ARTIFICIAL NEURAL NETWORK
CS-307

Project Report

# Sign Language Recognition

Submitted to Dr. Munwar Iqbal

Submitted By
21-CS-103 UMAIR FAROOQ
21-CS-109 ZAIN MEHMOOD

DEPARTMENT OF COMPUTER SCIENCE, UET TAXILA

# Table of Contents

## Abstract:

Individuals with hearing impairments, commonly referred to as deaf or hard of hearing, rely on sign language as their primary mode of communication instead of spoken language. However, for those who do not understand sign language, so communicating with the community can be quite challenging. Therefore, there is an extreme need to develop an application capable of recognizing sign language gestures or actions facilitating communication between the hearing and deaf communities. Despite the recognition of American Sign Language (ASL) within American society there remains a scarcity of educational ASL applications, particularly those equipped with real-time sign recognition systems. The Leap Motion controller offers a promising solution by enabling real-time and accurate recognition of ASL signs. Leveraging this technology presents an opportunity to create a learning application aimed at enhancing ASL proficiency. Input features for the classification model include characteristics like sphere radius, finger angles, and finger position distances. Experimental results demonstrate an average accuracy rate of 99.44% for recognizing the 26 ASL alphabets, with a 91.82% accuracy rate in 5-fold cross-validation using the Leap Motion controller. But the existing systems are costly and complex because of expensive sensers, hardware to run heavy models, etc.

In this we are using CCN for the American Sign Language (ASL) recognition. The model using classic MNIST trained on 27,455 samples and test on 7172 samples for the static signs excluding J and Z because of gesture motion (3D). And the accuracy in about 100% in training and validation.

## Introduction:

The ability to communicate is vital to the survival of humans. It is a basic, yet effective method used for communicating ideas, emotions, and perspectives. But a significant percentage of the global population is incapable of doing so. Many people experience speaking difficulties, losing their hearing, or both of them. A hearing problem refers to the partial or total capability of hearing in either one or both ears. On the other hand, being mute is a disorder that negatively impacts speech and renders the affected individual incapable of speaking. If a person becomes deaf-mute in their early years, it may interfere with their capacity to learn language and cause language impairment, often referred to as hearing mutism.

These conditions rank among the most frequent disabilities around the globe. A statistical analysis of children with disabilities that are physical, over the past decade shows a spike in an increasing number of newborns born with difficulties with hearing, which puts them at a communication disadvantage with the world around them.

A World Health Organization (WHO) research estimates that 278 million individuals across the globe were living with a hearing problem in 2005. After ten (10) years, this number had increased to 360 million, a rise that was almost 14%. The total number has been rising at an exponential rate ever since. According to the World Health Organization's most recent study, 466 million individuals worldwide—or 5% of the total population—had hearing loss in 2019. Of these, 432 million (or 83%) were older individuals, and 34 million of them (17%) were children.

Additionally, the World Health Organization (WHO) predicted that by the year 2050, the population would have doubled to a total of 900 million. It is necessary to remove the communication barrier that negatively impacts the lives and social connections of the rapidly increasing number of deaf-mute individuals.

But however, there is a clear answer to this problem in the fields of image detection and machine learning. Real-time captioning for Zoom meetings and other virtual conferences can be created by applying machine learning technologies to automatically recognize Sign Language signals. This would work in conjunction with voice-based captioning to create a two-way online communication system for individuals with hearing impairments, so dramatically increasing the accessibility to such assistance for the people who require them.

Furthermore, people can communicate by combining hand and mouth gestures with facial emotions. Three primary variations exist in sign language. They are:

1) Non-manual features: Hand gestures, body postures, facial expressions, and tone of voice are all utilized in communication.
2) Word-level sign expressing: every gesture symbolizes a whole word.
3) Hand vocabulary: An alphabet or number is represented by a single gesture.

We used the Hand vocabulary and used ASL [1] data set for the procedure in the present paper. The literature currently being published mostly includes two different kinds of approaches. One is recognizing and capturing the motion with a specialized gadget. Using deep learning to identify the movements that the hands are making is the second method. However, the drawback of this strategy is that the deep learning models' construction and computation are highly costly, and the specialized tools needed to use them are not widely accessible or affordable.

Since Keras is an open-source package that offers an integration with Python for artificial neural networks, we are employing it. Keras serves as the TensorFlow library's interface. A software library that is free and open source for machine learning and deep learning as well as artificial intelligence is known as TensorFlow.

## Literature:

Literature review of the problem shows that there have been several approaches to address the issue of gesture recognition in video using a lot of different methods. Past research has suggested several methods for the recognition of ASL, including the usage of motion gloves, Kinect Sensor, image processing with 3D cameras and leap motion controllers.

In one [2] The authors combined Gaussian Tree Augmented Naive Bayes Classifier and Bayesian Network Classification algorithms with Hidden Markov Models (HMM) to determine face expression from footage captured on video.

In another paper by Deep Kothadiya [3], a model powered by deep learning that can identify words from the individual's gestures. To identify signs for isolated Indian Sign Language (ISL) frames of video, LSTM and GRU are employed. With one layer of LSTM and one layer of GRU, the suggested model achieves approximately 97% accuracy across 11 distinct signs.

Francois et al. [4] also published an article, on the use of 2D and 3D appearance techniques for Human Posture Recognition in the video Frame. The paper discusses modelling posture in 3D and applying PCA to identify shapes from a stationary camera. The disadvantage of this method is that it uses intermediary gestures, which could cause ambiguity during training and, as a result, reduce prediction accuracy.

C.K.M. Lee [5] with his fellow researchers, published a paper on American sign language ASL with recognition and training methods with recurrent neural network. Given that ASL alphabets contain both static and dynamic signs (J, Z), the Long-Short Term Memory Recurrent Neural Network with k Nearest Neighbour approach is used. This is because the classification method relies on processing input sequences. The classification model uses extracted characteristics as input, such as spherical radius, angles within your fingertips, and spacing between finger positions.

A total of 2600 instances—100 samples for each alphabet—are used to train the computational model. The findings of the experiment showed that, when using a leap motion controller, the recognition rate of 26 ASL alphabets produces an average accuracy rate of 99.44% and 91.82% in five-fold cross-validation.

