mathematics

MDPI

Article

Deep Learning in Sign Language Recognition: A Hybrid Approach for the Recognition of Static and Dynamic Signs

Ahmed Mateen Buttar [1], Usama Ahmad [1], Abdu H. Gumaei [2,*], Adel Assiri [3], Muhammad Azeem Akbar [4,*] and Bader Fahad Alkhamees [5]

[1] Department of Computer Science, University of Agriculture Faisalabad, Faisalabad 38000, Pakistan; ahmedmatin@hotmail.com (A.M.B.); usamaahmad.cui@gmail.com (U.A.)
[2] Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
[3] Management Information Systems Department, College of Business, King Khalid University, Abha 61421, Saudi Arabia; adaseri@kku.edu.sa
[4] Software Engineering Department, LUT University, 15210 Lahti, Finland
[5] Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia; balkhamees@ksu.edu.sa
* Correspondence: a.gumaei@psau.edu.sa (A.H.G.); azeem.akbar@lut.fi (M.A.A.)

Abstract: A speech impairment limits a person's capacity for oral and auditory communication. A great improvement in communication between the deaf and the general public would be represented by a real-time sign language detector. This work proposes a deep learning-based algorithm that can identify words from a person's gestures and detect them. There have been many studies on this topic, but the development of static and dynamic sign language recognition models is still a challenging area of research. The difficulty is in obtaining an appropriate model that addresses the challenges of continuous signs that are independent of the signer. Different signers' speeds, durations, and many other factors make it challenging to create a model with high accuracy and continuity. For the accurate and effective recognition of signs, this study uses two different deep learning-based approaches. We create a real-time American Sign Language detector using the skeleton model, which reliably categorizes continuous signs in sign language in most cases using a deep learning approach. In the second deep learning approach, we create a sign language detector for static signs using YOLOv6. This application is very helpful for sign language users and learners to practice sign language in real time. After training both algorithms separately for static and continuous signs, we create a single algorithm using a hybrid approach. The proposed model, consisting of LSTM with MediaPipe holistic landmarks, achieves around 92% accuracy for different continuous signs, and the YOLOv6 model achieves 96% accuracy over different static signs. Throughout this study, we determine which approach is best for sequential movement detection and for the classification of different signs according to sign language and shows remarkable accuracy in real time.

Keywords: You Only Look Once (YOLO); Long Short-Term Memory (LSTM); deep learning; confusion matrix; convolutional neural network (CNN); MediaPipe holistic

MSC: 68T07; 68T45; 68Q09

Citation: Buttar, A.M.; Ahmad, U.; Gumaei, A.H.; Assiri, A.; Akbar, M.A.; Alkhamees, B.F. Deep Learning in Sign Language Recognition: A Hybrid Approach for the Recognition of Static and Dynamic Signs. *Mathematics* **2023**, *11*, 3729. https://doi.org/10.3390/math11173729

Academic Editors: Viacheslav Voronin, Yigang Cen and Evgenii Semenishchev

Received: 15 July 2023
Revised: 17 August 2023
Accepted: 24 August 2023
Published: 30 August 2023

1. Introduction

Only a few people, such as the instructors and family members of deaf individuals, can interpret sign language. Informal hints and regular gestures in a specific pattern are two types of sign language. A deaf person can communicate their thoughts via the use of a specific hand gesture and esoteric facial expressions, rather than by using words (as opposed to body language). An intentional hint with the same structure as the community's spoken language is a formal gesture.

Humans require efficient communication as a critical component to prevent and address problems. The most prevalent cause of improper communication is a lack of information about the language used among the parties involved, which is a crucial factor. This problem has persisted for a long time because members of active speech impairment (hearing- and speech-impaired) groups must interact with other individuals in society. Although gestures can be used to address this issue, they are frequently ineffective and take a long period of time to communicate a straightforward message. People from the hearing- and speech-impaired groups find this inconvenient because it occurs frequently and in a variety of different sectors. People might, for instance, be unable to send communications to the authorities. Other times, they might be impeded despite possessing all of the required skills and abilities; they may be prevented from engaging in some economic pursuits such as agriculture due to their poor interpersonal communication skills.

Sign language recognition techniques based on sensors are not the best option, according to one of the major findings of the aforementioned study. This is a result of the comparatively expensive and complicated hardware arrangements that they require. This explains why vision-based models have been preferred by the majority of researchers. There have been many studies on this topic, but it is clear that sign language recognition model development is still a challenging area of study. The greatest challenge is in identifying an appropriate model to address the challenges of continuous signs that are independent of the signer. Different signers' speeds, durations, and many other factors make it challenging to create a model with high accuracy and continuity.

There are various applications for the recognition of sign language, which is one of the hot topics in artificial intelligence and data science. Sign language detectors using human action recognition are a top priority because they can aid the active speech-impaired community in communicating with the general public. Sign language detectors are also designed based on different sensor technologies. For example, ref. [1] used a sensor-based glove for sign classification from the gestures of signers and presented the results in a meaningful phrase. For continuous sign language recognition, one can utilize an LSTM model [2]. For sign sentence recognition, the authors employed a four-gated LSTM cell coupled with a 2D CNN architecture. They also gave each sentence its own label. A deep learning approach based on gesture can be used to derive real-time text from a video feed while also detecting and monitoring sign language [3]. Framing, image preprocessing, and feature extraction are all steps in the system architecture, being performed according to hand positions and movement, etc.

To recognize sign language, human–computer interaction techniques typically employ either conduct-based or vision-based systems. In the first scenario, the signer frequently uses a human–computer interface (HCI) tool to capture their actions, and they are processed by a unique algorithm embedded in the system to generate the desired output. The sensor glove is an excellent example of sign language recognition based on behavior. Vision-based systems, on the other hand, can capture data from a signer even if they are not in close proximity to the human–computer interface hardware [4]. An automated camera, such as a webcam on a computer, records their actions. Once the data have been processed by the necessary algorithm to generate the desired output format, the results are displayed by the suitable output device. The use of deep learning to improve the recognition of human activity is a good example of the aforementioned strategy.

The goal of this study is to create an accurate sign language classifier that is capable of recognizing ASL signs in both their static and moving configurations. LSTM with a skeleton and a YOLOv6 model are trained on a custom dataset to create a final hybrid model. The proposed approach aims to attain the following objectives:

- To increase sign detection's accuracy;
- To detect static and dynamic signs sequentially in real time;
- To create an efficient method to detect signs in sign language using a hybrid approach from videos and display them with their meanings and confidence levels.

The YOLO and long short-term memory models were built, trained, and evaluated with the use of a multi-class confusion matrix. The training and testing accuracy of the YOLOv6 model were 96.00% and 92.00%, respectively, for static signs. On other hand, the LSTM model achieved 92.00% and 88.00%, respectively, for continuous signs. We fill a research gap by providing a holistic solution to sign language recognition, encompassing the entire spectrum of sign types. We believe that this approach not only enhances the accuracy but also broadens the applicability of sign language recognition systems.

2. Background

The interpretation and communication of American Sign Language (ASL) involves not merely a sequence of gestures but an intricate interplay of visual, spatial, and linguistic elements. As technology advances, so does our ability to decode and replicate this form of communication. This background section offers an extensive review of seminal works and evolving methodologies aimed at understanding and automating ASL recognition. From foundational datasets to the latest neural network architectures, the literature captures the journey of researchers and technologists as they unravel the nuances of ASL and its potential interfaces with human–computer interaction. We aim to trace the trajectory of ASL's intersection with technology.

The world has progressed to a point at which it can understand American Sign Language, including its visual and spatial components. ASL has its own phonology, morphology, and grammar, as with any other natural language. Signers use their hands to communicate. They achieve this by producing semantic information through the use of arm gestures, hand gestures, expressions of the face, and movements of the body and head in order to convey words as well as feelings. This is why human action recognition is the best approach for continuous sign and object detectors for static sign detection [5].

Golestani et al. [6] provided a model for the general public to access the American Sign Language Lexicon Video dataset, which includes video clips of more than 3300 ASL signals and several synchronized video clips presenting the different signs of sign language in different environments and from different angles. Gloss labels with the start and end times for signs, labels for both the beginning and end of hand forms, and the categorization of sign types according to morphology and articulation are a few examples of the linguistic annotations. Each morpheme for a compound sign has an annotation in the collection. The dataset also includes several video sequences, a camera from different angle sequences, different software for skin feature extraction, and numeric ID labels for different signs in order to teach computer vision-based sign language categorization. The author demonstrated the challenges associated with language annotation and classification. They also gave an example of a computer vision application that used the ASLLVD.

