates is 1629.

We took these 1629 coordinates, ordered them in a NumPy array in the order of pose, face, left hand, and lastly right hand. This means we generated a vector of length 1629 for each frame we capture, and to enable dynamic sign language detection, we decided to go with 30 frames per one single input.

The initial model was trained on 3 words: Hello, I love you, and Thanks, since they few of the most basic signs one can learn. Each sign contained a total of 30 series of inputs (videos), each input contained 30 NumPy arrays representing the coordinates of each landmark in every frame.

### Phase Two

After phase one, the resulted model was heavier than needed since it processed 1629 points each frame, so in order to increase prediction time and reduce the training cost as well as model size, we removed the face mesh which reduced the array size from 1629 to just 225 points.

### Phase Three

After finishing phase two, we noticed that the model predictions differ based on the location of the user in the frame as well as the distance from the lens which changes the X, Y, and Z coordinates which in turn change the input data.

To be able to combat this issue, we designed an algorithm that calculates a middle point between the user's left and right shoulders which acted as a reference point, and then calculated the distance from each keypoint collected to that midpoint. So instead of having an array of 225 of X, Y, and Z coordinates, we now have an array of 225 distances of our reference point. Which solved our problem and produced a more robust model.

## Experimental Evaluation and Results

To optimize the model after phase three of the methodology, we began to train the model, the training error was low while the validation error was really high which resulted in poor F1 scores which lead us to believe the error was because of the overfitting which happened due to the large number of training epochs.

In order for us to solve this problem, we had to plot the training and validation errors versus the number of epochs to either find the elbow after which the error saturates, or to find the lowest training and validation errors before the overtraining happens.

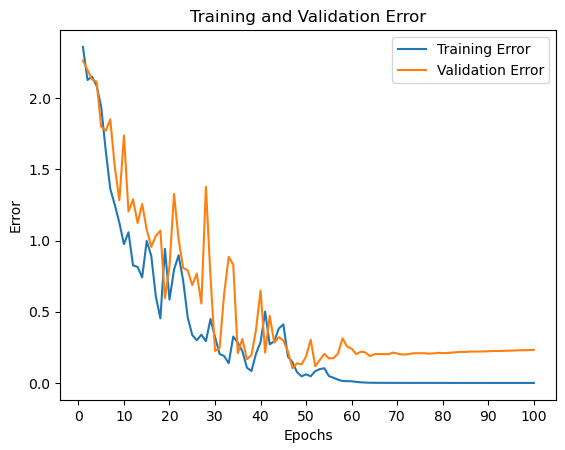


Figure 31 DynamicSL 100 Epochs Error Graph

As we can see from figure 31, the elbow or saturation point is at around epochs 50 - 60. So, to further investigate, we trained the model on 60 epochs, 50 epochs, and 20 epochs, to see the difference in the F1 score (which is what matters the most).

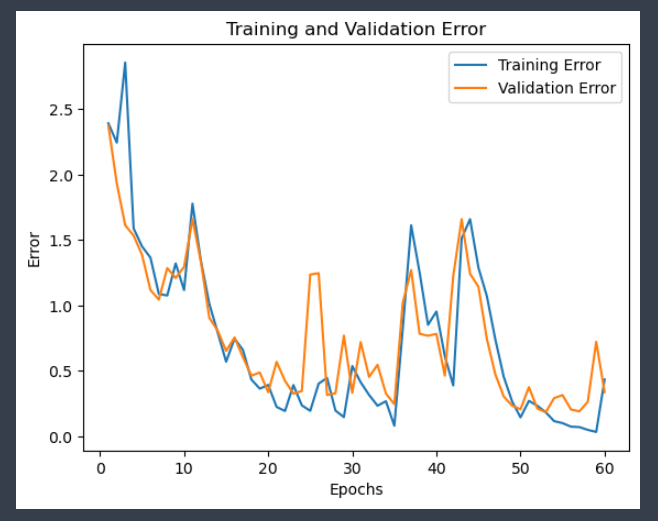
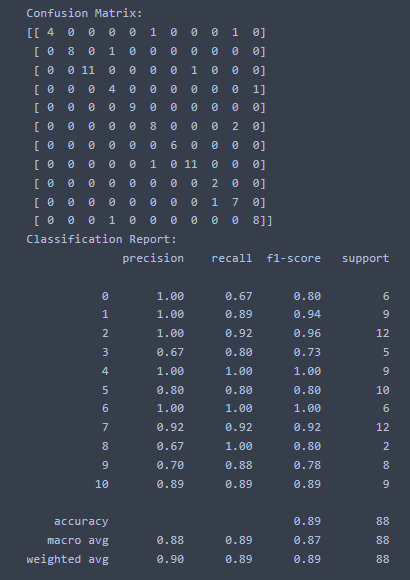


Figure 32 DynamicSL 60 Epochs Report

Figure 33 DynamicSL 60 Epochs Error Graph

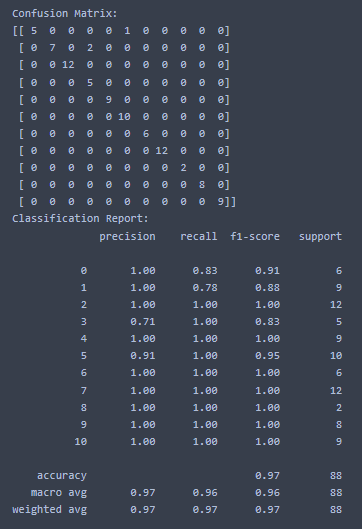
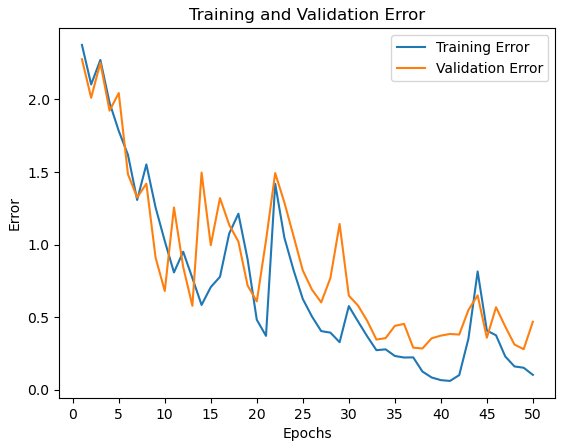
We also plotted the confusion matrix and classification report on the test set in order for us to gain better insight on each model as we train them. As we can see from figure 33, the F1 score is 0.89 which is really good considering the fact that there is a lower point at 50 epochs, so we inspected that point.