Similar work by Kumud et al. [6], explains the process of continuously recognizing Indian sign language. In order to extract frames from video data, the research suggests pre-processing the data, identifying important frames from the data, then extracting more features, recognition, and optimization.

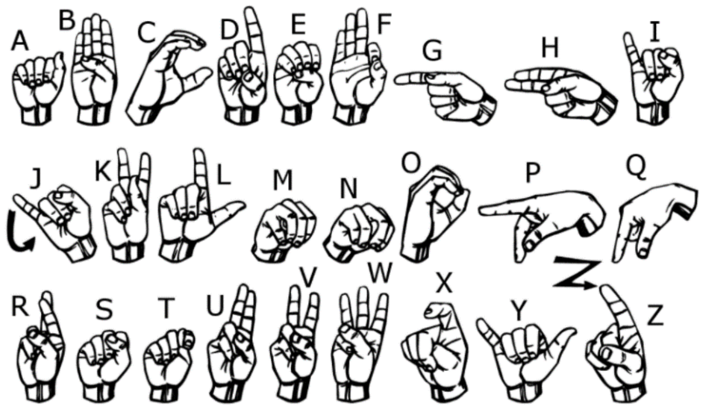
The video is first converted to an RGB frame stream in order to begin pre-processing. Every frame has the same measurements. With the aid of HSV, skin colour segmentation is utilized to extract skin region. The acquired photos were transformed into binary format. The gradient in between each frame was calculated in order to extract the critical frames. And an oriental histogram was used to extract the features from the key frames. The following distances were used for classification: Euclidean, Manhattan, Chess Board, and Mahalanobis.

## Methodology:

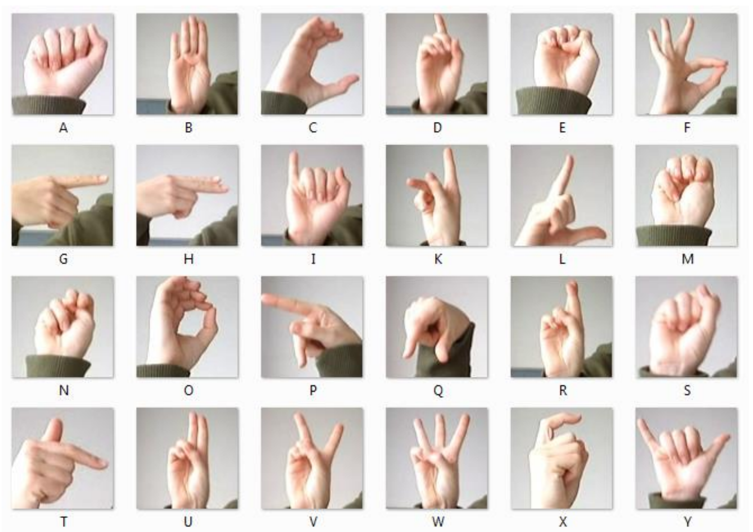
American Sign Language (ASL) is the complete, conversational language with similar linguistic properties as of spoken languages, with the grammatical structures that is distinct from English. Hand and facial movements are used in ASL communication. Many deaf and hard of hearing folks in North America use it as their primary means of communication, and some hearing people can also.

### Dataset:

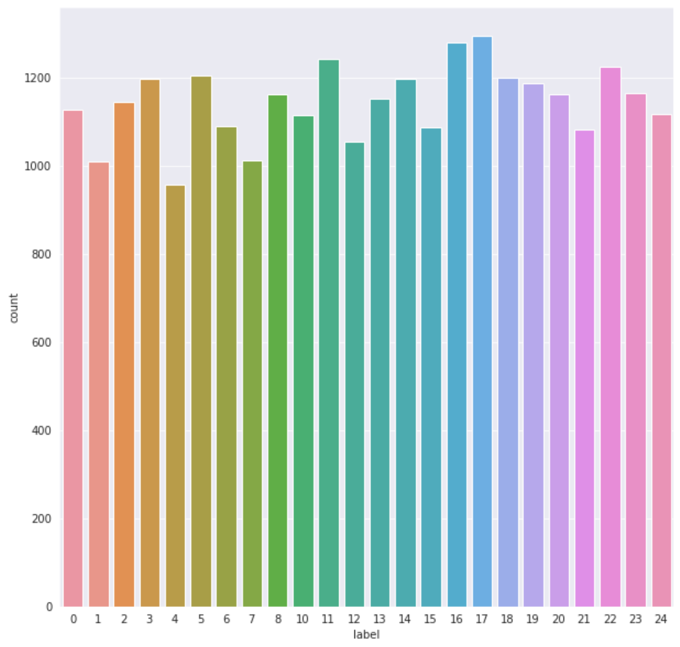
The format of the dataset is designed to closely resemble the traditional MNIST [1]. With the exception of scenarios in which 9=J or 25=Z are represented by gesture motions; every training and test sample displays a label (0–25) as a one-to-one map for every alphabetic letter A–Z. As in the figure below



The test data (7172 samples) along with the training dataset (27,455 samples) include a header row labelled "label, pixel1, pixel2, pixel3, .... pixel784, representing a single 28x28 pixel



picture having grayscale values ranging between 0-255. The training data and test data are about half the size of the typical MNIST but otherwise similar. The original data for the hand gesture image includes several users repeating the move against various backgrounds. The small number of colour photographs (1704) that were included and were not cropped around the hand region that was important gave the Sign Language MNIST data.



An ImageMagick-based image pipeline was utilized for the generation of new data which involved resizing, grayscale scaling, cropping to hands-only, and making at least 50+ variations like rotation, flipping, etc. to increase the quantity.