Gunji et al. [7] proposed a deep learning technique in which frames are annotated with the form and orientation of the right hand based on the annotation of the right hand's position in the single signer setup's training phase level using 50 $\times$ 70-pixel right hand photos that have been cropped. With the aim of automatically identifying shared hand shapes between signals, this annotation enables the evaluation of clustering methods in the context of sign language identification. To categorize the image into one of the ASL alphabets, PCA features that are taken from the image are used. The output of ASL gesture recognition is text, which can subsequently be translated into speech. The program enables people who have hearing and speech impairments to communicate through computers. Due to the videos' low temporal resolution and the pace at which people sign, several frames exhibited motion blur effects and were therefore disregarded for hand shape and orientation annotation. Because the hand traveled more than five pixels in Euclidean distance between 31,640 of the 46,282 video frames in the training set, they were automatically removed.

Tolentino et al. [8] used a method of determining and categorizing major activities using a person's hands, arms, face, and occasionally the head, which are recognized as gestures. Its relevance in the planning of intellectually sophisticated human–computer

interfaces for various presentations makes this an important area of study. They range from sign language to medical analysis to actual assurance. As a result of the rapid expansion of computer knowledge, human–computer communication technologies are steadily becoming the most popularly studied research topic. Additionally, the significance of signs emphasizes their evaluation in the area of research into human–computer communication. A signal is the most common, innate, and informal human–computer communication resource; it is intensively researched and used. One of the topics that is now being extensively researched is gesture detection based on visualization. One of the most common ways to exchange information is through gestures, used as a kind of signal. In order to improve cooperation between the deaf and those without impairments, communication by gestures acknowledgment (SLR) aims to establish a strong and reliable mechanism to record gesture-based communication into words or speech. SLR, one of the critical areas of human–computer interaction (HCI) research, is of increased interest in an HCI society.

Jiang et al. [9] adopted Adam as an approach that uses first-order gradients and adaptive predictions of lower-order moments to optimize stochastic objective functions. The technique is easy to apply, efficient in terms of computing, memory-efficient, and effective for situations requiring a large number of parameters or data, and it is invariant to the diagonal resizing of the gradients. This approach can be used to resolve problems with gradients that are exceedingly noisy and/or sparse, as well as non-stationary goals. In most cases, the hyperparameters may be tweaked to a reasonable degree and have intuitive meanings. Adam was built upon a number of linkages to related algorithms that are shown. The authors also discuss the characteristics of the theoretical convergence of the algorithm and produce a rate of convergence bound that is superior to the most impressive outcomes attained by the online framework for convex optimization. Evidence from empirical studies demonstrates that Adam outperforms other stochastic optimization approaches in practical applications. Finally, they discuss the Adam variant Jiang et al. [9], which is based on the infinity norm. Moreover, color images were filtered to lessen digitally induced noise. In addition, morphological operations were applied to the binary image that resulted from the skin segmentation process in order to better highlight the features. When using the FCM technique, they were able to increase their word recognition from 25% to 75% across a set of 40 ISL words.

Bantupalli and Xie [10] analyzed an image of a hand showing the sign alphabet, which was used as the system's input. At a specified moment in time, the camera captures the image. The picture is then stored in a disc. The test image and the pre-saved sign letter images are then loaded. The histograms of all the photos are produced once they are converted from the BGR color space to the HSV color space. Then, the Bhattacharyya distance—a metric to determine how similar the histograms are—is computed and compared. A search for the equivalent English alphabet is then conducted on the image whose histogram is closer to that of the test image or has a lower Bhattacharyya value. After this, the output is generated as the alphabet.

Kothadiya et al. [11] used the skeleton-based graph technique to construct a multi-model-based sign language recognition system to detect isolated signs. The author proposed a SAM-SLR framework as a means of isolating indicators, and then used the SL-GCN and SSTCN models to produce key points and extract features from the skeleton. This allowed the author to successfully identify the isolated indicators. The AULTS dataset was used to evaluate the proposed approach.

Iyer et al. [12] suggested a neural network architecture to derive real-time text from a video feed while also detecting and monitoring sign language. Framing, image preprocessing, and feature extraction are all steps in the system architecture. According to hand positions and movement, etc., the hand's point of interest (POI) is used to depict these hand characteristics. With this method, the author was able to extract 55 distinct features, which were then used as inputs to the proposed convolutional neural network-based architecture developed for symptom prediction. This step was taken in advance of the research.

The recognition and interpretation of American Sign Language (ASL) is undeniably complex, given the multifaceted components, ranging from visual–spatial elements to distinct phonology and grammar. The rich field of ASL recognition has been connected with technological advances over the years, as gleaned from an extensive literature review. Golestani et al. [6] illuminated the challenges inherent in language annotation and their role in computer vision applications. In a progressive manner, Gunji et al. [7] introduced deep learning techniques, tackling specific challenges such as motion blur in frames. As Tolentino et al. [8] indicated, gestures form an essential facet of human–computer interaction, with growing significance in diverse realms. Jiang et al. [9] introduction of the Adam approach testifies to the constant pursuit of optimization, while Arora and Bantupalli et al. [10] focused on histogram-based image recognition. The more recent works Kothadiya et al. [11] and Neidle et al. [3] move towards skeleton-based techniques and real-time text extraction, respectively. The breadth of the literature showcases the ongoing and multidimensional evolution of ASL recognition. However, a more in-depth investigation into this chronological array of information is crucial to discern patterns, draw parallels, and inductively analyze the direction in which the field is moving. It is evident that while the knowledge pool is vast, synthesizing these findings into cohesive insights is the next pivotal step.

3. Research Methodology

The detection of different signs in sign language, such as "hello", "bye", "welcome", "thank you", "I love you", and "I am fine", and the alphabets belonging to ASL are included in this study. According to the state of the art, we develop a machine learning model to detect static as well as dynamic signs. Only taking the inputs via a webcam, without using any other type of sensor, means that our detector is a touch-less technology that uses two different models to detect the signs [13]. A video-based custom dataset is used. The raw videos related to these different signs are collected from YouTube videos and a webcam by using OpenCV under different background environments, and we perform different preprocessing techniques to obtain the required form of data. The dataset consists of different classes of these signs, and videos related to each sign class are separately stored in a folder. We generate the frame data, also stored in the same folder of the specific sign class in a sequence, because each frame belongs to the previous one and the next [14]. After the collection of data, we perform different preprocessing steps to obtain the required outputs, as shown in Figure 1.

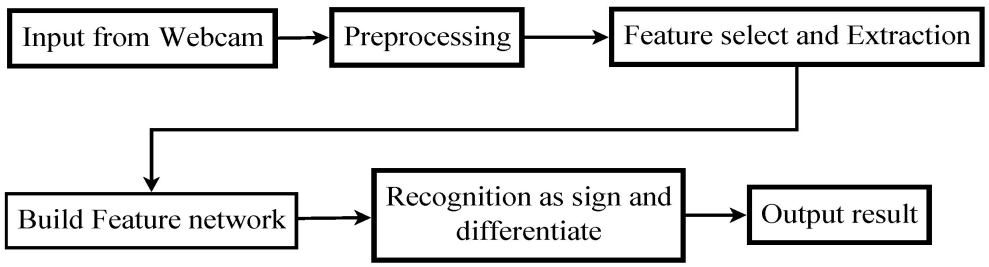

Figure 1. Research workflow.

3.1. LSTM with Skeleton Model

Firstly, we train a deep learning model by using a combination of two different techniques (LSTM, skeleton) to extract key features from each frame of the videos of sign language in a sequential manner [15]. The model training consists of different steps. First of all, video clips are converted into sequence frames; then, characteristics are extracted from each frame. Secondly, classification is performed, which detects different signs in real time [16]. Then, we analyze the proposed model's accuracy through the F1 score, precision, recall, and confusion matrix.

