

SIGN LANGUAGE RECOGNITION

MINOR PROJECT REPORT

Submitted by

NARENDRAN GR

20BCM028

Under the Guidance of

Mr. V. RAJASEKARAN MCA., M.Phil., MBA., M.Tech., MSc(Yoga).,(PhD)

Assistant professor

Department of Computer Science

In partial fulfillment of the requirements for the award of the degree of

Bachelor of Science in Computer Science

of Bharathiar University



PSG COLLEGE OF ARTS & SCIENCE DEPARTMENT OF COMPUTERSCIENCE

An Autonomous College-Affiliated to Bharathiar University

Accredited with 'A++' grade by NAAC (4th Cycle)

College with Potential for Excellence

(Status awarded by the UGC)

STAR College Status Awarded by DBT-MST An

ISO 9001:2015 Certified Institution

Coimbatore -641014

DECEMBER 2022

PSG COLLEGE OF ARTS & SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

An Autonomous College-Affiliated to Bharathiar University

Accredited with 'A++' grade by NAAC (4th Cycle)

College with Potential for Excellence

(Status awarded by the UGC)

STAR College Status Awarded by DBT-MST

An ISO 9001:2015 Certified Institution

Coimbatore -641 014

CERTIFICATE

This is to certify that this Project work entitled “**SIGN LANGUAGE RECOGNITION**” is a bonafide record of work done by **NARENDRAN G R** for the award of Degree of Bachelor of Science in Computer Science of Bharathiar University.

Signature of the Faculty Guide

Signature of the HoD

Submitted for Viva-Voce Examination held on_____.

Internal Examiner

External Examiner

DECLARATION

I, **NARENDRAN G R (20BCM028)**, hereby declare that this project work entitled **“SIGN LANGUAGE RECOGNITION”**, is submitted to PSG College of Arts & Science (AUTONOMOUS), Coimbatore in partial fulfillment for the award of degree is a record of original work done by me under the guidance and supervision of **Mr. V. RAJASEKARAN MCA., M.Phil., MBA., M.Tech., MSc(Yoga), (PhD)** Assistant Professor Department of Computer Science, PSG College of Arts & Science, Coimbatore.

This Project work has not been submitted by me for the award of any other Degree/ Diploma/ Associate ship/ Fellowship or any other similar degree to any other university.

PLACE: Coimbatore

NARENDRAN G R

DATE:

(20BCM028)

ACKNOWLEDGEMENT

My venture stands imperfect without dedicating find gratitude to a few people who have contributed a lot towards the victorious completion for find project. I would like to thank **Mr.L.Gopalakrishnan**, Managing Trustee, PSG & Sons Charities, for providing me a prospect and surroundings that made the work possible. I take this opportunity to express a deep sense of gratitude to **Dr T Kannaian, Secretary** of PSG College of Arts and Science, Coimbatore for permitting and doing the needful towards the successful completion of the project.

I express a deep sense of gratitude and sincere thanks to Principal **Dr D Brindha M.Sc., M.Phil., Ph.D., M.A(yoga)** for her valuable advice and concern on students.

I am very thankful to **Dr A Anguraj., M.Sc., M.Phil., PhD., Vice Principal (Academics), Prof M Umarani., M.Com., M.Phil.,** Faculty-In-Charge (Student affairs) for their support.

I kindly and sincerely thank **Dr.T. Revathi MCA., MPhil., PhD., Associate Professor and Head of the Department of Computer Science** for her whole hearted help to complete this project successfully by giving valuable suggestions. I convey find heartiest and passionate sense of thankfulness to find project guide **Mr. V. Rajasekaran MCA., M.Phil., MBA., M.Tech., MSc(Yoga).,(PhD)** Assistant professor, Department of Computer Science, for his timely suggestion which had enabled me in completing the project successfully.

This note of this acknowledgement will be incomplete without paying for heartfelt devotion to find parents, find friends and other people, for their blessings, encouragement, financial support and the patience, without which it would have been possible for me to complete this job.

S.no.	Table of Contents	page no.
1	Introduction	1
	1.1 Abstract	1
	1.2 Problem statement	1
	1.3 Project Overview	2
2	Modules	4
3	System Configuration	5
	3.1 Hardware Specifications	5
	3.2 Software Specifications	5
	3.3 Requirements Python	6
	3.4 Requirements Conda	6
4	System Analysis	7
	4.1 Existing System	7
	4.2 Proposed System	7
5	Software Overview	8
	5.1 Python	8
	5.2 Conda	8
	5.3 Tensorflow	8
	5.4 Keras	9
	5.5 OpenCV	9
6	Keywords & Definition	10
	6.1 Feature Extraction and Representation	10
	6.2 Artificial Neural Network (ANN)	10
	6.3 Convolutional Neural Network (CNN)	11

	6.3.1 Convolutional Layer	12
	6.3.2 Pooling Layer	12
	6.3.2.1 Max Pooling	12
	6.3.2.2 Average Pooling	13
	6.3.3 Fully connected Layer	13
	6.3.4 Final Output Layer	13
7	Methodology	14
	7.1 Data Set Generation	14
	7.2 Gesture Classification	15
	7.3 Finger spelling sentence Formation	18
8	System Design	19
	8.1 System Build	19
	8.2 System Processing Diagram	20
	8.3 Data Flow Diagram	21
	8.4 Use Case Diagram	23
9	Training & Testing	24
10	Challenges Faced	26
11	Result	27
12	Conclusion	29
13	Future Scope	30
14	Bibliography	31
15	Appendix	32

1.INTRODUCTION

1.1 ABSTRACT

Sign language is one of the oldest and most natural forms of language for communication, but since most people do not know sign language and interpreters are very difficult to come by, we have come up with a real time method using neural networks for fingerspelling based on American sign language.

An enhanced image processing model for sign language recognition was developed. The system was aimed at helping in the education and interaction with the speech impaired in the learning environment. Object-Oriented Analysis and Design Methodology were adopted in this approach. The system was implemented using MATLAB. The SURF algorithm was used to perform feature extraction on the input images in order to enhance detection of interest points on the images.

In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures. Our method provides 95.7 % accuracy for the 26 letters of the alphabet.

1.2 PROBLEM STATEMENT

Speech impaired people use hand signs and gestures to communicate. Normal people face difficulty in understanding their language. Hence there is a need for a system which recognizes the different signs, gestures and conveys the information to the normal people.

It bridges the gap between physically challenged people and normal people.

1.3 PROJECT OVERVIEW



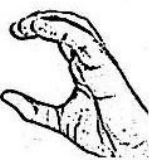



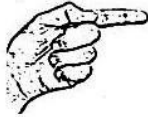
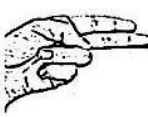

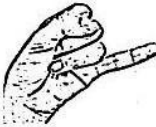





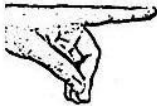










American sign language is a predominant sign language Since the only disability Deaf and Dumb (hereby referred to as D&M) people have is communication related and since they cannot use spoken languages, the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behavior and visuals. D&M people make use of their hands to express different gestures to express their ideas with other people. Gestures are the non-verbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language. A sign language is a language which uses gestures instead of sound to convey meaning combining hand-shapes, orientation and movement of the hands, arms or body, facial expressions and lip-patterns. Contrary to popular belief, sign language is not international. These vary from region to region.

Sign language is a visual language and consists of 3 major components:

FingerSpelling	Word level Sign vocabulary	Non-manual features
Used to spell words letter by letter.	Used for the majority of the communication.	Facial expressions and tongue ,mouth and body position.

Minimizing the verbal exchange gap among D&M and non-D&M people turns into a desire to make certain effective conversation among all. Sign language translation is among one of the most growing lines of research and it enables the maximum natural manner of communication for those with hearing impairments. A hand gesture recognition system offers an opportunity for deaf people to talk with vocal humans without the need of an interpreter. The system is built for the automated conversion of ASL into textual content and speech.