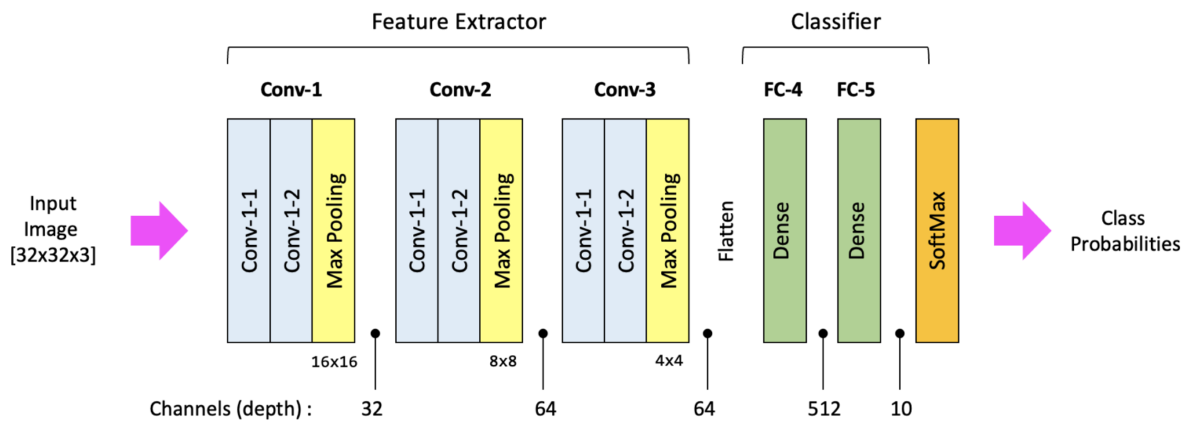
## Data Processing:

To prevent overfitting, we must purposefully enlarge the overall size of the collected dataset. We have to expand the overall size of the existing dataset. In order to produce the variations, the training data undergoes modification through slight adjustments. Data augmentation methods are the techniques that improve the training data in a way that modifies the data set's array representation while maintaining the label. Grayscales, vertical and horizontal flips, random crops, colour issues, translations, rotations, and much more are common augmentations performed by individuals. We can easily double or quadruple the number of training instances and build a very powerful model by applying only a few of these adjustments to our training data.

## Architecture:

After processing dataset images, the CNN model must be compiled to recognize all of the classes of information being used in the data. Normalization of the data must also be added to the data, equally balancing the classes with less images. We are using Keras as it is a free and open source project library that provides the artificial neural networks through the Python interface. The Keras basically, acts as an interface for the famous TensorFlow library. TensorFlow is also a free and an open source software library, for machine learning models and artificial intelligence. It has been adaptable to many different tasks, although it concentrates mostly on deep neural network training and inference. For Google's personal usage in research and production, the Google Brain team built it.

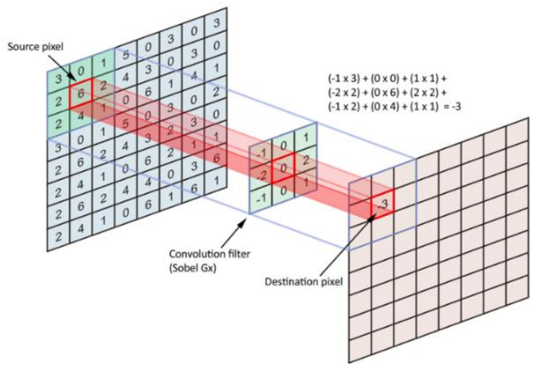
Implementation of CNN in Keras and TensorFlow, as show in figure

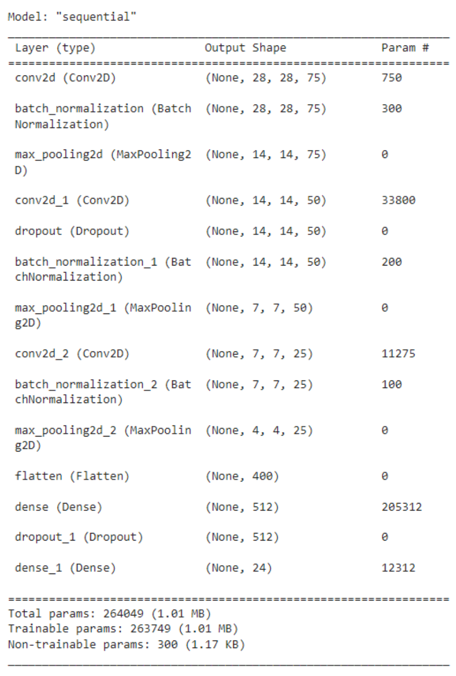


Keras is focused on being user-friendly, modular, and extendable in order to facilitate rapid experimentation with deep neural networks. The principal developer and maintainer of it is François Chollet, an engineer at Google, and it was created as a part of the ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) research project. Keras supports convolutional and recurrent neural networks in addition to normal standard neural networks. Additionally, widely used utility layers such as batch normalization, pooling, and dropout are supported.

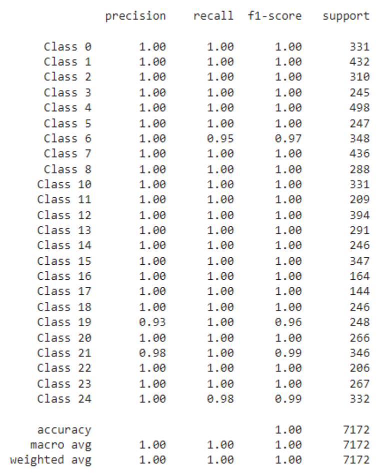
Convolutional neural networks are a unique kind of artificial intelligence application that process input from images using a unique mathematical technique for manipulating matrix known as the convolution operation.

- This goes on with a convolution, which produces a third, smaller matrix by multiplying two matrices.
- The network generates a feature map that describes an input image by applying a filter (or the kernel).
- A filter, typically a 2x2 or 3x3 matrix, has been moved throughout the image matrix during the convolution process. To get a single number that describes that input space, the corresponding numbers within both the matrices undergo multiplication and addition. The image is covered in iterations of this procedure. The following figure illustrates how this works.