3.2. YOLOv6 Model

In the second approach, we train the YOLOv6 deep learning model on the same custom dataset. The YOLOv6 training consists of different steps [17]. First of all, we perform the labeling of a custom dataset by using the labeling tool. Then, we train YOLOv6 to generate weight files. After training, again, we determine the accuracy, F1 score, precision, recall, and confusion matrix for our model [18].

In this study, we use hybrid approaches for American Sign Language detection for both static and continuous signs, as shown in Figure 2. We train two different models for static and continuous signs, respectively, YOLOv6 and LSTM with the skeleton model [19].

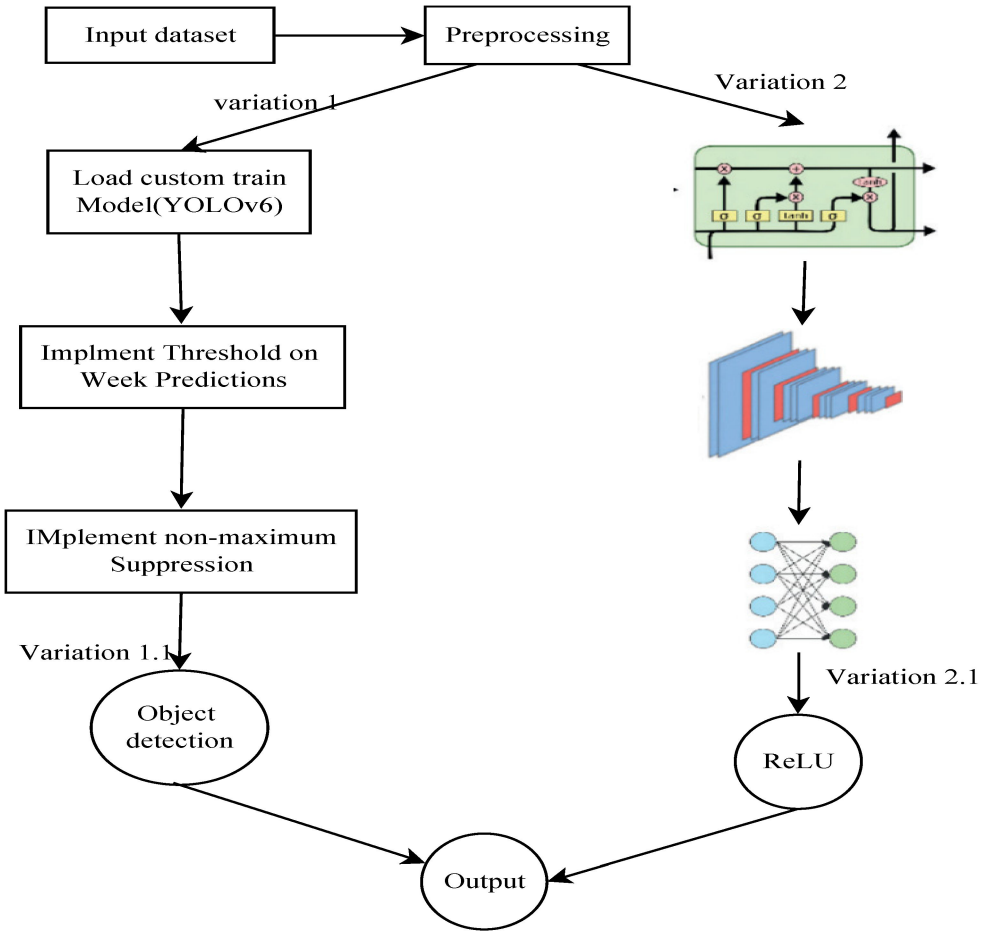

Figure 2. Research methodology.

3.3. LSTM with Skeleton Model Training

3.3.1. Data Collection

Using the DSL10 dataset, experiments are conducted for dynamic sign language recognition. We create and evaluate this dataset for the current study. The DSL10 dataset is made up of 750 films that are randomly divided into two groups for testing and training purposes. All of the videos in the DSL10 dataset were shot with a standard webcam in indoor settings with standard lighting. Because they were all shot at 30 frames per second, each video has the same number of frames and running time (FPS). The total number of frames for each clip was split into separate datasets, with an equal number of frames for each sign language video frame, as the models necessitate.

3.3.2. Preprocessing

The preprocessing block, shown in Figure 1, processes the various frame sizes in the videos for input homogeneity and frame normalization [20]. In order for the model to man-

age inter-database discrepancies caused by the use of various montages and digitalization techniques, input signal homogeneity is required. In particular, the model is amenable to input under a wide range of conditions that balance reducing the input dimensionality with maintaining the signal qualities needed to carry out the sign scoring task. In the same vein as conventional epoch-based scoring processes, videos are segmented using a 30-s window, yielding input patterns of a certain size that are given as input to the subsequent convolutional neural network processing block. After this, a Gaussian standard process is used to normalize the amplitude of each of these input patterns. The input signal can be filtered as a preliminary processing step, but this is not required [21]. In order to improve the final model's generalizability, this section focuses on omitting signals that are not essential but are dataset-specific.

3.3.3. Feature Extraction using MediaPipe

The use of the hands and postural estimation are essential to sign language; however, because of the constant movement, DSL has several challenges. These challenges include identifying the hands and determining their size, shape, and motion. For these issues, MediaPipe was utilized as a solution. It calculates poses for each frame and extracts the key points for both hands' three X, Y, and Z dimensions; these key points are shown in Figure 3. The hand location in relation to the body was predicted and tracked using the posture estimation technique. A list of key points for hands and position estimation is produced by the MediaPipe framework as its output.

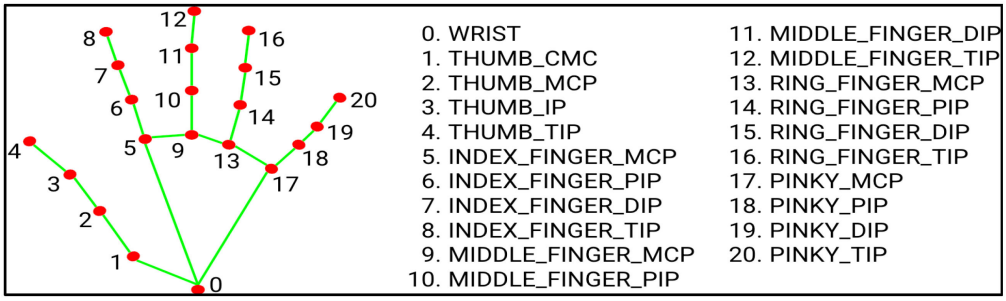

Figure 3. MediaPipe hand landmarks.

MediaPipe's hand-picked 21 most important points, the X, Y, and Z coordinates of the hands' key points in three-dimensional space, are calculated. Therefore, the following is the total sum of extracted hand key points.

$$\begin{aligned} \textit{Total keypoints in hands} = \textit{No. of hands} \times \textit{key points in hand} \times \textit{Three dimensions} = \\ (2 \times 21 \times 3) = 126 \textit{ key points} \end{aligned} \quad (1)$$

Figure 4 shows MediaPipe's pose estimation work; it identifies 33 key points in the image [22]. They are computed in the X, Y, and Z dimensions, so they are not simply based on how visible an object is. The visibility of a given frame's pixel is indicated by a value (hidden by another body part). Because of this, we can calculate the total number of key points recovered by the posture estimation.

$$\begin{aligned} \textit{Total keypoints in pose} = \textit{key points in pose} \times (3 \textit{ dimensions} + \textit{Visibility}) = \\ (33 \times (3+1)) = 132 \textit{ key points} \end{aligned} \quad (2)$$

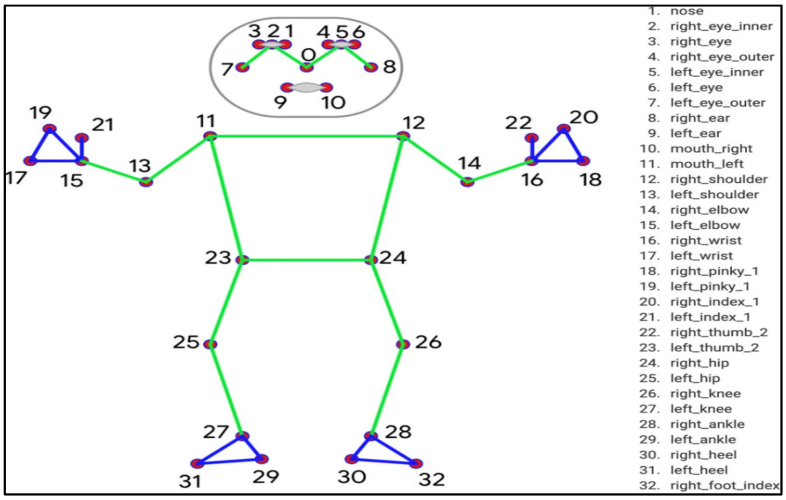

Figure 4. MediaPipe body pose points.