Figure 34 DynamicSL 50 Epochs Error Graph

Figure 35 DynamicSL 50 Epochs Report

We can see here in figure 35, the F1 score is really high at an amazing 0.97 which is the highest we’ve reached so far. We also tried decreasing the number of epochs just to make sure it doesn’t get better.

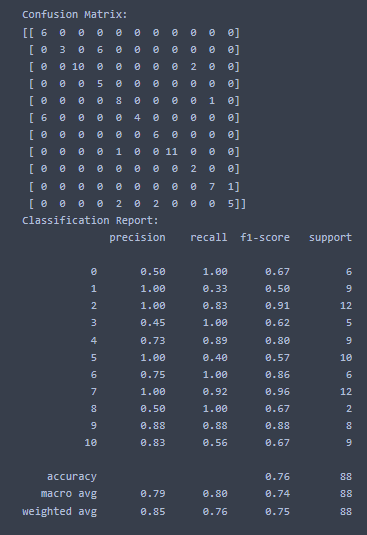
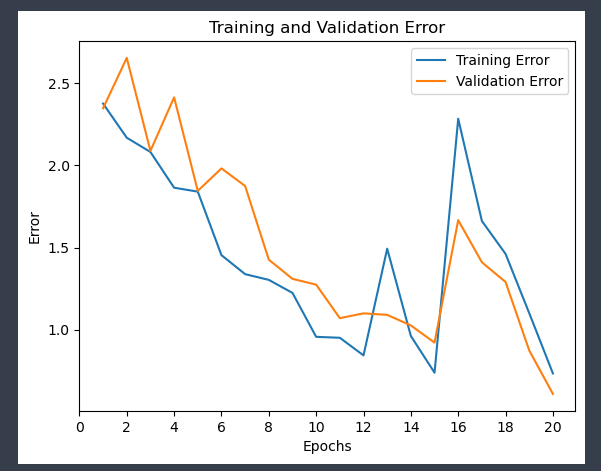
We can see here that if we lowered the number of epochs to 20, the F1 score is reduced to just 0.76 which is much worse than what we had on epoch 50.

Figure 36 DynamicSL 20 Epochs Error Graph

Figure 37 DynamicSL 20 Epochs Report

Which means that finally, we stuck with the 50 epochs model.

Finally, we plotted a heatmap using the 50 epochs model in order for us to better understand the relation between true and predicted labels. As we can see, the model performs very well as it predicts most of the words perfectly, except for hello which has 2 wrong predicted instances, and dance which has just 1 wrong predicted instance.

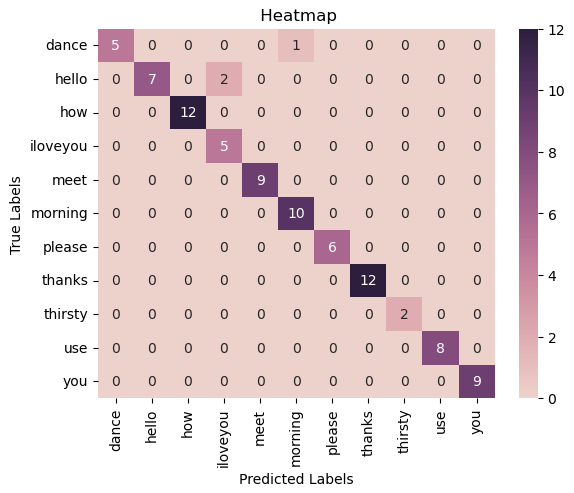


Figure 38 DynamicSL Heatmap

# BIOMETRIC SCANNER

Hand biometric recognition has become an increasingly popular method of identifying individuals in a variety of applications, from access control to mobile payments. In recent years, machine learning techniques have been applied to the field of hand biometrics, enabling accurate and efficient recognition of individuals based on the unique characteristics of their hands.

This study utilized a laptop camera and the MediaPipe library to capture and process a dataset of hand signs for eight different words. The images were captured at a consistent distance, and hand landmarks were extracted using the MediaPipe Hands library. Additional features, including finger lengths and palm width, were calculated based on the extracted landmarks, resulting in a set of 11 numerical features per image.

To associate each data point with a corresponding user, the study created a list of user labels and replaced each string value in the labels list with a corresponding integer value. This allowed for the creation of a panda DataFrame that included the extracted feature values and corresponding user labels.

The resulting dataset was split into training, validation, and testing subsets, and machine learning techniques were applied to predict the user who produced each image based on the extracted features. By demonstrating the efficacy of machine learning techniques in identifying individuals based on hand biometrics, this study provides valuable insights into the potential applications of hand biometric recognition in a range of fields.

## Dataset

This study utilized a laptop camera and the MediaPipe library to capture and process a dataset of hand signs for eight different words. The images were captured at a distance ranging from 60 to 70 centimeters from the camera by the detection of the user's face using the MediaPipe library to ensure consistent positioning.

