1.INTRODUCTION

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.

1.1 SIGN LANGUAGE

It is a language that includes gestures made with the hands and other body parts, including facial expressions and postures of the body.It used primarily by people who are deaf and dumb. There are many different sign languages such as British, Indian (ISL) and American sign languages. British sign language (BSL) is not easily intelligible to users of American sign Language (ASL) and vice versa .

A functioning signing recognition system could provide a chance for the inattentive communication with non-signing people without the necessity for an interpreter. It might be wont to generate speech or text making the deaf more independent. Unfortunately there has not been any system with these capabilities thus far. During this project our aim is to develop a system which may classify signing accurately.

Indian Sign Language (ISL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. ISL is expressed by movements of the hands and face. It is the primary language of almost all Indians who are deaf and hard of hearing, and is used by many hearing people as well.

1.2 SIGN LANGUAGE AND HAND GESTURE RECOGNITION

The process of converting the signs and gestures shown by the user into text is called sign language recognition. It bridges the communication gap between people who cannot speak and the general public. Image processing algorithms along with neural networks are used to map the gesture to appropriate text in the training data and hence raw images/videos are converted into respective text that can be read and understood.

Dumb people are usually deprived of normal communication with other people in the society. It has been observed that they find it really difficult at times to interact with normal people with their gestures, as only a very few of those are recognized by most people. Since people with hearing impairment or deaf people cannot talk like normal people, they have to depend on some sort of visual communication most of the time. Sign Language is the primary means of communication in the deaf and dumb community. As like any other language it has also got grammar and vocabulary but uses visual modality for exchanging information. The problem arises when dumb or deaf people try to express themselves to other people with the help of these sign language grammars. This is

because normal people are usually unaware of these grammars. As a result it has been seen that communication of a dumb person are only limited within his/her family or the deaf community. The importance of sign language is emphasized by the growing public approval and funds for international projects. At this age of Technology the demand for a computer based system is highly demanding for the dumb community. However, researchers have been attacking the problem for quite some time now and the results are showing some promise. Interesting technologies are being developed for speech recognition but no real commercial product for sign recognition is actually there in the current market. The idea is to make computers to understand human language and develop user friendly human computer interfaces (HCI). Making a computer understand speech, facial expressions and human gestures are some steps towards it. Gestures are the non-verbally exchanged information. A person can perform innumerable gestures at a time. Since human gestures are perceived through vision, it is a subject of great interest for computer vision researchers. The project aims to determine human gestures by creating an HCI. Coding of these gestures into machine language demands a complex programming algorithm. In our project we are focusing on Image Processing and Template matching for better output generation.

1.3 IMAGE PROCESSING

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms a core research area within engineering and computer science disciplines too. Image processing basically includes the following three steps:

- Importing the image via image acquisition tools.
- Analyzing and manipulating the image.
- Output in which the result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are preprocessing, enhancement, and display, information extraction.

- **Digital image processing:**

Digital image processing consists of the manipulation of images using digital computers. Its use has been increasing exponentially in the last decades. Its applications

range from medicine to entertainment, passing by geological processing and remote sensing. Multimedia systems, one of the pillars of the modern information society relies heavily on digital image processing. Digital image processing consists of the manipulation of those finite precision numbers. The processing of digital images can be divided into several classes: image enhancement, image restoration, image analysis, and image compression. In image enhancement, an image is manipulated, mostly by heuristic techniques, so that a human viewer can extract useful information from it.

Digital image processing is to process images by computer. Digital image processing can be defined as subjecting a numerical representation of an object to a series of operations in order to obtain a desired result. Digital image processing consists of the conversion of a physical image into a corresponding digital image and the extraction of significant information from the digital image by applying various algorithms.

- **Pattern recognition**

On the basis of image processing, it is necessary to separate objects from images by pattern recognition technology, then to identify and classify these objects through technologies provided by statistical decision theory. Under the conditions that an image includes several objects, the pattern recognition consists of three phases, as shown in Fig.

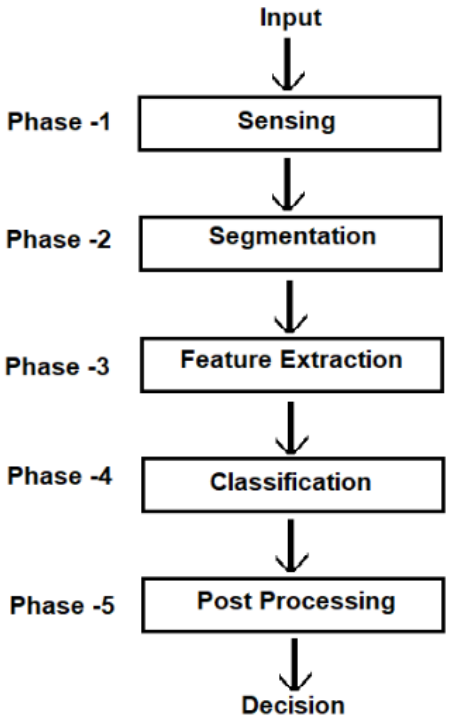


Fig 1.3.1 Phases of pattern recognition

The first phase is sensing which we can use camera or pre recorded footage to give as an input to the system.The second phase includes image segmentation and object separation.

In this phase, different objects are detected and separate from other backgrounds. The third phase is the feature extraction. In this phase, objects are measured. The measuring feature is to quantitatively estimate some important features of objects, and a group of the features are combined to make up a feature vector during feature extraction. The fourth phase is classification. In this phase, the output is just a decision to determine which category every object belongs to. Therefore, for pattern recognition, what input are images and what output are object types and structural analysis of images. The final stage is post processing in which other changes can be made to classified output to help in accurate and fast decision making. The structural analysis is a description of images in order to correctly understand and judge the important information of images.

1.4 MOTIVATION

The 2011 Indian census cites roughly 1.3 million people with "hearing impairment". In contrast to that, numbers from India's National Association of the Deaf estimates that 18 million people –roughly 1 percent of the Indian population -- are deaf. These statistics formed the motivation for our project. As these speech impairment and deaf people need a proper channel to communicate with normal people there is a need for a system . Not all normal people can understand the sign language of impaired people. Our project hence is aimed at converting the sign language gestures into text that is readable for normal people.

1.5 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. So we aim to propose a sign language detection system.

1.6 ORGANIZATION OF THESIS

The book is organized as follows:

Part 1: The various technologies that are studied are introduced and the problem statement is stated along with the motivation to our project.

Part 2: The Literature survey is put forth which explains the various other works and their technologies that are used for Sign Language Recognition.

Part 3: Explains the methodologies in detail, represents the architecture and algorithms used.

Part 4: Represents the project in various designs.

Part 5: Provides details on how we implemented the chosen algorithms and incorporated them in our architecture

Part 6: Provides the experimental analysis, the code involved and the results obtained.

Part 7: Concludes the project and provides the scope to which the project can be extended.

2. LITERATURE SURVEY

2.1 INTRODUCTION

The domain analysis that we have done for the project mainly involved understanding the neural networks

2.1.1 TENSORFLOW

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. Features: TensorFlow provides stable Python (for version 3.7 across all platforms) and C APIs; and without API backwards compatibility guarantee: C++, Go, Java, JavaScript and Swift (early release). Third-party packages are available for C#, Haskell Julia, MATLAB,R, Scala, Rust, OCaml, and Crystal."New language support should be built on top of the C API. However, not all functionality is available in C yet." Some more functionality is provided by the Python API. Application: Among the applications for which TensorFlow is the foundation, are automated image-captioning software, such as DeepDream.