In this project we primarily focus on producing a model which can recognize Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the images below.

BLANK						
						
						
						

American Sign Language Alphabets

2.MODULES

2.1 Module Description

A Module Description provides detailed information about the module and its supported components, which is accessible in different manners.

The modules are,

- Input Gesture
- Predict Gesture
- Word Creation
- Sentence Creation

Input Gesture

Giving gesture (Hand sign) to the model as an input with the Good quality Camera.

There should be proper light in the background with no objects or noises or symbols behind Gesture.

Predict Gesture

With the help of Input given, the model will classify the gesture and Identify the symbol.

Word Creation

The model will predict and display the symbol. If that symbol repeats for 20 frames, then that symbol is created as a word.

Sentence Creation

If the Blank symbol (ie., No gesture) is given for 20 frames then the word will be created as Sentence and the model will predict for the next word.

3.SYSTEM CONFIGURATION

3.1 HARDWARE REQUIREMENTS

COMPONENTS	REQUIREMENTS
CPU	AMD GPU device X86_64 processor
Camera	GOOD QUALITY 3mp & higher
RAM	4GB & Higher

3.2 SOFTWARE REQUIREMENTS

COMPONENTS	REQUIREMENTS
Operating System	Windows 7 or higher (64-bit)
Python	Python 3.7
PIP	Version 19.0 or higher
Visual C++	Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019
NVIDIA	CUDA Toolkit 11.2
NVIDIA	cuDNN SDK 8.1.0

3.3 REQUIREMENTS PYTHON

Python modules:

- OPEN CV(computer vision)
- TENSORFLOW
- OS
- NUMPY
- STRINGS
- PILLOW
- TKINTER
- KERAS
- PYENCHANT (Python bindings for the Enchant spell checking system)
- CYHUNSPELL (A wrapper on hunspell for use in Python)

3.4 REQUIREMENTS CONDA

Create Virtual Environment

```
conda create --name my_env python=3.7
```

Conda modules:

- PROTOBUF
- PYQT
- SCIKIT-LEARN
- SCIPY

4.SYSTEM ANALYSIS

4.1 EXISTING SYSTEM

The existing system is a manual process. It is difficult to understand the sign of the D&M people for the non D&M people. Normal people are not aware of their signs. It is time consuming to understand their language.

DISADVANTAGES OF EXISTING SYSTEM

- Unable to understand
- Time consumption

4.2 PROPOSED SYSTEM

The objective of the proposed system is to reduce the time consuming and make the system more user friendly, efficient, accurate and fast processing. It reduces the difficulty to understand the sign language of the D&M people to the non-D&M people.

ADVANTAGES OF PROPOSED SYSTEM

- More user friendly
- Understanding of Sign Language
- Reduces the time consuming

5.SOFTWARE OVERVIEW

5.1 PYTHON

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python is used in machine learning for a variety of reasons.

1. It is a very concise and readable language, which makes it easy to develop algorithms.
2. Python has a large number of well-developed libraries, which can be used to develop machine learning algorithms.
3. Python is fast enough for most machine learning tasks.

5.2 CONDA

Conda is an open source package management system and environment management system for installing multiple versions of software packages and their dependencies and switching easily between them. It works on Linux, OS X and Windows, and was created for Python programs but can package and distribute any software.

5.3 TENSORFLOW

TensorFlow is an end-to-end open-source platform for Machine Learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in Machine Learning and developers easily build and deploy Machine Learning powered applications.

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition.

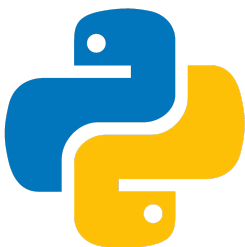
5.4 KERAS

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

5.5 OpenCV:

OpenCV (Open-Source Computer Vision) is an open-source library of programming functions used for real-time computer-vision.

It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE.



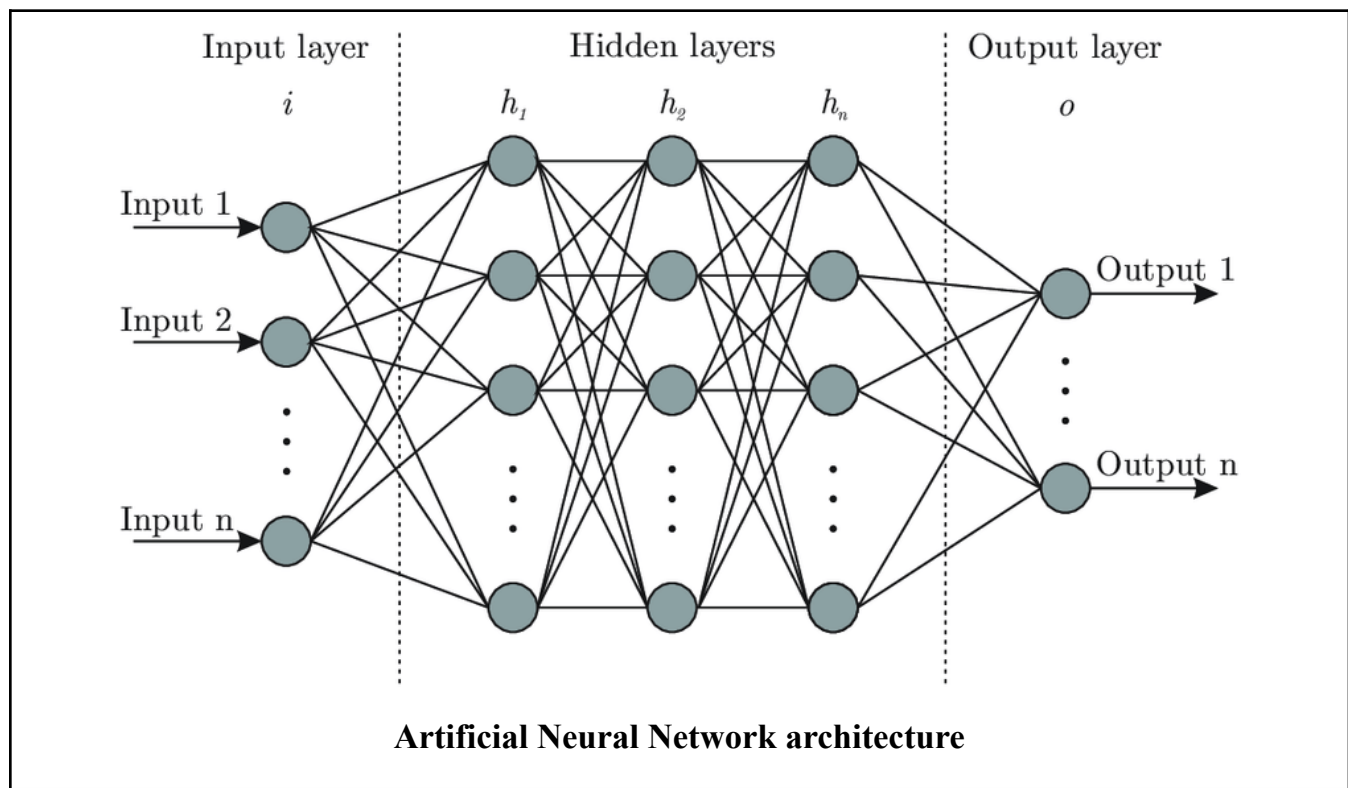
6.KEY WORDS & DEFINITIONS

6.1 Feature Extraction and Representation:

The representation of an image as a 3D matrix having dimension as of height and width of the image and the value of each pixel as depth (1 in case of Grayscale and 3 in case of RGB). Further, these pixel values are used for extracting useful features using CNN.

6.2 Artificial Neural Network (ANN):

Artificial Neural Network is a connection of neurons, replicating the structure of the human brain. Each connection of a neuron transfers information to another neuron. Inputs are fed into the first layer of neurons which processes it and transfers to another layer of neurons called hidden layers. After processing information through multiple layers of hidden layers, information is passed to the final output layer.

