

# Real-time recognition of American sign language using long-short term memory neural network and hand detection

Reham Mohamed Abdulhamied<sup>1,2</sup>, Mona M. Nasr<sup>2</sup>, Sarah N. Abdulkader<sup>2</sup>

<sup>1</sup>Department of Software Engineering, Faculty of Computers Science, October University for Modern Science and Arts, Cairo, Egypt

<sup>2</sup>Faculty of Computers and Artificial Intelligence, Helwan University, Cairo, Egypt

## Article Info

### Article history:

Received Feb 23, 2022

Revised Dec 5, 2022

Accepted Dec 10, 2022

### Keywords:

Action detection

Hand gesture

LSTM model

MediaPipe

Sign language

## ABSTRACT

Sign language recognition is very important for deaf and mute people because it has many facilities for them, it converts hand gestures into text or speech. It also helps deaf and mute people to communicate and express mutual feelings. This paper's goal is to estimate sign language using action detection by predicting what action is being demonstrated at any given time without forcing the user to wear any external devices. We captured user signs with a webcam. For example; if we signed "thank you", it will take the entire set of frames for that action to determine what sign is being demonstrated. The long short-term memory (LSTM) model is used to produce a real-time sign language detection and prediction flow. We also applied dropout layers for both training and testing dataset to handle overfitting in deep learning models which made a good improvement for the final result accuracy. We achieved a 99.35% accuracy after training and implementing the model which allows the deaf and mute communicate more easily with society.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



## Corresponding Author:

Reham Mohamed Abdulhamied

Department of Software Engineering, Faculty of Computers Science

October University for Modern Science and Arts

Kairo, Egypt

Email: reham.abdulhamied@gmail.com

## 1. INTRODUCTION

The earliest forms of communication for the deaf and mute people provided by the use of sign language, which appeared in Spain during the 17th century to assist people who could not speak or hear. This is one of the languages in which each letter of the alphabet is generated by putting the hands on certain signs. Hand movements are only one component of sign language; other aspects include facial expressions, lip movements, and body movements.

Sign language assists deaf and mute people with special abilities in developing their mental, verbal, and sign skills, as well as the reduction of internal and psychological pressures experienced by the deaf and mute, and the development of social, cognitive, and cultural relations among deaf and mute people [1]. Several schools were established in a variety of countries around the world to help the deaf and mute to learn sign language, but it needs more work to be expanded, particularly in Arab societies. It is critical to integrate deaf and mute children with other students in schools, as well as to provide them with dedicated classes within schools, in order to provide them with a better social and educational life, as well as to raise their self-confidence levels.

There are over 300 different sign languages in use worldwide [2]. Different sign languages use various sign language alphabets. For instance, the alphabets of Indian sign language and Italian sign language are very different from those of American sign language (ASL) as shown in Figure 1. Thus, regional differences in sign

language exist. There are still over 150,000 spoken English terms without an ASL equivalent. Furthermore, there is no universal sign symbol for any name of a person, place, brand, or title [3].



Figure 1. English letters represented in hand signals

There are 5 methods to learn sign language: oral education, hand signals, allusion, finger spell, intonation and total contact [1], [4]. Many researches have attempted to represent sign language (SL) using hand-crafted elements. For example, local binary patterns (LBP) are employed in histogram of oriented gradients (HOG), and its extension HOG-3D is applied to hand and joint locations [5]. In recent years, deep convolutional neural networks have had a major influence on video-related activities, recurrent neural networks (RNNs), for example, have demonstrated substantial performance in learning the temporal relationships in sign spotting and human action identification, as well as gesture recognition and sign spotting. For continuous SL recognition, several recent techniques utilizing neural networks have been developed. However, neural networks can only learn frame-wise representations in these studies, for sequence learning, and hidden Markov models (HMMs) are used. However, the frame-by-frame labelling used for deep neural network training is noisy. Given their restricted representation power, HMMs may find it difficult to learn complicated dynamic variations [3].

Researchers have used a number of images capturing systems to gather sign pictures for categorization. The webcam, the data glove, the Kinect, and the leap motion controller all fall within this category. (Taskiran *et al.* [6] 2018 are among these devices). A camera or webcam is often the tool of choice for many scientists since it allows for more natural connection between people and computer without the need for extra equipment. Data glove-based data collection has been shown to be more accurate than data glove-based data acquisition. Because it is more expensive and cumbersome for customers, it is still a viable option. Using Kinect is a popular and efficient method of interaction. For the most part, it's capable of supporting a video stream with color and depth swiftly differentiates the backdrop from genuine sign picture and extracts 3D hand motion trajectory trajectories from the image. However, Kinect has a major drawback in that it costs a lot of money. The range of motion of the move controller is restricted. However, it is a less expensive and more accurate alternative to the Kinect [7].

From picture capture to classification, the challenges of creating sign language recognition are numerous. Researchers are currently trying to figure out the optimal approach for acquiring images. The difficulties of picture pre-processing arise when images are collected using a camera. Using an active sensor device, on the other hand, can be pricey. Researchers have various disadvantages while using classification algorithms. Due to the huge variety of recognition methods available, researchers are unable to focus on a single best strategy. Choosing one way to concentrate on leads to the testing of other methods that might be more suited for sign language recognition [8]. Researchers who experiment with a variety of approaches rarely fully establish one strategy. Defeating these obstacles could be the focus of future research in this topic. In the future, more research on this area may be focused on overcoming these challenges.