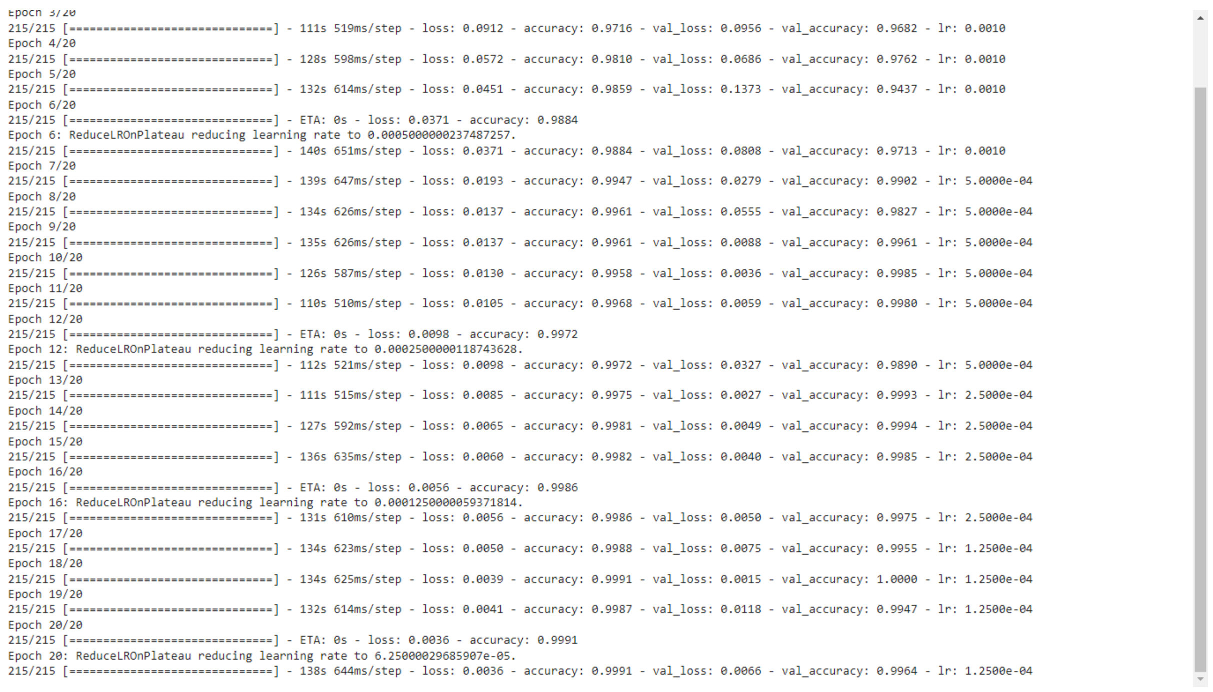

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter (Sobel Gx)
Destination pixel



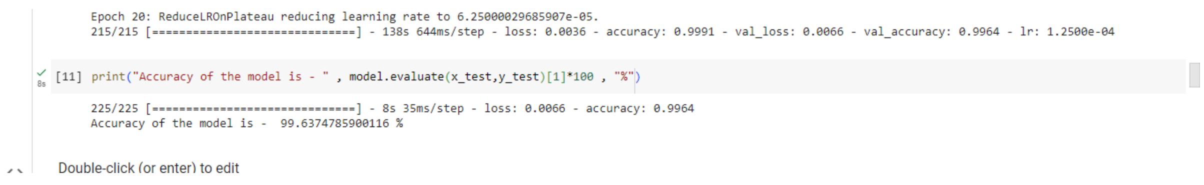
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 75)        750

 batch_normalization (Batch  (None, 28, 28, 75)        300
 Normalization)

 max_pooling2d (MaxPooling2  (None, 14, 14, 75)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 50)        33800

 dropout (Dropout)           (None, 14, 14, 50)        0

 batch_normalization_1 (Bat  (None, 14, 14, 50)        200
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 50)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 25)          11275

 batch_normalization_2 (Bat  (None, 7, 7, 25)          100
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 25)          0
 g2D)

 flatten (Flatten)           (None, 400)               0

 dense (Dense)               (None, 512)               205312

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 24)                12312

=================================================================
Total params: 264049 (1.01 MB)
Trainable params: 263749 (1.01 MB)
Non-trainable params: 300 (1.17 KB)
_________________________________________________________________
```



```
              precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00       331
     Class 1       1.00      1.00      1.00       432
     Class 2       1.00      1.00      1.00       310
     Class 3       1.00      1.00      1.00       245
     Class 4       1.00      1.00      1.00       498
     Class 5       1.00      1.00      1.00       247
     Class 6       1.00      0.95      0.97       348
     Class 7       1.00      1.00      1.00       436
     Class 8       1.00      1.00      1.00       288
    Class 10       1.00      1.00      1.00       331
    Class 11       1.00      1.00      1.00       209
    Class 12       1.00      1.00      1.00       394
    Class 13       1.00      1.00      1.00       291
    Class 14       1.00      1.00      1.00       246
    Class 15       1.00      1.00      1.00       347
    Class 16       1.00      1.00      1.00       164
    Class 17       1.00      1.00      1.00       144
    Class 18       1.00      1.00      1.00       246
    Class 19       0.93      1.00      0.96       248
    Class 20       1.00      1.00      1.00       266
    Class 21       0.98      1.00      0.99       346
    Class 22       1.00      1.00      1.00       206
    Class 23       1.00      1.00      1.00       267
    Class 24       1.00      0.98      0.99       332

