Sign Language Translator Using Hand Pose Estimation and LSTM

Ahmed Ahmed, Bassel Halabi, Chris Amgad, Feras Awaga, Mohammed Mostafa Safo, Seif ElDin El Moghazy

Supervised By: Dr. Hesham Eraqi, Dr Mohamed Mostafa

[#]*Computer Science and Engineering Department, The American University in Cairo*
Cairo, Egypt

Abstract----Communication between people has always been a great challenge over history, especially when the person in front of you is deaf or mute. Their only form of communication comes through sign language which requires a specialist to understand it. Thus, we propose a sign language interpreter that provides dual communication. It receives sign language and converts it to text then audio and also can take audio, convert it to text then generate a series of pictures having the predicted gestures. Our implementation will require us to use deep learning technologies such as Long Short-Term Memory (LSTM), hand pose estimation using mediapipe, and Natural Language Processing (NLP).

1. Introduction

Over the past century, there has been a communication gap between deaf and mute people and the rest of the people. According to the World Health Organization [1], over 5% of the world's population, which is 467 million human beings, suffer from some degree of hearing loss of which 432 million are adults and 34 million children. It is essential for them to express themselves and be able to carry on with their day-to-day activities. Hence, sign language is the method by which they can communicate and voice out their opinions but there has to be a translator in order for a non-sign language user to understand. With the number of deaf and mute people on the rise, the need for more translators and specialists increases. Bridging this gap between people with hearing and speaking disabilities and the rest of the people has led us to come up with the idea of a real-time sign language interpreter. This implementation aims to capture gestures from a deaf or mute person using a camera, translate these signs to text and convert this text to audio to give the sense of a real conversation between the two parties. Similarly, the application can receive audio from a person, convert it to text and then produce a series of signs representing the sentence said by the person. If possible, there will be an option for an AI-generated video of a person performing these signs. The sign languages that should be accepted are the American Sign Language (ASL) and hopefully Egyptian Sign Language. Almost every country has a different sign language. For example, for the same language, there can be 2 different sets of sign gestures. The American Sign Language is an entirely different language than the British Sign Language (BSL) so Americans with hearing and speaking disabilities cannot communicate with their British counterparts although they speak the same language. Deep learning techniques will be used to aid with our proposed implementation and these involve mediapipe to handle significant feature extraction, RNNs to try to process sequences of moves and capture words that need multiple gestures.

2. Literature

A huge amount of research was made in the field and multiple approaches were taken by different researchers in the field to find a solution to bridge the gap between people who talk using sign language and those who do not.

In one research, researchers proposed an approach [1] using a dataset that consisted of 46 gestures. In order to process the videos of hand gestures, their architecture included a CNN model for spatial features, in addition to an LSTM model to extract temporal information. Their method consisted of breaking down the video of the hand gesture into x numbers of frames, with each frame being labeled with the gesture's original label and fed to the CNN model to start the training process. The CNN model is then responsible for generating a prediction for each frame, which will be then sent to the RNN for temporal features training. This approach yielded an accuracy of 95.217%.

In another research [2] that was conducted at the university of science and technology of china, The data contained 100 sign language words that are widely used in daily life. 50 signers played each word 5 times. So each word had 250 samples. They divided their dataset into 2 subsets one for training and the other for testing. In order to come out with optimal results, they varied the LSTM hidden units between

256 and 1024. They obtained the best results by setting the LSTM hidden units to 512 hidden units. This approach yielded an accuracy of 63.3%.

There is also research [3] that was done by students at Zagazig University on Egyptian sign language.The Data consisted of only 9 words. Their CNN-LSTM architecture involved using a pre-trained CNN for feature extraction from input data along with LSTMs for sequence prediction. In the LSTM Model, to interpret features across time steps they Used 2 Methods; The first method was pre-training the CNN model on their data. And the second one was passing the predicted labels from the CNN model to an LSTM. The last hidden layer of the CNN was the input to the LSTM. After extracting the bottleneck features, They used a network consisting of a single layer of 512 LSTM units. They achieved 90% accuracy using CNN only CNN and LSTM gave an accuracy of 72%.