2.1.2 OPENCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision.[1] Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel[2]). The library is cross-platform and free for use under the open-source BSD license.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition

Stereopsis stereo vision: depth perception from 2 cameras

- Structure from motion (SFM).
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning

library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

AForge.NET, a computer vision library for the Common Language Runtime (.NET Framework and Mono).

ROS (Robot Operating System). OpenCV is used as the primary vision package in ROS.

VXL, an alternative library written in C++.

Integrating Vision Toolkit (IVT), a fast and easy-to-use C++ library with an optional interface to OpenCV.

CVIPtools, a complete GUI-based computer-vision and image-processing software environment, with C function libraries, a COM-based DLL, along with two utility programs for algorithm development and batch processing.

OpenNN, an open-source neural networks library written in C++.

List of free and open source software packages

- OpenCV Functionality
- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching,
- videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA acceleration (gpu)

Image-Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it. If we talk about the basic definition of image processing then “Image processing is the analysis and

manipulation of a digitized image, especially in order to improve its quality".

Digital-Image

An image may be defined as a two-dimensional function f(x, y), where x and y are spatial(plane) coordinates, and the amplitude of any pair of coordinates (x, y) is called the intensity or gray level of the image at that point.

In another word An image is nothing more than a two-dimensional matrix (3-D in case of coloured images) which is defined by the mathematical function f(x, y) at any point is giving the pixel value at that point of an image, the pixel value describes how bright that pixel is, and what color it should be.

Image processing is basically signal processing in which input is an image and output is image or characteristics according to requirements associated with that image.

Image processing basically includes the following three steps :

1. Importing the image
2. Analyzing and manipulating the image
3. Output in which result can be altered image or report that is based on image analysis

Applications of Computer Vision:

Here we have listed down some of major domains where Computer Vision is heavily

used.

- Robotics Application
- Localization − Determine robot location automatically
- Navigation
- Obstacles avoidance
- Assembly (peg-in-hole, welding, painting)
- Manipulation (e.g. PUMA robot manipulator)
- Human Robot Interaction (HRI) − Intelligent robotics to interact with and serve people
- Medicine Application
- Classification and detection (e.g. lesion or cells classification and tumor detection)
- 2D/3D segmentation
- 3D human organ reconstruction (MRI or ultrasound)
- Vision-guided robotics surgery
- Industrial Automation Application
- Industrial inspection (defect detection)
- Assembly

- Barcode and package label reading
- Object sorting
- Document understanding (e.g. OCR)
- Security Application
- Biometrics (iris, finger print, face recognition)
- Surveillance − Detecting certain suspicious activities or behaviors
- Transportation Application
- Autonomous vehicle
- Safety, e.g., driver vigilance monitoring

2.1.3 KERAS

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Features: Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, anda host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU) principally in conjunction with CUDA.Keras applications module is used to provide a pre-trained model for deep neural networks. Keras models are used for prediction, feature extraction and fine tuning. This chapter explains about Keras applications in detail.

Pre-trained models

Trained model consists of two parts: model Architecture and model Weights. Model weights are large files so we have to download and extract the feature from the ImageNet database. Some of the popular pre-trained models are listed below,

- ResNet

- VGG16
- MobileNet
- InceptionResNetV2
- InceptionV3

2.1.4 NUMPY

NumPy (pronounced /ˈnʌmpaɪ/ (NUM-py) or sometimes /ˈnʌmpi/ (NUM-pee)) is a library for the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open source software and has many contributors.

Features: NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is aplotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as a universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Limitations: Inserting or appending entries to an array is not as trivially possible as itis with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy np.concatenate([a1,a2]) operation does not actually link the two

arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave, numexpr and Numba. Cython and Pythran are static-compiling alternatives to these.

2.1.5 MACHINE LEARNING

Everyone in this world communicate with each other by voice, there is a small percentage of people who are unable to communicate with everyone because of hearing disability, millions of people are suffering this disability and not all school curriculums have sign language which the mute people use to communicate, so most of the crowd is unable to understand what a mute person wants to say. Sign language is expressed in the form of hand gestures along with some non-verbal expressions. It is not the same as non-verbal gestures and hence is difficult to understand for normal people unless trained. With the technology we have and in the modern era we are able to use artificial intelligence to solve the problems of our mankind, with the help of machine learning we are able to construct a bridge which breaks the communication barrier between a normal person and a mute person. Artificial Intelligence has made break-through solutions to a lot of our problems in modern life. With advanced technology in our age we are able to cross human intelligence at one point. Our sign language translator majorly focuses on indian sign language which is been used currently by mute people in india, our sign language translator has machine learning models which has a good accuracy rate, making it an essential tool for communication, our system takes in image input and video input, recognises the gestures and signs in the footage, predicts the output with the help of the trained machine learning algorithm and converts it into english language, with the user-friendly graphical user interface (GUI) we can provide the output on the screen, making it translate the images and video footage of signs and gestures into readable text, breaking the communication barrier and helping people communicate with each other, our supervised machine learning model has been trained with thousands of image datasets which include various lighting conditions, skin tone and other various scenarios where we probably use the system, this include video datasets of signs, making it possible to recognise some of the signs we use in our daily life, All the diverse dataset helps the model train better and making the model able to detect the signs and gestures in most the cases. The trained model is supervised and has satisfactory accuracy results.

Our goal is to predict alphabets, numbers and basic signs in the Indian sign language, making it a little easier to communicate with any person even if he/she has no idea about sign language.

A Sign Language is a language in which communication between people is made by visually transmitting the sign patterns to express the meaning. It is a replacement of speech for hearing and speech impaired people. Thus, because of which has attracted many researchers in this field from long. Many researchers have been working in different sign languages like American Sign Language, British Sign Language, Taiwanese Sign Language, etc. but few works have made progress on Indian Sign Language.

The hearing impaired people become neglected from the society because the normal people never try to learn ISL nor try to interact with the hearing impaired people. This becomes a curse for them and so they mostly remain uneducated and isolated. Thus recognition of sign language was introduced which has not only been important from engineering point of view but also for the impact on society.

Studies prove that Children feel more lonely when they are silent and the problem of this communication gap will add to their depression. There are also a numerous number of problems prevailing due to this communication gap, our project aims to act as a bridge to close this distance and make the flow of thoughts more easy.

2.1.6 NEURAL NETWORKS

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

A neural network works similarly to the human brain's neural network. A "neuron" in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis.

A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or

output signals to which input patterns may map. Hidden layers fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.

Areas of Application

Following are some of the areas where ANN is being used. It suggests that ANN has an interdisciplinary approach in its development and applications.

- Speech recognition
- Multilayer networks
- Kohonen self-organizing feature map
- Character recognition
- Neocognition
- Signature verification application
- Human face recognition
- Deep learning

1. Speech Recognition

Speech occupies a prominent role in human-human interaction. Therefore, it is natural for people to expect speech interfaces with computers. In the present era, for communication with machines, humans still need sophisticated languages which are difficult to learn and use. To ease this communication barrier, a simple solution could be, communication in a spoken language that is possible for the machine to understand.

Great progress has been made in this field, however, still such kinds of systems are facing the problem of limited vocabulary or grammar along with the issue of retraining of the system for different speakers in different conditions. ANN is playing a major role in this area. Following ANNs have been used for speech recognition.

