

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

***Course Code: CSE-491***

***Course Name: Computer & Systems Engineering Graduation Project***

***Graduation Project Supervisor***

**DR. Haytham Azmi Mahmoud Elmiligi**

**Ain Shams University**

**Faculty of Engineering**

**Fall Semester – 2022**

**Students Personal Information**

|  |  |  |
| --- | --- | --- |
| **NAME** | **ID** | **MAJOR** |
| Khaled Medhat Mahmoud Khalifa | 18P3557 | CESS |
| Mohamed Magdy Mostafa Sayed | 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**

**Yomna Hussien Mohamed Abd El Hamid**

**Khaled Medhat Mahmoud Khalifa Mahmoud Mourad Youssef**

**Mohamed Sayed Awwad Badawy**

Table of Contents

[I. INTRODUCTION 5](#_Toc124027184)

[II. RELATED WORK 6](#_Toc124027185)

[III. DATASET AND FEATURES 17](#_Toc124027186)

[Dataset Description 17](#_Toc124027187)

[Pre-processing and data augmentation 19](#_Toc124027188)

[IV. METHODOLOGY 21](#_Toc124027189)

[V. EXPERIMENTAL EVALUATION AND RESULTS 23](#_Toc124027190)

[I. Initial Version of Models 23](#_Toc124027191)

[II. Final Version of Models 26](#_Toc124027192)

[VI. CONCULSION 32](#_Toc124027193)

[VII. REFERENCES 33](#_Toc124027194)

Table of Figures

[Figure 1 Top 25 Words Histogram 18](#_Toc124027195)

[Figure 2 Top 25 Static Words Histogram 19](#_Toc124027196)

[Figure 3 Full Image of Word “Shirt” 20](#_Toc124027197)

[Figure 4 Cropped Image of Word “Shirt” 21](#_Toc124027198)

[Figure 5 Flipped Image 22](#_Toc124027199)

[Figure 6 Initial Models Validation Accuracy 24](#_Toc124027200)

[Figure 7 Initial Mdoels Validation Loss 24](#_Toc124027201)

[Figure 8 Inception-V3 Heatmap 25](#_Toc124027202)

[Figure 9 Initial VGG-16 Heatmap 25](#_Toc124027203)

[Figure 10 Initial MobileNet-V2 Heatmap 25](#_Toc124027204)

[Figure 11 Initial ResNet50-V2 Heatmap 25](#_Toc124027205)

[Figure 12 Initial Models Accuracies 26](#_Toc124027206)

[Figure 13 Initial Models F1-Scores 26](#_Toc124027207)

[Figure 14 Models Validation Accuracy 27](#_Toc124027208)

[Figure 15 Models Validation Loss 27](#_Toc124027209)

[Figure 16 EfficientNet Validation Accuracy 28](#_Toc124027210)

[Figure 17 EfficientNet Validation Loss 28](#_Toc124027211)

[Figure 18 VGG-16 Heatmap 29](#_Toc124027212)

[Figure 19 VGG-19 Heatmap 29](#_Toc124027213)

[Figure 20 ResNet50-V2 Heatmap 29](file:///C:\Users\user\Downloads\Graduation%20Project.docx#_Toc124027214)

[Figure 21 MobileNet-V2 Heatmap 29](#_Toc124027215)

[Figure 22 ResNet101-V2 Heatmap 30](#_Toc124027216)

[Figure 23 EfficientNet-B7 Heatmap 30](#_Toc124027217)

[Figure 24 EfficientNet-V2L Heatmap 30](#_Toc124027218)

[Figure 25 EfficientNet-V2M 30](#_Toc124027219)

[Figure 26 Models Accuracy Comparison 31](#_Toc124027220)

[Figure 27 Models F1-Score Comparison 31](#_Toc124027221)

[Figure 28 VGG-16 Results 32](#_Toc124027222)

**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, 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.**

**Keywords— deep learning, sign language, computer vision, hand gesture recognition, convolution neural network (CNN)**

# INTRODUCTION

People who have trouble speaking or hearing use sign language as a way of communication. Non-verbal movements in sign language are used by people to convey their thoughts and feelings. However, non-signers find it very challenging to comprehend these gestures, so qualified sign language interpreters are required for medical and legal appointments, as well as for educational and training sessions. Our system seeks to close this communication gap and let the deaf and the mute carry out their regular activities by using a simple application.