Acquiring signer independence is a challenging endeavor due to the fact that the manner in which a sign is articulated is susceptible to a great deal of interpersonal variation. Normalizing the features themselves will not solve this issue; rather, the categories level is where it has to be addressed. As a result, specific speech recognition adaption algorithms were developed and refined so that they could take into consideration the

peculiarities of sign languages. For rapid adaptation to new signers, the suggested recognition system employs a method that combines maximum likelihood linear regression and maximum a posteriori estimate [9].

In this paper, we'll see how to estimate sign language using action detection by applying a series of frames and predicting what action is being represented at that particular moment. For example, if we're using (thank you), it'll take the complete set of frames for that specific action to figure out what sign is being displayed. The long short-term memory (LSTM) model is used to create a real-time sign language detection and prediction flow. The ultimate goal is to create a real-time sign language detection stream with the help of the LSTM model.

The following is an outline for what will be included in the remaining sections of the paper: section 2 provides details about related work, section 3 has information on the applied system, which includes a study of the dataset, methodology, and evaluation metrics, and section 4 gives the conclusion.

Das *et al.* [10], a deep learning technique was developed for the classification algorithm of ASL using a camera and a depth sensor, Microsoft's Kinect is a good example of this. With the use of a simple webcam and a predetermined dataset, they were able to tackle this issue, if sufficient useful training material is provided, which can be done on a regular basis and included into the processing pipeline previously stated. If there is enough meaningful training data, this can be done on a regular basis and put into the processing pipeline that was previously defined. The training and validation accuracies of this convolutional neural network (CNN) architecture beat earlier models. The proposed CNN design resulted in a decreased training and validation loss total. The greatest recognition rate of their model is 94.34%.

Taskiran *et al.* [6], the CNN network was trained using a dataset gathered by Massey university's institute of information and mathematical sciences in 2011. The data collection contains, there were 900 images, with 25 samples for each of 36 characters (26 letters and 10 numerals). The photos were utilized as training data in 80% of the cases, and test data in 20% of the cases. The test data were chosen at random each time. After 20 training epochs, the CNN network was able to perform at its best in a test. The system was designed to function in real-time when network training was completed. Each of the 36 characters has been examined 10 times for real-time system control. Their real-time method obtained 98.05% on the test.

Saquib and Rahman [11], it has been developed that data gloves equipped with the appropriate sensors can recognize fingerspelling in both American sign language (ASL) and Bengali sign language (BSL). Even in resource-constrained settings, the approaches provided may be employed. The system was able to distinguish between static and dynamic alphabetic symbols with remarkable precision. Up to 96% accuracy was achieved by the method. Furthermore, A new approach of assessing continuous symbols from a stream of run-time data was given in this paper.

Aly *et al.* [12], based on depth pictures and principal component analysis network (PCANet) characteristics, this study provides a new efficient approach for user independent American fingerspelling identification. Hand segmentation is done on the depth image using a thresholding method. To enhance performance, the depth values of the segmented hand region are normalized. The PCANet deep learning architecture is used to learn hand shape features. When compared to color images, features extracted from depth photos can manage cross-user changes and image condition variations, resulting in promising outcomes. The suggested system is tested using a publicly available benchmark dataset obtained from a variety of users, and it achieves an average accuracy of 88.7%. The following Table 1 shows the accuracy of each method involved in sign language recognition:

Table 1. Comparison with the other related work

Reference	Type	Method	Accuracy
Das <i>et al.</i> [10]	Alphabets	Deep CNN	94.30%
Taskiran <i>et al.</i> [6]	Alphabets and Numbers	CNN	98.05%
Saquib and Rahman [11]	Alphabets	KNN	96.14%
		Random Forest	96.13%
		ANN	95.87%
		SVM	94.91%
Aly <i>et al.</i> [12]	Alphabets	SVM	88.70%

## 2. METHOD

In a range of applications, such as gesture recognition and augmented reality effects, our hand tracking technology may be used. A straightforward strategy for computing motions on top of the expected hand skeleton is described here. Each finger's state, such as whether it is bent or straight, is determined by the accumulated angles of joints, in the first place, before any other method is utilized. When a set of specified motions are assigned to each finger state, the mapping process is complete. When a set of preset motions are assigned to each finger state, a mapping is performed. Basic static gestures can be estimated with reasonable accuracy using this simple but effective technique. In addition to static gesture recognition, a series of

landmarks may be utilized to anticipate dynamic motions, which is an improvement over the current method. Another idea is to superimpose augmented reality features on front of the skeleton's bones. Hand-based augmented reality impacts are really popular right now.

### 2.1. Dataset

The dataset “American sign language” preserved by Neidle *et al.* [13], this data set produced with a hundred different signs. Three times, a single signer does the same sign under three different lighting situations and at three different signing speeds. Using three different lighting conditions and signing speeds, a single signer performs each sign three times. The videos were recorded on webcam. Each video was split down into pictures each frame and then reduced to 300 frames before being supplemented to enhance the data set for every sign to 2400 photos in total. As part of the augmentations, the photos were resized and rotated at random to create a more dynamic look. In addition, the data set was divided into two parts: a training data set with 1800 records and followed by a test data set with the rest of the records. The training data set had 1800 records and the test data set contained the remaining records.

We used Lana dataset scene2-front camera that contains 20 words. We've acquired a lot of data on all of our different important spots, so we'll be gathering data on our hands, bodies, and faces, after that, we'll save them as numpy arrays. After that, we'll design a deep neural network for classification using LSTM layers. Then we go ahead and predict that temporal component, so that we can predict the action from a number of frames—not just a single frame—as we have in the past, then we'll integrate it all together using opencv and actually predict in real time using a camera as shown in Figure 2.