2. Multilayer networks

Multilayer networks with recurrent connections

3. Kohonen self-organizing feature map

The most useful network for this is Kohonen Self-Organizing feature map, which has its input as short segments of the speech waveform. It will map the same kind of phonemes as the output array, called feature extraction technique. After extracting the features, with the help of some acoustic models as back-end processing, it will recognize the utterance.

4. Character Recognition

It is an interesting problem which falls under the general area of Pattern Recognition. Many neural networks have been developed for automatic recognition of handwritten characters, either letters or digits. Following are some ANNs which have been used for character recognition.

Multilayer neural networks such as Backpropagation neural networks.

5. Neocognitron

Though back-propagation neural networks have several hidden layers, the pattern of connection from one layer to the next is localized. Similarly, neocognitron also has several hidden layers and its training is done layer by layer for such kinds of applications.

6. Signature Verification Application

Signatures are one of the most useful ways to authorize and authenticate a person in legal transactions. Signature verification technique is a non-vision based technique.

For this application, the first approach is to extract the feature or rather the geometrical feature set representing the signature. With these feature sets, we have to train the neural networks using an efficient neural network algorithm. This trained neural network will classify the signature as being genuine or forged under the verification stage.

7. Human Face Recognition

It is one of the biometric methods to identify the given face. It is a typical task because of the characterization of “non-face” images. However, if a neural network is well trained, then it can be divided into two classes namely images having faces and images that do not have faces.

First, all the input images must be preprocessed. Then, the dimensionality of that image must be reduced. And, at last it must be classified using a neural network training algorithm. Following neural networks are used for training purposes with preprocessed image

Fully-connected multilayer feed-forward neural network trained with the help of backpropagation algorithm For dimensionality reduction, Principal Component Analysis PCA is used.

8. Deep Learning:

Deep-learning networks are distinguished from the more commonplace single-hidden- layer neural networks by their depth; that is, the number of node layers

through which data must pass in a multistep process of pattern recognition.

Earlier versions of neural networks such as the first perceptrons were shallow, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as “deep” learning. So deep is not just a buzzword to make algorithms seem like they read Sartre and listen to bands you haven’t heard of yet. It is a strictly defined term that means more than one hidden layer.

In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer’s output. The further you advance into the neural net, the more complex the features your nodes can recognize, since they aggregate and recombine features from the previous layer.

This is known as feature hierarchy, and it is a hierarchy of increasing complexity and abstraction. It makes deep-learning networks capable of handling very large, high dimensional data sets with billions of parameters that pass through nonlinear functions.

Above all, these neural nets are capable of discovering latent structures within unlabeled, unstructured data, which is the vast majority of data in the world. Another word for unstructured data is raw media; i.e. pictures, texts, video and audio recordings. Therefore, one of the problems deep learning solves best is in processing and clustering the world’s raw, unlabeled media, discerning similarities and anomalies in data that no human has organized in a relational database or ever put a name to.

For example, deep learning can take a million images, and cluster them according to their similarities: cats in one corner, ice breakers in another, and in a third all the photos of your grandmother. This is the basis of so-called smart photo albums.

Deep-learning networks perform automatic feature extraction without human intervention, unlike most traditional machine-learning algorithms. Given that feature extraction is a task that can take teams of data scientists years to accomplish, deep learning is a way to circumvent the chokepoint of limited experts. It augments the powers of small data science teams, which by their nature do not scale.

When training on unlabeled data, each node layer in a deep network learns features automatically by repeatedly trying to reconstruct the input from which it draws its samples, attempting to minimize the difference between the network’s guesses and the probability distribution of the input data itself. Restricted Boltzmann machines, for example, create so-called reconstructions in this manner

In the process, these neural networks learn to recognize correlations between certain relevant features and optimal results – they draw connections between feature signals and what those features represent, whether it be a full reconstruction, or with labeled data.A deep-learning network trained on labeled data can then be applied to

unstructured data, giving it access to much more input than machine-learning nets.

2.1.7 CONVOLUTION NEURAL NETWORK

Convolutional neural networks (CNN) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for searching through among users photos.

Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point.

To solve this problem the computer looks for the characteristics of the base level. In human understanding such characteristics are for example the trunk or large ears. For the computer, these characteristics are boundaries or curvatures. And then through the groups of convolutional layers the computer constructs more abstract concepts.In more detail: the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.

Applications of convolution neural network:

1. Decoding Facial Recognition

Facial recognition is broken down by a convolutional neural network into the following

major components -

- Identifying every face in the picture
- Focusing on each face despite external factors, such as light, angle, pose, etc.
- Identifying unique features

Comparing all the collected data with already existing data in the database to match a face with a name.

A similar process is followed for scene labeling as well.

2. Analyzing Documents

Convolutional neural networks can also be used for document analysis. This is not just useful for handwriting analysis, but also has a major stake in recognizers. For a machine to be able to scan an individual's writing, and then compare that to the wide

database it has, it must execute almost a million commands a minute. It is said with the use of CNNs and newer models and algorithms, the error rate has been brought down to a minimum of 0.4% at a character level, though its complete testing is yet to be widely seen.