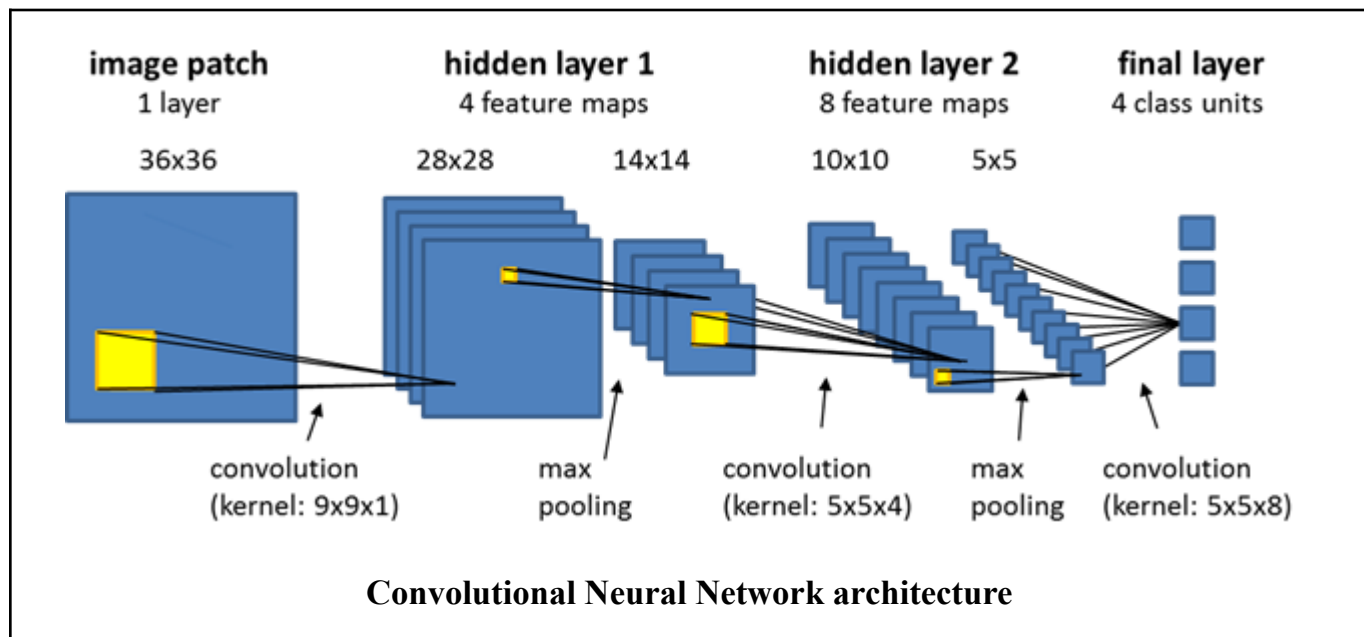


These are capable of learning and have to be trained. There are different learning strategies:

- Unsupervised Learning
- Supervised Learning
- Reinforcement Learning

6.3 Convolutional Neural Network (CNN):

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.



6.3.1 Convolution Layer

In convolution layer we take a small window size [typically of length 5×5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slide the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position.

As we continue this process we will create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color.

6.3.2 Pooling Layer

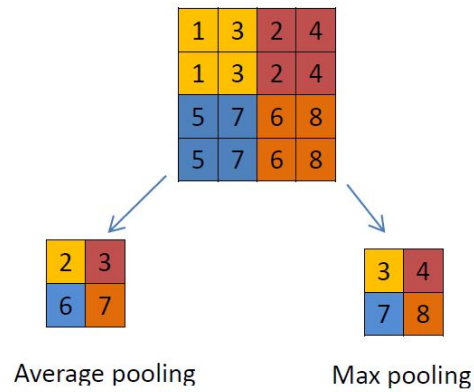
We use a pooling layer to decrease the size of the activation matrix and ultimately reduce the learnable parameters. There are two types of pooling:

6.3.2.1 Max Pooling:

In max pooling we take a window size [for example window of size 2×2], and only take the maximum of 4 values. We'll slide this window and continue this process, so we'll finally get an activation matrix half of its original size.

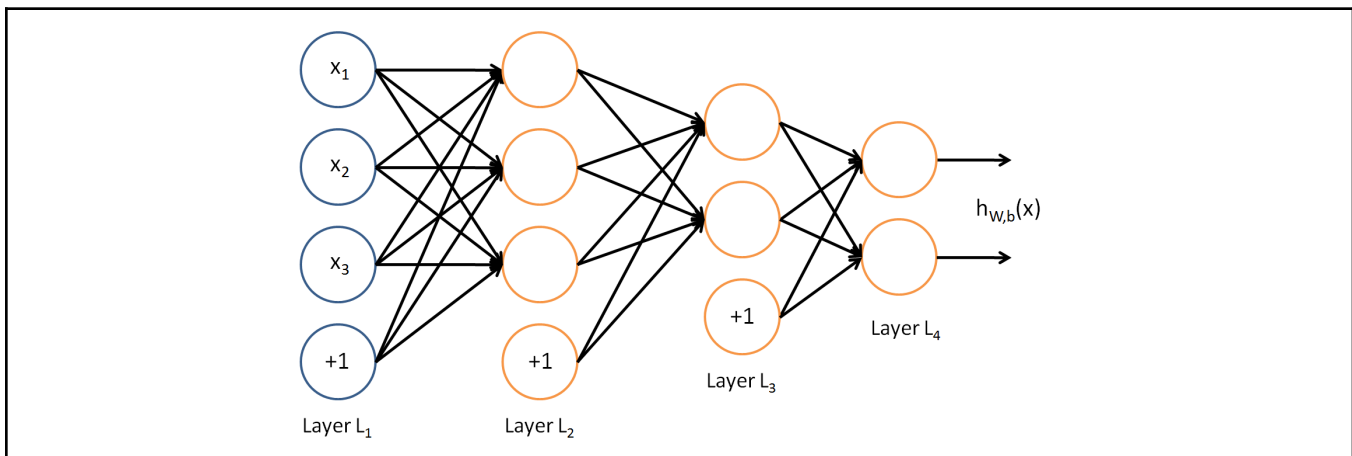
6.3.2.2 Average Pooling:

In average pooling, we take advantage of all Values in a window.



6.3.3 Fully Connected Layer

In convolution layer, neurons are connected only to a local region, while in a fully connected region, we will connect all the inputs to neurons.



6.3.4 Final Output Layer

After getting values from a fully connected layer, we will connect them to the final layer of neurons having count equal to total number of classes], that will predict the probability of each image to be in different classes.

