Virtual Assistance in Deep Learning

Muhammad Amjad Bin Abdul Malik
B. Eng. Electronic Enginering
Hochschule Hamm-Lippstadt
Lippstadt, Germamy
muhammad-amjad-bin.abdul-malik@hshl.stud.de

***Abstract*—This essay discusses a model of virtual assistance for mute and auditory people who are unable to use or benefit from traditional virtual assistance, which primarily employs audio as the primary channel. Since it is so difficult for hearing-impaired communities to communicate, dynamic sign language was created. In dynamic sign language, vocabulary is represented by hand and body gestures. To accomplish our goal of creating the solution model, we have used important frameworks and tools like MediaPipe Holistic with LSTM Deep Learning. All of the important details and steps are will be explained throughout this paper.**

I. Introduction

An application software called Virtual assistant which also known as Artificial Intelligent (AI) or digital assistant is an application program that can comprehend voice commands and human language. One can say that it actually imitates the jobs and works formerly performed by a personal assistant or a secretary. This is because, with the information and data collected from human commands, it can carry out the task for its user such as making phone calls, reading aloud an email or some text messages, doing some dictations, reminding the user of an upcoming meeting or appointment and the list goes on [1].

Virtual assistants primarily need an aid of the internet connectivity in order to operate with their functionalities since they are cloud-based software programs. These software programs include 'Google Assistant' on Android devices, 'Cortana' or 'Braina' on Microsoft devices and 'Siri' on Apple devices. Though in the meantime, we can use some features on 'Siri' and 'Braina' offline without connecting to the internet to do some simple tasks. Basically, we can encounter and utilize all the stated examples of virtual assistant software programs or applications in our daily life by just discovering them through our mobile phones, tablets, laptops or even the devices connected to them like our headphones. The virtual assistant features may help and assist us partly, aside from just using the devices solely for their main purpose [1].

On the other hand, there are also devices that are dedicated to providing us the actual virtual assistance experience with the features really embedded inside the devices which make them a very important part or essence in home automation [1]. The ones that are available from Google and Amazon are currently the most popular devices out there. For example, we can just activate them by calling or saying the word, "okay google" or "Alexa" to initiate the dictation. The device will then be ready to receive a command when there appears a light signal on the device. Some simple commands can be depicted like "Play lo-fi music." or a simple question like "What is the weather like today?" Those requests are then stored and processed in the cloud.

There is without a doubt that these application programs provide a lot of ease to human lives by reducing our time and increasing our daily productivity. Therefore, we can see that a lot of virtual assistant devices have been part of our lives. But does virtual assistant really give benefits to all of us regardless of our inabilities? If not, what are the solutions? These questions will be elaborated on and explained in this paper starting with the problem statement, and proposed solution, followed by the methodology of the approach used, the results and finally the conclusion.

II. Problem Statement

While the virtual assistant is undeniably convenient to a lot of us, an argument came into the debate when we talk about those who are auditory or verbally impaired. Would the features in virtual assistant be practical and give any benefits to them as it does to normal people? Personal assistant devices are getting more and more common due to rising technological trends. However, every existing Virtual Assistant in today's date is found to be voice automated, which needs an audio input, thereby making it unusable by Deaf-mutes and people with certain disabilities [2]. Unlike any other handicap disabilities, deaf-mutes have a variety of effects on one's ability to communicate. Thus, we can see that there is a big communication gap when they try to express their thoughts through speech and understand other normal people through listening.

A survey by the World Federation of the Deaf (WFD) found that 328 million adults and 32 million children worldwide or more than 5% of the total population have a hearing impairment. In this situation, sign language might come in handy by reducing the communication gap between deaf-mutes and regular people, facilitating regular interaction. It also makes communication become much more effortless and simplified. In sign language, one's feelings can be expressed by hand gestures, body movements, facial expressions, and emotions [3].

Identify applicable funding agency here. If none, delete this.