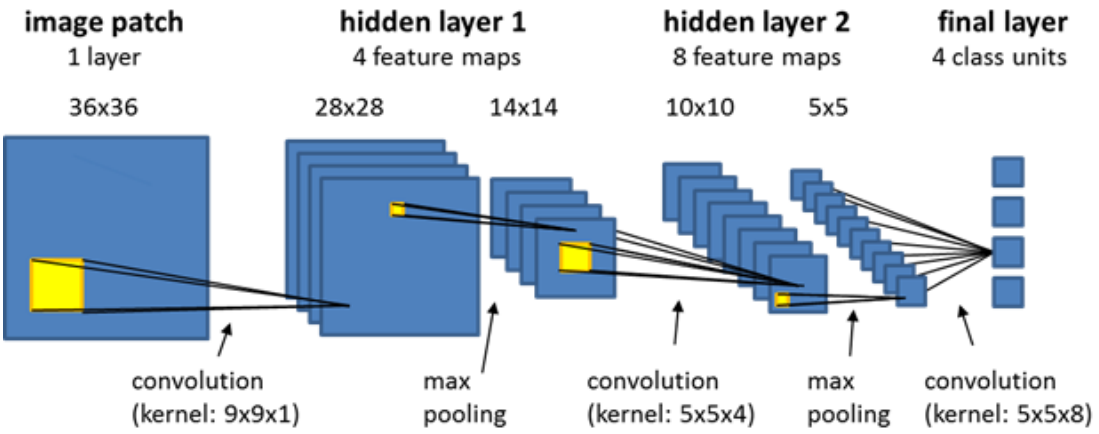


Fig 2.1.7.1 Layers involved in CNN

2.1.8 LONG SHORT TERM MEMORY

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

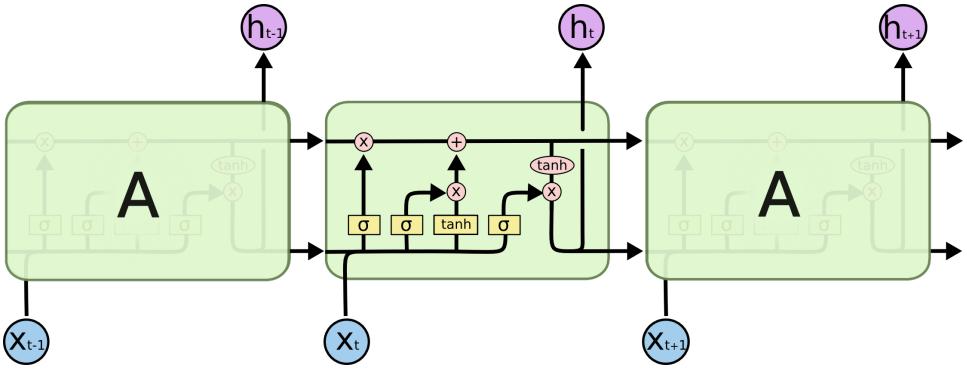
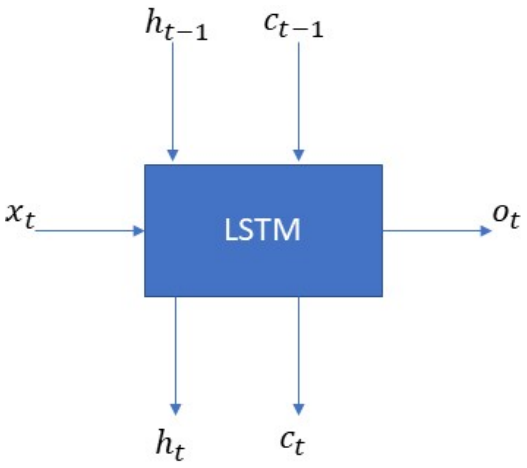


Fig 2.1.8.1 Long Short Term Memory



$$f_t = \sigma_g\ (W_f \times\ x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\ (W_i \times\ x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\ (W_o \times\ x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\ (W_c \times\ x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

f_t *is the forget gate*
i_t *is the input gate*
o_t *is the output gate*
c_t *is the cell state*
h_t *is the hidden state*

Fig 2.1.8.2 Long Short Term Memory Equations

Applications of LSTM

LSTM models need to be trained with a training dataset prior to its employment in real-world applications. Some of the most demanding applications are discussed below:

- **Text generation**

Language modeling or text generation, that involves the computation of words when a sequence of words is fed as input. Language models can be operated at the character level, n-gram level, sentence level or even paragraph level.

- **Image Processing**

Image processing, that involves performing analysis of a picture and concluding its result into a sentence. For this, it's required to have a dataset consisting of a good amount of pictures with their corresponding descriptive captions. A model that has already been trained is used to predict features of images present in the dataset. This is photo data. The dataset is then processed in such a way that only the words that are most suggestive are present in it. This is text data. Using these two types of data, we try to fit the model. The work of the model is to generate a descriptive sentence for the picture one word at a time by taking input words that were predicted previously by the model and also the image.

- **Speech and Handwriting Recognition**

Speech recognition, or speech-to-text, is the ability of a machine or program to identify words spoken aloud and convert them into readable text. Rudimentary speech recognition software has a limited vocabulary and may only identify words and phrases when spoken clearly. More sophisticated software can handle natural speech, different accents and various languages.Speech recognition uses a broad array of research in computer science, linguistics and computer engineering. Many modern devices and text-focused programs have speech recognition functions in them to allow for easier or hands-free use of a device.

Handwriting recognition (HWR), also known as handwritten text recognition (HTR), is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition. Alternatively, the movements of the pen tip may be sensed "on line", for example by a pen-based computer screen surface, a generally easier task as there are more clues available. A handwriting recognition system handles formatting, performs correct segmentation into characters, and finds the most plausible words.

- **Music Generation**

Music generation which is quite similar to that of text generation where LSTMs predict musical notes instead of text by analyzing a combination of given notes fed as input.

- **Language Translation**

Language Translation involves mapping a sequence in one language to a sequence in another language. Similar to image processing, a dataset, containing phrases and their translations, is first cleaned and only a part of it is used to train the model. An encoder-decoder LSTM model is used which first converts input sequence to its vector representation (encoding) and then outputs it to its translated version.

2.1.9 Scikit-image

Scikit-image is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.The scikit-image project started as scikits,.image, by Stéfan van der Walt. Its name stems from the notion that it is a "SciKit", a separately-developed and distributed third-party extension to SciPy. The original

codebase was later extensively rewritten by other developers. Of the various scikits, scikit-image as well as scikit-learn were described as well-maintained and popular in November 2012.

Scikit-image has also been active in the Google Summer of Code.The scikit-learn project started as scikits,learn, a Google Summer of Code project David Cournapcau, Its name stems from the notion that it is a SciKit, a separately- eloped and distributed third-party extension to SciPy. The original codebase was later written by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre ramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as well-maintained and popular in November 2012. Scikit-image is largely written in Python, with some core algorithms written in Cython to achieve performance.

2.1.10 Mediapipe

Mediapipe offers open source cross-platform, customizable ML solutions for live and streaming media. We can use mediapipe for different applications like face detection , gesture recognition , posture evaluation and many other machine learning related applications. It is especially useful in gesture recognition and detection thanks to its feature to extract face and pose landmarks. It is very accurate and reliable so it is frequently used in gesture prediction. Although it provides a frame we still need to write our own model and train it to be able to recognize the patterns in the gestures.

3. PROBLEM ANALYSIS

3.1 EXISTING SYSTEM

There are a few already existing solutions to recognise the signs in sign language like

1. Sensor gloves
2. Software algorithms

3.1.1 SENSOR GLOVES

Sensor gloves are synthetic gloves with in-built sensors which are able to detect the position of the hands, fingers and knuckles. The position of the hands, dingers and knuckles are plotted in 3d obtaining a configuration. These configurations have already defined signs and converted accordingly.

They are highly accurate because of their static lookup nature. The system directly maps the identified configuration with an already configured gesture. They are easy to use i.e., no additional training required to use these gloves , once after wearing the gloves the user can use gestures like he would regularly do without any gloves. The implementation implicitly does not involve any machine learning algorithms.

The use of physical hardware makes the system expensive and as a result a very small population will be able to afford it making it highly not feasible. Carrying gloves wherever you go, wearing it all the time makes users feel conscious and uncomfortable and also considering only the hand gestures gives no room to get the context of the expression.

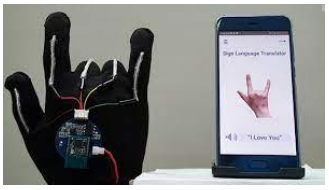


Fig 3.1.1.1 Sensor Gloves

3.1.2 SOFTWARE ALGORITHMS

Software Algorithms collectively refers to the machine learning and artificial algorithms using computer vision and gesture recognition to translate sign language. Different models are trained on different instances of datasets enabling the algorithms to be able to recognise the gesture in sign language.

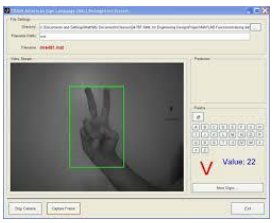


Fig 3.1.2.1 Software Algorithms

Due to low implementation cost they are usually highly recommendable and used. It also can offer high accuracy if designed and trained correctly. They can be easily deployed and scaled making them available and reliable. As the number of signs is huge it takes a long time to train them. Achieving a high accuracy rate may be difficult due to a major part of the system being uncertain.

3.1.3 BACKGROUND STUDY

In Literature survey we have gone through other similar works that are implemented in the domain of sign language recognition.The summaries of each of the project works are mentioned below.

1. Sign Language Recognition (SLR) system, which is required to recognize sign languages, has been widely studied for years.The studies are based on various input sensors, gesture segmentation, extraction of features and classification methods.This paper aims to analyze and compare the methods employed in the SLR systems, classifications methods that have been used, and suggests the most promising method for future research. Due to recent advancement in classification methods, many of the recent proposed works mainly contribute on the classification methods, such as hybrid methods and Deep Learning. This paper focuses on the classification methods used in prior Sign Language Recognition systems. Based on our review, HMM- based approaches have been explored extensively in prior research, including its modifications.