In another research [4], researchers from Prince Mohammad bin Fahd University were developing an automatic detection system for the Arabic Sign Language (ArSL) that was based on the use of deep convolutional neural networks (CNN). To carry this study out, a dataset of almost 54,000 pictures was used which contained all of the Arabic alphabet consisting of 28 letters. The parameters of the CNN model were changed in every test to check the accuracy of the model and try to reach an architecture that yields a high accuracy with ArSL. In their model, they could only detect letters but not words or sentences. The accuracy they managed to achieve was 97.6%.

Kartik et al. [5] proposed a model that converts American Sign Language (ASL) to text using deep learning techniques involving Convolutional Neural Networks (CNN). The dataset used for their implementation was 87002 images chosen randomly from the ASL dataset where 78000 were used for training and 8700 for testing (ratio of 80:20). Before these images are fed to the CNN, some preprocessing needs to take place to decrease the computational power required. The proposed architecture consists of 2 convolutional layers, 2 max-pooling layers and an ADAM optimization function. They managed to achieve a training accuracy of 98.68% and 90% validation accuracy.

Mehreen Hurroo and Mohammad Elham [6] proposed an American sign language recognition System using CNN and computer vision. The hand gestures would be captured by the webcam of the computer or laptop. The first step in their proposed system was creating their own dataset of 2000 images consisting of 10 classes of alphabets (A,B,C,D,K,N,O,T and Y). They enlisted the help of 2 signers who each performed each gesture 200 times in different lighting conditions. Out of the 2000 images, 1600 were used for training and 400 for testing. They achieved results of over 90% accuracy on their dataset.

3. Implementation

For the implementation, we had two different directions to tackle with the first being the sign to text pipeline and the second being the audio to sign pipeline. For the first pipeline, there were multiple stages that we went through to reach a fully functional model. First, we had to gather a dataset large enough to start the training process. Second, we had to perform some preprocessing on this collected dataset to enhance it. Finally, we had to choose a model that is capable of fulfilling our objective and tackles our problem in the most efficient way possible. Concerning the second pipeline, it also consisted of several steps in order to achieve the full cycle of communication. To begin with, audio has to be detected from the user via a microphone before being converted to text. Afterward, NLP techniques are applied on each word to get its origin so that it could be mapped to a certain sign. In the following sections, we will illustrate further how each pipeline was completed.

1. Sign to Text

a. Dataset

The first phase of the implementation was the collection of the dataset. Our initial approach was to tackle public open-source datasets and maintain a primary source of data within our research paper. Based on our findings, the two most prominently open-source datasets are Microsoft ASL and WLASL (Word-Level American Sign Language). As per table 1, there are multiple subsets of Microsoft ASL ranging from 100 classes up to 1000 classes, and finally, there is the WLASL having 2000 classes. After carefully analyzing every one of the given datasets with reference to their published research papers, we concluded that the number of videos per class did not convey the actual amount of data we tried to access due to the fact that many of the data were corrupted or inaccessible. As a result, we were left down with a significant number of missing data. Our main goal was to construct our outcomes established upon an optimal dataset in terms of the number of classes and the ratio of the missing videos to the existing videos. After analyzing each and every one of the following public datasets, the ratio of missing videos to the existing videos was the least in the WLASL dataset in contrast to the other subsets of Microsoft datasets, putting into account that WLASL also had the greatest number of classes; thus, making WLASL our optimal public dataset choice for our research. Although WLASL still managed to satisfy our outlined requirements, the number of videos per class was not sufficient as we had a mean of 9.2 videos per class, which was not enough to boost the performance and accuracy of our model. As stated before, the missing data of the public dataset was the actual reason behind ending up with such a low amount of videos per class; therefore, it was still required to find another way to improve our dataset.

Set	Class	Samples
MS-ASL100	100	5736
MS-ASL200	200	9719
MS-ASL500	500	17823
MS-ASL1000	1000	25513
WLASL	2000	21083

Table 1. Datasets

Another approach that we chose to go for is to manually collect our data with an aim of increasing the number of videos per class. Our manual data collection process focused on a wide range of public open-source databases and educational websites. Although this stage was the most time-consuming, it was a priority for us to ensure an acceptable amount of data to base our experiments on as well as guarantee dependable and consistent final results. Eventually, the WLASL dataset was merged with our manually collected data, and the mean of the number of videos per class significantly increased to 25.2. At this point, we were ready to move on to the next phase, which is the data preprocessing.