As seen in Figure 5, MediaPipe retrieves 468 key points for the face. Lines connect the dots representing the 468 landmarks to show the contours of the face, eyes, lips, and brows [22]. Exact values are determined for the X, Y, and Z spaces. Thus, the following formula is used to determine how many key points are extracted from the face.

$$\begin{gathered} Total\ keypoints\ for\ face = Total\ key\ points\ of\ face \times 3\ dimensions = \\ (468 \times 3) = 1404\ key\ points \end{gathered} \tag{3}$$

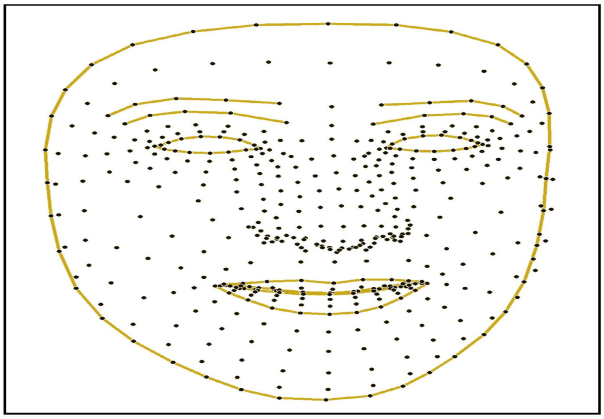

Figure 5. MediaPipe face mesh.

The following formula is used to obtain the total number of key points for each frame, excluding key points for the face.

$$\begin{gathered} Face\ mesh\ keypoints\ per\ frame = Key\ points\ in\ pose + key\ points\ in\ hands = \\ = (132 + 126) = 258\ key\ point \end{gathered} \tag{4}$$

If we sum all the key points in every frame, which includes the face, we obtain

$$\begin{gathered} Total\ number\ of\ keypoints\ in\ model = Key\ points\ of\ face + \\ total\ key\ points\ of\ hands\ total\ key\ points\ in\ pose = (126 + 132 + 1404) \\ = 1662\ key\ points \end{gathered} \tag{5}$$

To extract the crucial points for each frame of the video, this technique is utilized throughout. In every video from the DSL10 dataset, the positions of the hand, torso, and face are identified together with an assessment of their shape and orientation.

The MediaPipe framework offers a number of options to deal with time-series data, including the detection of faces, face meshes, hands, and poses [23]. Some DSL recognition can be inaccurate, and there may be security concerns. The movement of signs can cause problems. Using long short-term memory (LSTM), MediaPipe, and bi-directional LSTM, our study mitigates these drawbacks (BILSTM).

3.3.4. Dataset Properties

The DSL10 dataset was designed as a live sign language dataset, so high-quality recordings are not required for this study's methodology. The DSL10 dataset was readily available for this study, and its use was not difficult. "Hello", "I love you", "thank you", "me", and "sorry" are some of the signs included in the DSL10 dataset of ten commonly used DSL words. There were a total of 30 recordings for each sign, so the signers who created the videos spent 3 h recording the material. Utilizing a camera and the Video Capture feature of the OpenCV package, the data were recorded. With a 640 × 480 resolution and a 30 frame per second frame rate, each video was one second long. The videos were captured utilizing a model developed to create the dataset, and the following suggested guidelines were used.

Signer body: The entire body of the signer must be visible throughout the video.
Signer movement: The camera frame must contain all movement details.

- Background: It is preferable to record the dataset against a steady background devoid of any other hands or faces besides the signer's.
- Lighting: To ensure that all of the key points are visible, it is preferable to record in a well-lit environment.
- Camera: The position of the webcam should be fixed. Ensure that the video is as steady and sharp as possible.
- Number of videos and sequence length: It is important to determine the length of the video clip as well as the number of frames that are included in the sequence before commencing the process of recording.
- Quality: We opted for a regular webcam instead of a specialized sensor because commercially available sensors are of different sizes or larger resolutions.

3.4. Building and Training the LSTM Model

The same preprocessed input dataset was used to import NumPy arrays from MPData, which were mapped and classified as indicated in the previous technique; it was utilized once more for the purpose of training and testing the LSTM model. This was done so that the model could be trained and tested utilizing the Adam optimizer as shown in Figure 6. The training and validation of the model took place over the course of one hundred iterations and we made use of Google Colab for instructional needs, as shown in Figure 7.

```
In [ ]: 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'))

In [ ]: res = [.7, 0.2, 0.1]

In [ ]: actions[np.argmax(res)]

In [14]: model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])

In [ ]: model.fit(X_train, y_train, epochs=100, callbacks=[tb_callback])
```

Figure 6. Model parameter setting.

```
0.9452
Epoch 33/70
73/73 [==============================] - 15s 204ms/step - loss: 0.0198 - accuracy: 0.9966 - val_loss: 0.1103 - val_accuracy:
0.9726
Epoch 34/70
73/73 [==============================] - 13s 182ms/step - loss: 0.0887 - accuracy: 0.9760 - val_loss: 0.2069 - val_accuracy:
0.9452
Epoch 35/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0355 - accuracy: 0.9932 - val_loss: 0.1648 - val_accuracy:
0.9589
Epoch 36/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0222 - accuracy: 0.9966 - val_loss: 0.1908 - val_accuracy:
0.9178
Epoch 37/70
73/73 [==============================] - 13s 183ms/step - loss: 0.0105 - accuracy: 1.0000 - val_loss: 0.1827 - val_accuracy:
0.9589
Epoch 38/70
73/73 [==============================] - 13s 184ms/step - loss: 0.0642 - accuracy: 0.9760 - val_loss: 0.1956 - val_accuracy:
0.9178
```

Figure 7. LSTM model training and fitting for 100 iterations.

The architectures of RNN and long short-term memory allow for the extraction of features over arbitrary time periods. They were designed for the categorization, processing, and prediction of time series with known time delays but unknown lengths [20]. Because LSTM has relative intensity gaps, it outperforms competing sequences of action in learning models from videos, such as hidden Markov models and other RNNs. The looping arrow on the LSTM, which denotes its recursive nature, is known as the cell state. As a result, the cell state has data from the previous interval. It is modified by a remember vector situated beneath the cell state and is then adjusted by input modification gates.

The information must be multiplied by the forget gate in order for it to be removed from the cell state equation. At the same time, fresh information must be added through the output of the input gates. The forget gate multiplies a number to determine which information the cell state must forget. A zero value represents the necessary matrix position. If the forget gate output value is one, the data remain in the cell state. After this, the algorithm makes use of the input by applying a sigmoid function to it, along with a weight and the state that was hidden earlier. Typically, the task of selecting the data to be fed into the LSTM is delegated to a save vector, which is also referred to as an input gate. The domain of this sigmoid function includes the numbers 0 and 1. Because the equation for the cell state is the sum of all the previous cell states, this does not eliminate the memory; rather, it adds to it. There are a few names for working memory and a few for the focus vector. One is the "hidden state", while the other is the "output gate". Figure 8 is a magnified view of a long short-term memory cell, showcasing the sigmoid functions [8].

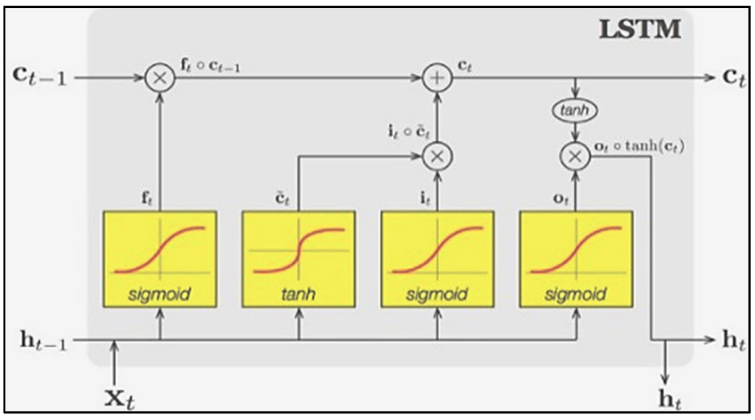

Figure 8. LSTM cell with sigmoid gate. $\otimes$ pointwise multiplication; $\oplus$ pointwise addition; σ sigmoid-activated NN; h_{t-1} input data; c_{t-1} previous state; *tanh* tanh-activated NN.