   This study is based on various input sensors, gesture segmentation, extraction of features and classification methods. This paper aims to analyze and compare the methods employed in the SLR systems, classifications methods that have been used, and suggests the most reliable method for future research. Due to recent advancement in classification methods, many of the recently proposed works mainly contribute to the classification methods, such as hybrid method and Deep Learning. Based on our review, HMM-based approaches have been explored extensively in prior research, including its modifications.Hybrid CNN-HMM and fully Deep Learning approaches have shown promising results and offer opportunities for further exploration.

2. Chat applications have become a powerful media that assist people to communicate in different languages with each other. There are lots of chat applications that are used by different people in different languages but there are not such chat applications that facilitate communication with sign languages. The developed system is based on Sinhala Sign language. The system has four main components as text messages are converted to sign messages, voice messages are converted to sign messages, sign messages are converted to text messages and sign messages are converted to voice messages. Google voice recognition API is used to develop speech character recognition for voice messages.The system has been trained for the speech and text patterns by using some text parameters and signs of Sinhala Sign language isdisplayed by emoji. Those emoji and signs that are included in this system will bring the normal people more close to the disabled people. This is a 2 way communication system but it uses a pattern of gesture recognition which is not very reliable in getting appropriate output.

3. Computer recognition of sign language is an important research problem for enabling communication with hearing impaired people. This project introduces an efficient and fast algorithm for identification of the number of fingers opened in a gesture representing an alphabet of the Binary Sign Language. The system does not require the hand to be perfectly aligned to the camera. The project uses an image processing system to identify, especially English alphabetic sign language used by the deaf people to communicate. The basic objective of this project is to develop a computer based intelligent system that will enable dumb people significantly to communicate with all other people using their natural hand gestures. The idea consisted of designing and building up an intelligent system using image processing, machine learning and artificial intelligence concepts to take visual inputs of sign language hand gestures and generate easily recognizable forms of outputs. Hence the objective of this project is to develop an intelligent system which can act as a translator between the sign language and the spoken language dynamically and can make the communication between people with hearing impairment and normal people both effective and efficient. The system is we are implementing for Binary sign language but it can detect any sign language with prior image processing

4. One of the major drawbacks of our society is the barrier that is created between disabled or handicapped persons and the normal person. Communication is the only medium by which we can share our thoughts or convey the message but for a person with disability (deaf and dumb) faces difficulty in communication with normal person. For many deaf and dumb people , sign language is the basic means of communication. Sign language recognition (SLR) aims to interpret sign languages automatically by a computer in order to help the deaf communicate with hearing society conveniently. Our aim is to design a system to help the

person who trained the hearing impaired to communicate with the rest of the world using sign language or hand gesture recognition techniques. In this system, feature detection and feature extraction of hand gestures is done with the help of SURF algorithm using image processing. All this work is done using MATLAB software. With the help of this algorithm, a person can easily trained a deaf and dumb.

5. Speech impairment is a disability which affects one's ability to speak and hear. Such individuals use sign language to communicate with other people. Although it is an effective form of communication, there remains a challenge for people who do not understand sign language to communicate with speech impaired people. The aim of this paper is to develop an application which will translate sign language to English in the form of text and audio, thus aiding communication with sign language. The application acquires image data using the webcam of the computer, then it is preprocessed using a combinational algorithm and recognition is done using template matching. The translation in the form of text is then converted to audio. The database used for this system includes 6000 images of English alphabets. We used 4800 images for training and 1200 images for testing. The system produces 88% accuracy.

- **DISADVANTAGES OF EXISTING SYSTEM**

    1. Sensor gloves, which is a hardware based solution needs the user to wear the gloves whenever they want to recognise the signs which makes the user feel uncomfortable
    2. And the user needs to carry the system and the gloves handy with him/her which is not at all feasible
    3. Software algorithm to recognise signs, which is a software based solution is somewhat better when compared with the Sensor gloves as software algorithms does not require any external devices to be carried by the user.
    4. But the existing software algorithm is limited to some extent to fulfill the communication gap between physically challenged people and normal people. It is able to recognise only the alphabet and number of sign languages. It could not solve the communication barrier between mute people and normal people.
    5. Most of the existing systems are able to recognise the alphabets and the digits of sign languages which are only limited to image based recognition which is not really feasible in the real time scenario.

3.2 PROPOSED SYSTEM

Our proposed system is a sign language recognition system using convolution neural networks which recognizes various hand gestures by capturing video and converting it into frames. Then the hand pixels are segmented and the image it obtained

and sent for comparison to the trained model. Thus our system is more robust in getting exact text labels of letters.

3.2.1 SYSTEM ARCHITECTURE

The Video Input is taken as input and each frame is taken out of the video. The Video filter and processor labels the hand gestures in the frames. These labeled frames are then sent to the learning model. This model now gets trained by the sent data. After the gestures are recognised, they're sent to the sentence generator. The Sentence generator takes the text and phrases the sentence, words accordingly for understandable output to the user. This output of the Sentence generator is shown as Output which is an understandable sentence.

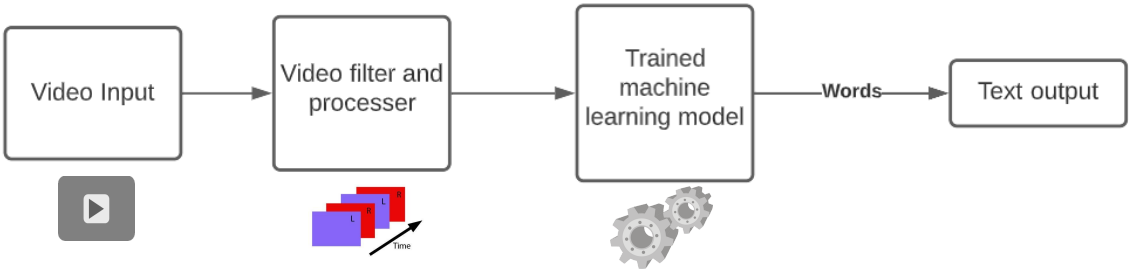


Fig 3.2.1.1 System Architecture

Video input

Recorded video in real time which is captured from camera from the computer system web cam. It takes the video and generated frames and passes them to the video filter.

Video filter and processor

The video filter takes the generated frames and identifies the objects and extracts the areas of interest from each frame . We use object detection algorithms to identify hands and faces.

Trained machine learning model

A trained machine learning model that can recognise the signs from the areas of interest and also predict the context based on the facial expressions.The model generates english words corresponding to the signs in the video.

Text Output

The text output generated from the machine learning model is shown on the screen or played out loud using device audio.

3.2.2 IMPORTANCE OF FACIAL EXPRESSIONS

Face emotion recognition is very crucial for exactly identifying the emotion of the user with which we can predict the context of the conversion more accurately. The example shows the same gesture with different meaning based on facial expression.