However, this problem might become more critical in the education sector or at the workplace. As we all know, sign language also differs by region, hence it might be challenging to distinguish between them all. For example, American Sign Language (ASL) is not the same as other sign languages like British Sign Language (BSL) or any other sign language. To simplify, sign language can be seen as a normal language where there are a lot of differences between different languages. Continuing with our earlier points, there can be scenarios even in the special school for impaired students, where the teacher is not that proficient and hence can hamper the students' education. The same issue can be seen in the workplace where the employer or the colleague could not understand the workers who are deaf-mutes. Apart from that, there have been times when deaf-mutes were struggling to communicate with first responders in an emergency. Two communities only depend and rely on human-based translators can be inconvenient and costly. These events show how crucial it is to be solved with an immediate effect.

III. Proposed Solution

The events mentioned in the section before show how crucial they are to be solved with an immediate effect. A system with an interface that can assist people with speaking or listening difficulties is therefore required to bridge the communication gap between the deaf-mutes and the other. This paper focuses on research that gives an idea of creating a Real-Time Hand Gesture Recognition System that can detect and recognize hand gestures and movements in real-time. Hand gesture recognition is a technique that leverages neural networks and algorithms by analyzing hand movements and then identifying their patterns of action [6].

Hand gestures can add another dimension of communication between humans and computers and the other way around [2], thus making it possible for people with hearing or speaking difficulties to utilize the available digital technology and also communicate with the outside world better without needing any help from a human-based interpreter. In other words, it is an alternative way for someone who has hearing or speaking inabilities to interact with others through a machine or computer by just simply using hand gestures [6], especially when they are communicating with someone who has limited or zero knowledge about sign language since hand gestures and hand movements can be quite confusing to understand [4].

Therefore, it doesn't matter if someone lacks their own voice or is unable to talk clearly. That is where this initiative shines. These individuals can readily communicate with these gadgets through an interface that accepts hand gestures as input and outputs text for others to understand them. This initiative has the potential to close the gap between those with disabilities and rapidly developing technology. Not only that, but it might also help deaf-mutes to equally enjoy their social and personal life. Designing such an interface will make them find their freedom while using such technologies and might boost their confidence in this digital age [2]. With the development of Artificial Intelligence which includes adopting technologies from deep learning and machine learning techniques, the objective of overcoming the communication barrier and simplification the integration process between people with deaf-mutes disabilities and normal people can be achieved.

IV. Methodology Of Approach Used

There are several approaches to reaching our goal of making a Real-Time Hand Gesture Recognition system. Hand gestures basically can be divided into 2 categories which are static and dynamic gestures. A static gesture is a single image that represents a specific shape and pose of the hand, meanwhile dynamic gesture is a moving gesture, represented by a series of images. The chosen approach and strategy in this paper help build a dynamic hand gesture recognition [3], which then may develop a system that can predict and display the description of the sign language performed by the user when employing hand gestures through the webcam [4]. The technique used or applied incorporates a framework called MediaPipe Holistic Solution from Google for extracting hand landmarks and an artificial neural network called Long Short-Term Memory (LSTM) to train and recognize the gestures.

By creating our own dataset using Google's Mediapipe Holistic Solution and utilizing the Open Source Computer Vision (OpenCV) library to detect the landmarks from the hand in real-time detection, it helps us eliminate the once major challenge faced in realizing the Hand Gesture Recognition System, which was the non-uniform background and segmentation of hands from the background. Therefore, the system will accurately identify landmarks from one's hands regardless of whether they are in their car, house, or on the streets. The designed system consists of three primary components that are sequentially coupled. The pre-processing module, where the data is processed to input into the final LSTM module, where the training for gesture recognition is done, comes after the dataset module when the landmarks are extracted, and the dataset construction process is completed. A camera renders the live real-time feed and sends it through the model, and the name of the gesture is displayed as text on the screen once the model is generating satisfactory results after refining the hyperparameters [6]. "Fig. 1" shows the architecture of the proposed Hand Gesture Recognition System.