    accuracy                           1.00      7172
   macro avg       1.00      1.00      1.00      7172
weighted avg       1.00      1.00      1.00      7172
```

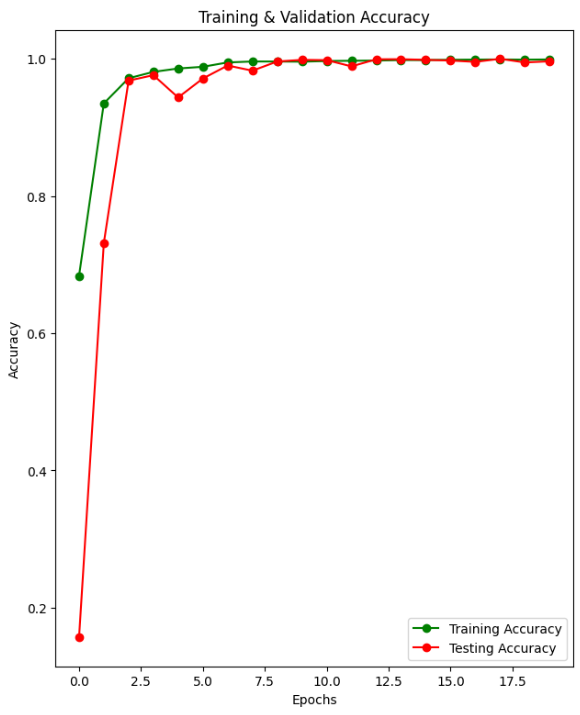


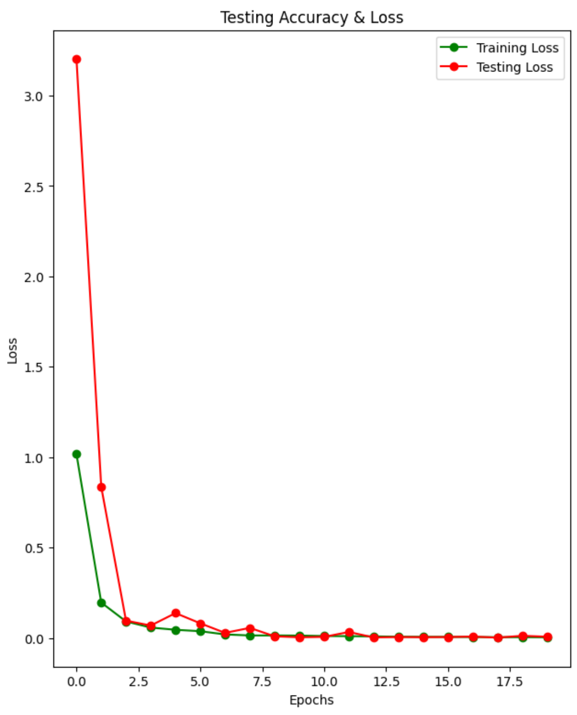
```
Epoch 3/20
215/215 [==============================] - 111s 519ms/step - loss: 0.0912 - accuracy: 0.9716 - val_loss: 0.0956 - val_accuracy: 0.9682 - lr: 0.0010
Epoch 4/20
215/215 [==============================] - 128s 598ms/step - loss: 0.0572 - accuracy: 0.9810 - val_loss: 0.0686 - val_accuracy: 0.9762 - lr: 0.0010
Epoch 5/20
215/215 [==============================] - 132s 614ms/step - loss: 0.0451 - accuracy: 0.9859 - val_loss: 0.1373 - val_accuracy: 0.9437 - lr: 0.0010
Epoch 6/20
215/215 [==============================] - ETA: 0s - loss: 0.0371 - accuracy: 0.9884
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
215/215 [==============================] - 140s 651ms/step - loss: 0.0371 - accuracy: 0.9884 - val_loss: 0.0808 - val_accuracy: 0.9713 - lr: 0.0010
Epoch 7/20
215/215 [==============================] - 139s 647ms/step - loss: 0.0193 - accuracy: 0.9947 - val_loss: 0.0279 - val_accuracy: 0.9902 - lr: 5.0000e-04
Epoch 8/20
215/215 [==============================] - 134s 626ms/step - loss: 0.0137 - accuracy: 0.9961 - val_loss: 0.0555 - val_accuracy: 0.9827 - lr: 5.0000e-04
Epoch 9/20
215/215 [==============================] - 135s 626ms/step - loss: 0.0137 - accuracy: 0.9961 - val_loss: 0.0088 - val_accuracy: 0.9961 - lr: 5.0000e-04
Epoch 10/20
215/215 [==============================] - 126s 587ms/step - loss: 0.0130 - accuracy: 0.9958 - val_loss: 0.0036 - val_accuracy: 0.9985 - lr: 5.0000e-04
Epoch 11/20
215/215 [==============================] - 110s 510ms/step - loss: 0.0105 - accuracy: 0.9968 - val_loss: 0.0059 - val_accuracy: 0.9980 - lr: 5.0000e-04
Epoch 12/20
215/215 [==============================] - ETA: 0s - loss: 0.0098 - accuracy: 0.9972
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
215/215 [==============================] - 112s 521ms/step - loss: 0.0098 - accuracy: 0.9972 - val_loss: 0.0327 - val_accuracy: 0.9890 - lr: 5.0000e-04
Epoch 13/20
215/215 [==============================] - 111s 515ms/step - loss: 0.0085 - accuracy: 0.9975 - val_loss: 0.0027 - val_accuracy: 0.9993 - lr: 2.5000e-04
Epoch 14/20
215/215 [==============================] - 127s 592ms/step - loss: 0.0065 - accuracy: 0.9981 - val_loss: 0.0049 - val_accuracy: 0.9994 - lr: 2.5000e-04
Epoch 15/20
215/215 [==============================] - 136s 635ms/step - loss: 0.0060 - accuracy: 0.9982 - val_loss: 0.0040 - val_accuracy: 0.9985 - lr: 2.5000e-04
Epoch 16/20
215/215 [==============================] - ETA: 0s - loss: 0.0056 - accuracy: 0.9986
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
215/215 [==============================] - 131s 610ms/step - loss: 0.0056 - accuracy: 0.9986 - val_loss: 0.0050 - val_accuracy: 0.9975 - lr: 2.5000e-04
Epoch 17/20
215/215 [==============================] - 134s 623ms/step - loss: 0.0050 - accuracy: 0.9988 - val_loss: 0.0075 - val_accuracy: 0.9955 - lr: 1.2500e-04
Epoch 18/20
215/215 [==============================] - 134s 625ms/step - loss: 0.0039 - accuracy: 0.9991 - val_loss: 0.0015 - val_accuracy: 1.0000 - lr: 1.2500e-04
Epoch 19/20
215/215 [==============================] - 132s 614ms/step - loss: 0.0041 - accuracy: 0.9987 - val_loss: 0.0118 - val_accuracy: 0.9947 - lr: 1.2500e-04
Epoch 20/20
215/215 [==============================] - ETA: 0s - loss: 0.0036 - accuracy: 0.9991
Epoch 20: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
215/215 [==============================] - 138s 644ms/step - loss: 0.0036 - accuracy: 0.9991 - val_loss: 0.0066 - val_accuracy: 0.9964 - lr: 1.2500e-04
```



```
    Epoch 20: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
    215/215 [==============================] - 138s 644ms/step - loss: 0.0036 - accuracy: 0.9991 - val_loss: 0.0066 - val_accuracy: 0.9964 - lr: 1.2500e-04