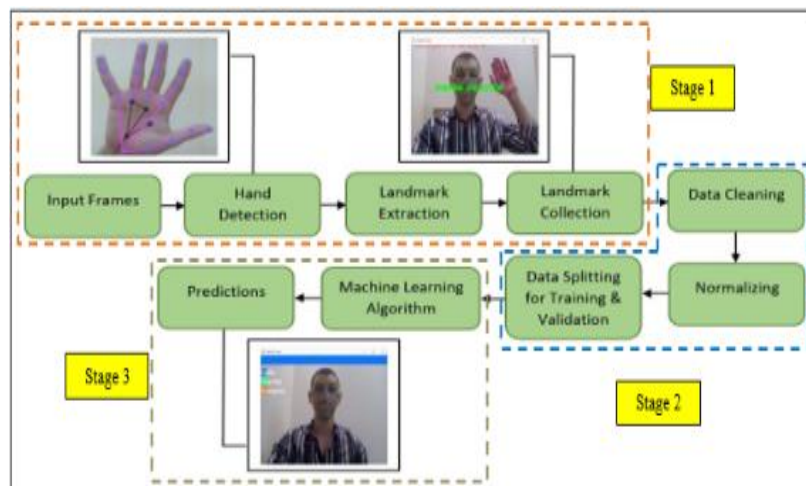


Figure 2. Proposed system architecture

## 2.2. Architecture

### 2.2.1. Stage 1: pre-processing of images to get multi-hand landmarks using MediaPipe

Developers may use MediaPipe to create cross-platform multi-modal experiences for their consumers using machine learning pipelines (data in the form of video, audio, or any time series) [14]. Google's big and diversified dataset has been used to train a large number of human body detection and tracking algorithms in MediaPipe. The proposed model used the skeleton of nodes and landmarks or edges to track significant places on different aspects of the human body. In three dimensions, all coordinate points are normalized. Graphs are utilized in Tensorflow lite models built by Google developers to make information flow more adaptable and configurable Li *et al.* [15]. Pipelines in MediaPipe are made up of nodes on a graph, which are often provided in the form of a ptxt file, this is a simple text file. These nodes are linked to C++ files. These files serve as the foundation for MediaPipe's base calculator class.

In this research, MediaPipe tool is utilized to identify hand movements, and the LSTM model is used to predict which signs and movements are being presented [1]. The MediaPipe tool is a machine learning technique algorithm that examines individual hands in a video stream. To detect the hand, MediaPipe uses two separate machine learning models. The image quality is adjusted to a normal 256\*256 size joint photographic experts group (JPEG) grayscale, and then a contour filter is applied after being initially preprocessed. When the second MediaPipe model is applied to the picture, it reduces it to a regular-sized bounding box after trying

to identify the palm of the hand if one can be found inside the image. Following the first identification of the bounding box, MediaPipe will need far less computer resources in order to find the hand inside the image in later rounds of the procedure. If the palm remembers where it was in the previous frame, it will be simpler to identify it in the succeeding frame, given that the palm does not shift much from the previous frame. After detecting the palm, MediaPipe uses its hand landmark detection model. As previously stated, to construct a collection of 21 hand landmarks, we start with a picture of the hand that is standardized. The coordinates of x and y, each point's z coordinate, which reflects relative depth, is normalized to the standard image size, which is an integer between 1 and -1. Observe the little blue dots in Figure 3.

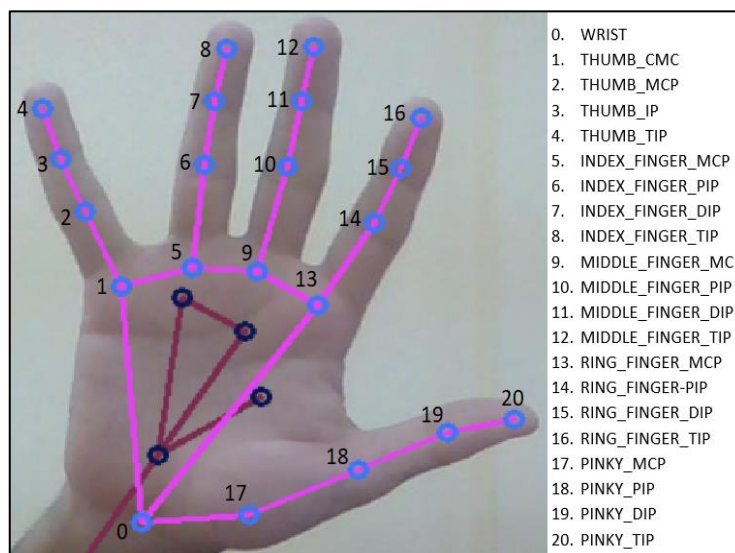


Figure 3. With a live video feed, a collection of linked hand landmarks is applied to the original hand picture

**a. MediaPipe integration**

A python framework has been constructed on top of MediaPipe to incorporate our MediaPipe, and to improve the MediaPipe Hand detection algorithm, we included hand landmark output. All of MediaPipe's inputs and outputs are predefined, and it runs every frame or at certain intervals. A hand tracking project in MediaPipe recognizes and tracks the movement of a hand based on the presence of 20 predefined landmarks on the recognized hand. For each landmark [16], a three-dimensional coordinate has been assigned that has been standardized to a defined range of possible values before being plotted on a map of that landmark. A new version of this hand tracking example program has been added that includes two more cases. First, we produced a customized dataset of comma separated values (CSV) files that contained hand landmarks contains a variety of hand movements, and then we compared it to that dataset to see how well it performed. The MediaPipe build in question will be referred to as MediaPipe trainer. An example hand picture utilized in training, as well as a set of hand landmark coordinates generated by MediaPipe, are shown in Figure 2. Using the MediaPipe tool as a module in our main program, we could detect video feed landmarks in real time, which would be useful in the second scenario.