A. Hand Detection Using Holistic Mediapipe Framework

The dynamic sign language (DSL) that we are going to work on is made up of a series of actions that are rapid and remarkably identical. Correspondingly, spotting the hands and identifying their orientation and shape is quite challenging for DSL. It was at this point that MediaPipe started to aid in fixing these issues [mediapipe]. Pose, hand, and face are the three parts of the mediapipe's holistic pipeline, which as a whole offers a wide range of solutions. A total of 543 distinct landmarks are extracted using the holistic approach which are 468 landmarks for face, 33 landmarks for pose

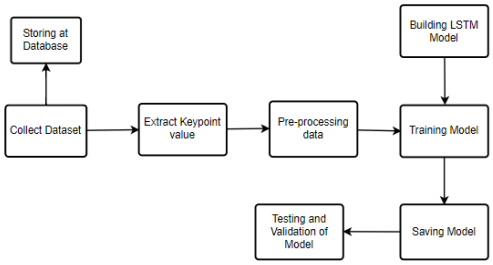

Fig. 1. Architecture of the proposed solution model.

and 21 landmarks for hand [6]. It uses Machine Learning to understand those 3D local landmarks from just one frame.

A frame with an open source and cross-platform is called Mediapipe. Other pre-trained machine learning solutions are included, including those for object detection, hand detection, location measurement, and face detection. In our case study, it is an effective method for detecting the hand and fingers [4]. As seen in "Fig. 3", it extracts the key points for both hands' three dimensions (X, Y, and Z) and estimates postures for each frame in "Fig. 2".

Fig. 2. Sequence of frames collected for our dataset [3].

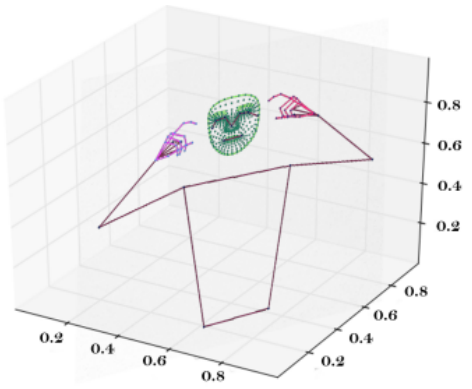

Fig. 3. Keypoints of hands and body in 3D space [3].

The hand placement in regard to the body was predicted and tracked using the pose estimation technique. The set of keypoints for hands and stance estimation is produced using the MediaPipe framework. The keypoints, X, Y, and Z for both hands are computed in the three-dimensional space. "Fig. 4" illustrates how MediaPipe retrieves 21 keypoints for each hand. As a result, the number of hands' keypoints that were extracted is determined as follows with 'KP' indicating the keypoints:

$$\text{KP in Hands} \ * \ 2 \ * \ 3\text{D} \ = (21 * 3 * 2) = 126 \ \text{KP} \quad (1)$$

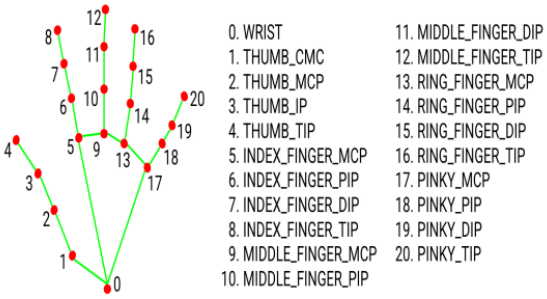

Fig. 4. The order and labels for keypoints that exist in the hands [3].