[11] print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")

    225/225 [==============================] - 8s 35ms/step - loss: 0.0066 - accuracy: 0.9964
    Accuracy of the model is -  99.6374785900116 %
```

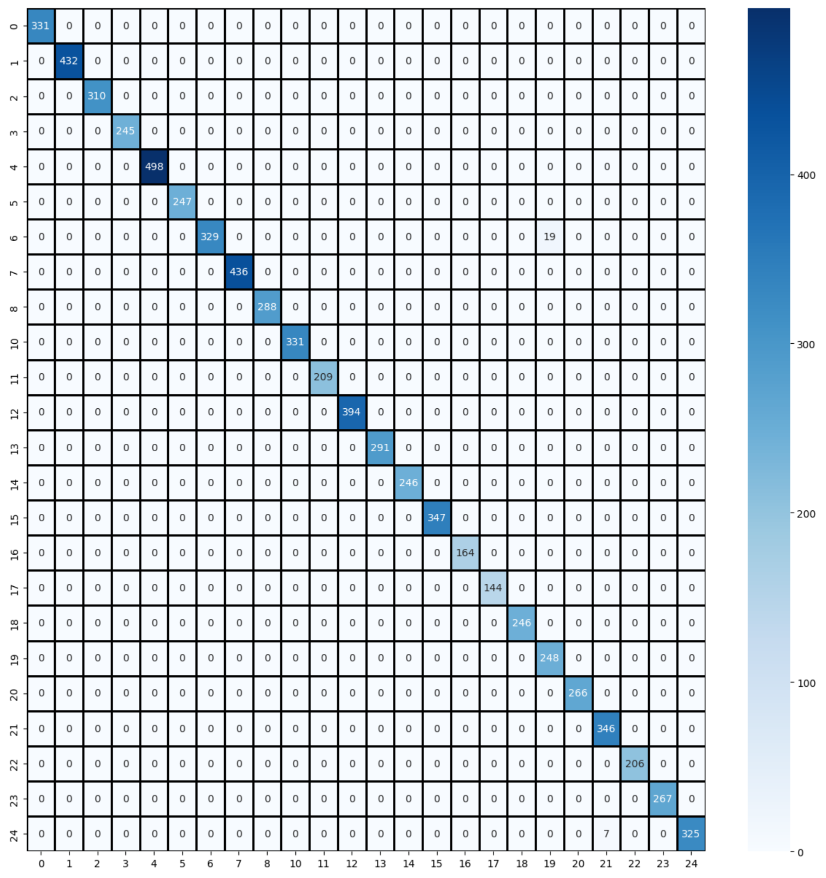
The summary of the model is given, we achieved the accuracy on average about 100% in the final run. Below figures are the model summary and evaluation graphs;





From the given confusion matrix below, it is clear that the trained model can label maximum of the data quite accurately. The confidence value of the sign/gesture it recognizes is also excellent. Accuracy of the model is about 100 % (as tested and give by the model)

(J and Z aren't included because of gesture/3D motions)



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 331 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 432 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 310 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 245 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 498 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 247 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 329 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 19 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 436 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 288 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 331 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 209 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 394 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 291 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 246 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 347 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 164 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 144 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 246 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 248 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 266 | 0 | 0 | 0 | 0 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 346 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 206 | 0 | 0 |
| 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 267 | 0 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 325 |

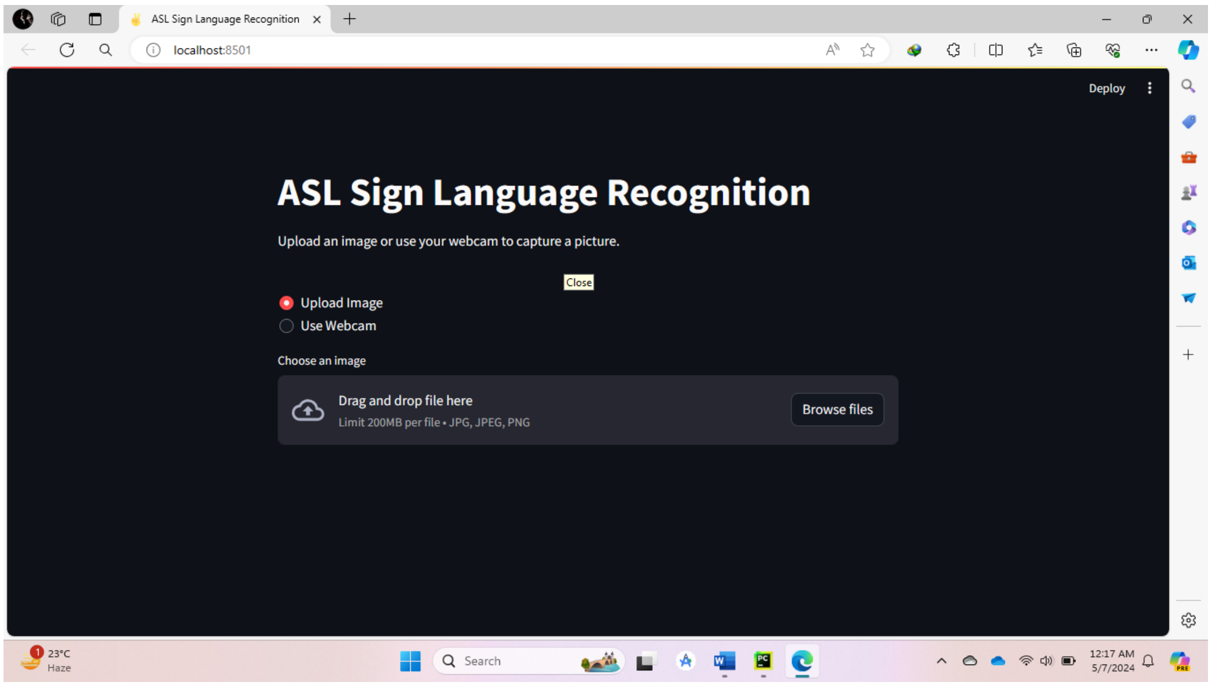
The developed model accurately detects and classifies Sign Language symbols with about 100% training accuracy.

Now the model is trained and tested and have 99% validation accuracy rate. Its time to develop and test the model on real application using webcam input and run this developed model on real time video stream.

## Realtime System Implementation and Evaluation:

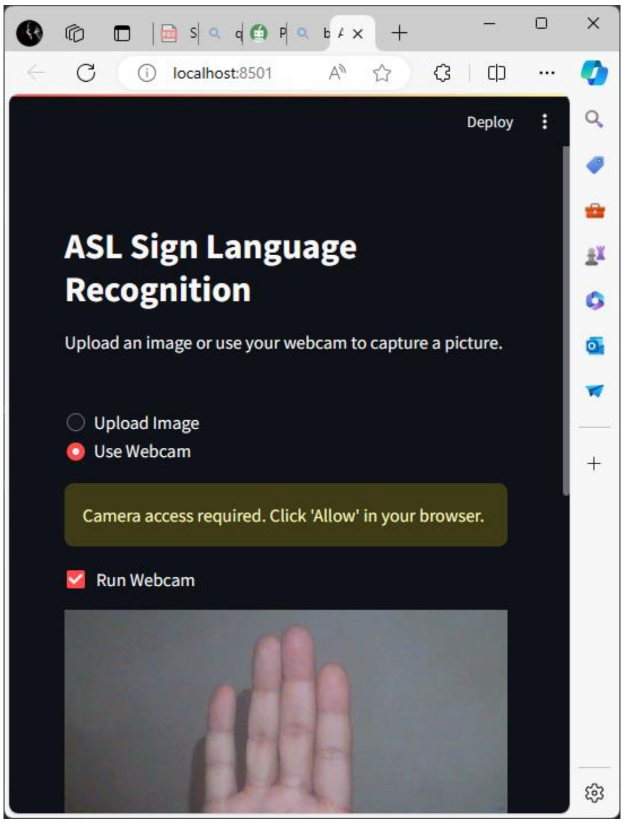
Now, we are using two popular libraries known as Streamlit and Open-CV. We can take webcam input and run our developed model on real time system or upload an image to predict the alphabet of ASL sign language. The driver code run the web based app using streamlit server:

```
# Streamlit run app.py
import streamlit as st_lit
import cv2
import numpy as np
from keras.models import load_model
from PIL import Image

# Load the trained model
model = load_model('ASL_Model.h5')
letter_predicted = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 
'U', 'V', 'W', 'X', 'Y']



# Function to preprocess and predict image
def predict_image(image):
    # Preprocess the image (adjust this part based on your preprocessing steps)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (28, 28))
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    image = np.expand_dims(image, axis=-1)

    # Predict
    predictions = model.predict(image)
    predicted_class = np.argmax(predictions)
    confidence = predictions[0][predicted_class]

    return letter_predicted[predicted_class], confidence



# Stream_lit app
def main():
    st_lit.set_page_config(page_title="ASL Sign Language Recognition", page_icon=":v:")
    st_lit.title("ASL Sign Language Recognition")

    st_lit.write("Upload an image or use your webcam to capture a picture.")

    # Choose between webcam and file upload
    option = st_lit.radio("", ("Upload Image", "Use Webcam"))

    if option == "Upload Image":
        # File uploader
        uploaded_file = st_lit.file_uploader("Choose an image", type=["jpg", "jpeg", "png"],
accept_multiple_files=False, key="fileUploader")

        if uploaded_file is not None:
            # Read and display the image
            image = np.array(Image.open(uploaded_file))
            st_lit.image(image, caption="Uploaded Image", use_column_width=True)
            # Predict
            if st_lit.button("Predict", key="predictButton"):
                prediction, confidence = predict_image(image)
                st_lit.success(f"Predicted character: {prediction}")
                st_lit.success(f"Confidence: {confidence * 100:.2f}%")

    elif option == "Use Webcam":
        # Webcam capture
        st_lit.warning("Camera access required. Click 'Allow' in your browser.")
        run_camera = st_lit.checkbox("Run Webcam", key="runWebcam")
        if run_camera:

            cap = cv2.VideoCapture(0)
            if not cap.isOpened():
                st_lit.error("Unable to access the camera. Please check your camera settings.")
                return

            _, frame = cap.read()
            st_lit.image(frame, caption="Webcam", channels="BGR", use_column_width=True)
            # Predict on captured image
            if st_lit.button("Predict", key="predictWebcam"):
                prediction, confidence = predict_image(frame)
                st_lit.success(f"Predicted character: {prediction}")
                st_lit.success(f"Confidence: {confidence * 100:.2f}%")