Each word in the dataset was captured by five different users. The dataset includes eight words: "bar," "blind," "Egypt," "gun," "hello," "mad," "my," and "stare." To ensure uniformity, each user contributed 30 images, resulting in a total of 150 images per word. All captured images have a resolution of 640X480 pixels and were taken under different environmental and lighting conditions. To augment the dataset, all images were vertically flipped, resulting in a total of 60 images per user and a grand total of 2400 images.

The use of a laptop camera and the MediaPipe library allowed for the efficient capture and processing of a large number of images, making it a practical and cost-effective method for generating a dataset of hand signs. The resulting dataset is a valuable resource for researchers studying sign language recognition and other related applications.

## Methodology

We utilized MediaPipe, a popular open-source framework for building cross-platform computer vision applications, to extract hand landmarks from a set of images. Specifically, we used the MediaPipe Hands library to detect the landmarks of the hands as shown in figure 29.

As part of the data preprocessing stage, we utilized a standardized naming convention for all captured images. Each image was labeled with its corresponding user label, such as "User1", "User2", etc., followed by the name of the word, such as "My", "Blind", etc. and finally the frame number of the captured image. The resulting image name followed this format: "User2\_ Blind­\_frame\_0.png".

To augment the dataset, we performed vertical flipping on all the images. The resulting flipped images were saved with a modified label that included the word "flipped" after the frame number, such as "User2\_Blind­\_frame\_0\_flipped.png". This naming convention ensured that the original and flipped images were easily distinguishable and organized in a systematic manner.

To obtain additional features for our machine learning model, we calculated the lengths of the thumb, index, middle, ring, and pinky fingers, as well as the palm width and the size of the whole hand palm based on the extracted landmarks. This resulted in a set of 11 numerical features per image.

We saved the extracted features in a variable called data, which was formatted as a list of lists. Each inner list represented a single data point, and each element in the inner list represented a feature value.

Since the images were saved with a standardized naming convention. Therefore, upon extracting hand features from the images, we created a list of user labels in the form of string values that extracts the user label from the image name. After gathering all users in the list, we replaced each string value with a corresponding integer value. For example, "User1" was replaced with 1, "User2" with 2, and so on. This resulted in a list of integers representing the user labels.

Finally, we created a pandas Data Frame and added the extracted feature values to it, along with their corresponding user labels in our list of integers. This allowed us to perform further analysis and modeling on the dataset, using machine learning techniques to predict the user who produced each image based on the extracted features.

As part of our analysis, we created three different scenarios to evaluate the effectiveness of our machine learning models.

The first scenario involved using an unlabeled image dataset containing our 11 extracted features, along with integer representation of all user labels.

The second scenario utilized a labeled image dataset, where we assigned a unique value for each word from the 8 words we have in our dataset.

The third scenario used a labeled and normalized dataset, where each image was assigned a unique label, and all features were normalized to a range between 0 and 1.

To train a machine learning model on this dataset, the data was split into three subsets: a training set, a validation set, and a testing set. The training set was composed of 80 percent of the entire dataset, corresponding to 1920 images. This set was used to train the machine learning model. The remaining 20 percent of the data was divided equally between the validation set and the testing set, each containing 240 images.

The validation set was used during the training process to validate the model's performance on data that it had not yet seen. This was used to adjust the model's hyperparameters to improve its performance. The testing set was used after the model had been trained and evaluated on the validation set to test the model's final performance on unseen data.

A picture containing text, plot, line, diagram

Description automatically generated

Figure 39 KNN K values

From the graph we can see that the highest accuracy is achieved when K value is 1 and that was common on all of the 3 scenarios we have, but the other models hyperparameters varies from scenario to scenario. We used GridSearchCV to automate the process of tuning the hyperparameters for the other models such as “C” in SVM model or “Max\_Depth” in Decision Tree Model and computing f1-scores of the models on the validation set.

## Experimental Evaluation and Results

In the beginning, it is necessary to establish clear definitions of key terms in machine learning.

Precision: Precision is a metric that measures the proportion of true positive predictions out of all positive predictions. It represents the exactness or correctness of positive predictions.

Recall: Recall is a metric that measures the proportion of true positive predictions out of all actual positive instances. It represents the completeness or coverage of positive predictions.

F1-score: F1-score is a metric that provides a harmonic mean of precision and recall. It balances precision and recall, providing a single summary metric of the performance of a classifier.

Accuracy: Accuracy is a metric that measures the proportion of correct predictions out of all predictions. It represents the overall correctness of predictions.

Specificity: Specificity is a metric that measures the proportion of true negative predictions out of all actual negative instances. It represents the ability of a classifier to correctly identify negative instances.

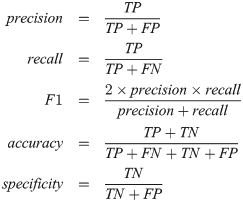


Figure 40 Necessary Equations

We trained and tested five different models, namely K-Nearest Neighbors (KNN), Random Forest, Decision Tree, Support Vector Machine (SVM), and Logistic Regression, allowing us to determine the most effective approach for our dataset. By comparing the F1-scores of each scenario, we were able to identify the optimal methodology for training our models and improving their overall performance.

We will now present visualizations of the performance metrics, namely recall, precision, F1 score, accuracy, and specificity, for all three scenarios. The purpose of this analysis is to determine the most effective model for predictive purposes among the five models under consideration.

In general, the inclusion of word labels and normalization tends to improve the overall performance of the models across different metrics, compared to the scenario without word labels. The inclusion of word labels provides additional information for the models to learn from, potentially improving their ability to classify the data accurately. Similarly, normalization of the data can help to mitigate the effects of varying scales and ranges among the features.

However, the specific impact of the different datasets on the performance of each individual model may vary.