33 keypoints are extracted by MediaPipe for pose estimation, as seen in "Fig. 5". The three dimensions of X, Y, and Z as well as visibility are determined in this space. A value indicating whether a point is visible or concealed (occluded by another body part) on a frame is called visibility. As a result, the number of keypoints recovered from the posture estimation is determined as follows:

$$\text{KP in Pose} \ * \ 3\text{D + Visibility} \ = 33 * (3+1) = 132 \ \text{KP} \quad (2)$$

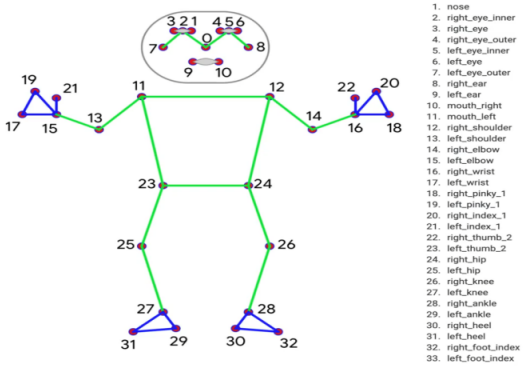

Fig. 5. The order and labels for keypoints that exist in pose [3].

For the face parts, MediaPipe extracts 468 keypoints. Contours around the face, eyes, lips, and brows are represented by lines connecting landmarks, while the 468 landmarks are represented by dots. They are calculated in the three-dimension space: X, Y, and Z. Thus, the number of extracted keypoints from the face is calculated as follows:

$$\text{KP in Face} \ * \ 3\text{D} \ = (468 * 3) = 1404 \ \text{KP} \quad (3)$$

To sum up, the total number of keypoints including in hands, pose and face for each frame are 1662 keypoints. To extract the crucial points for each frame of the video, this technique

is done throughout. In each video in the Dataset, the position of the hand, shoulders, and face are identified together with an evaluation of their form and orientation. Some of the solutions offered by the MediaPipe framework that we used in this paper for processing time-series data are Face Mesh, Hands, and Pose [3].

We created a function called Mediapipe detection that accepts two arguments. Firstly, the image being detected and secondly, the Mediapipe model we are using for detection which is also known as the 'holistic model'. The function changes the image's Blue, Green, Red (BGR) to Red, Green, Blue (RGB) format before passing it to a function called model.process() and storing the outcome. The function then returns the image and the result that was previously stored after converting the image back to BGR format. We define another function (draw_styled_landmarks), which accepts the returned output from the previous functions as its arguments. With the help of this function, we may see real-time hand detection by generating landmarks [6].

The X, Y, and Z coordinates of the key points are then extracted from the detection results following the successful completion of real-time hand detection. To execute the coordinate extraction, we write another function called extract keypoints, which uses the detection results as its parameter. The concatenated array of all the arrays containing the key point coordinates of the holistic model is what this function delivers. The data collection process calls this function. Even though we employed the holistic model and identified important details for the hand, face, and pose, we only displayed certain landmarks on the screen during the recognition phase in order to prevent clumps of landmarks on the screen [6]. An example of this is shown in "Fig. 6".

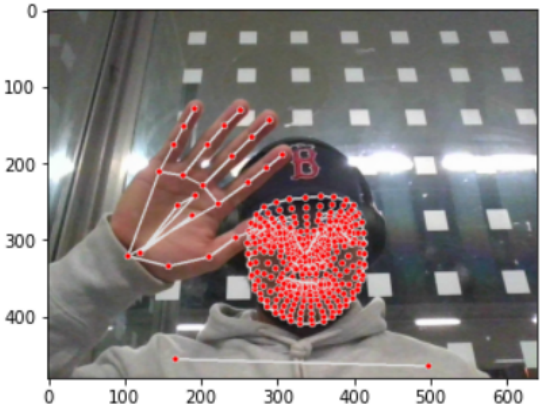

Fig. 6. Example of the recognition with landmarks.

Not to forget, the planned system also requires an 'eye' which can be an external camera, or in my case, I only used a webcam from my laptop to collect real-time photographs of hand gestures that can be transmitted for identification and classification in both training and testing. The testing process can be done right after we finished collecting data by capturing videos that produce the frames and completed with the training of datasets which will be explained later in this paper. Deep Learning models for visualization image processing basically benefit a lot from OpenCV's sophistication [2].