An additional approach that we have taken to increase the size of our dataset was to self-record ourselves using an HD video camera doing the ASL signs. Each individual in our team has performed the video signs for 100 classes of words then we merged these videos with our dataset. After running our model on this new dataset, it was concluded that the new videos have degraded the performance of the model due to the fact that the key points that were extracted from the videos were always at different positions, and orientations, and had different translation magnitudes than the ones that were originally present in the main dataset. We decided to disregard these videos from our final dataset and continue off with the first approach only. At this point, we were ready to move on to our next phase, which is the data preprocessing phase.

Figure 1. Diversity of our dataset that includes different signers at different conditions and environments

b. Data preprocessing

After collecting the dataset from multiple sources as stated earlier, we are now ready for the next step in our implementation: data preprocessing. Data preprocessing was divided into 3 stages: standardizing the number of frames, normalizing video dimensions, and augmentation.

I. Standardize the number of frames

This is one of the most important steps in our implementation because the number of frames is one of the inputs to the model we are proposing. Hence, we need to have a certain unified number for all our videos. Since sign language depends on a sequence of gestures, we cannot just take the first 40,50,80 frames of the video as this could lead to the loss of crucial information if a sign video has more frames than collected. Instead, we decided to standardize the number of frames to ensure the best results possible.

Therefore, our first approach to making all videos in our dataset have the same number of frames was to get the average number of frames in the whole dataset and apply it to all the videos. This average was almost 70 frames, but this turned out to be hugely unsuccessful when we started training. This was due to the fact that this large input to the model increased its complexity as it was difficult for the model to process a sequence of 70 frames; hence, yielding extremely poor results. Consequently, we decreased the number of frames per video to become 40 frames and this provided us with much more realistic and satisfying results. However, we decided to proceed with 30 frames per video as input to our proposed model to get better results.

II. Normalizing Dimensions

Our second step in preprocessing was to standardize the dimensions of the videos in the whole dataset. Similar to the first step, our initial approach was to get the average dimensions of the videos and resize the videos based on these dimensions. This also yielded poor results since this led to a decrease in the quality of the videos whether by increasing the dimensions or decreasing them causing some sequences not to be captured completely correct. Thus, we resorted to the option of normalization of the dimensions. This means that the height and width of all videos are within a specified range. The minimum height for the videos was 280 and the maximum was 420. For the width, the minimum was 520 and the maximum was 720. Table 2 has the range of dimensions of the videos after normalization.

	Minimum	**Maximum**
width	520	720
height	280	420

Table 2: Normalized dimensions

III. Augmentation

As stated earlier in the previous section, there was a major issue in gathering our dataset. The average number of videos per class (25.2) was still a small amount of data for us to start our training process. This was proven when we tried to test on 10 classes only and the results found were extremely unsatisfactory. Hence, we decided to augment our data meaning that we would enlarge the gathered dataset to an extent that it is ready to be fed to the model. A library called imgaug was used to perform our augmentation. First, we convert every video to a sequence of frames that are appended in an array. This is due to the fact that imgaug can only apply augmentation to images, not videos. Afterward, the required augmentation is applied to this sequence of frames before converting the resulting frames to a video once again.

Our first approach to augment our dataset was to certain types of augmentation such as adding specific noises to the videos like the additive gaussian noise. Other types of augmentation we thought of using were increasing or decreasing the brightness of the videos, sharpening and blurring the videos. However, the model we are proposing is gathering key points from the videos in the form of coordinates so these types of augmentations will not solve our problem.

Thus, our second and final approach was to apply one type of augmentation which is rotation. Rotation was the ideal augmentation since it will not alter or affect the quality of the videos so the key points gathered will not be distorted. Moreover, it helps solve the issue of the coordinates collected as the coordinates collected after rotation are now different from the original videos. The angles of rotation used were very small as they had values of 10,15,20,25,30 -10,-15,-20,-25,-30. Finally, we reached a mean of 121.93 videos per class which made us ready to start the training process.

c. Proposed Model Architecture

In the process of researching and designing the architecture of our model, the target was to come up with a powerful model in terms of classification accuracy, as well as a less complex model compared with complex CNN + RNN models as proposed in [1],[2], and [3] in order to fit with our aim. In order to achieve this, we built upon the power of LSTM-based models in the detection of sequences and temporal information in multiple frames and sequences. However, we introduced the usage of hand estimation techniques in order to decrease the complexity of our model by using MediaPipe, rather than using pre-trained CNNs.