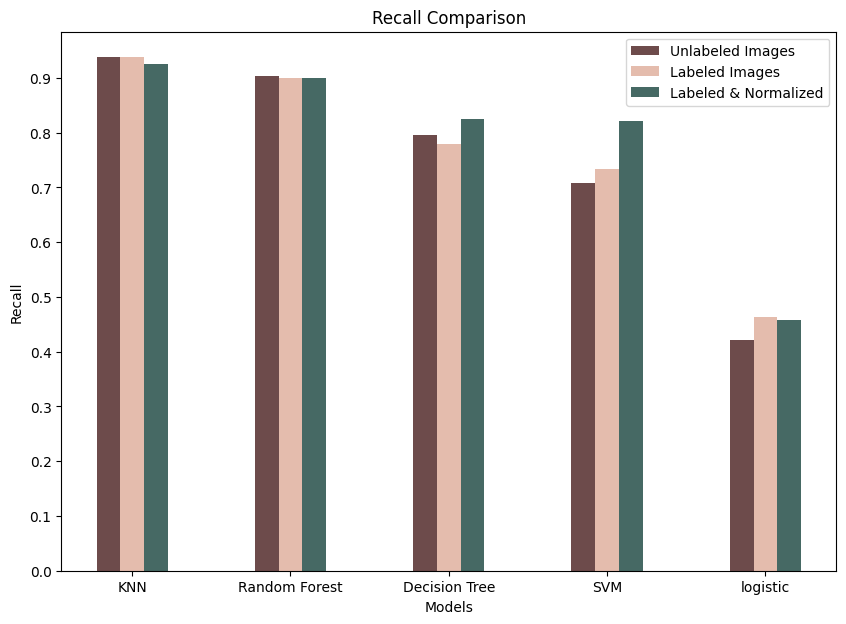


Figure 41 Recall Comparison

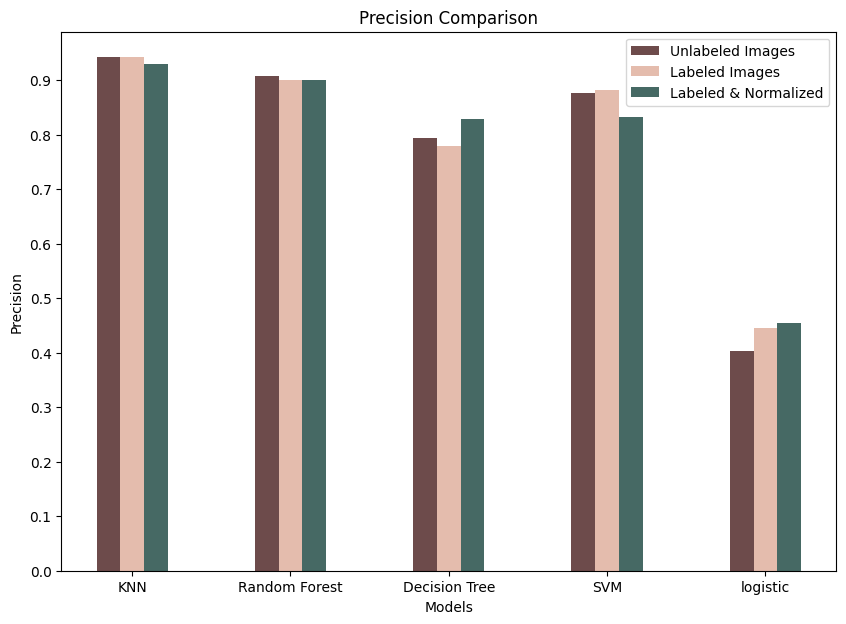


Figure 42 Precision Comparison

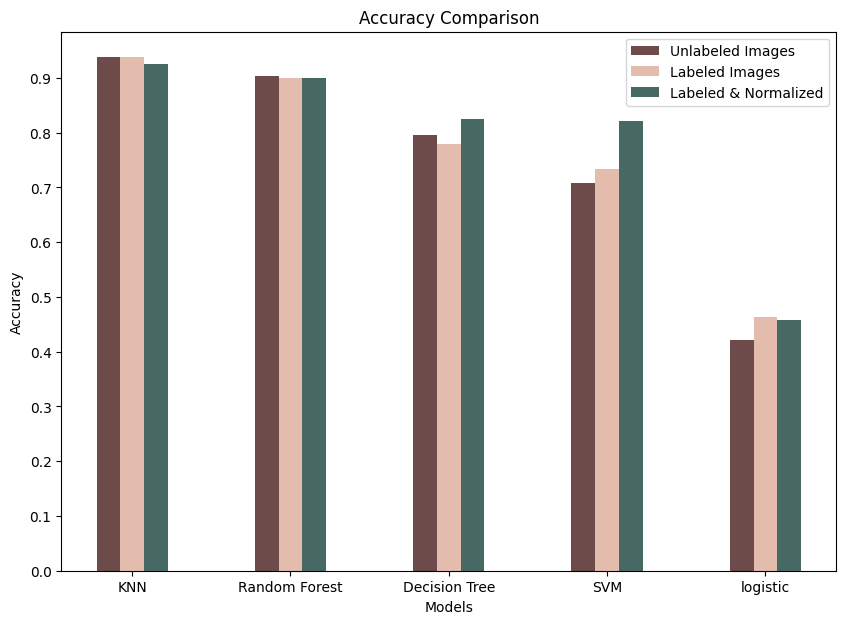


Figure 43 Accuracy Comparison

A picture containing text, screenshot, line, parallel

Description automatically generated

Figure 44 F1 score Comparison

Based on the accuracy graph, it appears that the KNN model is the best performing model. However, it is important to note that the F1 score is a more reliable metric for model selection, as it considers both recall and precision. Therefore, we will use the F1 score as the primary indicator of performance to determine the best model among the five models.