OpenCV is a huge open-source library for Computer Vision, Machine Learning and image processing. In our system, considering that there are two channels in a picture (the RGB and BGR channels), OpenCV uses BGR image format. So, when we read an image using cv2. imread(), it interprets in BGR format by default. We then use cvtColor() function to convert a BGR image to RGB, being that the image processing library we used, which is Numpy Library has RGB pixel orderings. In other words, we need to change the color channel order from BGR to RGB because Numpy Library is used for the numerical computation of pixel matrices representing the images in OpenCV [2].

B. Data Collection

Next, we move on to the collection of data. The first step in Data Collection is to establish folders to store the dataset that we are going to create and collect. To create a folder for the dataset, the os.path.join method is used. After we have gathered our own dataset for the training and testing of our deep learning model, the next step is to define an array of the gestures that we have chosen. For example, an array of 3 gestures, ('Hello', 'Thanks' and 'Sorry') is defined in order to train our model to recognize these gestures. We specify the total number of videos which is 30 videos for each action and each video consist of 30 frames. This means that the total number of frames that we are going to have is 2700. The next phase is data gathering in folders after the gesture folder has been created. The 30 video folders for each of the 3 movements are then created by running a loop. Afterward, the extract_keypoints() function is applied to gather the array of key points for each frame after collecting the video data for each gesture. Finally, each frame inside each video folder receives a copy of the NumPy array of key information that was gathered. The key points collected will be indicated as (.npy) files [6].

C. Deep learning (LSTM)

Huge amounts of data are needed for virtual assistant technologies, which are then supplied to platforms for machine learning, deep learning, speech recognition, and other types of artificial intelligence (AI). A virtual assistant's AI programming makes use of sophisticated algorithms to learn from data input and get better at guessing the users' needs as they interact with the assistant. Therefore, we have also included and implemented Deep Learning in our approach to the solution.

Deep Learning is essentially a subsection of the Machine Learning paradigm, primarily comprising the approach of multilayer neural networks. Neural networks are frequently used by deep learning to accomplish their goals. A neural network is composed of layers that alter input to a certain degree in order to produce the output [2]. Deep Learning algorithms can be divided into many categories such as Sequence to Sequence

(Seq2Seq), Multi-Layer Perceptrons (MLP), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and others. In our solution, a type of RNN architecture which is LSTM has been chosen to be implemented.

A sort of RNN architecture called long short-term memory (LSTM), is capable of keeping values at any intermission. Mainly used for a sequence of data or data in time series [6], they were created for the processing, categorization and prediction of time series with specific time delays and uncertain lengths. LSTMs have relative intensity gaps, which give them an advantage over competing sequence learning models like hidden Markov models and other RNNs [5]. It has gates that regulate the data exchange from a certain unit. The loop, which has the ability to store the previous history of information in the internal state memory is an essential part of the LSTM design, aiming to extract both temporal and spatial information from data [6].

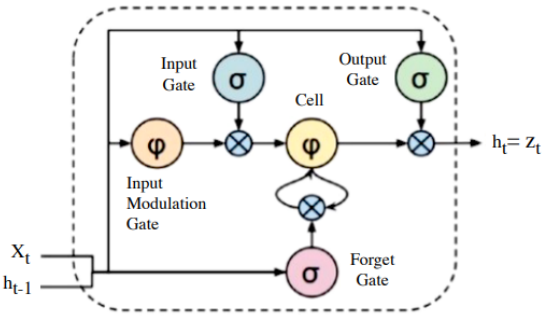

Fig. 7. LSTM cell [5].

The revolving arrow on the LSTM, which denotes its recursive nature, is known as the state of the cell. As a consequence of that, the cell state has the data from the preceding epoch. It is modified by a remember vector situated beneath the cell state and is then adjusted by input modulation gates. The forget gate is the standard name for this remembers vector [5].