I. MediaPipe Hand Pose Estimation

Mediapipe is a cross-platform framework developed by Google that uses machine learning techniques to detect 3D landmarks of the body. Mediapipe offers multiple options, including Hand tracking of 21 landmarks per hand, Holistic tracking of 33 landmarks of multiple joints in the body, and Face mesh for face landmarks analysis. Mediapipe can segment the landmarks from the RGB of video frames, which decreases the amount of data fed to the LSTM architecture. For our model, we capitalized on Mediapipe's hand pose detection algorithm in order to track the landmarks in each of the videos used for training and testing. Our experimentation phase included trying to use Mediapipe's As Mediapipe is used, the input shape fed to our LSTM architecture is (30, 126), 126 landmarks of both hands for each of the 30 frames.

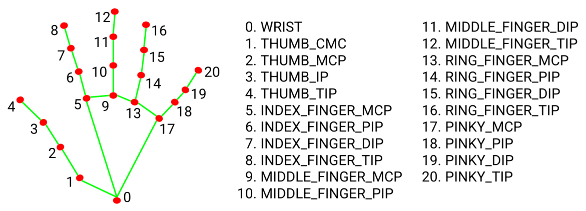

Figure 2. Hand detected landmarks

II. LSTM-based Architecture

In order to detect the sequence between each of the different frames, it was obvious that our choice would be based on sequential models. After thorough research of the literature, we based our model on Long short-term memory (LSTM) architecture. LSTM is a variation of a Recurrent neural network (RNN) that is capable of learning long-term variations, as its design and architecture aids in the learning of long sequences and dependencies, compared to regular RNN short-term memory and vanishing gradient problem. The core concept of the LSTM lies in the cell state, which acts as the memory that passes through all the network. The gates inside the LSTM cell tells the cell state what data should be neglected from the cell and what data shall be passed. These gates are the Forget gate, which decides what data shall be passed and what shall be forgotten, Input gate, which calculates the input to update the cell state, and finally the output gate, which calculates the next hidden state. The updated cell state is updated by a bitwise multiplication of the current cell state and the forget cell output and a bitwise addition with the input state output. By the mechanism of the cell state, the network can carry streams of data from the beginning till the end of the network.
In our proposed architecture, we used 3 LSTM layers with tanh activations and dropouts in between them to avoid overfitting, followed by a network of 3 dense layers for classification

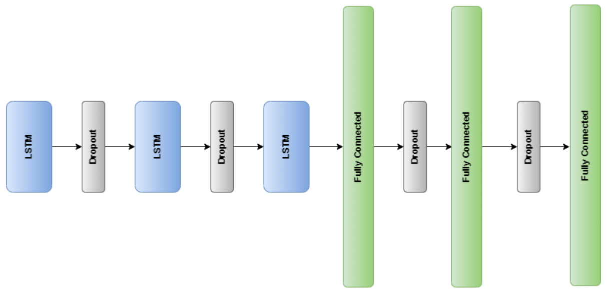

Figure 3. Model architecture

III. Optimizer and Loss function

For the optimizer, we chose the adam optimizer for its problem-independent, steady performance, in addition to its fast learning rate, which is shown in the model behavior and results in the following section. As for the loss function, we chose the categorical cross-entropy, which is famously used in multi-class classification problems, given how it regulates the loss with respect to the probabilities of the classes.

IV. Camera Detection Module

Following our preprocessing steps and training of our models, we started to design and implement a prototype for a camera detection module, in which we can test our model and approach in action. Capitalizing on mediapipe's quick and robust detections of key points, we were able to implement a module that is usable and accurate to a very high extent. In order to be able to generate predictions for a certain sign, our module detects key points for 30 consecutive frames, pauses for 500 milliseconds in order to indicate to the user that the 30 frames are over, and finally showcases the top 5 predictions to the user if the top prediction passed a 75% prediction confidence.

We opted for generating the top 5 predictions because some signs are similar to a great extent, resulting in the correct prediction being present within the first 5 predictions, rather than the very first prediction; therefore, we wanted to give the user the ability to choose the correct prediction among a pool of possible predictions.