The information that the CT-1 cell state should forget is defined by the forget gate, which is depicted by the first sigmoid on the left of the figure above. The input gate, which controls whether or not the LSTM retains the data, is defined by the second sigmoid and the tanh activation function. Data that pass through the final sigmoid output gate must be in the next hidden state.

The LSTM block has three layers: a first sequence configuration layer, an LSTM layer that only processes data in one direction, and a fully connected layer that is activated with SoftMax. The sequence configuration process generates the epoch-specific feature sequence k that corresponds to the current evaluation epoch. For a PSG recording with M epoch intervals, for a given epoch k, $k = 1 \ldots M$, the sequence $S(k)$ is composed as follows.

$$\begin{aligned}\left[F\left(k-\left[\tfrac{L-1}{2}\right]\right), F\left(K+1-\left[\tfrac{L-1}{2}\right]\right), \ldots\ldots\ldots\ldots, F(K-1), F(K), F(K+1), \ldots\ldots\ldots\ldots\right. \\ \left. \ldots\ldots\ldots\ldots, F\left(K-1+\left[\tfrac{L-1}{2}\right]\right)\right.\end{aligned} \tag{6}$$

where L and K stand for the ceiling and floor operations, L stands for the length of the sequence, and F stands for the input feature vector that corresponds to it, which, in this case, is derived from the CNN node that comes before it. In the event that L equals three, for instance, the sequence might be [F ($k-1$), [F ($k-2$), [F ($k-1$), [F ($k-2$), etc. In this study, a total of one hundred neurons were assumed to be concealed within the LSTM layer.

In contrast to the RNN layers that compose the first three, the final three are dense. In the next step, the optimal optimizer parameter value is used to assemble the layers. Parameters for the layer could be set to any value before training the model as required in the layer setting, as shown in Figure 9.

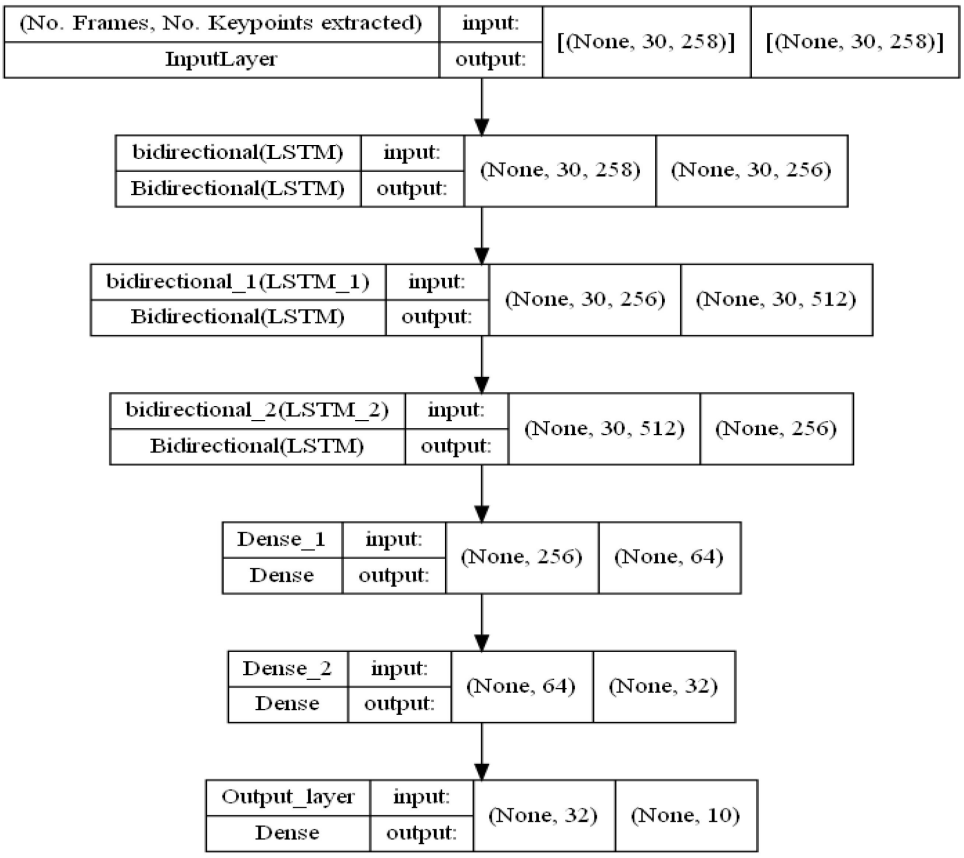

Figure 9. Layers used by LSTM for training.

3.5. Training Process of YOLOv6 Model

3.5.1. Dataset Collection

We collect a dataset consisting of 8000 photos separated into 26 classes, each class denoting a number and a letter, to be utilized as input data for the recognition system. Approximately 200 images were selected from the alphabet, as well as a few words for each class, as indicated in Table 1.

Table 1. Dataset properties.

Dataset Specification	Value
Image resolution	1920 × 1080
Image extension	JPG
Total images in dataset	1050
Total classes	26
Images in dataset per class	40
Image size	900–1000 Kb

As given in Table 2, 80 percent of the dataset was used for training, and the remaining 20 percent was used for testing. Table 2 and Figure 10 show how they were divided into four groups under different lighting conditions and backgrounds.

Table 2. Dataset under different condition.

Conditions	Description
Condition 1	Background 01, taken from 10–20 cm away
Condition 2	Background 01, taken from 30–40 cm away
Condition 3	Background 02, taken from 10–20 cm away
Condition 4	Background 02, taken from 30–40 cm away

Figure 10. Images under different background and lighting conditions.

3.5.2. Image Labeling

Using the labeling application "LabelIMG", which forms a box around the object to be detected and sets five values for the object, each class was given a name, as shown in Figure 11. Labels included a value or number for the class, two values for the object's location represented by X and Y, and additional values for the object's size, as seen in Figure 12. We stored these values in an XML file. Each image in the class, as well as the other classes, underwent this procedure.

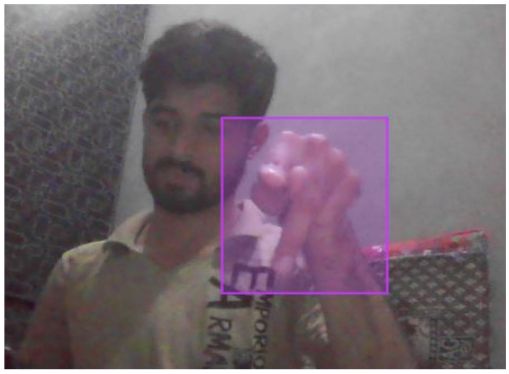

Figure 11. Labeling of images using labeling tool.

Figure 12. Object values.

3.5.3. Dataset Division

Following the collection and labeling of the images in YOLO format, the image and label files were divided by 80% and 20%, respectively, for training and validation.

3.5.4. YOLOv6 Selection and Training

YOLO is a well-known object detector in terms of accuracy and speed. At the time of the study, YOLOv6 is the latest version, which is the best one compared to other versions. When we obtained the dataset for the model's training, due to the large size of the photos, which caused training issues, the model did not perform well. Therefore, we changed the image size and repeatedly retrained the model using the new dataset. We trained the model for 400 iterations in the initial training; for training purposes, we used Google Collab, which took approximately 4 h for training purposes at 400 iterations. As shown in Figure 13, the accuracy increased over time; at iteration 100, it was 44%; at iteration 150, it was 96%; and at iteration 400, the rate of loss began to decline. It continued declining through iteration 200 and reached a low point at iteration 300, where it was 1.52. The first training stage finished at this point.