As seen in "Fig. 7", an LSTM model's architecture can be described as consisting of three gates and a memory cell module. First off, forget gate eliminates useless information from the cell state. Whatever knowledge could be disregarded and is no longer contextually relevant is regulated by the forget gate. The function of the input gate is to append significant data to the state of the cell. It is closely related and concerned with which new information we should include or update in our operational storage state data. The output gate's job is to extract important data from the current state of the cell and display it as output [6].

Whatever portion of data is currently held in the current state should be provided as the output in a specific instance. Short-term memory and long-term memory, which is a self-state, are the two states that the memory cell can be in. Based on how each gate operates, values between 0 and 1 are assigned to each gate. Information is passed based on the values between 0 and 1, where 1 means all information is passed and 0 denotes that no information is passed. A single LSTM module containing four interconnected layers is shown in "Fig. 8" [6].

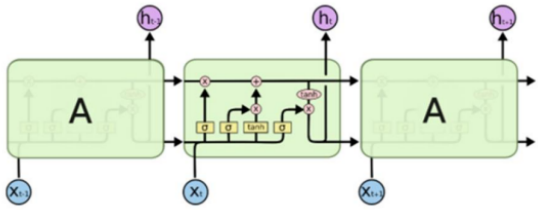

Fig. 8. LSTM module [6].

Our data consists of a collection of frames that have been sequentially saved with information about the locations of numerous landmarks that have been retrieved. Those frames are each saved as a NumPy array. The sequential model is used to construct the LSTM network. Followed by three layers of LSTM, there are three dense layers. For the first five layers, the "ReLu" activation function is utilized, and for the final dense layer, "softmax" is used. In order to deal with stochastic gradient descent, the optimizer "ADAM" is employed. After every epoch, the model's training and validation accuracy and loss are measured for evaluation by using Tensorboard from TensorFlow [6].

Because the ReLU activation function is the most often employed activation function in modern neural networks and because of its computational simplicity, it is chosen to be applied in this project. This function returns zero for negative inputs and returns its value of itself for positive inputs. Despite this, activation functions like Sigmoid, Tanh, and others are also implemented [6]. The summary of the model is shown in "Fig. 9".