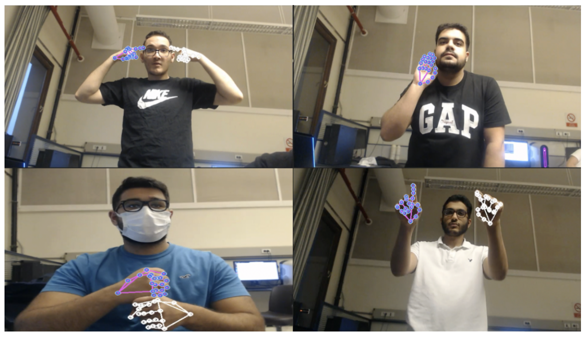

Figure 4. Camera Detection Module

2. Audio to Sign

Similar to the sign to text pipeline, there are 3 stages required to complete a smooth conversion from audio to sign: audio recognition, NLP, and mapping words to sign as shown in Figure 5. In this section, we will further discuss each stage in detail.

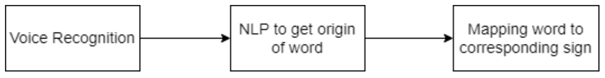

Figure 5. 3 stages to convert audio to sign

a. Voice recognition

The first step in this pipeline was to detect audio from a user via a microphone. Hence, we used the pyaudio python library in order to record the audio from the user after reading from a microphone. After retrieving the audio input, Google Speech API was used to convert the audio to text since it achieves state-of-the-art accuracy and capitalizes on Google's cutting-edge deep learning neural network algorithms for automatic speech recognition (ASR). Having converted the audio to text, we are ready for the next step: NLP.

b. NLP

This is the most essential phase in this second pipeline since it is the one responsible for getting the origin or root of each word fed from the first stage. Usually, words that people say have a certain degree of inflection meaning that they contain some sort of modification such as a prefix, suffix or infix. Thus, we made use of the python nltk package which is the Natural Language ToolKit package adopted for Natural Language Processing purposes such as tokenizing and eliminating inflection from words. Two of the techniques implemented to remove inflection are stemming and lemmatization with the latter being the one we eventually used.

Stemming is reducing inflection in words to their root forms or stem although the stem itself might not be part of the language. We experimented with two different types of stemmers for the English language: Porter and Lancaster Stemmers.

Porter Stemmer utilizes a technique called suffix stripping which is just removing the suffix from the end of the word such as the "s", "ing" or "ed". This means that the Porter Stemmer does not abide by linguistics; instead, it follows certain algorithmic rules to detect whether it is wise to remove the suffix or not rather than keeping a lookup table for the stems of words. Hence, it does not generate stems that are actually part of the English Language; yet, it is often used for its simplicity and speed.

The LancasterStemmer, also known as Paice-Husk stemmer, is a more aggressive and iterative approach with rules saved externally as opposed to the Porter Stemmer. Its algorithm consists of a table having around 120 rules indexed by the last letter of a suffix. Each rule either deletes or replaces the ending of the word and if no rule exists, the algorithm terminates. Another termination condition is if a word starts with a vowel and only two letters remain or if a word begins with a consonant and there are only three letters left. Otherwise, the iterations keep occurring and rules are repeated till a termination condition is satisfied. LancasterStemmer is a simplistic approach but the extensive number of iterations performed or over-stemming could lead to the stems not being linguistically correct. Since this is a problem encountered in both types of stemmers, we opted to test another technique which is lemmatization.

Lemmatization removes the inflection in words correctly whilst making sure that the origin of the word or lemma is linguistically accurate. For example, eats, eating, ate are all forms of the word eat, therefore eat is the lemma of all these words. Python NLTK provides WordNet Lemmatizer that uses the WordNet Database to find lemmas of words. Wordnet database is a lexical English database comprising around 117,000 nouns, verbs, adjectives, and adverbs. Words that carry the same meaning or form (string of letters) are grouped together. Moreover, the context for lemmatization to occur needs to be provided and this is achieved via the parts-of-speech parameter (POS). This indicates whether the word in question is a verb, noun, adjective or adverb. Thus, lemmatization provides context to the words in contrast to stemming while also associating words with the same meaning to one word. Since lemmatization retrieves a word that is part of the language, it is used where it is necessary to get valid words but it is more time consuming than stemming. The difference between the output of stemming and lemmatization is shown in Figure 6.