if __name__ == "__main__":
    main()
```



ASL Sign Language Recognition

Upload an image or use your webcam to capture a picture.


Deploy

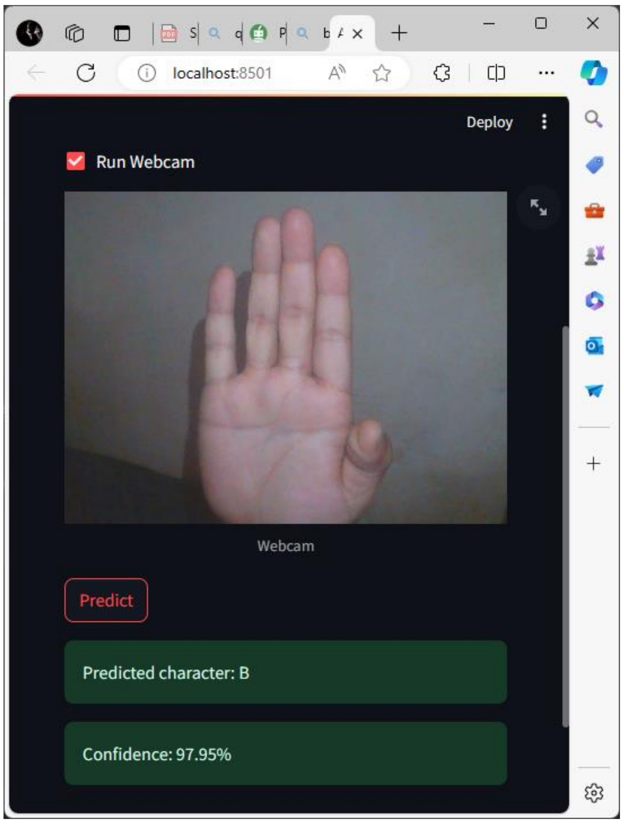
ASL Sign Language Recognition

Upload an image or use your webcam to capture a picture.

Upload Image
[✓ Use Webcam]

Camera access required. Click 'Allow' in your browser.

[✓ Run Webcam]

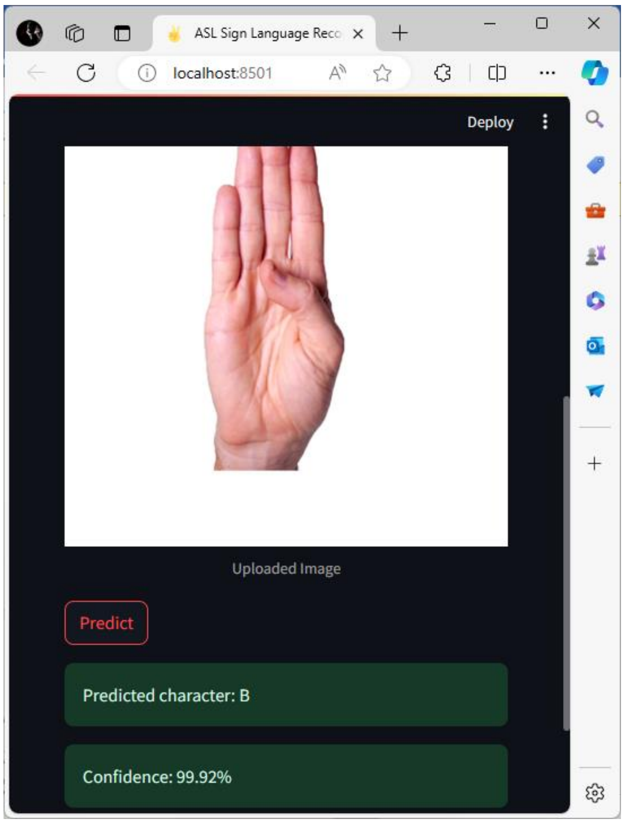

Deploy

[✓ Run Webcam]

Predicted character: B

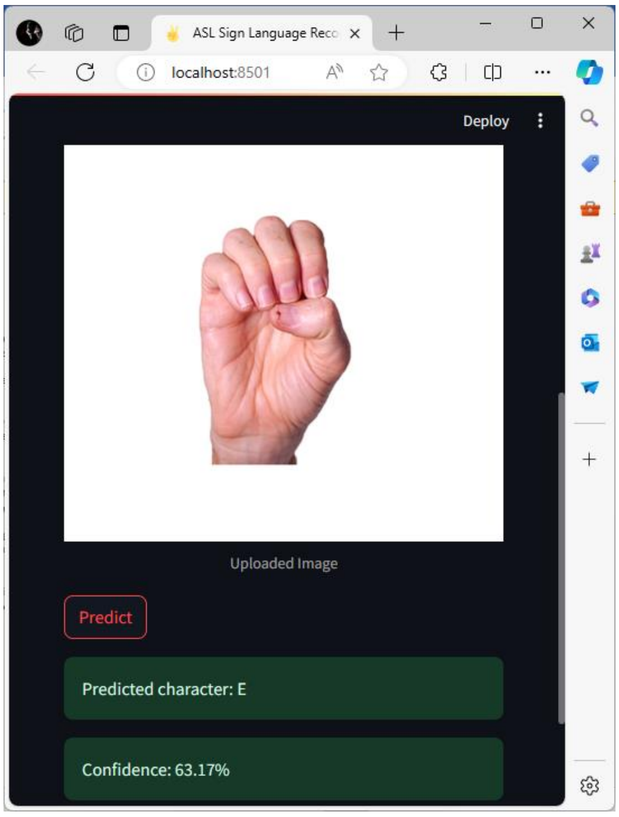
Confidence: 97.95%

As shown in the figure, the model accurately predicts the character being shown from the camera or by uploading an image. With the Predicted Character, the model also displays the confidence level of the classification from the trained CNN model.



Predicted character: B

Confidence: 99.92%



Predicted character: E

Confidence: 63.17%

## Conclusion:

In this article, a deep learning approach for sign language recognition system development is presented. The process consists of first training the data with a Convolutional Neural Network (CNN) technique and then utilizing it to identify sign language motions. As the accompanying graphs show, the study shows that the model obtains a recognition accuracy of 100% by using a significant amount of data sets for training each class.

Furthermore, a number of factors affect the system's real-time video recognition accuracy. When compared to situations when there is adequate lighting, accuracy is significantly reduced by insufficient light. Furthermore, the overall quality of the model has a major impact on how well the recognition system works. A model that has been trained on a large dataset performs more effectively. To create a strong classifier, it is imperative to include images that reflect different circumstances in the training process. Also, cam quality or image quality effects a lot in real time system performance.

## References

[1] "sign-language-mnist," [Online]. Available: https://www.kaggle.com/datasets/datamunge/sign-language-mnist.

[2] I. Cohen, N. Sebe, A. Garg, L. S. Chen and T. S. Huang, "Facial expression recognition from video sequences: temporal and static modeling," *Computer Vision and Image Undertaking,* vol. 91.

[3] D. Kothadiya, C. Bhatt, K. Sapariya, K. Patel, A.-B. Gil-González and J. M. Corchado, "Deepsign: Sign Language Detection and Recognition Using Deep Learning," *Electronics,* vol. 11, no. 11, 2022.

[4] B. Boulay, F. Bremond and M. Thonat, "Human Posture Recognition in Video Sequence," *IEEE International Workshop on VSPETS (Visual Surveillance and Performance Evaluation of Tracking and Surveillance),* 2003.

[5] C. Lee, K. K. Ng, C.-H. Chen, H. Lau, S. Chung and T. Tsoi, "American sign language recognition and training method with recurrent," *Expert Systems With Applications,* 2021.

[6] K. Tripathi, N. Baranwal and G. C. Nandi, "Continuous dynamic Indian Sign Language gesture recognition with invariant backgrounds by Kumud Tripathi," *Conference on Advances in Computing Communications and Informatics (ICACCI),* 2015.