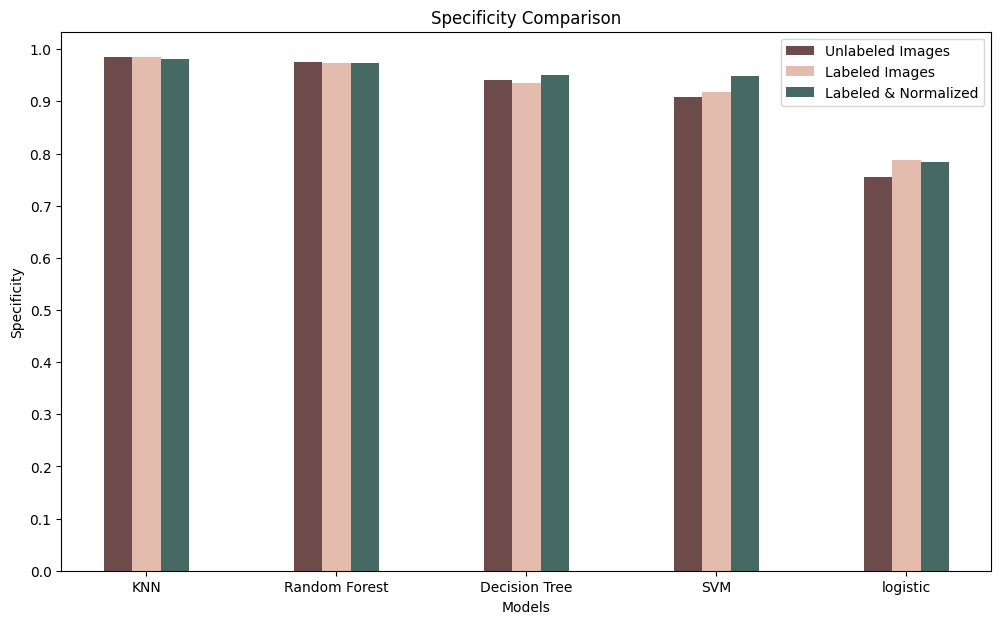


Figure 45 Specificity Comparison

As observed in the specificity graph, the first four models exhibit a high specificity of 0.9 or higher, except for the logistic regression model.

It is important to note that logistic regression is typically used for binary classification problems and may not be the optimal choice for a multi-class classification problem such as our problem. This may explain the comparatively lower specificity and accuracy observed for the logistic regression model relative to the other models.

Overall, our evaluation demonstrates that the KNN model outperformed the other models achieving an impressive F1-score of 94%. Using unlabeled images dataset

Now we will see how many correct results KNN has got correctly from our dataset

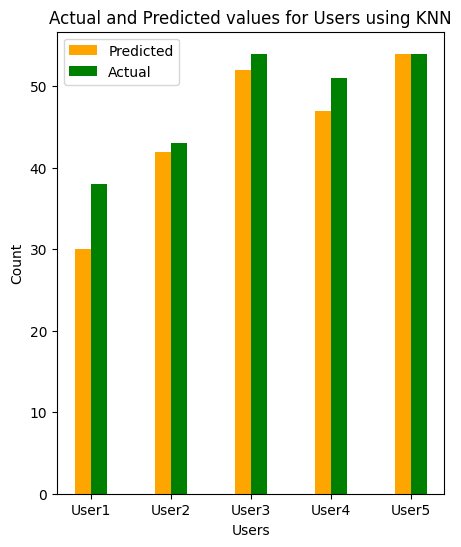


Figure 46 KNN Prediction on Unlabeled Image Dataset

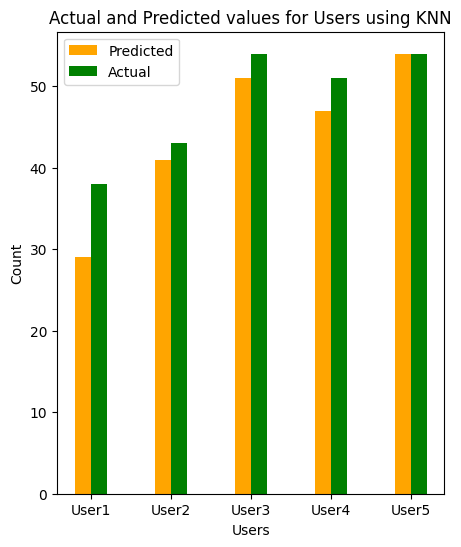


Figure 47 KNN Prediction on Labeled Image and Normalized Dataset

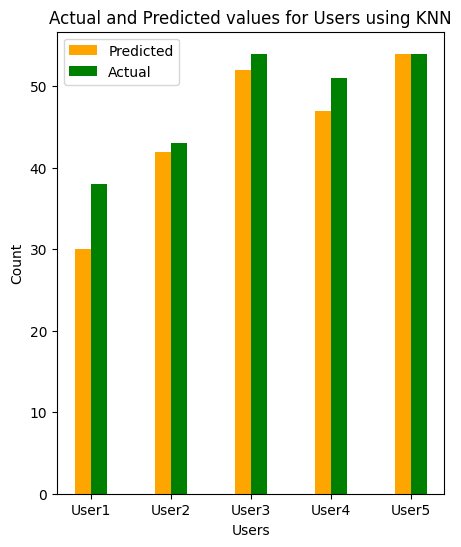


Figure 48 KNN Prediction on labeled Image Dataset

It appears that the performance of the KNN model is consistent across the scenarios with and without word labels, but the scenario with normalized data presents some challenges. Specifically, the KNN model's ability to make accurate predictions on user 3 in the scenario with normalized data

We will now compare the performance of the other four models across all three scenarios and assess their ability to make accurate predictions.