(HAPPY EXPRESSION) = I AM FINE

(DOUBTFUL EXPRESSION /QUESTION) = HOW ARE YOU

Fig 3.2.2.1 Importance of Facial Expression

4. SOFTWARE REQUIREMENTS SPECIFICATION

4.1 HARDWARE REQUIREMENTS

- **Camera:** Minimum 3MP camera with resolution of ()
- **Ram**: Minimum 8GB or higher
- **GPU**: 4GB dedicated
- **Processor**: Intel Pentium 4 or higher
- **HDD**: 10GB or higher
- **Monitor**: 15” or 17” color monitor
- **Mouse**: Scroll or Optical Mouse or Touchpad
- **Keyboard**: Standard 110 keys keyboard

4.2 SOFTWARE REQUIREMENTS

- **Operating System**:Windows,Mac,Linux
- **Packages Required:** Python,OpenCV,TensorFlow,Keras,Numpy,Mediapipe
- **IDE :** Jupyter Notebook / VS Code editor

4.3 DESCRIPTION OF SOFTWARE

4.3.1 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.

It is used for:

- web development (server-side),
- software development,
- mathematics,

- system scripting.

4.3.2 ANACONDA

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code

4.4 FUNCTIONAL REQUIREMENTS

A Functional Requirement (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional Requirements in Software Engineering are also called Functional Specification.

- Alphabet prediction
- Gesture prediction
- Face emotion detection

1. The System must be able to take video input from the user which contains Indian sign language gestures (in the form of alphabets) and recognize them. The signs may be made by any hand and the system must be able to recognize the gesture accurately. The predicted gesture must then be shown on the screen. The signs include (A-Z) and (1-9) in Indian sign language.
2. While gesture recognition is hard compared to finger spellings, It is mostly used and more reasonable to expect in any sign language. So, The system also provides a gesture prediction system that recognizes the signs that it has been trained on. Only one user's gestures can be recognized at any given time.

3. Facial emotion recognition plays an important role while recognizing the signs in any sign language. As illustrated the importance of facial expressions is unmistakably an important aspect to consider in order to say the exact feeling of the user. So the system also has an inbuilt face emotion recognition system that helps in predicting the signs more accurately.

4.5 NON-FUNCTIONAL REQUIREMENTS

Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs. Also known as system qualities, nonfunctional requirements are just as critical as functional Epics, Capabilities, Features, and Stories. They ensure the usability and effectiveness of the entire system. Failing to meet any one of them can result in systems that fail to satisfy internal business, user, or market needs, or that do not fulfill mandatory requirements imposed by regulatory or standards agencies. In some cases, non-compliance can cause significant legal issues (privacy, security, safety, to name a few).

- Speed
- Accuracy

1. As this is a real time system the users will depend on the system for their communication needs so a delay in the translations will waste the time of both the parties. The system uses machine learning algorithms which usually take some time to load and generate results after saving the trained model we can increase the speed of the system to match with real time performance.
2. As a tool used to decrease the communication gap between the people we expect high accuracy from the system. Any errors or inaccurate predictions of the signs will directly translate into a misunderstanding between the users. So, we tried different methods and different architectures in the same method to ensure the accuracy.

5. METHODOLOGY

5.1 SEGMENTED MODEL

The System comprises several phases due to its complex nature. The segmented design allows us to make changes that are required to improve accuracy and efficiency of the process.

Decomposed system workflow

Phase 1	Implementing object detection for accurac analysis,Algorithm selection(image datase
Phase 2	Include Face gestures and training using video data (3 words)
Phase 3	Training complete model(10 words) and including face emotion detection.

Table 5.1.1 Decomposed System Workflow

Phase 1

We take image dataset and train the model because working with video data is tedious.we can fix on the algorithm and parameters in this phase.

- Data Acquisition
- Training
- Evaluation

Phase 2

We now train the model with video data using 3 gestures initially by using the algorithm fixed in phase 1. After fixing the algorithm we also add the face gestures.

- Data Acquisition
- Training
- Evaluation

Phase 3

Train the model with complete 10 gestures after getting high accuracy we then need to work on face emotion detection.

- Data Acquisition

- Training
- Evaluation

5.2 TRAINING MODULE

Supervised machine learning:It is one of the ways of machine learning where the model is trained by input data and expected output data. To create such model, it is necessary to go through the following phases:

1. model construction
2. model training
3. model testing
4. model evaluation

Model construction:

It depends on machine learning algorithms. In this project case, it was neural networks.Such an algorithm looks like:

1. begin with its object: model = Sequential()

2. then consist of layers with their types: model.add(type_of_layer())

3. After adding a sufficient number of layers the model is compiled. At this moment

Keras communicates with TensorFlow for construction of the model. During model compilation it is important to write a loss function and an optimizer algorithm. It looks like: model.compile(loss= 'name_of_loss_function', optimizer= 'name_of_opimazer_alg') The loss function shows the accuracy of each prediction made by the model.

Before model training it is important to scale data for their further use.

Model training

After model construction it is time for model training. In this phase, the model is trained using training data and expected output for this data. It looks this way: model.fit(training_data, expected_output). Progress is visible on the console when the script runs. At the end it will report the final accuracy of the model.

Model Testing

During this phase a second set of data is loaded. This data set has never been seen by the model and therefore its true accuracy will be verified. After the model training is complete, and it is understood that the model shows the right result, it can be saved by: model.save("name_of_file.h5"). Finally, the saved model can be used in the real world. The name of this phase is model evaluation. This means that the model can be used to

evaluate new data.

Model evaluation

Machine learning models are only considered useful only if they have high accuracy in classification of the given data. So we must always measure the accuracy of of the model after constructing it, There are several metrics to measure the accuracy of the model and help evaluate it.

Confusion matrix

Confusion matrix is a very popular measure used while solving classification problems. It can be applied to binary Classification as well as multi class classification problems.

- true positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.
- true negatives (TN): We predicted no, and they don't have the disease.
- false positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- false negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

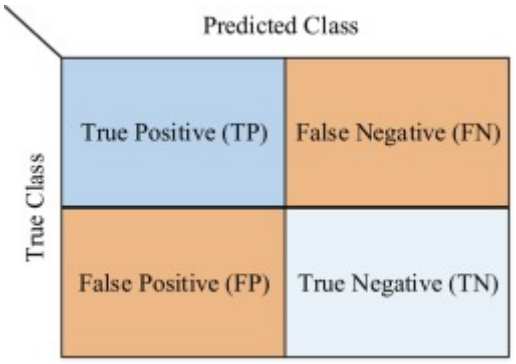


Fig 5.2.1 Confusion matrix

Accuracy is a well-known performance metric that is used to tell a strong classification model from one that is weak. Accuracy is, simply put, the total proportion of observations that have been correctly predicted. There are four (4) main components that comprise the mathematical formula for calculating Accuracy, viz. TP, TN, FP, FN, and these components grant us the ability to explore other ML Model Evaluation Metrics. The formula for calculating accuracy is as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Fig 5.2.2 Accuracy based on confusion matrix

Where:

- TP represents the number of True Positives. This refers to the total number of observations that belong to the positive class and have been predicted correctly.
- TN represents the number of True Negatives. This is the total number of observations that belong to the negative class and have been predicted correctly.
- FP is the number of False Positives. It is also known as a Type 1 Error. This is the total number of observations that have been predicted to belong to the positive class, but instead, actually, belong to the negative class.
- FN is the number of False Negatives. It may be referred to as a Type 2 Error. This is the total number of observations that have been predicted to be a part of the negative class but instead belong to the positive class.

5.3 PREPROCESSING

Uniform aspect ratio

- **Understanding aspect ratios**

An aspect ratio is a proportional relationship between an image's width and height. Essentially, it describes an image's shape.Aspect ratios are written as a formula of width to height, like this: For example, a square image has an aspect ratio of 1:1, since the height and width are the same. The image could be 500px × 500px, or 1500px × 1500px, and the aspect ratio would still be 1:1.As another example, a portrait-style image might have a ratio of 2:3. With this aspect ratio, the height is 1.5 times longer than the width. So the image could be 500px × 750px, 1500px × 2250px, etc.