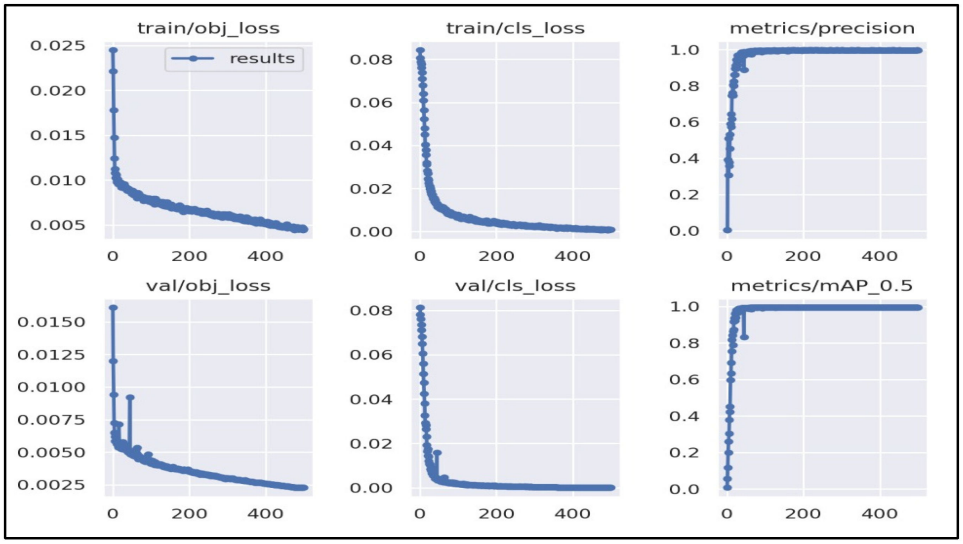

Figure 13. First training results from YOLOv6.

As we can see, when increasing the number of iterations, the training object loss and training class loss both decreased, and the mAP also increased up to a specific point and stopped at a specific point. We could not achieve a further positive effect by increasing the number of iterations. These losses also satisfied the tested data, with small amount of variation, but still provided satisfactory results, as seen in the lower row of Figure 13. Increasing the number of iterations does not guarantee better outcomes. Although we raised the number of iterations, the training was ended because we did not see any positive effects. The current average loss no longer decreased.

4. Evaluation of Proposed Models

In the following subsections, we describe the evaluation results of each proposed model, including examples of test images.

4.1. Evaluation of LSTM Model Results

The results of applying deep learning models to recognize common, straightforward message postures such as "hello", I love you", "thanks", "sorry", and "my name", and alphabets in American Sign Language, are discussed in this section. The data collected in this section did not rely on preprocessed information but rather were gleaned from the collection of different face and complete body landmarks using a computer webcam. The necessary dependencies had to be installed and exported in Python in order to accomplish this. Then, using the MediaPipe holistic, important details were extracted from the various facial and position markers. The fundamental point values for the various landmarks were then applied at various positions in the model. These images were 150, 30, and 1662 pixels in size, respectively. This implies that the breadth was 30 (inconsistent) and the length was 150. The CNN model, as previously stated, only accepts photos with identical height and width measurements. As a result, the LSTM model was developed, trained, and tested solely with the image dataset gathered for this model. The dataset was divided into two sets, with 70% used for training and 30% used for testing.

Accuracy can change in both directions during training and testing, with positive and negative results possible depending on how the created model performs. Both the testing and training of the LSTM model were monitored and left unattended for 40 iterations in this experiment. The accuracy was 37% during training and 33% during testing. As the training progressed, the rate of rise became unstable, and the training accuracy increased in an erratic manner. Furthermore, the rate of increase remained erratic even as the testing accuracy increased, as the forty-first iteration approached. The LSTM model had a 20.5% value, with the lowest training accuracy of 49.0%, and the model attained the highest training accuracy of 89.8%.

On the other hand, the long short-term memory model with MediaPipe achieved 25.5% to 91.47% accuracy across its testing range. The long short-term memory model's loss in both aspects of training and testing decreased somewhat with each iteration. Once more, the reduction was quite erratic.

Figure 14 represents the long short-term memory model's behavior in both aspects of testing and training with respect to accuracy. On the graph, the number of epochs is depicted along the X-axis, with a range from 0 to 41, and the accuracy is depicted along the Y-axis, with a range from 0 to 100 percent (where 1.0 represents 100 percent). The graph depicts a blue curve for training accuracy and an orange curve for testing accuracy. The accuracy rates of the long short-term memory model during training and testing are displayed versus the number of iterations.

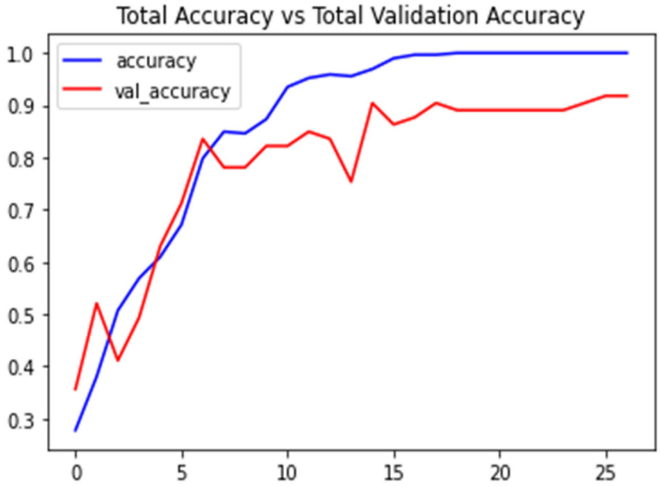

Figure 14. Training and validation accuracy of LSTM model.

While both the training and testing accuracy improved with an increasing number of epochs, the graph shows that the curves followed a somewhat unpredictable pattern. The graph shows that the training and testing accuracy peaked at the 36th and 40th iterations, while the training precision peaked around the twenty-fifth iteration. Regardless of the number of iterations, neither curve appears to be in a forced steady state. The graph below represents how much weight was lost during training and testing. The number of epochs is represented along the X-axis, with a range of 0 to 41, and losses during the training process are plotted on the Y-axis. Figure 15 represents the testing and training losses. The training losses are depicted by the blue line, while the testing losses are depicted by the red line. The relationship between the number of epochs used to train and test the loss function for the LSTM model is shown graphically in Figure 15.

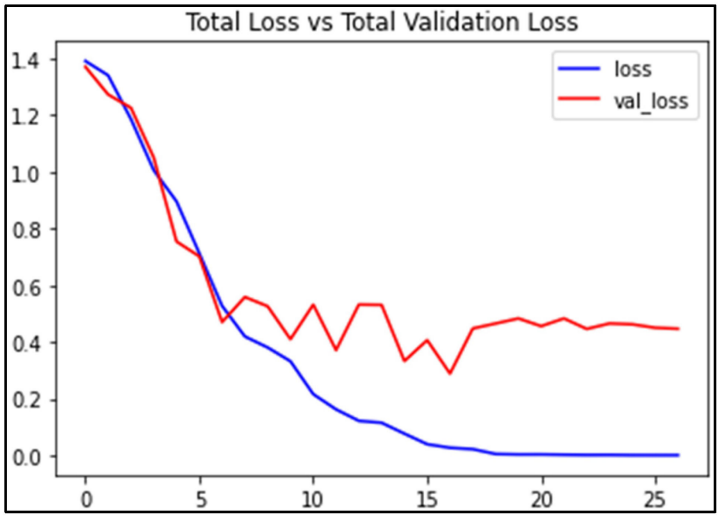

Figure 15. Training and validation loss of LSTM model.

The loss seemed to decrease over time in both the training and testing phases, but the trend was highly erratic, with sudden spikes and dips along the curve. This led to surprising results on the dataset. To perform a more in-depth analysis of these results, a confusion matrix for multiclass datasets was used, as in Figure 16 below.