To implement this, we made a model to recognize different hand gestures by using American sign language data set to train the model. The dataset contains a variety of hand gestures that were done repeatedly under various video and context situations. We used a Convolution Neural Network (CNN) model to extract spatial features from the video stream for Sign Language Recognition.

This technique will undoubtedly benefit society by assisting those who are physically disabled. A deaf-mute will be able to converse with regular people via computer with the help of this device. As a result, the main goal of this system is to make it feasible for deaf individuals to communicate with the rest of the world on a daily basis. The teaching and learning process is lagging at schools for the deaf because there are so few qualified sign language tutors. In such circumstances, this approach can be used to teach sign language, allowing anyone to learn or practice sign language.

# 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 MobileNet v2 320 × 320. The SSD MobileNet v2 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.

# DATASET AND FEATURES

## 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 no LSTMs were introduced to our models.
* 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)

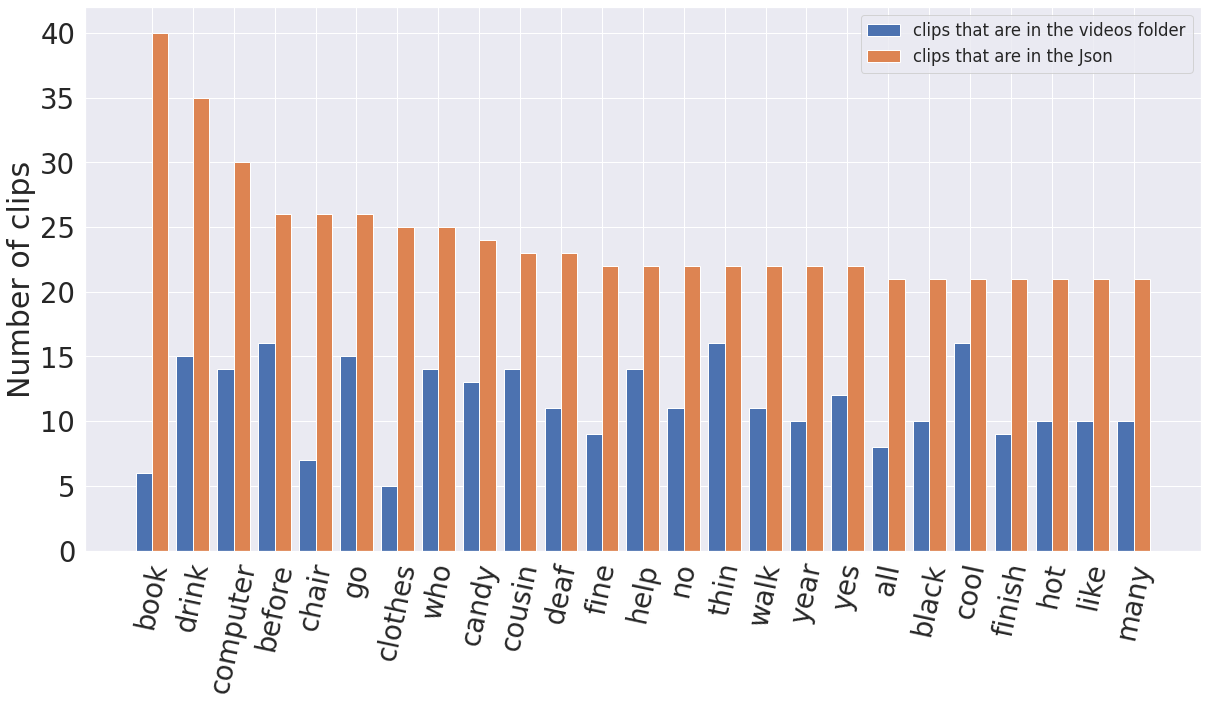


Figure 1 Top 25 Words Histogram

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

Then we tried training the model on the collected dataset, but the results weren’t so good neither on test pictures nor in a real-time setting.