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            442112

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 3)                 99

=================================================================
Total params: 596,675
Trainable params: 596,675
Non-trainable params: 0
```

Fig. 9. Mediapipe cite [1].

1) Justification of Tools Used: The LSTM model and the MediaPipe holistic were used primarily because they required less data to generate a hyper-accurate model, fewer parameters to increase speed, and a simpler model to enable simple motion recognition in real-time [5]. On top of that, LSTM is a recurrent network design that can train to span time gaps more than steps even in the situation of noisy, incompressible

input sequences, without sacrificing the ability to respond to minor time lags. To read values from a hand gesture, the nodes are taken advantage of. Every one of these nodes corresponds to the LSTM and detects the precise value of the gestures to produce the output [4].

2) Building and Training of LSTM Neural Networks: Importing important dependencies, such as the sequential models, LSTM layer, and dense layer, was the initial step. The setting up of a log and a Tensor Board callback for tracking neural network training came right after that. The chosen optimizer to be utilized and the loss functions (categorical cross entropy) for the multiclass classification model were then specified, and the models were then assembled and fitted. The measures were designed to track accuracy over the course of training. The model was then prepared for training and fitting. By defining the X-train and Y-train data as well as the number of epochs and callbacks, this was accomplished [5].

The model was then started on its training, and the Tensor board was used to track the training accuracy. The training may either be halted when the desired training accuracy was reached or until the required number of epochs had been finished. The ".h5" model was used to evaluate the predictions made after each projected action's probability had been determined in advance [5].

D. Tensorflow

A machine learning framework called TensorFlow incorporates information to develop deep learning models, which aids in forecasting and improving future outcomes. The primary benefit of utilizing TensorFlow is its ability to provide encapsulation, which enables the developer to concentrate on logical development by allowing them to leave the supervision of the model's tiny details to the library.

The nicest thing about utilizing the TensorFlow library is that it is a fully accessible library with a large number of models that have already been created. Deep Learning, in particular, benefits greatly from the TensorFlow library. Acknowledging the definition of two concepts is necessary to comprehend the theoretical application of Tensor Flow. Tensor in this context is an N-Dimensional Array, and Flow denotes a graph of operations. In TensorFlow, any mathematical operations are viewed as a graph of operations, where the nodes are operations and the edges are simply tensors.

In our solution, TensorFlow aids in model training, given the supplied dataset. When applied in conjunction with OpenCV, TensorFlow methods in object detection assist in the classification and identification of various hand gestures. TensorFlow can assist in categorizing and recognizing real-time hand gestures by analyzing hundreds of pictures. It enables the development of a model that can recognize 3D images, categorize them using 2D photos and analyze their pattern from its feed dataset [implementation]. Furthermore, we can utilize the TensorBoard features from TensorFlow to analyze the result of our training model which has been mentioned before in the previous section. However, we will go through the demonstration in the next section of Results and Discussion.

V. Results and Discussions

In this section, we will be discussing the results of the solution proposed starting from the result of our dataset training which is displayed by the tool TensorBoard, testing and evaluation of data by experimenting with some predictions, evaluation of data by using confusing matrix and lastly, the final test in real-time. All of these processes should be ideally done to see and monitor how well is the model actually performing.

The graph in "Fig. 10" demonstrates that even though training and testing accuracy both rise with an increase in epochs, the curves exhibit a somewhat erratic trend. The graph shows that the training and testing accuracy peaked at about the 40th iteration, while the training and testing precision peaked at about the 180th iteration. In whatever range of iteration testing, neither curve appears to demonstrate steady performance.

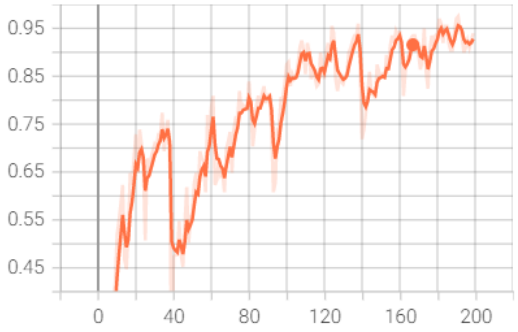

Fig. 10. Graph showing the accuracy of training.

"Fig. 11" shows the epoch loss. The graph's epoch is plotted on the X-axis, whereas the loss is plotted on the Y-axis. The loss in training is represented by the dark orange curve, whereas the loss in testing is represented by the light orange curve.

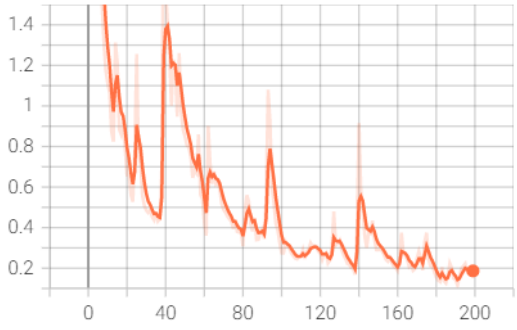

Fig. 11. Graph showing the epoch loss.

Even though loss during training and testing is generally decreasing, the trend is quite erratic, with sporadic abrupt rises and drops all throughout the curve. The dataset becomes more unpredictable as an outcome. A deeper analysis of these results was carried out using firstly, the simple prediction method

and subsequently using a more reliable method which is the confusion matrix.

The original dataset, which was split into training and testing data, served as the source for the dataset that was utilized for testing. Accuracy is the evaluation metric we employ. So what we are doing is basically comparing the data between the predicted data from 'Test Data' and the 'Train Data'. As we can see from "Fig. 12", both predicted data lead to the same action which is 'hello'. From this, we can say that the LSTM model is effectively operating. However, this is only a simple test of accuracy.