Figure 49 Models Prediction on Unlabeled Image Dataset

A picture containing text, screenshot, diagram, colorfulness

Description automatically generatedA picture containing text, screenshot, diagram, plot

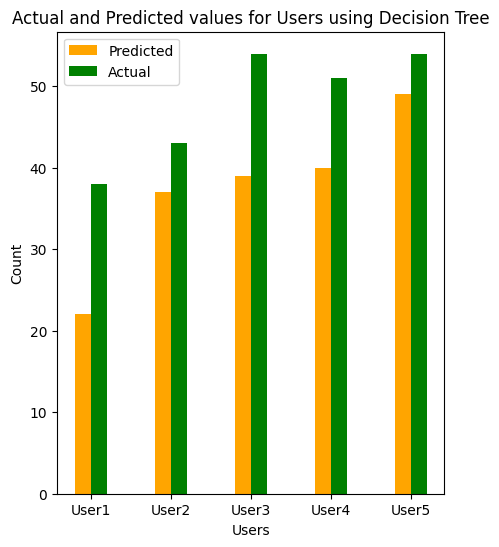
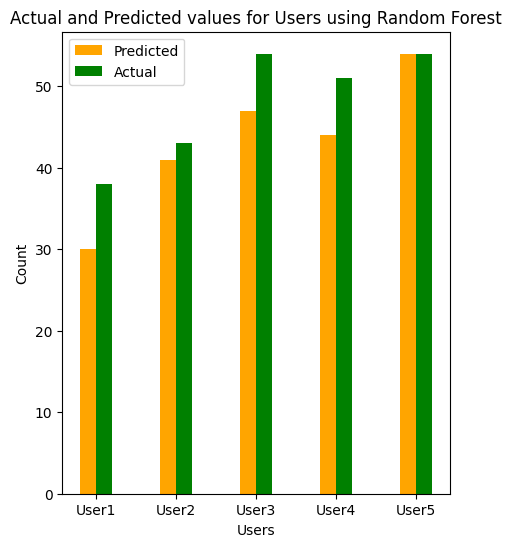
Description automatically generated

A picture containing text, screenshot, diagram, line

Description automatically generatedA picture containing text, screenshot, diagram, plot

Description automatically generated

Figure 50 Models Prediction on Labeled Image Dataset



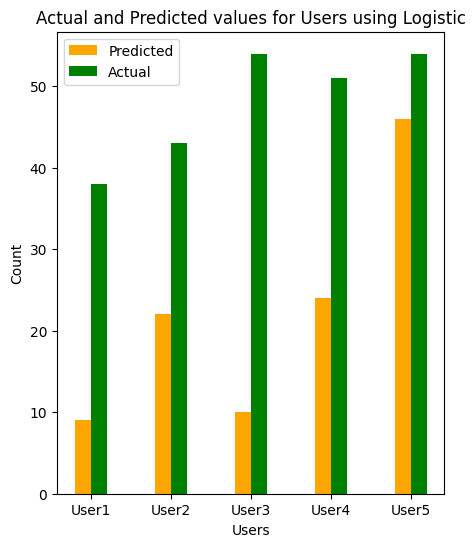
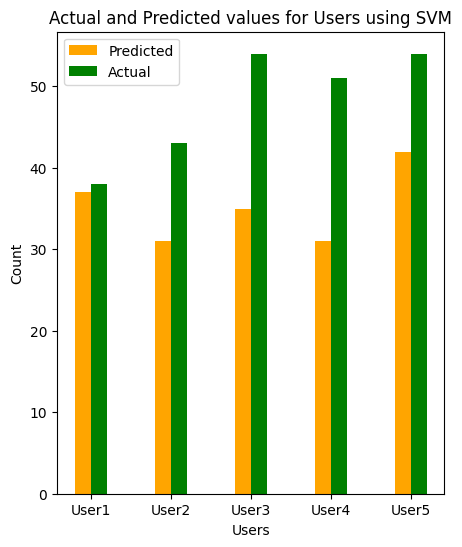
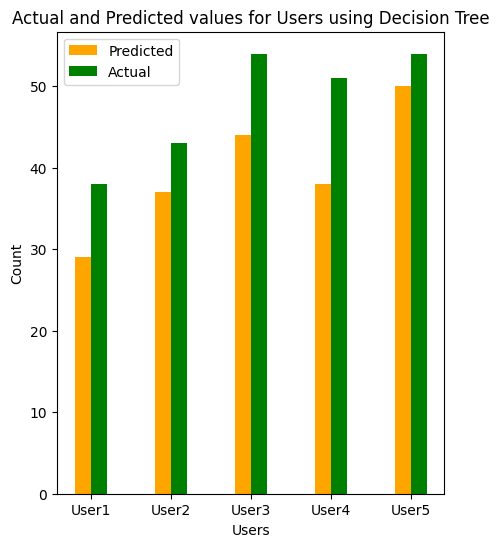
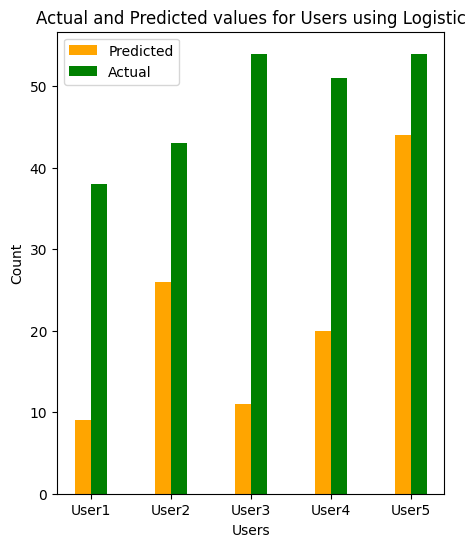
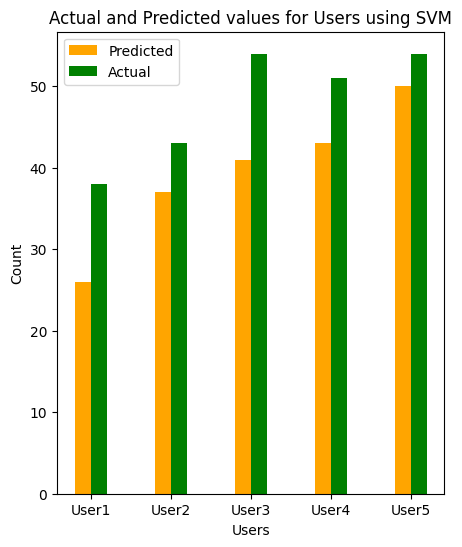


Figure 51 Models Prediction on Labeled Image and Normalized Dataset

A picture containing text, screenshot, diagram, colorfulness

Description automatically generated



Based on our analysis, the logistic regression model consistently exhibited the lowest predictive performance among the five models across all three scenarios.