The second iteration of the dataset was that we used YouTube and ASL Sign Bank [26] to crop frames of the signed words in a 1080 x 1080 resolution, we included the whole body, the hands, and the background for this iteration. We collected 40 images per sign all the same size, then we split the data 75% train and 25% validation (30 and 10 images respectively). Then we trained VGG16 model on the collected dataset, and the results weren’t good either.

For the third iteration, we wanted to train on the hands only, so we looked up ways of cropping the hands, the first method we found was LabelImg which was very easy, but not very optimal as we had to manually draw boxes around the entire dataset, and it wasn’t very optimal on the long run as well, because if we added data we’d have to manually update the boxes.

The second method we found was MediaPipe Hand-Detection module. MediaPipe is a Python library that contains a lot of automatic detection modules such as a Face Detector, or a Body Pose Detector, but we’ll only be using the Hand Detector in our project.

The final dataset contained 15 words, each word consists of 60 training images and 20 validation images, half of each is flipped to allow for both left and right-handed signs. All in a 350 x 350 resolution for uniform training.

## Pre-processing and data augmentation

For our initial model, we only focused on 12 static signs which are (bed, shirt, wrong, cow, full, water, show, sick, police, sandwich, father, have). Since the signs had an unequal number of videos, we had to collect videos from other sources so that each sign would have total of 40 videos, performed by different signers in varying lighting conditions and speed of signing with different yet simple backgrounds for the purpose of making our model more accurate and robust. We also had to remove corrupt sample files and replace them with new samples from different sources.

The added sample videos were from different sources such as ASLLRP Sign Bank [26], ASL Signbank [27] and other more sources. We then took a clean square frame from each sample video that represents the sign with constant 1080x1080 resolution and added them to folders so that we can use them as our 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 but the results were very poor and so we had to change both models and dataset.



Figure 3 Full Image of Word “Shirt”

Then we integrated the Hand Detection module from Media Pipe, and we used it to draw a bounding box around the detected hand. Then, we extracted that bounding box as a frame on its own. Then we used the extracted frame to fill a 350 x 350 box, the remaining areas are filled with white. We had to account for the orientation of the hand to fill either left and right or top and bottom. So, we found the larger dimension of the X and Y and filled the white spaces accordingly. If the X is larger, that means that the hand is horizontal, and we fill top and bottom parts, and if the Y is larger, that means that the hand is vertical, and we fill right and left.

Then we looped on all words, each one contained in a folder, and we applied the aforementioned method on it, and saved the pictures in a separate folder.

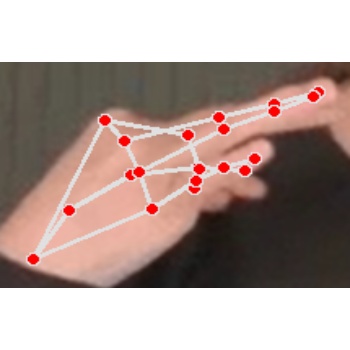


Figure 4 Cropped Image of Word “Shirt”

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

After trying different model architectures, we found the results of the other models to be either identical to the one we already used at the time or worse. So, we tried changing the pre-processing algorithm, but we couldn’t think of any better algorithm that could be both scalable and efficient.

So that brought us to number two, which is getting more data. After searching and getting as many images as we possibly could, it was still not enough. So, we gathered the data ourselves. We applied the pre-processing algorithm of cropping the bounding box and transferring it to a 350 x 350 image, then we added that if we pressed “S” we’d save the photo with a unique name that properly signifies what that word is (e.g. shirt\_2).