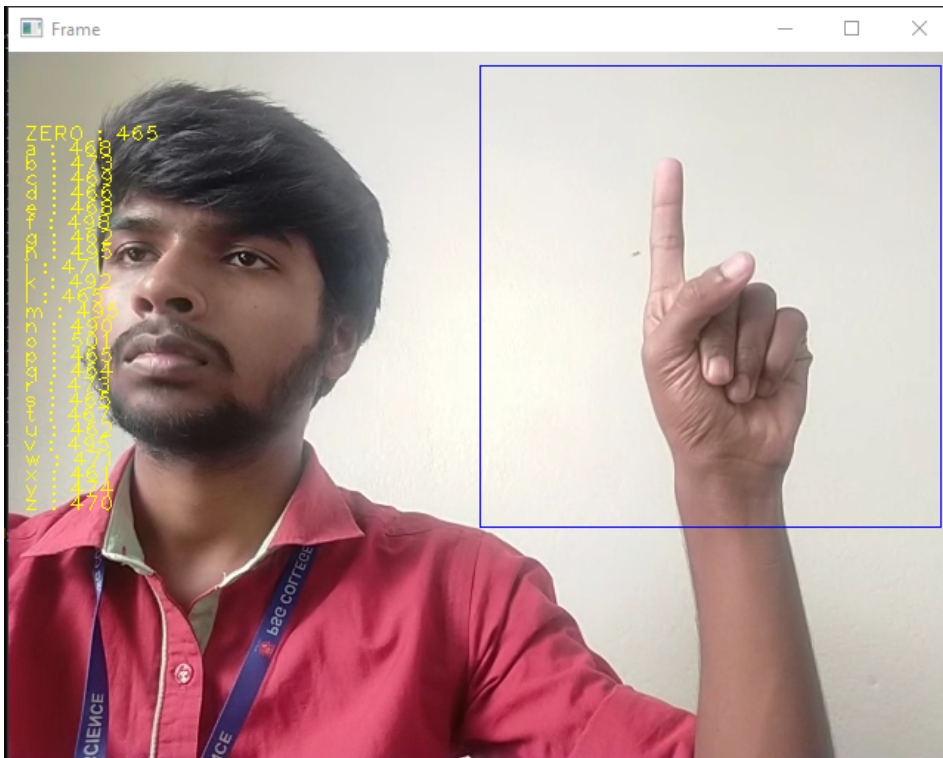
7.Methodology

The system is a vision-based approach. All signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

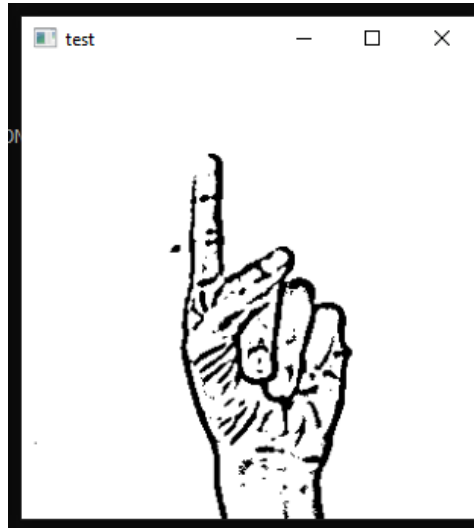
7.1 Data Set Generation

For the project we tried to find already made datasets but we couldn't find datasets in the form of raw images that matched our requirements. All we could find were the datasets in the form of RGB values. Hence, we decided to create our own data set. Steps we followed to create our data set are as follows. We used the Open computer vision (OpenCV) library in order to produce our dataset. Firstly, we captured around 400 images of each of the symbols in ASL (American Sign Language) for training purposes and around 150 images per symbol for testing purposes.

First, we capture each frame shown by the webcam of our machine. In each frame we define a Region Of Interest (ROI) which is denoted by a blue bounded square as shown in the image below:



Then, we apply Gaussian Blur Filter to our image which helps us extract various features of our image. The image, after applying Gaussian Blur, looks as follows:



7.2 GESTURE CLASSIFICATION

Our approach uses two layers of algorithm to predict the final symbol of the user

Algorithm Layer 1

- Apply Gaussian Blur filter and threshold to the frame taken with openCV to get the processed image after feature extraction.
- This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word.
- Space between the words is considered using the blank symbol.

Algorithm Layer 2

- We detect various sets of symbols which show similar results on getting detected.
- We then classify between those sets using classifiers made for those sets only.

Layer 1

CNN Model:

- **1st Convolution Layer:**

The input picture has a resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.

- **1st Pooling Layer:**

The pictures are down sampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is down sampled to 63x63 pixels.

- **2nd Convolution Layer:**

Now, these 63 x 63 from the output of the first pooling layer serve as an input to the second convolutional layer. It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 60 x 60 pixel image.

- **2nd Pooling Layer:**

The resulting images are down sampled again using a max pool of 2x2 and is reduced to 30 x 30 resolution of images.

- **1st Densely Connected Layer:**

Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an array of $30 \times 30 \times 32 = 28800$ values. The input to this layer is an array of 28800 values. The output of these layers is fed to the 2nd Densely Connected Layer. We are using a dropout layer of value 0.5 to avoid overfitting.

- **2nd Densely Connected Layer:**

Now the output from the 1st Densely Connected Layer is used as an input to a fully connected layer with 96 neurons.

- **Final layer:**

The output of the 2nd Densely Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying (alphabets + blank symbol).

- **Activation Function:**

We have used ReLU (Rectified Linear Unit) in each of the layers (convolutional as well as fully connected neurons).

ReLU calculates $\max(x, 0)$ for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features. It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time.

- **Pooling Layer:**

We apply Max pooling to the input image with a pool size of (2, 2) with ReLU activation function. This reduces the amount of parameters thus lessening the computation cost and reduces overfitting.

- **Dropout Layers:**

The problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. This layer "drops out" a random set of activations in that layer by setting them to zero. The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out.

- **Optimizer:**

We have used Adam optimizer for updating the model in response to the output of the loss function.

Adam optimizer combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm (ADA GRAD) and root mean square propagation (RMSProp).

Layer 2:

We are using two layers of algorithms to verify and predict symbols which are more similar to each other so that we can get as close as we can get to detect the symbol shown. In our testing we found that following symbols were not showing properly and were giving other symbols also:

- For D : R and U
- For U : D and R
- For I : T, D, K and I
- For S : M and N

So, to handle above cases we made three different classifiers for classifying these sets:

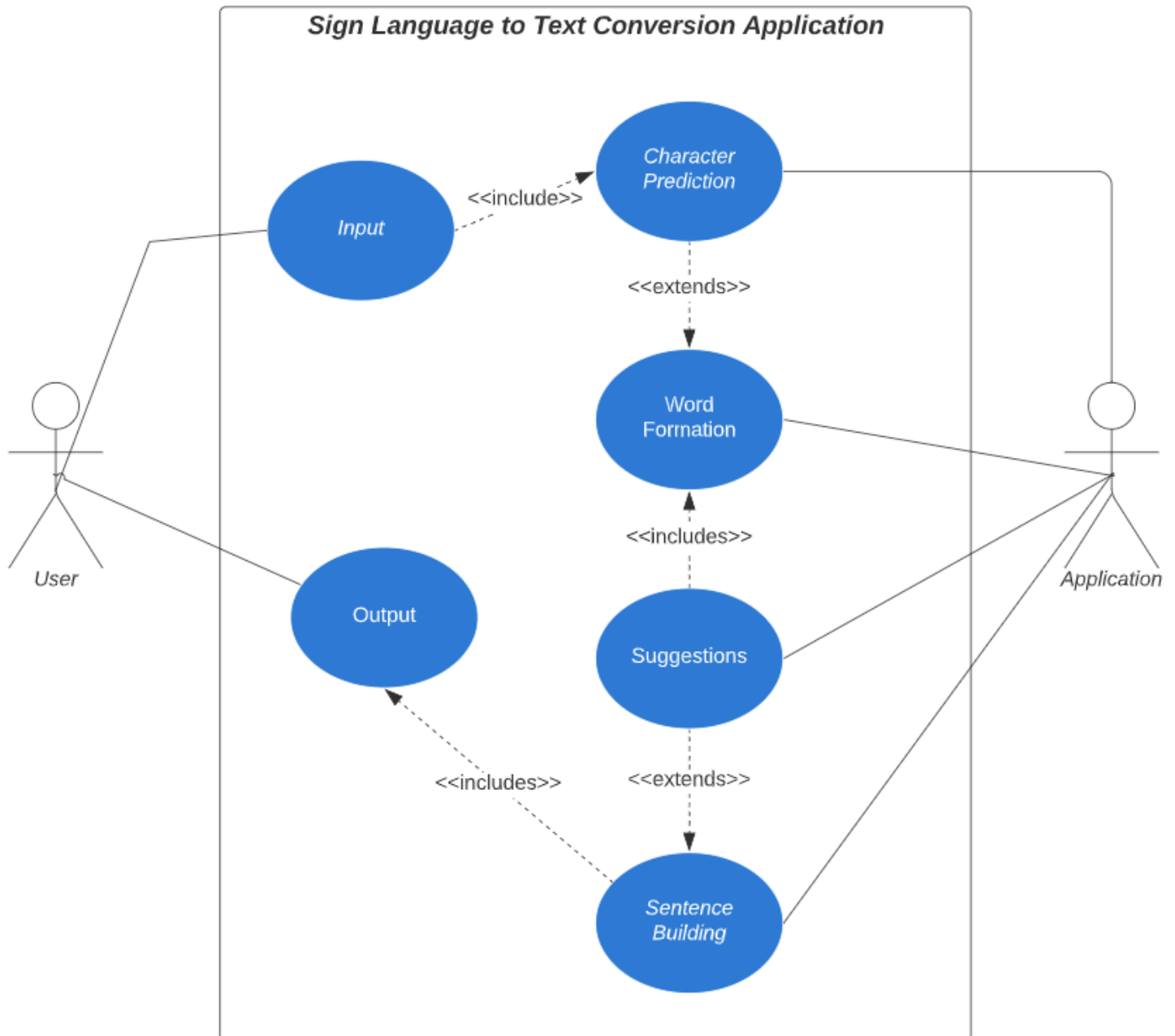
- {D, R, U}
- {T, K, D, I}
- {S, M, N}

7.3 Finger Spelling Sentence Formation Implementation:

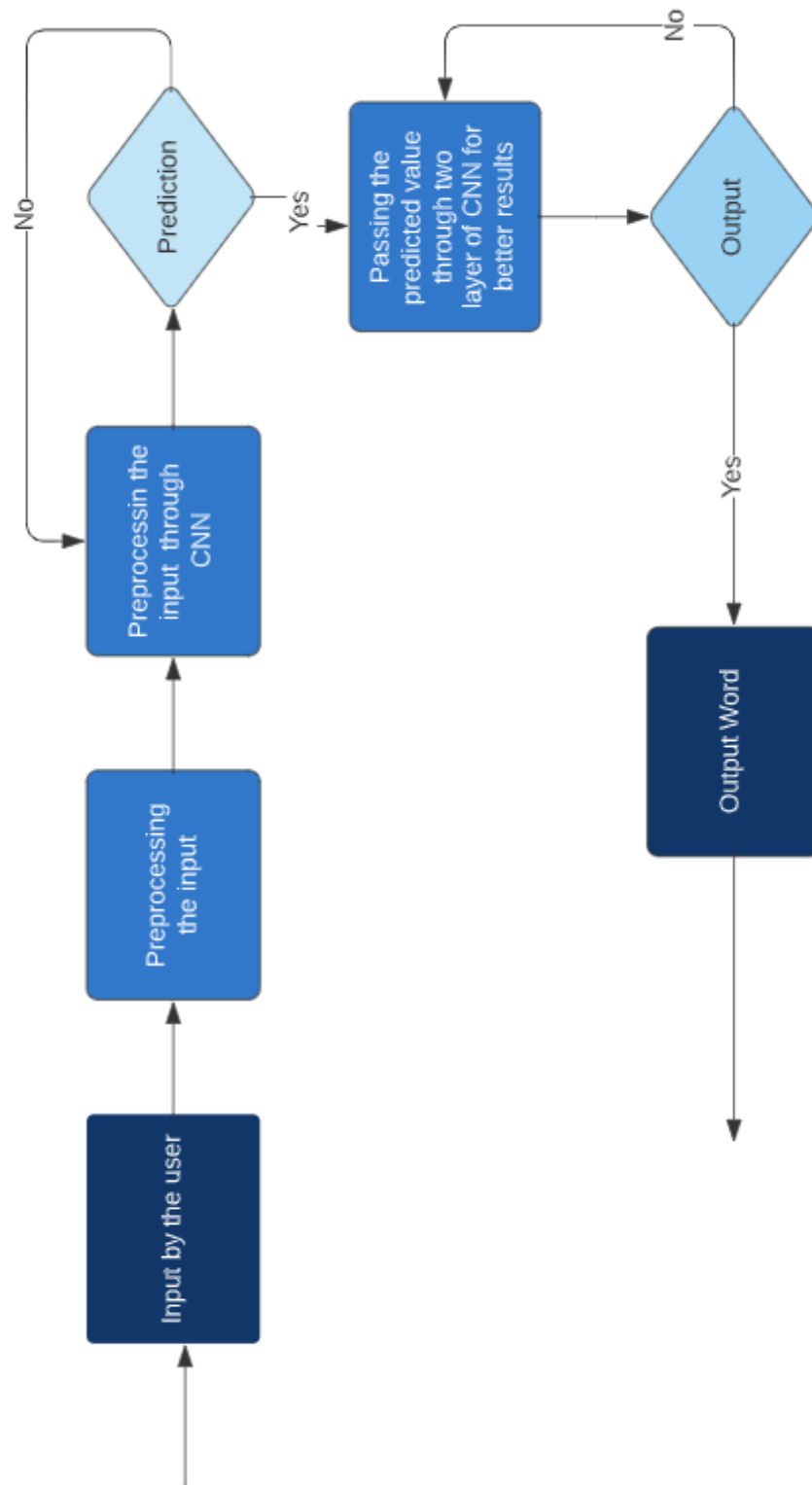
- Whenever the count of a letter detected exceeds a specific value and no other letter is close to it by a threshold, we print the letter and add it to the current string (In our code we kept the value as 20 and difference threshold as 19).
- Otherwise, we clear the current dictionary which has the count of detections of the present symbol to avoid the probability of a wrong letter getting predicted.
- Whenever the count of a blank (plain background) detected exceeds a specific value and if the current buffer is empty no spaces are detected.
- In other cases it predicts the end of word by printing a space and the current gets appended to the sentence below.

8.SYSTEM DESIGN

8.1 SYSTEM BUILD



8.2 SYSTEM PROCESSING DIAGRAM



8.3 DATA FLOW DIAGRAM(DFD)

The DFD is also known as a bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyze an existing system or model of a new one. A DFD can often visually “say” things that would be hard to explain in words and they work for both technical and non- technical.

There are four components in DFD:

1. External Entity
2. Process
3. Data Flow
4. Data Store

1) External Entity:

It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram. These are sources and destinations of the system’s input and output.

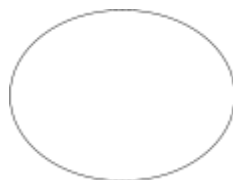
Representation:



2) Process:

It is just like a function that changes the data, producing an output. It might perform computations to sort data based on logic or direct the dataflow based on business rules.

Representation:



3) Data Flow:

A dataflow represents a package of information flowing between two objects in the data-flow diagram. Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

Representation:



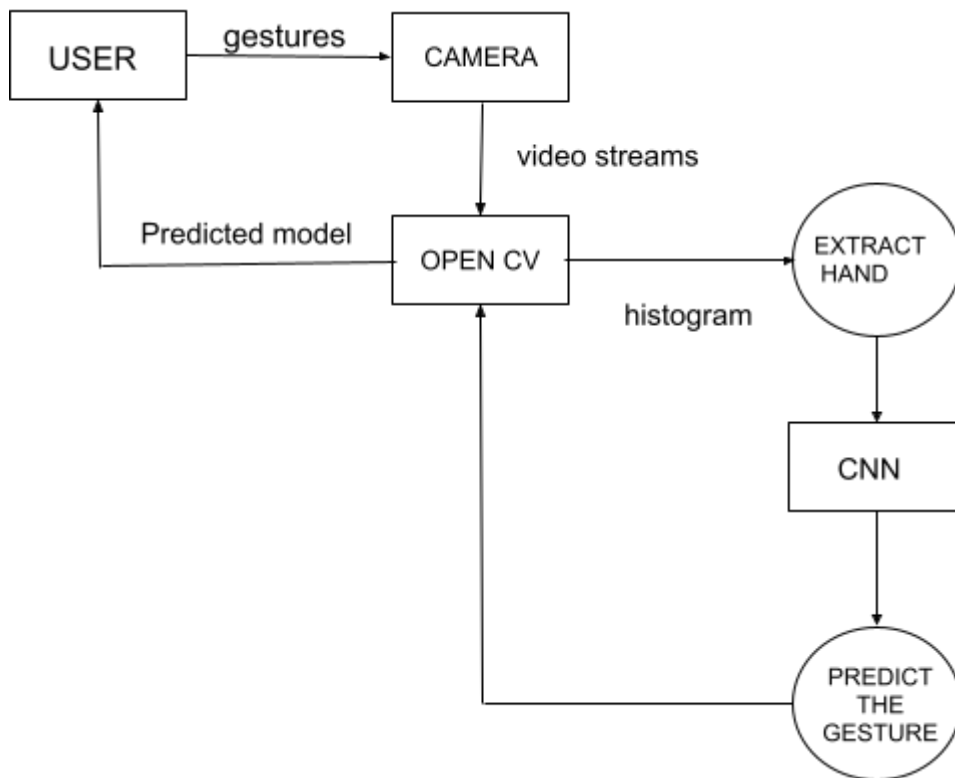
4) Data Store

These are the files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label.

Representation:



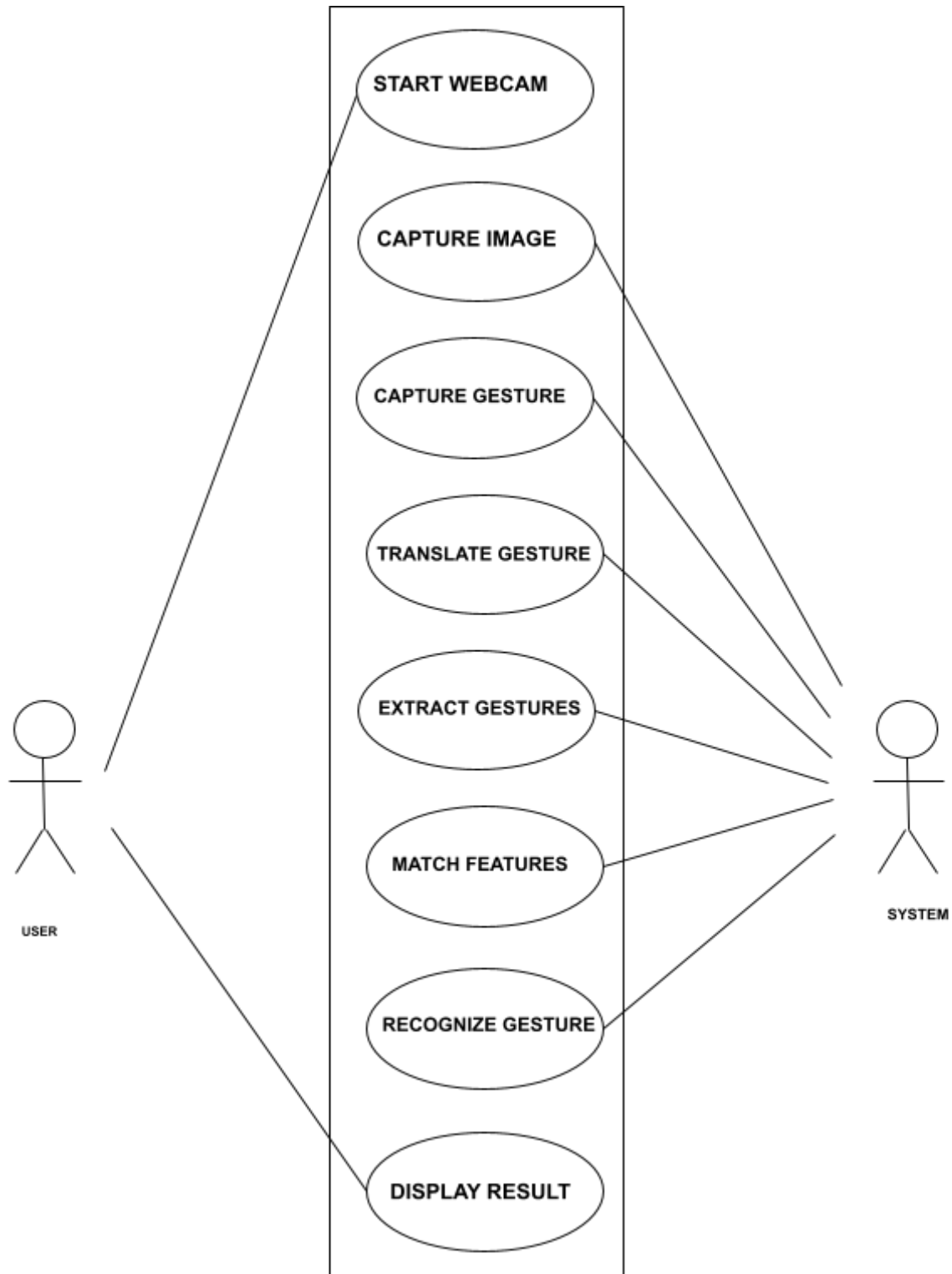
DFD



8.4 USE CASE DIAGRAM

A use case diagram is a graphical depiction of a user's possible interactions with a system.

The use cases are represented by either circles or ellipses.



9. TRAINING & TESTING

We convert our input images (RGB) into grayscale and apply gaussian blur to remove unnecessary noise. We apply an adaptive threshold to extract our hand from the background and resize our images to 128 x 128.

We feed the input images after pre-processing to our model for training and testing after applying all the operations mentioned above.

The prediction layer estimates how likely the image will fall under one of the classes. So, the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using the SoftMax function.

```
C:\Windows\system32\cmd.exe
Use the following CPU instructions in performance-critical operations: AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"

Layer (type)                 Output Shape              Param #
-----
conv2d (Conv2D)              (None, 126, 126, 32)     320
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)       0
conv2d_1 (Conv2D)            (None, 61, 61, 32)       9248
max_pooling2d_1 (MaxPooling (None, 30, 30, 32)       0
2D)
flatten (Flatten)            (None, 28800)             0
dense (Dense)                (None, 128)              3686528
dropout (Dropout)            (None, 128)              0
dense_1 (Dense)              (None, 96)               12384
dropout_1 (Dropout)          (None, 96)               0
dense_2 (Dense)              (None, 64)               6208
dense_3 (Dense)              (None, 27)              1755

Total params: 3,716,443
Trainable params: 3,716,443
Non-trainable params: 0


Found 13306 images belonging to 27 classes.
Found 4268 images belonging to 27 classes.
Epoch 1/5
1330/1330 [=====] - 1561s 1s/step - loss: 2.2741 - accuracy: 0.3076 - val_loss: 0.4183 - val_accuracy: 0.8662
Epoch 2/5
1330/1330 [=====] - 1318s 991ms/step - loss: 0.7228 - accuracy: 0.7539 - val_loss: 0.1001 - val_accuracy: 0.9704
Epoch 3/5
1330/1330 [=====] - 710s 533ms/step - loss: 0.4327 - accuracy: 0.8576 - val_loss: 0.0517 - val_accuracy: 0.9831
Epoch 4/5
1330/1330 [=====] - 920s 692ms/step - loss: 0.3037 - accuracy: 0.9027 - val_loss: 0.0234 - val_accuracy: 0.9939
Epoch 5/5
1330/1330 [=====] - 791s 594ms/step - loss: 0.2515 - accuracy: 0.9217 - val_loss: 0.0078 - val_accuracy: 0.9977
Model Saved
Weights saved
```

TRAINING MODEL

At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which are not the same as labeled value and is zero exactly when it is equal to the labeled value. Therefore, we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy.

As we have found out the cross-entropy function, we have optimized it using Gradient Descent. In fact with the best gradient descent optimizer is called Adam Optimizer.