**b. MediaPipe implementation**

Calculators are modular components that may be used to form a directed graph, which can be built using MediaPipe to create our hand tracking pipeline. When using MediaPipe, you get a variety of calculators that you can use to help with various tasks including model inference, media preparation, and data transformations throughout many multiple interfaces. These calculators may be expanded as needed. Several calculators are included with Mediapipe to help with activities such as model inference, data conversions and media processing across a broad variety of devices and platforms [5]. These calculators may be expanded as needed for example, TFLite graphical processing units (GPU) inference has been employed on the majority of modern smartphones. Figure 4 shows our MediaPipe graph for monitoring the movement of the hand. One subgraph is used for hand detection, while the other is used to compute landmarks.

The graph is partitioned into two subgraphs. When using MediaPipe, one of the most important features is that the palm detector only activates when it is needed (which isn't very often), this enables you save a significant amount of energy. All of this is performed by extracting the current video frame's hand position

from the previous frame's calculated hand landmarks [17], hence removing the requirement to employ the palm detector on each frame of the video stream in question.

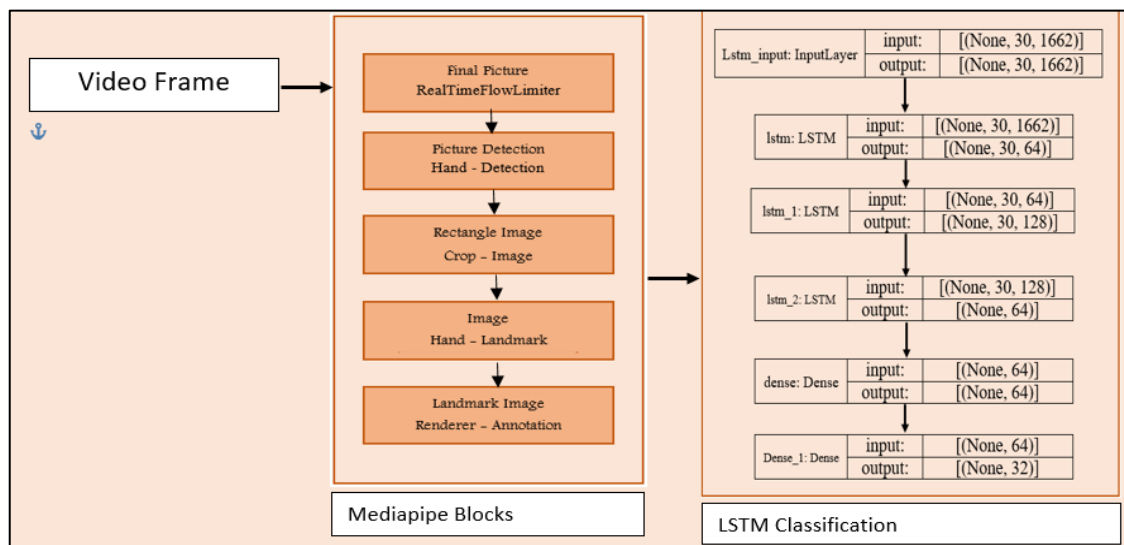


Figure 4. Proposed model architecture

### 2.2.2. Stage 2: data cleaning and normalization

Each image in the dataset is sent through stage 1 to collect all the data points under one file, where we are only considering the x and y coordinates from the detector [18]. The pandas' library function is then used to scrape this file and look for any entries that are null. Sometimes therefore, it is essential to eliminate these points to avoid bias while creating the prediction model. As a result, these issues must be cleared up otherwise bias will be introduced into the predictive model. Utilizing their indexes, rows with these null entries are looked for and deleted from the table. We normalized x and y coordinates after eliminating spots that weren't necessary for our system. The data file is then ready to be split into the training set and the validation set. 20% of the data is set aside for validating the model, while 80% of the data is kept for the training data of the model with various optimization and loss functions.

### 2.2.3. Stage 3: prediction using machine learning algorithm

Predictive analysis of different sign languages is performed using machine learning algorithms and LSTM algorithm. Hand pose estimation models are used to predict the hand joint positions in order to assist in sign recognition improvement. The objective is to classify as many data points correctly as possible at the real time to classify multiple classes of sign language words.

#### a. Background of RNN and LSTM

Different from traditional artificial neural network (ANN), recurrent neural network (RNN) additionally uses prior inputs to predict the output. It means that RNNs store and model the temporal sequence of prior inputs. Deep learning and machine learning are both aided by the usage of the LSTM architecture, which is an artificial RNN architecture. LSTM neural networks have feedback connections, as opposed to standard feed-forward neural networks [19]. Data points (such as images) can be processed individually, but entire sequences of data can be processed by the system. For example, LSTM is suitable to a variety of jobs. Intrusion detection systems perform tasks like unsegmented voice recognition, linked handwriting identification, and anomaly detection in network traffic.