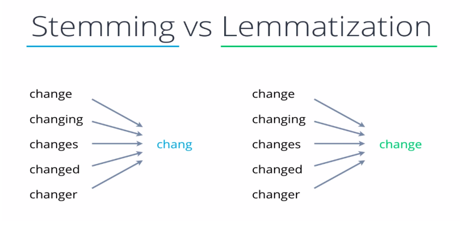

Figure 6. Stemming Vs Lemmatization

c. Mapping each word to a sign

After tokenizing the sentence into words and getting the root form of each word, we need to map performed by a unified signer of our choosing. If the word processing is part of our database, then its corresponding video is visualized as an output on the screen. However, if the word processed is not present as in the case of it being the name of a country or a person, then it is fingerspelled these words to their corresponding signs. We have created a database that consists of 100 words (verbs and nouns) in addition to the 26 English letters, where each gesture is played one after the other. An example of a fingerspelled word (Chris) is shown below in Figure 7.

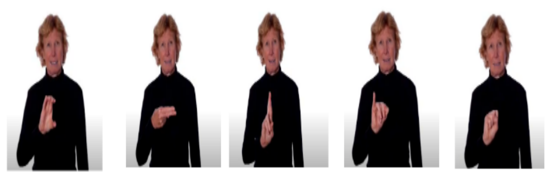

Figure 7. The word (Chris) fingerspelled

4. Experiments and Results

In order to have a better understanding of how our suggested model is performing with respect to our dataset, a series of experiments were conducted using various architectures that were trying to solve the same problem. Multiple approaches were showing auspicious results. Every approach used different datasets for training and testing which is a very crucial factor that was taken into consideration. This is because the different datasets could perform differently on different models and thus comes the need to unify the dataset used in training all the models to ensure that the results from all the models are fair. Moreover, to further ensure a fair comparison, all of the preprocessing steps of the input data was followed to ensure that the exact required data was being fed into the model. This included multiple aspects such as the type of augmenters used if any and the number of epochs. Several models that were showing promising results were not tested on our dataset because of the unavailability of the model. The criteria for testing all the models were specific to measure two very crucial points. The first is the accuracy of detecting sign language and the second is scalability. Scalability is very important because most of the literature tested on a small number of words which is not a very good indicator of how such models will perform on a large number of words. To perform those two tests, all of the models were tested on 100 words going through more than one stage. Every model was tested on 20, 40, 60, 80 then 100 words, and all were run a different number of epochs each based on the published paper of that model. Two models were used as a comparison, the initial was from The Australian National University [8] and implemented an I3D which was enhanced and improved from [11] to take into consideration Spatio-temporal information of signs. The second was a research from the FH Aachen University of Applied Sciences [9] which implemented a model consisting of a number of RNN layers and Mediapipe for key points collection of features needed. This model had three different variants where the number of RNN layers was changed and some of the hyperparameters of the model were changed.
The results are summarized in the following graph and table.

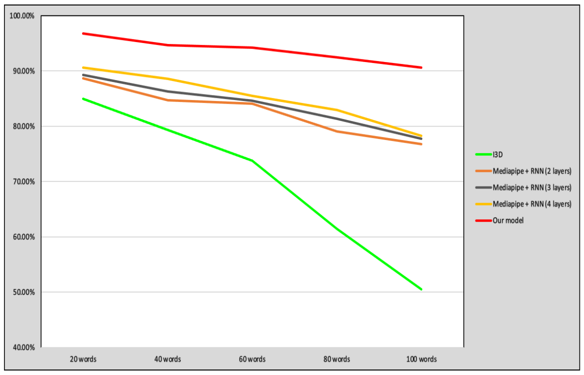

Figure 8. Models behavior with respect to number of classes

Model/Number of classes	20	40	60	80	100
I3D	85%	79.30%	73.80%	61.40%	50.50%
Mediapipe + RNN variant 1	88.70%	84.70%	84.05%	79.10%	76.80%
Mediapipe + RNN variant 2	89.30%	86.30%	84.60%	81.32%	77.70%
Mediapipe + RNN variant 3	90.60%	88.60%	85.50%	82.90%	78.24%
AUC	96.80%	94.67%	94.23%	92.50%	90.60%