Furthermore, the impact of using labeled and normalized datasets on model performance varied across the different models. Specifically, the scenario with labeled and normalized data resulted in improved performance for the decision tree and SVM models, compared to the other two scenarios. Conversely, the scenario with unlabeled data resulted in improved performance for the KNN and random forest models.

These findings suggest that the choice of dataset may have a differential impact on the performance of various models, and careful consideration should be given to selecting an appropriate dataset for a given modeling task.

# CONCLUSION

After the initial research, we concluded that there are many tools to use to be able to convert motion or hand position to images that can be read, some of which were Smart Gloves, Microsoft Kinect, and more. And the more complex the technology is, the higher the prediction accuracy in the end, but that also comes at the price of accessibility and possibly higher cost.

So, we decided to go with the basic camera that is available in almost every computer, laptop, and mobile phone which in turn increased the accessibility of our software, but from the results section, we could see that the model fails to consistently predict some words which is a good tradeoff if it’s meant to be deployed in a production environment.

For data acquisition, it’s a tedious process to manually browse internet videos, and images so it’s easier to learn the signs that we’re going to incorporate into our prediction system and automate the data capturing process. But the downside of this is that to be able to add more words, we need to learn each word manually, so it’s fine if the total number of words is minimal, but it’s not optimal for larger pools of words.

And to train our model, we looked at different artificial neural networks, but the one that was mostly used and for good reason was CNN, which is widely used for image recognition and prediction, so we used it to predict static sign language.

The model we used in the static sign language was the VGG-16 model which was employed in many articles and research papers, but we had to test for ourselves to be sure, and it was very accurate in predicting most words (way more than any other model).

From the comparison in the results section, our model’s accuracy scored an average of 96% which is better than a lot of papers, considering the fact that we’re predicting words and not just letters.

Then we learned that CNNs alone can’t be used to predict dynamic hand signs, because of the fact that CNNs only predict with the current image as input and doesn’t take into consideration the previous images, so we have to integrate Long Short-Term Memory modules in addition to our existing model.

After deciding to use the LSTM model, we found that it would be really heavy if it was combined with CNNs, as the CNNs alone were computationally expensive. So, we found that using coordinates instead of pixels proved to be a much better alternative not only because of the computation time, but also because of the fact that we can now detect both hands at the same time which is something we were not able to achieve using the CNN approach from before.

And after many experiments, we found that the model trained on 50 epochs is the best model as it was trained for long enough to effectively learn so it avoided underfitting, and stopped training just before overfitting so we achieved that perfect balance.

Machine learning models are known to have less training and testing time than Deep learning models due to the model simplicity which requires less computational time and also less resources because of the drastically smaller number of parameters in a deep learning neural network in comparison to machine learning models.

Since we need to check for the user identity using hand biometric, we need a model that can predict our hand correctly and in a short amount of time therefore we used a supervised machine learning model such as K Nearest Neighbor (KNN) to be our solution for this project. KNN with K=1 is our best model that achieved 96% for f1-score on our test set.

We also know that our problem is a multiclass problem since we identify the hand biometrics for five users, so we used four machine learning models capable of solving this multiclass problem and a binary classifier “Logistic Regression” to compare its prediction with other models.