Given the possibility of temporal delays of undetermined length between critical occurrences in a time series, it is crucial to consider the following [20]: LSTM networks are especially well-suited for time series data categorization, processing, and prediction because of their ability to learn from experience. When training typical RNNs, the LSTM technique was created in order to cope with the vanishing gradient issue that might arise. The benefit of LSTMs over RNNs is that they are generally unaffected by the length of the interval between two inputs. A one-of-a-kind memory unit was created by the LSTM, and three gates control how and when such information is updated and used: the input, output and forget gates. The LSTM also created a memory unit that

is unique in that it preserves historical information, meanwhile, three gates control the upgrading and application of that information [21]. Figure 5 depicts a tetrahedron in diagrammatic form. The following graphic shows a representation of an LSTM memory cell. In this paper, the composite function described below is used to implement the solution:

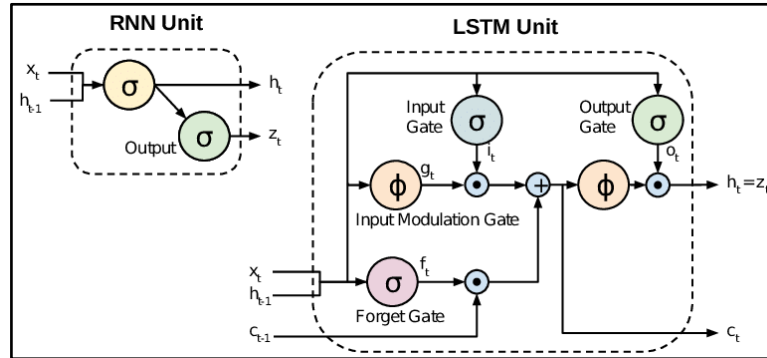


Figure 5. The typical structure of an LSTM block [11]

The following components are found in an LSTM unit: a cell, an input gate, an output gate, and a forget gate, among other things [22], Cell, input gate, output gate, and forget gate make up a typical LSTM unit, among other things. There are three gates, and they all limit how much information may enter and leave a cell, and the cell itself is capable of retaining values for an arbitrary amount of time, as explained before. We started with the input phase in the LSTM algorithm [11], where;  $X_t$ : is the current input,  $C_{t-1}$ : is the last unit memory and  $h_{t-1}$ : is the last unit output. Second the phase of forget gate which is represented by  $f_t$ . After that the update phase that state the new cell  $N_t$ . Lastly the output phase  $O_t$ . Algorithm 1 describe the LSTM phases implemented in this study.

**Algorithm 1. Long-short term memory algorithm**

```

The input phase:
 $X_t, C_{t-1}, h_{t-1}$ 
 $x_t$ : Current input;  $C_{t-1}$ : last unit memory;
 $h_{t-1}$ : last unit output

The phase of forget gate
 $C_{t-1}$ 
 $f_t = \text{Sigmoid}(W_{t_f}[h_{t-1}, X_t] + bias_f)$ 

The update phase: State of the new cell
 $i_t = \text{Sigmoid}(W_{t_i}[h_{t-1}, X_t] + bias_i)$ 
 $N_t = \text{Tanh}(W_{t_n}[h_{t-1}, X_t] + bias_n)$ 
 $C_t = C_{t-1}f_t + N_t i_t$ 

The output phase:
 $O_t = \text{Sigmoid}(W_{t_o}[h_{t-1}, X_t] + bias_o)$ 
 $Output_t = O_t \tanh(C_t)$ 

End
    
```

**2.3. Methodology (building and training LSTM model)**

In this proposed model we use Tensorflow [23] and Keras library [24] to import four main different things, the first is the sequential model to allow us build a sequential neural network. The next thing is the LSTM and a dense which is going to be our LSTM layer that gives us a temporal component for building our neural network and allows us to perform action detection and then we have passed through dense which is a normal fully connected layer. Also, we import Tensorboard which is a web app that's offered as part of the TensorFlow package that allows us to monitor our neural network training, to allow us to perform some logging inside of Tensorboard if we wanted to go and trace and monitor our model. The next thing is to create a log directory and set up our Tensorboard callbacks, and then we go and build our neural network architecture.

First up is that we are instantiating the model specifically the sequential application programming interface (API), then, we are adding three sets of LSTM layers then we have passed through 64 LSTM layers and a specified return underscore sequences equals true then we have specified the activation function, then, we

specified the input shape which is going to be 30 frames per prediction multiplied by 600 or 1662 values. After that we have created another two LSTM layers which is going to have 128 units specified return sequences equals true because we've got one more layer after it and then activation function then our next LSTM layer. LSTM is going to have 64 return sequences equals false (Hint: this line is going to have return sequences set to false because the next layer is a dense layer) and then activation function.

We specified that we want 64 dense units -or fully connected neural network neurons- then specified the activation function again and we add another dense layer, the final layer which is going to be our actions layer, then dense and then we specified effectively extracting function which are three neural network units and then we specified the activation equals softmax so this is going to return values that are within a probability of zero to one with the sum of all the values returned adding up to one.

After dividing the dataset into a validation dataset and a training dataset, the findings were as follows: the validation dataset used 20% of the data to fit the machine learning model, and the result was 99.3%. The training dataset utilised 80% of the data, and the results were 98.97%. After the installation of a new dropout layer in between the input (or visible layer) and the first hidden layer, the learning rate was scaled up by an order of magnitude, and the momentum was raised to 0.9. This was done in order to improve the accuracy of the model. The dropout rate was raised by 20%, which brought the total to 98.97%, and the dropout rate was set at 20%, which brought the total to 98.97%. Following the application of dropout to hidden neurons inside the model's main body. Dropout is then applied between the two hidden layers, as well as between the final hidden layer and the output layer. Once again, a 20% dropout rate is utilised as a weight limitation on those layers, and the output will be (98.87%). This is a significant step in the right direction, as it demonstrates that the model does well in both training and testing. As a result, with the help of dropout regularisation, we were able to solve the problem of overfitting in deep learning models.