- **Cropping to an aspect ratio**

Aside from using built in site style options , you may want to manually crop an image to a certain aspect ratio. For example, if you use product images that have the same aspect ratio, they'll all crop the same way on your site.

Option 1 - Crop to a preset shape

Use the built-in Image Editor to crop images to a specific shape. After opening the editor, use the crop tool to choose from preset aspect ratios.

Option 2 - Custom dimensions

To crop images to a custom aspect ratio not offered by our built-in Image Editor,

use a third-party editor. Since images don’t need to have the same dimensions to have the same aspect ratio, it’s better to crop them to a specific ratio than to try to match their exact dimensions. For best results, crop the shorter side based on the longer side.

- For instance, if your image is 1500px × 1200px, and you want an aspect ratio of 3:1, crop the shorter side to make the image 1500px × 500px.
- Don't scale up the longer side; this can make your image blurry.

Image scaling

- In computer graphics and digital imaging , image scaling refers to the resizing of a digital image. In video technology, the magnification of digital material is known as upscaling or resolution enhancement.
- When scaling a vector graphic image, the graphic primitives that make up the image can be scaled using geometric transformations, with no loss of image quality. When scaling a raster graphics image, a new image with a higher or lower number of pixels must be generated. In the case of decreasing the pixel number (scaling down) this usually results in avisible quality loss. From the standpoint of digital signal processing, the scaling of raster graphics is a two-dimensional example of sample-rate conversion, the conversion of a discrete signal from a sampling rate (in this case the local sampling rate) to another.

Data Augmentation

Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model. It is closely related to oversampling in data analysis.

5.4 DATASETS USED FOR TRAINING

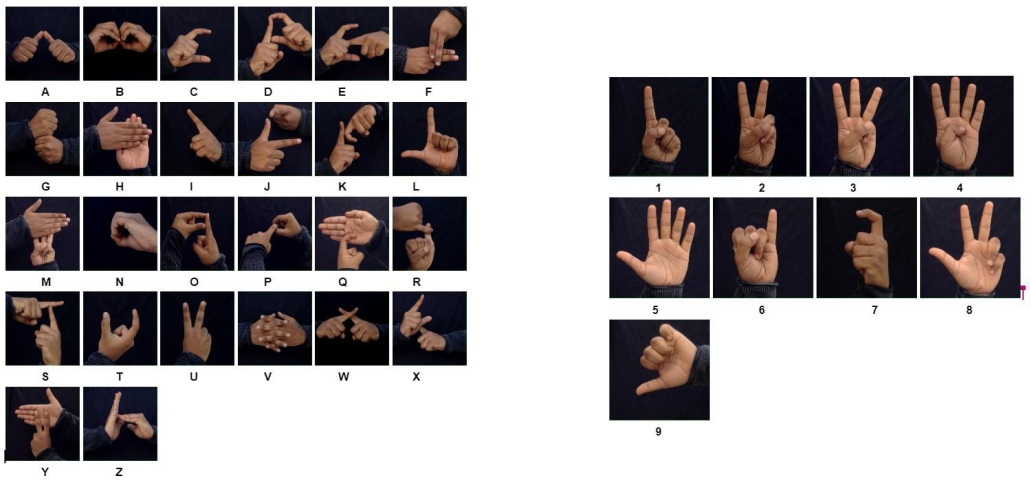


Fig 5.4.1 Classes used for training the model

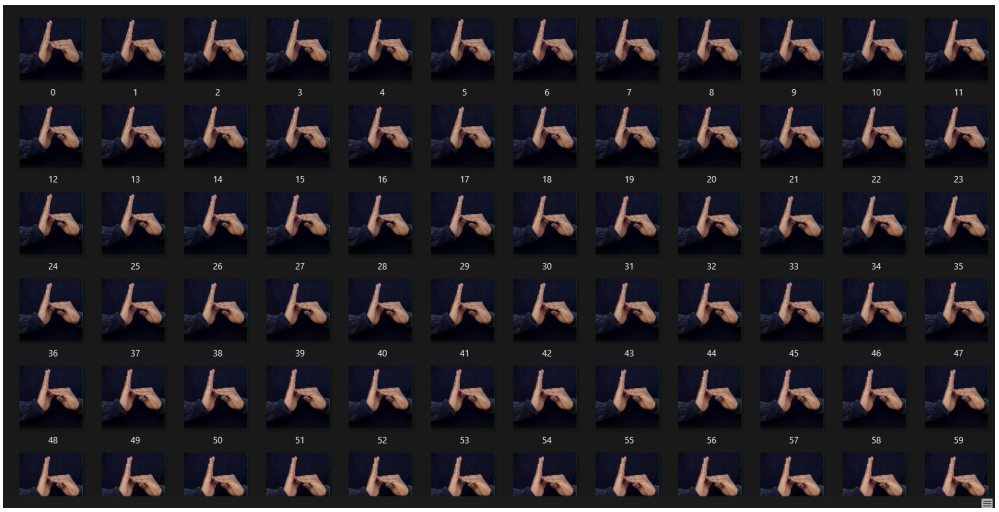
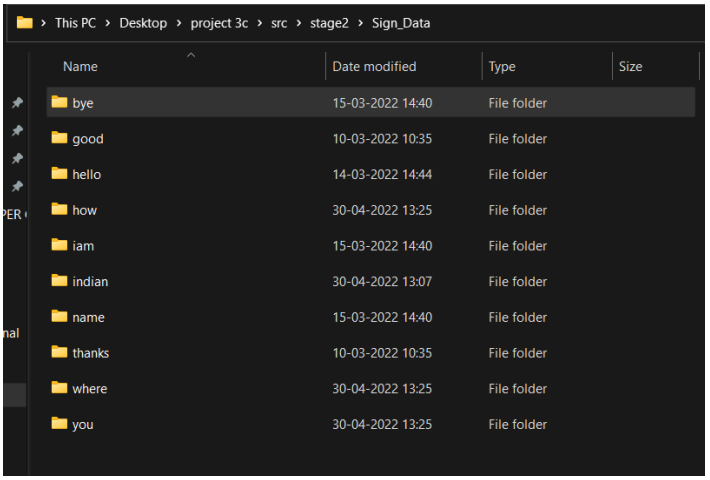


Fig 5.4.2 Sample pictures of training data



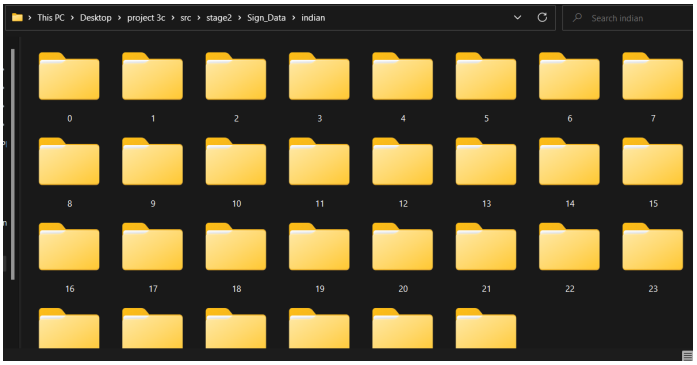


Fig 5.4. Training data given for Sign INDIAN

5.5 ALGORITHM

BackPropagation

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization. Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respect to all the weights in the network.