# REFERENCES

1. Brandon Garcia, Sigberto Alarcon Viesca, Real-time American Sign Language Recognition with Convolutional Neural Networks, Stanford University, Stanford, CA.
2. Veronica J., Schmalz, Real-time Italian Sign Language Recognition with Deep Learning, Freie Universität Bozen-Bolzano, Bozen, Italy.
3. Da Ma, Kaoru Hirota, Yaping Dai, Zhiyang Jia, Dynamic Sign Language Recognition Based on Improved Residual-LSTM network, School of Automation, Beijing Institute of Technology, Zhongguancun South Street, Haidian District, Beijing, China.
4. Jay Suthar, Devansh Parikh, Tanya Sharma, Avi Patel, (2021) Sign Language Recognition for Static and Dynamic Gestures, Global Journal of Computer Science and Technology: DNeural & Artificial Intelligence.
5. Mohamed S. Abdalla, Elsayed E. Hemayed (2013) Dynamic Hand Gesture Recognition of Arabic Sign Language using Hand Motion Trajectory Features, Engineering Cairo University, Egypt.
6. Shadman Shahriar, Ashraf Siddiquee, Tanveerul Islam, Abesh Ghosh, Rajat Chakraborty, Asir Intisar Khan, Celia Shahnaz, Shaikh Anowarul Fattah, (2018) Real-Time American Sign Language Recognition Using Skin Segmentation and Image Category Classification with Convolutional Neural Network and Deep Learning, Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology Dhaka-1205, Bangladesh
7. Sawant Pramada; Deshpande Saylee; Nale Pranita; Nerkar Samiksha Mrs.Archana S. Vaidya (2013) Sign Language Recognition Using Image Processing, GES’s R. H. Sapat College of Engineering, Management Studies and Research, Nashik (M.S.), INDIA
8. Kothadiya, D.; Bhatt, C.; Sapariya, K.; Patel, K.; Gil-González, A.-B.; Corchado, J.M. Deepsign: Sign Language Detection and Recognition Using Deep Learning. Electronics 2022, 11, 1780.
9. Prof. Radha S. Shirbhate; Mr. Vedant D. Shinde; Ms. Sanam A. Metkari; Ms. Pooja U. Borkar; Ms. Mayuri A. Khandge: Sign language Recognition Using Machine Learning Algorithm
10. Alaa H Al-Obodi, Ameerh M Al-Hanine, Khalda N Al-Harbi, Maryam S Al-Dawas, and Amal A. Al-Shargabi, A Saudi Sign Language Recognition System based on Convolutional Neural Networks, Department of Information Technology, College of Computer, QassimUniversity, Buraydah, Saudi Arabia
11. E. Kiran Kumar, Student Member, IEEE, P. V. V. Kishore, Senior Member, IEEE, A. S. C. S. Sastry, Member, IEEE, M. Teja Kiran Kumar, Student Member, IEEE, and D. Anil Kumar, Training CNNs for 3-D Sign Language Recognition with Color Texture Coded Joint Angular Displacement Maps
12. Wanbo Li, Hang Pu, Ruijuan Wang, School of Software, Sign Language Recognition Based on Computer Vision, Hanwei Internet of Things Research School Zhengzhou University Zhengzhou 450000, China.
13. Dr. Sandhya Arora; Ananya Roy; Recognition of Sign Language using Image Processing, Department of Computer Engineering Cummins College of Engineering for Women Pune, India
14. Sharmila Gaikwad; Akanksha Shetty; Akshaya Satam; Mihir Rathod; Pooja Shah, Recognition of American SignLanguage using Image Processing and Machine Learning, Department of Computer Engineering, MCT‟s Rajiv Gandhi Institute Of Technology, Mumbai, India.
15. B. Mocialov, G. Turner, K. Lohan, H. Hastie, (2017) Towards Continuous Sign Language Recognition with Deep Learning
16. Kshitij Bantupalli; Ying Xie, (2018) American Sign Language Recognition using Deep Learning and Computer Vision, Department of Computer Science Kennesaw State University Kennesaw, USA
17. Ching-Hua Chuan, Eric Regina, Caroline Guardino, American Sign Language Recognition Using Leap Motion Sensor
18. G.Anantha Rao, K.Syamala , P.V.V.Kishore, A.S.C.S.Sastry11 Biomechanics and Vision Computing Research Center, Department of ECE,K.L. University, Green Fields, Vaddeswaram, Guntur (DT), Andhra Pradesh, INDIA. Department of ECE, Avanthi Institute of Engineering and Technology, INDIA. (2018) Deep Convolutional Neural Networks for Sign Language Recognition
19. Paulo Trigueiros; Fernando Ribeiro; and Luís Paulo Reis; (2014) Vision-based Portuguese Sign Language Recognition System, nstituto Politécnico do Porto, Porto, Portugal
20. Sharvani Srivastava, Amisha Gangwar, Richa Mishra, Sudhakar Singh (2022) Sign Language Recognition System using TensorFlow Object Detection API (arXiv:2201.01486)
21. Kanchan Dabre; Surekha Dholay, (2014) Machine Learning Model for Sign Language Interpretation using Webcam Images, Department of Computer Engineering Sardar Patel Institute of Technology Student of M.E.(Computer) Mumbai, India.
22. Shashank Salian; Aditya Suresh; Pranav Ganorkar; Indu Dokare; Dhiren Serai, (2017) Proposed System for Sign Language Recognition, Department of Computer Engineering V.E.S. Institute of Technology, Mumbai, India.
23. G.Anantha Rao, K.Syamala, P.V.V.Kishore, A.S.C.S.Sastry1, (2018) Deep Convolutional Neural Networks for Sign Language Recognition, Biomechanics and Vision Computing Research Center, Department of ECE, K.L. University, Green Fields, Vaddeswaram, Guntur (DT), Andhra Pradesh, INDIA
24. Runpeng Cui , Hu Liu , and Changshui Zhang (2019) A Deep Neural Framework for Continuous Sign Language Recognition by Iterative Training
25. https://www.kaggle.com/datasets/risangbaskoro/wlasl-processed?select=wlasl\_class\_list.txt WLASL (World Level American Sign Language) Video, Kaggle dataset
26. https://dai.cs.rutgers.edu/dai/s/signbank
27. https://aslsignbank.haskins.yale.edu/
28. M. Quinn, J.I. Olszewska, British Sign Language Recognition In The Wild Based On Multi-Class SVM
29. Kanchan Dabre, Surekha Dholay, Machine Learning Model for Sign Language Interpretation using Webcam Images
30. Vision-Based Portuguese Sign Language Recognition System
31. Veronica J. Schmalz1,2, Real-time Italian Sign Language Recognition with Deep Learning
32. Pranjal Srivastava. Essentials of deep learn-ing: Introduction to long short term memory. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm, 2017.
33. Sepp Hochreiter and J ̈urgen Schmidhuber. Long short-term memory, Jo-  
    hannes Kepler University Linz, 1997.
34. Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015
35. Marcus Rohrbach Subhashini Venugopalan Sergio Guadarrama Kate Saenko  
    Trevor Darrell Jeff Donahue, Lisa Anne Hendricks. Long-term recurrent  
    convolutional networks for visual recognition and description, 2016.
36. Samy Bengio Oriol Vinyals, Alexander Toshev and Dumitru Erhan. Show  
    and tell: A neural image caption generator, 2015.
37. Lorenzo Torresani Jamie Ray Yann LeCun Du Tran, Heng Wang and  
    Manohar Paluri. Learning spatiotemporal features with 3d convolutional  
    networks, 2015.
38. KyungHyun Cho Junyoung Chung, Caglar Gulcehre and Yoshua Bengio.  
    Empirical evaluation of gated recurrent neural networks on sequence mod-  
    eling, 2015.