Sign Language To Text Conversion



Character : Y

Word : BOY

Sentence :

TESTING

10.CHALLENGES FACED

- There were many challenges faced during the project. The very first issue we faced was concerning the data set. We wanted to deal with raw images and that too square images as CNN in Keras since it is much more convenient working with only square images.
- We couldn't find any existing data set as per our requirements and hence we decided to make our own data set. The second issue was to select a filter which we could apply on our images so that proper features of the images could be obtained and hence then we could provide that image as input for CNN model.
- We tried various filters including binary threshold, canny edge detection, Gaussian blur etc. but finally settled with Gaussian Blur Filter.
- More issues were faced relating to the accuracy of the model we had trained in the earlier phases. This problem was eventually improved by increasing the input image size and also by improving the data set.

11.RESULT

We have achieved an accuracy of 95.8% in our model using only layer 1 of our algorithm, and using the combination of layer 1 and layer 2 we achieve an accuracy of 98.0%, which is a better accuracy than most of the current research papers on American sign language.

- Most of the research papers focus on using devices like Kinect for hand detection.
- In Sign Language Recognition using CNN by Pigou L., Dieleman S., Kindermans P.J., Schrauwen B., they build a recognition system for Flemish sign language using convolutional neural networks and Kinect and achieve an error rate of 2.5%.
- In Sign Language recognition by Zaki, M.M., Shaheen, the model is built using a hidden Markov model classifier and a vocabulary of 30 words and they achieve an error rate of 10.90%.
- In Japanese fingerspelling recognition they achieve an average accuracy of 86% for 41 static gestures in Japanese sign language.
- Using depth sensors, the map achieved an accuracy of 99.99% for observed signers and 83.58% and 85.49% for new signers.
- They also used CNN for their recognition system. One thing should be noted that our model doesn't use any background subtraction algorithm while some of the models present above do that.

So, once we try to implement background subtraction in our project the accuracies may vary. On the other hand, most of the above projects use Kinect devices but our main aim was to create a project which can be used with readily available resources. A sensor like Kinect not only isn't readily available but also is expensive for most of the audience to buy and our model uses a normal webcam of the laptop hence it is a great plus point.

Below are confusion matrices for our model.

		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Confusion Matrix	A	147	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	2	0	0
	B	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0
	C	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	145	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	135	0	0	0	0	0	4	0	0	0	0	0	1	0	0	2	10	0	0	0
	G	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	H	1	0	0	0	0	0	7	143	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
	I	0	0	0	33	0	0	0	0	108	0	2	0	0	0	0	0	0	0	0	7	1	0	0	0	0
	J	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	K	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	L	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0
	M	0	0	0	0	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	0	0	0	0	0
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	147	1	0	0	0	0	0	0	0	0	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	
S	0	0	0	0	1	0	0	0	0	0	0	0	0	1	10	0	0	0	132	0	0	0	0	8	0	
T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	151	0	0	0	0	0	
U	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	115	0	0	0	0	
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	1	0	0	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	149	0	0	0	
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	148	0	
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	
Confusion Matrix	A	147	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	2	0	0	
	B	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	
	C	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	D	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	E	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	F	0	0	0	0	0	135	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	3	10	0	0	0
	G	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
	H	1	0	0	0	0	0	7	143	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	
	I	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
	J	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	K	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	L	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	
	M	0	0	0	0	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	0	0	0	0	0	
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0	
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	
	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	147	1	0	0	0	0	0	0	0	
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	
	S	0	0	0	0	1	0	0	0	0	0	0	0	0	0	10	0	0	0	133	0	0	0	0	8	0	0
	T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	151	0	0	0	0	0	
	U	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	1	0	0	
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	149	0	0	
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	148	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Algo 1 + Algo 2																									

12.CONCLUSION

- ✓ In this report, a functional real time vision based American Sign Language recognition for D&M people have been developed for asl alphabets.
- ✓ We achieved final accuracy of 98.0% on our data set. We have improved our prediction after implementing two layers of algorithms wherein we have verified and predicted symbols which are more similar to each other.
- ✓ This gives us the ability to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate.

13.FUTURE SCOPE

- ❖ We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms.
- ❖ We are also thinking of improving the Pre Processing to predict gestures in low light conditions with a higher accuracy.
- ❖ This project can be enhanced by being built as a web/mobile application for the users to conveniently access the project. Also, the existing project only works for ASL; it can be extended to work for other native sign languages with the right amount of data set and training. This project implements a finger spelling translator; however, sign languages are also spoken in a contextual basis where each gesture could represent an object, or verb. So, identifying this kind of a contextual signing would require a higher degree of processing and natural language processing (NLP).
- ❖ We are also thinking of adding translation of the text to voice.

14.BIBLIOGRAPHY

- [1] T. Yang, Y. Xu, and “A., Hidden Markov Model for Gesture Recognition”, CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, May 1994.
- [2] Pujan Ziaie, Thomas Muller, Mary Ellen Foster, and Alois Knoll “A Naïve Bayes Munich, Dept. of Informatics VI, Robotics and Embedded Systems, Boltzmannstr. 3, DE-85748 Garching, Germany.
- [3] https://docs.opencv.org/2.4/doc/tutorials/imgproc/gaussian_median_blur_bilateral_filter/gaussian_median_blur_bilateral_filter.html
- [4] Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.
- [5] [aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural Networks-Part-2/](https://aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/)
- [6] <http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php>
- [7] Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham
- [8] Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision-based features. Pattern Recognition Letters 32(4), 572–577 (2011).
- [9] N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," 2017 Nicograph International (NicoInt), Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9
- [10] Byeongkeun Kang, Subarna Tripathi, Truong Q. Nguyen” Real-time sign language fingerspelling recognition using convolutional neural networks from depth map” 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)
- [11] Number System Recognition (<https://github.com/chasinginfinity/number-sign-recognition>)
- [12] <https://opencv.org/>
- [13] <https://en.wikipedia.org/wiki/TensorFlow>
- [14] https://en.wikipedia.org/wiki/Convolutional_neural_network
- [15] <http://hunspell.github.io/>

15.APPENDIX

Application code

```
from PIL import Image, ImageTk
import tkinter as tk
import cv2
import os
import numpy as np
from keras.models import model_from_json
import operator
import time
import sys, os
import matplotlib.pyplot as plt
from hunspell import Hunspell
import enchant
from string import ascii_uppercase

class Application:

    def __init__(self):
        self.hs = Hunspell('en_US')
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.current_image2 = None
        self.json_file = open("model\model-bw.json", "r")
        self.model_json = self.json_file.read()
        self.json_file.close()
        self.loaded_model = model_from_json(self.model_json)
        self.loaded_model.load_weights("model\model-bw.h5")
        self.json_file_dru = open("model\model-bw_dru.json" , "r")
        self.model_json_dru = self.json_file_dru.read()
```

```

self.json_file_dru.close()
self.loaded_model_dru = model_from_json(self.model_json_dru)
self.loaded_model_dru.load_weights("model\model-bw_dru.h5")
self.json_file_tkdi = open("model\model-bw_tkdi.json" , "r")
self.model_json_tkdi = self.json_file_tkdi.read()
self.json_file_tkdi.close()
self.loaded_model_tkdi = model_from_json(self.model_json_tkdi)
self.loaded_model_tkdi.load_weights("model\model-bw_tkdi.h5")
self.json_file_smn = open("model\model-bw_smn.json" , "r")
self.model_json_smn = self.json_file_smn.read()
self.json_file_smn.close()
self.loaded_model_smn = model_from_json(self.model_json_smn)
self.loaded_model_smn.load_weights("model\model-bw_smn.h5")
self.ct = {}
self.ct['blank'] = 0
self.blank_flag = 0

for i in ascii_uppercase:
    self.ct[i] = 0

print("Loaded model from disk")

self.root = tk.Tk()
self.root.title("Sign Language To Text Conversion")
self.root.protocol('WM_DELETE_WINDOW', self.destructor)
self.root.geometry("900x900")

self.panel = tk.Label(self.root)
self.panel.place(x = 100, y = 10, width = 580, height = 580)

self.panel2 = tk.Label(self.root) # initialize image panel
self.panel2.place(x = 400, y = 65, width = 275, height = 275)