```
In [82]: res = model.predict(X_test)
         1/1 [==============================] - 0s 30ms/step

In [96]: actions[np.argmax(res[3])]
Out[96]: 'hello'

In [97]: actions[np.argmax(y_test[3])]
Out[97]: 'hello'
```

Fig. 12. Evaluation of data using simple prediction method.

Next, we can also evaluate our data using the Confusion Matrix for more precise results of our model's performance. In this process, we import a couple of metrics from Scikit Learn tool to evaluate the performance of our model. Basically, a multi-label confusion matrix is going to give us a confusion matrix for each one of our different labels and this allows us to evaluate what's being detected as a true positive and a true negative and what's been detected as a false positive and a false negative. From "Fig. 13", we can observe that, with regard to the test dataset, an accuracy of around 94 percent was reached.

```
In [68]: from sklearn.metrics import multilabel_confusion_matrix, accuracy_score

In [69]: yhat = model.predict(X_train)
         3/3 [==============================] - 0s 28ms/step

In [70]: ytrue = np.argmax(y_train, axis=1).tolist()
         yhat = np.argmax(yhat, axis=1).tolist()

In [71]: multilabel_confusion_matrix(ytrue, yhat)
Out[71]: array([[[55,  3],
                 [ 2, 25]],

                [[56,  0],
                 [ 0, 29]],

                [[54,  2],
                 [ 3, 26]]], dtype=int64)

In [72]: accuracy_score(ytrue, yhat)
Out[72]: 0.9411764705882353
```

Fig. 13. Evaluation of data using Confusion Matrix.

Finally, a real-time test is demonstrated to realize the application of the methods of our solution which we already went through from the beginning of the paper. A webcam is used to detect hand gestures and if there are any gestures that are recognized by our system, it will print out the meaning behind the gestures onto the screen and a text-based gesture recognition will be displayed via live feed. "Fig. 14" shows the final result of the demonstration. As we can observe, the system still can manage to recognize the gesture performed correctly even when the user wears a face mask and a cap.

Fig. 14. Testing in real-time.

VI. Conclusion

Throughout this paper, the system methods and solutions have been explained step by step with a demonstration of the results included. Despite the simple model system that I have done which only consists of 3 hand gestures to be recognized, I hope that the reader can understand the content of the paper and grasp the idea of how the system works. Besides, the reader can also apply them as a stepping stone in implementing some of the ideas in more deeper and advanced systems that most likely require more complicated techniques. I suppose that future studies should go in the direction of concentrating on dynamic sign sentences in real-time video with real-life scenarios. Moreover, expanding the dataset and building a preparation algorithm for datasets with various video lengths.

References

[1] Botelho, B. (2017, October 31). What is a Virtual Assistant (AI assistant)? - definition from whatis.com. SearchCustomerExperience. Retrieved October 29, 2022, from https://www.techtarget.com/searchcustomerexperience/definition/virtual-assistant-AI-assistant

[2] Someshwar, D., Bhanushali, D., Chaudhari, V., Nadkarni, S. (2020). Implementation of virtual assistant with sign language using deep learning and tensorflow. 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA).

[3] Samaan, Gerges H., et al. "MediaPipe's Landmarks with RNN for Dynamic Sign Language Recognition." Electronics, vol. 11, no. 19, 8 Oct. 2022, p. 3228.

[4] Kate, R., Brahmabhatt, P., Dhopte, S., Tushar, T. (2022). Hand Gesture Recognition System Using Holistic Mediapipe, 09(International Research Journal of Engineering and Technology (IRJET)).

[5] Dhulipala, S., Adedoyin, F. F., Bruno, A. (2022). Sign and Human Action Detection Using Deep Learning. Journal of Imaging, 8(7), 192.

[6] Agrawal, S., Chakraborty, A., Rajalakshmi, M. (2022). Real-Time Hand Gesture Recognition System Using MediaPipe and LSTM. International Journal of Research Publication and Reviews, Vol 3, 2509–2515.