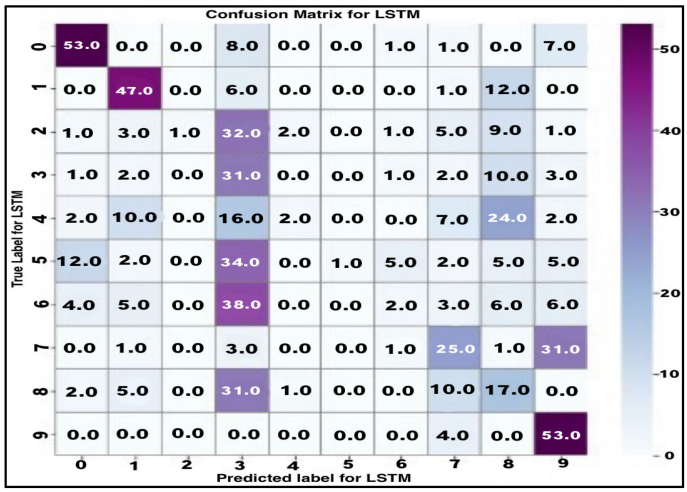

Figure 16. Confusion matrix for LSTM model.

The rows and columns in the confusion matrix each reflect the actual class index number, whereas columns represent the anticipated class index number. The values that are shown in the diagonal of the confusion matrix are the ones that are projected to be correct based on the dataset. On the other hand, it also represents the wrongly predicted classes according to the dataset in each class. The classes that are predicted to have the highest value at each level of the class in each row and column are shaded with a darker color in order to more clearly display the patterns in the behavior and performance of the model [8]. The confusion matrix of the LSTM model is shown in Figure 16 to demonstrate the classification performance.

The majority of the highest anticipated values across all levels of the classes are clearly dispersed randomly across the matrix. The absence of a discernible pattern in the diagonal values of the matrix indicates that the LSTM model's ability to forecast numerical sign language does not follow a consistent or reliable pattern. The model had average precision of 80% while only having a recall rate of 52% of the data. The averages for recall and precision were 57% and 82%, respectively. For dynamic signs, the long short-term memory with MediaPipe model performed better during both aspects of training and testing, as seen in Figure 17.

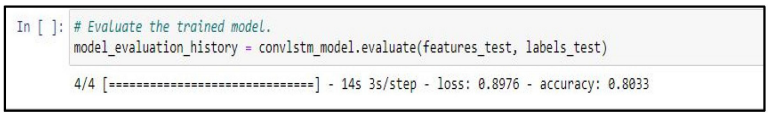

Figure 17. Classification results of LSTM model on video data.

4.2. Evaluation of YOLOv6 Model Results

In the study presented, we developed an American Sign Language (ASL) recognition system leveraging the capabilities of the YOLOv6 approach, a cutting-edge object detection model. Rather than relying on previously established datasets, we curated our own custom dataset for ASL, ensuring an updated and possibly more comprehensive set of sign language data. The performance metrics are quite impressive: for static images, the system achieved a mean average precision (mAP) of 96%. This essentially means that in 96 out of 100 cases, the model correctly identified and recognized the ASL signs presented in the photographs. Despite the high accuracy, there was an average loss value of 0.81, suggesting that there is potential for the model's predictions to be refined further. When it comes to real-time video analysis, the model demonstrated a capability to process at a rate of 28.9 frames per second, making it a viable tool for applications that demand instant ASL sign interpretation. A comparative analysis was also conducted, wherein the YOLO approach was compared against the LSTM network. The findings favored YOLO,

especially for the recognition of static signs, which are hand gestures that maintain a steady posture rather than those involving dynamic movements. To further enhance the model's accuracy and predictive capabilities, we recommend augmenting the dataset with more images. Additionally, a dedicated table, referred to as Table 3, is provided. This table offers a granular breakdown of the results for each class or type of sign in the dataset, detailing the accuracy and quantifying the true and false predictions.

Table 3. Calculated results for different classes.

Class Name	Average Precision (ap)	True Positive (tp)	False Positive (fp)
A	98.8%	40	1
B	100.00%	40	0
C	100.00%	40	0
D	100.00%	40	0
E	99.8%	39	1
F	98.00%	37	4
G	100.00%	40	0
H	100.00%	40	0
I	100.00%	40	0
J	88.00%	32	9
K	98.20%	39	1
L	97.00%	38	0
M	86.00%	35	7
N	100.00%	40	0
O	100.00%	40	0
P	96.00%	38	2
Q	96.00%	39	2
R	100.00%	40	0
S	100.00%	40	0
T	97.40%	34	6
U	88.00%	36	7
V	90.00%	36	5
W	92.00%	35	5
X	91.00%	30	4
Y	99.00%	38	1
Z	88.00%	31	8

4.3. Examples of Test Images

The comparison of the performance results for training in terms of precision, recall, the F1 score, and mAP is given in Table 4 and Figure 18.

Table 4. Overall YOLO performance.

Results	First Time Training	Second Time Training
Precision	0.86	0.96
Recall	0.92	0.96
F1 score	0.89	0.96
TP	1510	1533
TF	215	67
FN	90	67
Average IoU	68.86%	74.67
mAP	90.44%	96.22%

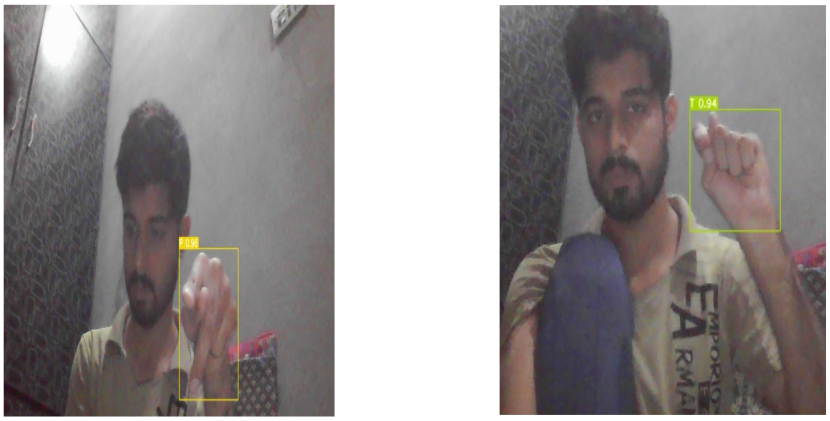

Figure 18. YOLO resulting images for T and P.

5. Summary and Discussion

This part provides an in-depth discussion of the study's findings, including how these findings and the accompanying literature review relate to the study's stated goals, hypotheses, and pertinent secondary issues.

The goal of this research was to determine whether deep learning and computer vision models could help those who are unable to speak but can understand sign language and communicate with the rest of society. The study's objective was to generate a hybrid model from two deep learning models for ASL interpretation and prediction: long short-term memory with skeleton and YOLO [24]. The questions above required both a critical analysis of the current literature on deep learning and a hybrid experiment [22]. In this hybrid experiment, we collected multiple datasets, used them to train two deep learning models (YOLO and LSTM), and then tested their performance in predicting ASL, and both of them showed specific features in specific applications. The YOLO model's accuracy improved with more training iterations, and this effect was seen in both settings. The model was better able to learn from the training set and make reliable predictions after being given more training cycles. The obtained results show that the YOLO model's accuracy improves with more time spent in training, as measured by the number of iterations and images. On the other hand, given the constraints of the number of iterations, LSTM performed better.

6. The Implications for Research and Practice

The hybrid method was broken down into two distinct approaches, and each of them used a specific type of dataset. The first strategy involves using deep learning models to predict static signs in American Sign Language using a custom dataset. The second strategy includes using a webcam to generate a dataset to train deep learning models to recognize and forecast continuous or dynamic signs (motion of hands mixed with facial expressions) in American Sign Language. The YOLO and long short-term memory models were built, trained, and evaluated with the use of a multi-class confusion matrix in the first method. The training and testing accuracy of the CNN were improved to 96.00% and 92.00%, respectively, thanks to this strategy.

There was a strong relationship between the total number of iterations and the accuracy of both the training and the testing. Experimental results showed that the YOLOv6 model developed in this study consistently predicted a high percentage of static hand gestures in both training and testing, as measured by the multi-class confusion matrix. There was an average weighted precision of 96% and average recall of 92% with the YOLOv6 model.

According to these results, the YOLO algorithm is superior to the LSTM model in recognizing static signs, but not in recognizing continuous ones. The YOLO model achieved 96% training accuracy, 95% testing accuracy, and 96% precision after only 97 epochs, indicating that it has a great deal of promise in accurately predicting static signs in American Sign Language. These results support the first hypothesis, which proposes that the combination of deep learning and computer vision models can greatly enhance the communicative

capabilities of people with speech impairments by providing high accuracy, precision, and reliability.