#### 2.4. Evaluation metrics review

Because of the fact that LSTM can manage only 3D datasets, we utilized the Reshape function of the Numpy package to convert the values into 3D arrays. This was done because of the aforementioned reason. This completes the collection of inputs that our model requires. Our model has two LSTM layers, each consisting of 256 units, and the layer that comes after them is called a Dense layer. This is all discussed in the section where the model is created. The training accuracy, the training loss, the validation accuracy, and the validation loss were selected as the parameters to use in the analysis and evaluation of our model. Immediately after the conclusion of the training, we displayed the parameters of our model and then saved the model as a.h5 file. Epoch accuracy and epoch loss are the two plots that we generated for our training dataset; they are shown in Figure 6 and Figure 7, respectively.

```

model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))

```

Figure 6. The LSTM and dense layers model architecture

Now we are going to import a couple of metrics to evaluate the performance of our model [13]. As a result, we imported a multi-label confusion matrix, which will provide us with a confusion matrix for each of our labels. This helps us to determine what constitutes a genuine positive and true negative, as well as what constitutes a false positive and false negative.

To analyze results, we implemented a performance matrix to examine the data for each dataset, including accuracy, recall, precision, and F1 score [10]. Accuracy is defined as the proportion of correctly predicted data points among all the data points. The number of all accurate predictions regarding the ASLR can be calculated as the determines the total number of data set elements.

At this point, we are compiling a full categorization report based on the values that were anticipated. A traditional model may provide one of four distinct types of responses: a true negative (also known as TN), a true positive (also known as TP), a false negative (also known as FN), or a false positive (also known as FP) (false positive). TP is the case when both the real and predicted cases are true, TN is the case when both the real and predicted cases are false, FN is the case when the real case is positive but the predicted case is negative, and FP is the case when the real case is negative but the predicted case is positive. TN stands for the situation in which both the real case and the predicted case are negative. The ability of a classifier to avoid incorrectly labelling



instances as positive when they really are negative is referred to as precision. We used performance matrices like as accuracy, recall, precision, and F1 score in order to conduct research on the outcomes of each of the datasets [10]. The amount of accurately anticipated data points out of all the data points is referred to as accuracy. Address space layout randomization (ASLR) can be calculated as the number of all correct predictions to calculate the overall number of elements in the data set.

At this point, we are compiling a full categorization report based on the values that were anticipated. A traditional model may provide one of four distinct types of responses: a true negative (also known as TN), a true positive (also known as TP), a false negative (also known as FN), or a false positive (also known as FP) (False Positive). TP is the case when both the real and predicted cases are true, TN is the case when both the real and predicted cases are false, FN is the case when the real case is positive but the predicted case is negative, and FP is the case when the real case is negative but the predicted case is positive. TN stands for the situation in which both the real case and the predicted case are negative. Precision is the capacity of the classifier to avoid labelling an instance as positive when it is, in fact, negative.

$$ASLR = \frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

$$PSLR = \frac{TP}{TP+FP} \tag{2}$$

$$RSLR = \frac{TP}{TP+FN} \tag{3}$$

Peak sidelobe ratio (PSLR) is a measure of how accurate a model is in terms of how many of the predicted positives turn out to be positive in practice. PSLR is a great statistic to employ if there is a large cost associated with a false positive result. Relative sea level rise (RSLR) calculates how much of the real positives are captured by our model based on how many of them are marked as positive [22]. When there is a substantial cost associated with false negatives, we will use RSLR as the model metric. As shown in (2) and (3), respectively, represent the mathematical formulations of precision and recall [25].

The F-measure method, which is referenced in (4), offers a method to integrate accuracy and recall into a single measure that takes into consideration both of these features. It corrects inequalities in categorization that may arise. The confusion matrix was also analyzed in order to have a better understanding of the many errors that are produced by the classifier. In the key to the confusion matrix, a number of correct and incorrect guesses are presented, along with count values and a classification breakdown for each.

$$FSLR = \frac{2 \times P \times R}{P+R} \tag{4}$$

After running for about 2000 epochs and testing the model by two users, we've already got a categorical accuracy of 0.9935 as described in Figure 7 and our epoch loss shown in Figure 8 could potentially go a little bit lower. The more epochs we train the more accuracy and lower loss we get. Here are some screenshots from the proposed model as shown in Figure 9. The features are extracted from the first three pictures, and then the predicted result is shown in the last picture.