After finishing, we’d have gathered 40 images for each word. And to make the model more robust, and allow for both left and right-handed signs, we looped on all folders, and vertically flipped the images. After flipping, we ended up with 80 images for each word, which were then split into 60 training (30 normal and 30 flipped) and 20 validation (10 normal and 10 flipped) all of size 350 x 350.

The final dataset consisted of 15 words: “Drink”, “Food”, “Full”, “Have”, “Hello”, “I”, “I love you”, “Police”, “Prefer”, “Shirt”, “Telephone”, “Water”, “Wrong”, “Yes”, and “You”. Each word consists of 60 training images and 20 validation images for a total of 900 training images and 300 validation images all of resolution 350 x 350.

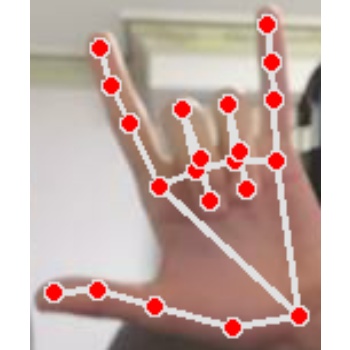
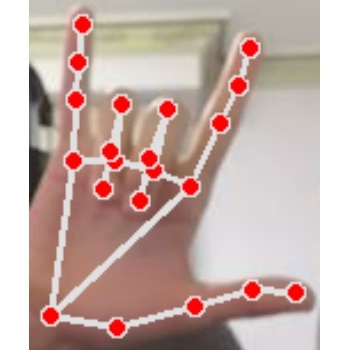


Figure 5 Flipped Image

# METHODOLOGY

We first thought about the data collection process, then we thought about what color format will we use, then how to automatically detect the hands, and lastly, which model to use.

At first, we passed the entire 1080x1080 images with people and not just their hands, and the accuracy was almost close to 0.So, we thought about normalizing the data by only detecting the hands, which is why we imported **MediaPipe’s Hand Detection Module** which detects hands in a rendered frame and draws a bounding box around it. Then we wanted to generate an automatically cropped dataset for scalability, so the process was as follows:

1. Loop on all images in all folders
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. Save the 350x350 image in a new directory for each word
7. Flip the images vertically
8. Split into training and validation

We tried a bunch of different models, but the best ones at first after real time testing were VGG-16, MobileNet-V2, and ResNet-V2-50. Then we kept testing different models with emphasis on the aforementioned three in a real time environment.

The final used model was VGG-16, it was better than the VGG-19 maybe because of overfitting or vanishing gradient problems due to the increased number of layers.

Then the next problem that arose was the transitional periods between each word, where the model would predict a sign with high confidence (99% at times) so we had to find a workaround. First, we thought about giving the user a cooldown period between each sign and the next one, but that would sort of ruin the experience.

The second thing we thought of was graphing the predicted values of each word as percentages of the total and creating a threshold, because we thought that the confidence would be significantly lower when predicting signs during transitions, but that was not the case at all. In fact, the model would predict with 99% or even 100% confidence. So the threshold we made was at 99.999999% which means that if the confidence is higher than that number, then it shows the predicted sign, if it’s not, then it doesn’t show the predicted sign. That almost solved the problem of transitions, but it didn’t work in all cases, so we had to figure out a workaround in cohesion with this method.

So, that brought us to the fourth idea, which was lowering the framerate. At first, we lowered the capture framerate which was not ideal, but it solved the prediction issue. Then we decided to keep the capture framerate the same but lower the prediction’s so that the model skips most frames between one sign and the next, which made it basically transition frames.

Then we connected our model with the front-end using Flask and rendered it on an HTML page for the user to see.