This study's primary approach involved developing two separate deep learning models—a YOLO model and a long short-term memory (LSTM) model—and then fusing them into a hybrid method. The datasets were collected, used for model construction, and trained, tested, and evaluated using a multi-class confusion matrix in this experiment. The accuracy of the training and testing data, as well as the precision and reliability/consistency of the sign language prediction, were used as evaluation metrics. The prediction of static hand gestures in American Sign Language was performed with the help of user-supplied data, while a key point dataset was used to indicate continuous signs (hand movement combine with facial expressions).

7. Conclusions and Future Work

According to the study's premise, the YOLO model is the most accurate, precise, and reliable option for static signs. A positive correlation was established between the length of the model's training period and the accuracy of both training and testing. The length of the training period was defined by the number of iterations and the number of images in each dataset. The YOLO model that was developed as a result was quite accurate. If more signals and iterations were employed to improve the training and testing accuracy, then the model could be used to accommodate more than one form of sign language, which would increase its efficiency. However, LSTM with the skeleton model performed very well for continuous sign classification, but it can only be used with a small number of signs, and as the number of signs is increased, the accuracy of the signs decreases. For this reason, we employed a hybrid approach for both static and continuous signs in sign language. In the future work, we plan to combine the developed models with fuzzy logic to deal with uncertainty in the outputs, to improve the detection rate in the presence of noise and changes in lighting and contrast.

Author Contributions: Conceptualization, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; methodology, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; software, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; validation, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; formal analysis, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; investigation, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; resources, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; data curation, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; writing—original draft preparation, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; writing—review and editing, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; visualization, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; supervision, A.M.B., U.A., A.H.G., A.A., M.A.A. and B.F.A.; project administration, A.H.G., A.A., M.A.A. and B.F.A.; funding acquisition, A.H.G., A.A., M.A.A. and B.F.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by King Salman Center for Disability Research through Research Group (No. KSRG-2023-105).

Data Availability Statement: Data will be available by request from the corresponding author.

Acknowledgments: The authors extend their appreciation to the King Salman Center for Disability Research for funding this work through Research Group No. KSRG-2023-105.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Trigueiros, P.; Ribeiro, F.; Reis, L.P. Vision-based Portuguese Sign Language Recognition System. In *New Perspectives in Information Systems and Technologies*; Springer International Publishing: Cham, Switzerland, 2014; Volume 1, pp. 605–617.
2. Agarwal, S.R.; Agrawal, S.B.; Latif, A.M. Sentence Formation in NLP Engine on the Basis of Indian Sign Language using Hand Gestures. *Int. J. Comput. Appl.* **2015**, *116*, 18–22.
3. Neidle, C.; Thangali, A.; Sclaroff, S. Challenges in Development of the American Sign Language Lexicon Video Dataset (ASLLVD) Corpus. In Proceedings of the LREC2012 5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon, Istanbul, Turkey, 27 May 2012; pp. 1–8.
4. Kingma, D.P.; Ba, J.L. Adam: A method for stochastic optimization. *arXiv* **2015**, arXiv:1412.6980.

5. Arora, S.; Roy, A. Recognition of sign language using image processing. *Int. J. Bus. Intell. Data Min.* **2018**, *13*, 163–176.
6. Golestani, N.; Moghaddam, M. Human activity recognition using magnetic induction-based motion signals and deep recurrent neural networks. *Nat. Commun.* **2020**, *11*, 1551. [CrossRef]
7. Gunji, B.M.; Bhargav, N.M.; Dey, A.; Zeeshan Mohammed, I.K.; Sathyajith, S. Recognition of Sign Language Based on Hand Gestures. *J. Adv. Appl. Comput. Math.* **2022**, *8*, 21–32. [CrossRef]
8. Tolentino, L.K.S.; Serfa Juan, R.O.; Thio-ac, A.C.; Pamahoy, M.A.B.; Forteza, J.R.R.; Garcia, X.J.O. Static sign language recognition using deep learning. *Int. J. Mach. Learn. Comput.* **2019**, *9*, 821–827. [CrossRef]
9. Jiang, S.; Sun, B.; Wang, L.; Bai, Y.; Li, K.; Fu, Y. Skeleton aware multi-modal sign language recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Virtual, 19–25 June 2021; pp. 3408–3418.
10. Bantupalli, K.; Xie, Y. American Sign Language Recognition using Deep Learning and Computer Vision. In Proceedings of the 2018 IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, 10–13 December 2018; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2019; pp. 4896–4899.
11. Kothadiya, D.; Bhatt, C.; Sapariya, K.; Patel, K.; Gil-González, A.B.; Corchado, J.M. Deepsign: Sign Language Detection and Recognition Using Deep Learning. *Electronics* **2022**, *11*, 1780. [CrossRef]
12. Iyer, R.; Shashikant Ringe, P.; Varadharajan Iyer, R.; Prabhulal Bhensdadiya, K. Comparison of YOLOv3, YOLOv5s and MobileNet-SSD V2 for Real-Time Mask Detection Comparison of YOLOv3, YOLOv5s and MobileNet-SSD V2 for Real-Time Mask Detection View project Comparison of YOLOv3, YOLOv5s and MobileNet-SSD V2 for Real-Time Mask Detection. *Artic. Int. J. Res. Eng. Technol.* **2021**, *8*, 1156–1160.
13. Lin, H.; Hong, X.; Wang, Y. Object Counting: You Only Need to Look at One. *arXiv* **2021**, arXiv:2112.05993.
14. Dhulipala, S.; Adedoyin, F.F.; Bruno, A. Sign and Human Action Detection Using Deep Learning. *J. Imaging* **2022**, *8*, 192. [CrossRef] [PubMed]
15. Alvarez-Estevez, D.; Rijsman, R.M. Inter-database validation of a deep learning approach for automatic sleep scoring. *PLoS ONE* **2021**, *16*, e0256111. [CrossRef] [PubMed]
16. Al-Shaheen, A.; Çevik, M.; Alqaraghuli, A. American Sign Language Recognition using YOLOv4 Method. *Int. J. Multidiscip. Stud. Innov. Technol.* **2022**, *6*, 61. [CrossRef]
17. Forster, J.; Schmidt, C.; Koller, O.; Bellgardt, M.; Ney, H. Extensions of the Sign Language Recognition and Translation Corpus RWTH-PHOENIX-Weather. In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), Reykjavik, Iceland, 26–31 May 2014.
18. Mekala, P.; Gao, Y.; Fan, J.; Davari, A. Real-time sign language recognition based on neural network architecture. In Proceedings of the 2011 IEEE 43rd Southeastern Symposium on System Theory, Auburn, AL, USA, 14–16 March 2011; pp. 195–199. [CrossRef]
19. Huang, Y.; Huang, J.; Wu, X.; Jia, Y. Dynamic Sign Language Recognition Based on CBAM with Autoencoder Time Series Neural Network. *Mob. Inf. Syst.* **2022**, *2022*, 3247781. [CrossRef]
20. Dang, C.N.; Moreno-García, M.N.; De La Prieta, F. Hybrid Deep Learning Models for Sentiment Analysis. *Complexity* **2021**, *2021*, 9986920. [CrossRef]
21. Kaluri, R.; Pradeep Reddy, C.H. Sign gesture recognition using modified region growing algorithm and Adaptive Genetic Fuzzy Classifier. *Int. J. Intell. Eng. Syst.* **2016**, *9*, 225–233. [CrossRef]
22. Aly, S.; Aly, W. DeepArSLR: A novel signer-independent deep learning framework for isolated arabic sign language gestures recognition. *IEEE Access* **2020**, *8*, 83199–83212. [CrossRef]
23. Samaan, G.H.; Wadie, A.R.; Attia, A.K.; Asaad, A.M.; Kamel, A.E.; Slim, S.O.; Abdallah, M.S.; Cho, Y.I. MediaPipe's Landmarks with RNN for Dynamic Sign Language Recognition. *Electronics* **2022**, *11*, 3228. [CrossRef]
24. Al-Hammadi, M.; Muhammad, G.; Abdul, W.; Alsulaiman, M.; Bencherif, M.A.; Alrayes, T.S.; Mathkour, H.; Mekhtiche, M.A. Deep learning-based approach for sign language gesture recognition with efficient hand gesture representation. *IEEE Access* **2020**, *8*, 192527–192542. [CrossRef]