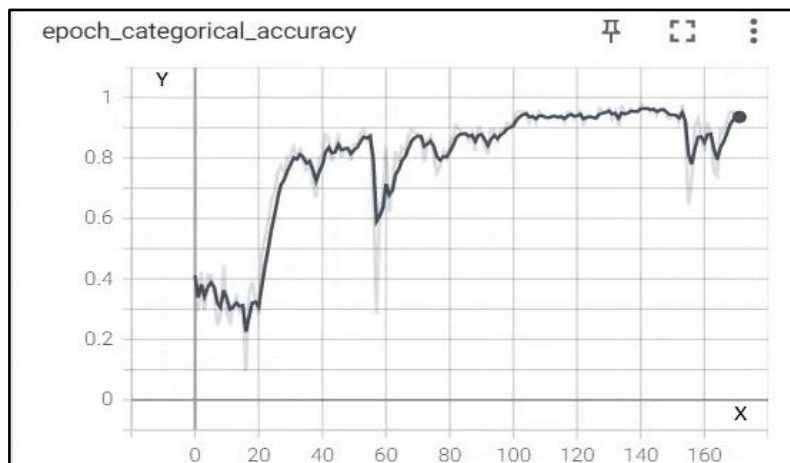


Figure 7. Epoch accuracy for model train

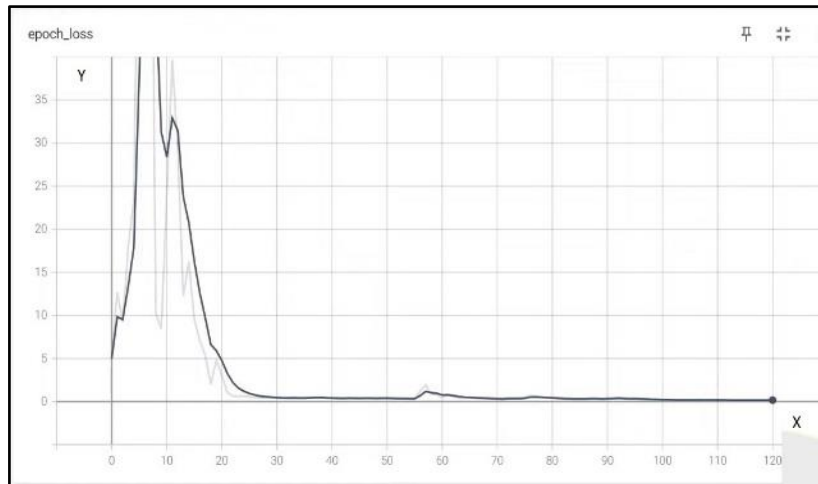


Figure 8. Epoch loss for model train



Figure 9. Real-time American sign language recognition

### 3. CONCLUSION




The recognition of sign language became one of the recent challenges that getting the attention now days. In this paper we consider a recognition system using real time video frames instead of extracting frames and hand features from a video and filtering them and combine all the frames in one video to recognize the word. The recognition with the video frames using LSTM model reduced the required data, the detection was faster, and the model needs less time to train. The final result accuracy we got was about 99.35% more accurate than related work. One of the most difficult aspects of developing a sign language recognition system is dealing with the signer's background and the contrast in light. To facilitate sign capturing, we've developed a minimalistic background, the user started testing the model with three different signs, the model started with

the collection stage to identify language with real-time restriction. As a result, we're working on real-time video-based sign language identification with real-time restrictions, such as non-uniform backdrops, changeable illumination, and a system independent of the signer itself. In order to recognise objects (segmentation), gradient masking is used. It's done because of the contrast between the object and the background. Using different segmentation operators to get the thresholds. Using threshold values and the edge, you may create a binary mask of the image. When applying binary gradient masking, vertical structuring elements are commonly used.