# 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.

|  |
| --- |
| Figure 6 Initial Models Validation Accuracy |
| Figure 7 Initial Mdoels Validation Loss |

|  |  |
| --- | --- |
| Figure 8 Inception-V3 Heatmap | Figure 9 Initial VGG-16 Heatmap |
| Figure 10 Initial MobileNet-V2 Heatmap | Figure 11 Initial ResNet50-V2 Heatmap |

The heatmaps show that the accuracies on the initial dataset are only remotely good on the VGG-16 and MobileNet-V2. The other two models didn’t have good accuracies as well, but the ResNet50-V2 produced better results when used in a real-time environment than the Inception-V3, so we decided to keep experimenting on the VGG-16, MobileNet-V2, and the Inception-V3 models.

|  |  |
| --- | --- |
| Chart, bar chart  Description automatically generated  Figure 12 Initial Models Accuracies | Chart, bar chart  Description automatically generated  Figure 13 Initial Models F1-Scores |

The final comparison shows that the VGG-16 is the most accurate model due to both high accuracy and F1-Score which makes it the most suitable out of all the models we experimented on.

## 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 7 epochs on the final dataset produced these results.

|  |
| --- |
| Figure 14 Models Validation Accuracy |
| Figure 15 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:

|  |
| --- |
| Figure 16 EfficientNet Validation Accuracy |
| Figure 17 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 more clear insight as to how each model behaves and how accurate each model is.

|  |  |
| --- | --- |
| Figure 18 VGG-16 Heatmap | Figure 19 VGG-19 Heatmap |
| Figure 20 ResNet50-V2 Heatmap | Figure 21 MobileNet-V2 Heatmap |
| Figure 22 ResNet101-V2 Heatmap | Figure 23 EfficientNet-B7 Heatmap |
| Figure 24 EfficientNet-V2L Heatmap | Figure 25 EfficientNet-V2M |

From these heatmaps, we can observe that the almost all models produce very poor results, except for VGG-16 and VGG-19 with the former being the being slightly better.

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

|  |
| --- |
| Figure 26 Models Accuracy Comparison |
| Figure 27 Models F1-Score Comparison |

These two bar charts prove that the VGG-16 is the best model to use for our purposes.

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

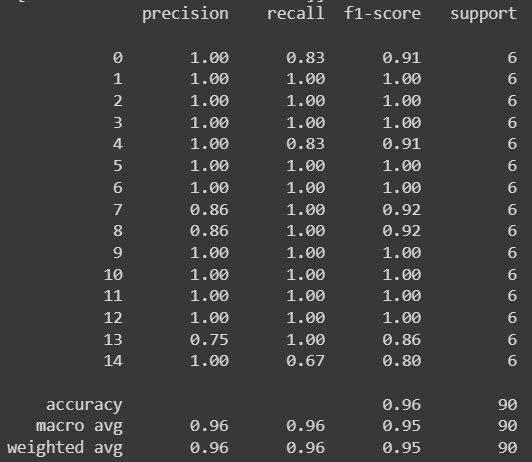


Figure 28 VGG-16 Results

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

# CONCULSION

# 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. 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
18. Paulo Trigueiros; Fernando Ribeiro; and Luís Paulo Reis; (2014) Vision-based Portuguese Sign Language Recognition System, nstituto Politécnico do Porto, Porto, Portugal
19. Sharvani Srivastava, Amisha Gangwar, Richa Mishra, Sudhakar Singh (2022) Sign Language Recognition System using TensorFlow Object Detection API (arXiv:2201.01486)
20. 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.
21. 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.
22. 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
23. Runpeng Cui , Hu Liu , and Changshui Zhang (2019) A Deep Neural Framework for Continuous Sign Language Recognition by Iterative Training
24. <https://www.kaggle.com/datasets/risangbaskoro/wlasl-processed?select=wlasl_class_list.txt> WLASL (World Level American Sign Language) Video, Kaggle dataset
25. <https://dai.cs.rutgers.edu/dai/s/signbank>
26. <https://aslsignbank.haskins.yale.edu/>