```

```

self.T = tk.Label(self.root)
self.T.place(x = 60, y = 5)
self.T.config(text = "Sign Language To Text Conversion", font = ("Courier", 30, "bold"))

self.panel3 = tk.Label(self.root) # Current Symbol
self.panel3.place(x = 500, y = 540)

self.T1 = tk.Label(self.root)
self.T1.place(x = 10, y = 540)
self.T1.config(text = "Character :", font = ("Courier", 30, "bold"))

self.panel4 = tk.Label(self.root) # Word
self.panel4.place(x = 220, y = 595)

self.T2 = tk.Label(self.root)
self.T2.place(x = 10, y = 595)
self.T2.config(text = "Word :", font = ("Courier", 30, "bold"))

self.panel5 = tk.Label(self.root) # Sentence
self.panel5.place(x = 350, y = 645)

self.T3 = tk.Label(self.root)
self.T3.place(x = 10, y = 645)
self.T3.config(text = "Sentence :", font = ("Courier", 30, "bold"))

self.bt1 = tk.Button(self.root, command = self.action1, height = 0, width = 0)
self.bt1.place(x = 26, y = 745)

self.bt2 = tk.Button(self.root, command = self.action2, height = 0, width = 0)
self.bt2.place(x = 325, y = 745)

self.bt3 = tk.Button(self.root, command = self.action3, height = 0, width = 0)
self.bt3.place(x = 625, y = 745)

```



```

self.str = ""
self.word = " "
self.current_symbol = "Empty"
self.photo = "Empty"
self.video_loop()

def video_loop(self):
    ok, frame = self.vs.read()
    if ok:
        cv2image = cv2.flip(frame, 1)
        x1 = int(0.5*frame.shape[1])
        y1 = 10
        x2 = frame.shape[1]-10
        y2 = int(0.5*frame.shape[1])
        cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
        cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGBA)
        self.current_image = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=self.current_image)
        self.panel.imgtk = imgtk
        self.panel.config(image=imgtk)
        cv2image = cv2image[y1:y2, x1:x2]
        gray = cv2.cvtColor(cv2image, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray,(5,5),2)
        th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_IN
V,11,2)
        ret, res = cv2.threshold(th3, 70, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
        self.predict(res)
        self.current_image2 = Image.fromarray(res)
        imgtk = ImageTk.PhotoImage(image=self.current_image2)
        self.panel2.imgtk = imgtk

```

```

self.panel2.config(image=imgtk)
self.panel3.config(text=self.current_symbol,font=("Courier",50))
self.panel4.config(text=self.word,font=("Courier",40))
self.panel5.config(text=self.str,font=("Courier",40))
predicts=self.hs.suggest(self.word)
"""if(len(predicts) > 0):
    self.bt1.config(text=predicts[0],font = ("Courier",20))
else:
    self.bt1.config(text="")
if(len(predicts) > 1):
    self.bt2.config(text=predicts[1],font = ("Courier",20))
else:
    self.bt2.config(text="")
if(len(predicts) > 2):
    self.bt3.config(text=predicts[2],font = ("Courier",20))
else:
    self.bt3.config(text="")
if(len(predicts) > 3):
    self.bt4.config(text=predicts[3],font = ("Courier",20))
else:
    self.bt4.config(text="")
if(len(predicts) > 4):
    self.bt5.config(text=predicts[4],font = ("Courier",20))
else:
    self.bt5.config(text="")"""
self.root.after(30, self.video_loop)
def predict(self,test_image):
    test_image = cv2.resize(test_image, (128,128))
    result = self.loaded_model.predict(test_image.reshape(1, 128, 128, 1))
    result_dru = self.loaded_model_dru.predict(test_image.reshape(1, 128, 128, 1))
    result_tkdi = self.loaded_model_tkdi.predict(test_image.reshape(1, 128, 128, 1))
    result_smn = self.loaded_model_smn.predict(test_image.reshape(1, 128, 128, 1))
    prediction={}

```

```

prediction['blank'] = result[0][0]
inde = 1
for i in ascii_uppercase:
    prediction[i] = result[0][inde]
    inde += 1
#LAYER 1
prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
self.current_symbol = prediction[0][0]
#LAYER 2
if(self.current_symbol == 'D' or self.current_symbol == 'R' or self.current_symbol == 'U'):
    prediction = {}
    prediction['D'] = result_dru[0][0]
    prediction['R'] = result_dru[0][1]
    prediction['U'] = result_dru[0][2]
    prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
    self.current_symbol = prediction[0][0]

if(self.current_symbol == 'D' or self.current_symbol == 'T' or self.current_symbol == 'K' or
self.current_symbol == 'T'):
    prediction = {}
    prediction['D'] = result_tkdi[0][0]
    prediction['T'] = result_tkdi[0][1]
    prediction['K'] = result_tkdi[0][2]
    prediction['T'] = result_tkdi[0][3]
    prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
    self.current_symbol = prediction[0][0]

if(self.current_symbol == 'M' or self.current_symbol == 'N' or self.current_symbol == 'S'):
    prediction1 = {}
    prediction1['M'] = result_smn[0][0]
    prediction1['N'] = result_smn[0][1]
    prediction1['S'] = result_smn[0][2]
    prediction1 = sorted(prediction1.items(), key=operator.itemgetter(1), reverse=True)

```

```

if(prediction1[0][0] == 'S'):
    self.current_symbol = prediction1[0][0]
else:
    self.current_symbol = prediction[0][0]
if(self.current_symbol == 'blank'):
    for i in ascii_uppercase:
        self.ct[i] = 0
self.ct[self.current_symbol] += 1
if(self.ct[self.current_symbol] > 15):
    for i in ascii_uppercase:
        if i == self.current_symbol:
            continue
        tmp = self.ct[self.current_symbol] - self.ct[i]
        if tmp < 0:
            tmp *= -1
        if tmp <= 10:
            self.ct['blank'] = 0
            for i in ascii_uppercase:
                self.ct[i] = 0
            return
self.ct['blank'] = 0
for i in ascii_uppercase:
    self.ct[i] = 0
if self.current_symbol == 'blank':
    if self.blank_flag == 0:
        self.blank_flag = 1
        if len(self.str) > 0:
            self.str += " "
        self.str += self.word
        self.word = ""
    else:
        if(len(self.str) > 16):
            self.str = ""

```

```

        self.blank_flag = 0
        self.word += self.current_symbol
def action1(self):
    predicts=self.hs.suggest(self.word)
    if(len(predicts) > 0):
        self.word=""
        self.str+=" "
        self.str+=predicts[0]
def action2(self):
    predicts=self.hs.suggest(self.word)
    if(len(predicts) > 1):
        self.word=""
        self.str+=" "
        self.str+=predicts[1]
def action3(self):
    predicts=self.hs.suggest(self.word)
    if(len(predicts) > 2):
        self.word=""
        self.str+=" "
        self.str+=predicts[2]
def action4(self):
    predicts=self.hs.suggest(self.word)
    if(len(predicts) > 3):
        self.word=""
        self.str+=" "
        self.str+=predicts[3]
def action5(self):
    predicts=self.hs.suggest(self.word)
    if(len(predicts) > 4):
        self.word=""
        self.str+=" "
        self.str+=predicts[4]
def destructor(self):

```

```
print("Closing Application...")
self.root.destroy()
self.vs.release()
cv2.destroyAllWindows()
```

```
def destructor1(self):
    print("Closing Application...")
    self.root1.destroy()
```

```
print("Starting Application...")
(Application()).root.mainloop()
```