## REFERENCES

- [1] S. Sharma, K. Kumar, and N. Singh, "Deep eigen space based ASL recognition system," *IETE Journal of Research*, vol. 68, no. 5, pp. 3798–3808, Sep. 2022, doi: 10.1080/03772063.2020.1780164.
- [2] W. Liu, Y. Fan, and Z. ZhongZhang, "RGBD video based human hand trajectory tracking and gesture recognition system," *Mathematical Problems in Engineering*, vol. 2015, 2014, doi: 10.1155/2015/863732.
- [3] D. Uebersax, J. Gall, M. Van den Bergh, and L. Van Gool, "Real-time sign language letter and word recognition from depth data," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Nov. 2011, pp. 383–390, doi: 10.1109/ICCVW.2011.6130267.
- [4] R. Sharma, R. Khapra, and N. Dahiya, "Sign language gesture recognition," *International Journal of Recent Research Aspects*, vol. 7, no. 2, pp. 14–19, 2020.
- [5] A. Halder and A. Tayade, "Real-time vernacular sign language recognition using mediapipe and machine learning," *International Journal of Research Publication and Reviews*, no. 2, pp. 9–17, 2021.
- [6] M. Taskiran, M. Killioglu, and N. Kahraman, "A real-time system for recognition of american sign language by using deep learning," in *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, Jul. 2018, pp. 1–5, doi: 10.1109/TSP.2018.8441304.
- [7] Y. Bin, Z.-M. Chen, X.-S. Wei, X. Chen, C. Gao, and N. Sang, "Structure-aware human pose estimation with graph convolutional networks," *Pattern Recognition*, vol. 106, p. 107410, Oct. 2020, doi: 10.1016/j.patcog.2020.107410.
- [8] B. M. Wilson, "Evaluating and Improving the SEU reliability of artificial neural networks implemented in SRAM-based FPGAs with TMR," Thesis, Brigham Young University, 2020.
- [9] U. von Agris, J. Zieren, U. Canzler, B. Bauer, and K.-F. Kraiss, "Recent developments in visual sign language recognition," *Universal Access in the Information Society*, vol. 6, no. 4, pp. 323–362, Feb. 2008, doi: 10.1007/s10209-007-0104-x.
- [10] P. Das, T. Ahmed, and M. F. Ali, "Static hand gesture recognition for american sign language using deep convolutional neural network," in *2020 IEEE Region 10 Symposium (TENSYP)*, 2020, pp. 1762–1765, doi: 10.1109/TENSYP50017.2020.9230772.
- [11] N. Saquib and A. Rahman, "Application of machine learning techniques for real-time sign language detection using wearable sensors," in *Proceedings of the 11th ACM Multimedia Systems Conference*, May 2020, pp. 178–189, doi: 10.1145/3339825.3391869.
- [12] W. Aly, S. Aly, and S. Almotairi, "User-independent american sign language alphabet recognition based on depth image and PCANet features," *IEEE Access*, vol. 7, pp. 123138–123150, 2019, doi: 10.1109/ACCESS.2019.2938829.
- [13] C. Neidle, A. Thangali and S. Sclaroff, "Challenges in development of the American sign language lexicon video dataset (ASLLVD) Corpus," *5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon, LREC. 2012*, May 27, 2012.
- [14] F. Zhang *et al.*, "MediaPipe Hands: on-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, Jun. 2020, [Online]. Available: <http://arxiv.org/abs/2006.10214>.
- [15] Y. Li, X. Chen, X. Zhang, K. Wang, and Z. J. Wang, "A sign-component-based framework for Chinese sign language recognition using accelerometer and sEMG Data," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 10, pp. 2695–2704, Oct. 2012, doi: 10.1109/TBME.2012.2190734.
- [16] Z. J. Liang, S. Bin Liao, and B. Z. Hu, "3D convolutional neural networks for dynamic sign language recognition," *Computer Journal*, vol. 61, no. 11, pp. 1725–1736, 2018, doi: 10.1093/comjnl/bxy049.
- [17] O. Koller, S. Zargarani, H. Ney, and R. Bowden, "Deep sign: hybrid CNN-HMM for continuous sign language recognition," in *British Machine Vision Conference 2016, BMVC 2016*, 2016, vol. 2016-September, pp. 136.1-136.12, doi: 10.5244/C.30.136.
- [18] G. Li *et al.*, "Hand gesture recognition based on convolution neural network," *Cluster Computing*, vol. 22, no. S2, pp. 2719–2729, Mar. 2019, doi: 10.1007/s10586-017-1435-x.
- [19] W. Chen *et al.*, "A survey on hand pose estimation with wearable sensors and computer-vision-based methods," *Sensors*, vol. 20, no. 4, p. 1074, Feb. 2020, doi: 10.3390/s20041074.
- [20] D. Avola, M. Bernardi, L. Cinque, G. L. Foresti, and C. Massaroni, "Exploiting recurrent neural networks and leap motion controller for the recognition of sign language and semaphoric hand gestures," *IEEE Transactions on Multimedia*, vol. 21, no. 1, pp. 234–245, Jan. 2019, doi: 10.1109/TMM.2018.2856094.
- [21] F. Gomez-Donoso, S. Orts-Escolano, and M. Cazorla, "Accurate and efficient 3D hand pose regression for robot hand teleoperation using a monocular RGB camera," *Expert Systems with Applications*, vol. 136, pp. 327–337, Dec. 2019, doi: 10.1016/j.eswa.2019.06.055.
- [22] A. A. Hosain, P. S. Santhalingam, P. Pathak, H. Rangwala, and J. Kosecka, "FineHand: learning hand shapes for American sign language recognition," in *Proceedings - 2020 15th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2020*, Nov. 2020, pp. 700–707, doi: 10.1109/FG47880.2020.00062.
- [23] J. V. Dillon *et al.*, "TensorFlow distributions," *arXiv preprint arXiv:1711.10604*, 2017.
- [24] T. B. Arnold, "kerasR: R interface to the keras deep learning library," *The Journal of Open Source Software*, vol. 2, no. 14, p. 296, Jun. 2017, doi: 10.21105/joss.00296.
- [25] B. Bagby, D. Gray, R. Hughes, Z. Langford, and R. Stonner, "Simplifying sign language detection for smart home devices using Google MediaPipe," *Bradenbagby*, 2021.




**BIOGRAPHIES OF AUTHORS**

**Reham Mohamed Abdulhamied**    is Teaching Assistant in Software Engineering Department, Faculty of Computer Sciences at MSA University, Giza, Egypt. She Holds a BSc. degree in Software Engineering. Her research areas are artificial intelligence, machine learning, deep learning and data engineering. She can be contacted at email: reham.abdulhamied@gmail.com.



**Mona M. Nasr**    is Professor in Information Systems department at Faculty of Computers & Artificial Intelligence, Helwan University. She Holds a PhD. degree in Information Systems. Her research areas are e-Learning, Big Data, Data Science, Cloud Computing and Knowledge Discovery. Prof. Mona held many positions such as (Chief Information Officer (CIO) for Helwan University HU, Scientific Computing Center Manager at HU, Head of Information Systems Department at Faculty of Computers and Artificial Intelligence HU, Vice Dean for Community Service and Environmental Development at FCAI-HU). She can be contacted at email: m.nasr@helwan.edu.eg.



**Sarah N. Abdulkader**    is an Assistant Professor at Helwan University, graduated in 2005, obtained her Master's degree in 2011 for a research in the field of Vehicular Ad-hoc Networks, became a member of HCI Lab in 2013. She obtained her PhD degree in 2015. Her PhD research work focuses on the exploration and incorporating of brain signals in security systems. It uses the non-invasively collected brain waves for claimed-identity authentication. Her current research interests includes HCI, pattern recognition and machine learning. She can be contacted at email: nabil.sarah@gmail.com.