Table 3. Models Accuracy Results

5. Limitations

The obstacles were related to data collection, camera orientation, and multiple word variations. First, data collection, the amount of data that was collected is enough for the recognition of the 100 classes. However, in order to satisfy the recognition of all the words in the ASL, scaling up the model is needed. Therefore, multiple sources of data must be added to training data. There are not enough sources in order to cover the whole language which is the main limitation for scaling up the model. Second, camera orientation, different cameras can capture frames with different resolutions; therefore, if the resolution is low, the key points detection algorithm may fail in detecting multiple points which will lead to a decrease in the model's performance. Finally, multiple word variation, there exists multiple word variations depending on the context of the word in the sentence. For example, the word "Right" can be understood as correct or as direction. In order to simplify this problem, we minimized our dataset in order to contain one variation for each word that has multiple variations.

6. Future Work

In the upcoming period of time, we will be aiming to:

1. Enhance our model further, either by manipulating the configuration of our current model or by increasing the videos per word in our dataset. We will be aiming to support the words in ASL, with the interest in making our model and data public for public use, in order to help researchers interested in the topic to further enhance our work for the community,
2. Work more and enhance our camera detection algorithm to make it more smooth and more usable,
3. Add a description to class names in order to handle multiple word variations.
4. Finally, Deploy the model on multiple systems that will have a great impact to the community

7. Conclusion

To conclude, after we discovered the problem in our dataset in the first pipeline, we decided to increase our dataset per class to a mean of 22.69 videos per class before pre-processing and 121.93 videos per class after preprocessing. We trained and tested our LSTM and mediapipe hand pose estimation model on 5 different stages: 20, 40, 60, 80, 100 words, all whilst achieving benchmark results that exceeded other models tackling the same problem as depicted in the results section. On the other hand, we showcased the methodology for the audio to sign translation, which is not our main contribution to the research, but it is crucial to complete the whole cycle of communication and bridge the gap between people with speaking and hearing disabilities and others.

8. References

[1] S.Masood, A.Srivastava, H.Thuwal, M.Ahmad. "Real-Time Sign Language Gesture (Word) Recognition from Video Sequences Using CNN and RNN"

[2] Liu, Tao, et al. "Sign Language Recognition with Long Short-Term Memory." *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, https://doi.org/10.1109/icip.2016.7532884.

[3] Elhagry, Ahmed Adel Gomaa, and Rawan Gla Elrayes. *Egyptian Sign Language Recognition Using CNN and LSTM*. https://arxiv.org/ftp/arxiv/papers/2107/2107.13647.pdf.

[4] Latif, Ghazanfar. Mohammad, Nazeeruddin. AlKhalaf, Roaa. AlKhalaf, Rawan. Alghazo, Jaafar and Majid A. Khan, " An Automatic Arabic Sign Language Recognition System based on Deep CNN: An Assistive System for the Deaf and Hard of Hearing" *International Journal of Computing and Digital Systems.* July 2020

[5] K. P.V.S.M.S., S. K.B.V.N.S., R. V.N.V.S., Prakash P. "Sign Language to Text Conversion Using Deep Learning". In: Ranganathan G., Chen J., Rocha Á. (eds) Inventive Communication and Computational Technologies. Lecture Notes in Networks and Systems, vol 145. Springer, Singapore, 2021. https://doi.org/10.1007/978-981-15-7345-3_18.

[6] M. Hurroo and M. Elham, "Sign Language Recognition System using Convolutional Neural Network and Computer Vision," in INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT), Volume-9 Issue 12, December 2020.

[7] http://dai.cs.rutgers.edu/dai/s/signbank

[8] D. Li, C. R. Opazo, X. Yu, and H. Li, "Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison," *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020.

[9] L. Antonio. Dom`enech, **"** Sign Language Recognition ASL Recognition with MediaPipe and Recurrent Neural Networks", *FH Aachen University of Applied Sciences*, 2020

[10] R. M. McGuire, J. Hernandez-Rebollar, T. Starner, V. Henderson, H. Brashear and D. S. Ross, "Towards a one-way American sign language translator," Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings., 2004, pp. 620-625, doi: 10.1109/AFGR.2004.1301602.

[11] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In CVPR, 2017. 5, 7