Optimizer(Adam)

Adam can be looked at as a combination ofRMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using the moving average of the gradient instead of the gradient itself like SGD with momentum. Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. Its name is derived from adaptive moment estimation, and the reason it's called that is because Adam uses estimations of first and second moments of gradient descent to adapt the learning rate for each weight of the neural network. Now, what is the moment ? N-th moment of a random variable is defined as the expected value of that variable to the power of n. More formally:

Loss Function(categorical cross entropy)

Categorical cross entropy is a loss function that is used for single label categorization. This is when only one category is applicable for each data point. In other words, an example can belong to one class only.

Note. The block before the Target block must use the activation function Softmax.

5.6 SEGMENTATION

Image segmentation is the process of partitioning a digital image into multiple segments(sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyse.Modern image segmentation techniques are powered by deep learning technology. Here are several deep learning architectures used for segmentation:

Why does Image Segmentation even matter?

If we take an example of Autonomous Vehicles, they need sensory input devices

like cameras, radar, and lasers to allow the car to perceive the world around it, creating a digital map. Autonomous driving is not even possible without object detection which itself involves image classification/segmentation.

How Image Segmentation works

Image Segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labeled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image. A common technique is to look for abrupt discontinuities in pixel values, which typically indicate edges that define a region.Another common approach is to detect similarities in the regions of an image. Some techniques that follow this approach are region growing, clustering, and thresholding. A variety of other approaches to perform image segmentation have been developed over the years using domain-specific knowledge to effectively solve segmentation problems in specific application areas.

5.7 CLASSIFICATION

CONVOLUTION NEURAL NETWORK

Image classification is the process of taking an input(like a picture) and outputting its class or probability that the input is a particular class. Neural networks are applied in the following steps:

1)One hot encode the data: A one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

2)Define the model: A model said in a very simplified form is nothing but a function that is used to take in certain input, perform certain operation to its beston the given input (learning and then predicting/classifying) and produce the suitable output.

3)Compile the model: The optimizer controls the learning rate. We will be using Adam as our optimizer. Adam is generally a good optimizer to use for many cases.The adam optimizer adjusts the learning rate throughout training. The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.

4)Train the model: Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization.

5)Test the model

A convolutional neural network convolves learned features with input data and uses 2D convolution layers.

Convolution Operation

In purely mathematical terms, convolution is a function derived from two given functions by integration which expresses how the shape of one is modified by the other.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

Fig 5.7.1 Formula for convolution operation

Here are the three elements that enter into the convolution operation:

- Input image
- Feature detector
- Feature map

Steps to apply convolution layer

You place it over the input image beginning from the top-left corner within the borders you see demarcated above, and then you count the number of cells in which the feature detector matches the input image.

- The number of matching cells is then inserted in the top-left cell of the feature map
- You then move the feature detector one cell to the right and do the same thing. This movement is called a and since we are moving the feature detector one cell At times, that would be called a stride of one pixel.
- What you will find in this example is that the feature detector's middle-left cell with the number 1 inside it matches the cell that it is standing over inside the input image. That's the only matching cell, and so you write “1” in the next cell in the feature map, and so on and so forth.
- After you have gone through the whole first row, you can then move it over to the next row and go through the same process. There are several uses that we gain from deriving a feature map. These are the most important of them: Reducing the size of the input image, and you should know that the larger your strides (the movements across pixels), the smaller your feature map.

Relu Layer

Rectified linear unit is used to scale the parameters to non negative values.We get pixel values as negative values too . In This layer we make them as 0's. The purpose of applying the rectifier function is to increase the non-linearity in our images. The reason we want to do that is that images are naturally non-linear. The rectifier serves to break up the linearity even further in order to make up for the linearity that we might impose on an image when we put it through the convolution operation. What the rectifier function does to an image like this is remove all the black elements from it, keeping only those carrying a positive value (the gray and white colors).The essential difference between the non-rectified version of the image and the rectified one is the progression of colors. After we rectify the image, you will find the colors changing more abruptly. The gradual change is no longer there. That indicates that the linearity has been disposed of.

Pooling Layer

The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input This process is what provides the convolutional neural network with the "spatial variance" capability. In addition to that, pooling serves to minimize the size of the images as well as the number of parameters which, in turn, prevents an issue of "overfitting" from coming up. Overfitting in a nutshell is when you create an excessively complex model in order to account for the idiosyncrasies we just mentioned. The result of using a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input. They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same location. This Capability added by pooling is called the model's invariance to local translation.

Fully Connected Layer

The role of the artificial neural network is to take this data and combine the features into a wider variety of attributes that make the convolutional network more capable of classifying images, which is the whole purpose of creating a convolutional neural network. It has neurons linked to each other ,and activates if it identifies patterns and sends signals to output layer .the output layer gives output class based on weight values, For now, all you need to know is that the loss function informs us of how accurate our network is, which we then use in optimizing our network in order to increase its effectiveness. That requires certain things to be altered in our network. These include the weights (the blue lines connecting the neurons, which are basically the synapses), and the feature detector since the network often turns out to be looking for the wrong features and has to be reviewed multiple times for the sake of optimization.This full connection process practically works as follows:

- The neuron in the fully-connected layer detects a certain feature; say, a nose.
- It preserves its value.
- It communicates this value to the classes' trained images.

6. DESIGN

6.1 DATA FLOW DIAGRAM

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:

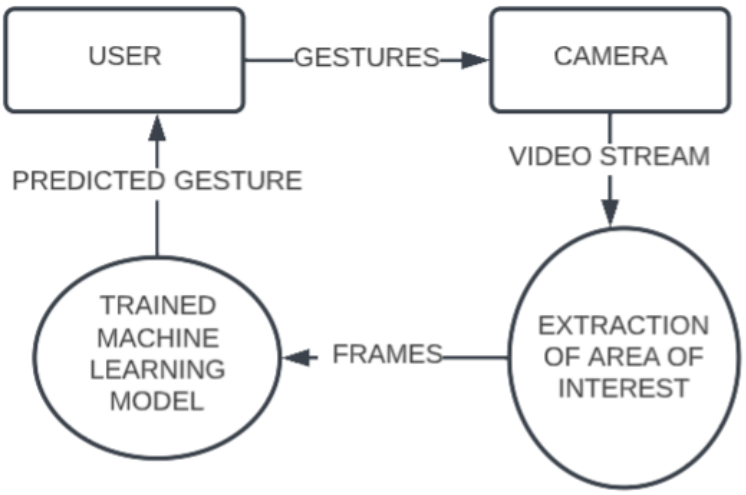


Fig 6.1.1 Data Flow Diagram for Sign Language Recognition

6.2 UML DIAGRAMS

UML stands for Unified Modeling Language. Taking SRS document of analysis as input to the design phase drawn UML diagrams. UML is only a language so it is just one part of the software development method. The UML is process independent, although optimally it should be used in a process that should be driven, architecture-centric, iterative, and incremental. The UML is a language for visualizing, specifying, constructing, documenting the articles in a software- intensive system. It is based on diagrammatic representations of software components. A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of the system.

A modeling language such as the UML is thus a standard language for software blueprints. The UML is a graphical language, which consists of all interesting systems. There are also different structures that can transcend what can be represented in a programming language.

These are different diagrams in UML.

6.2.1 USE CASE DIAGRAM

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system. Use case describes the behavior of the system as seen from the actor's point of view. It describes the function provided by the system as a set of events that yield a visible result for the actor.

Purpose of Use Case Diagrams

The purpose of a use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams(activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from the other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modeled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows :

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

How to Draw a Use Case Diagram?

- Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases.
- We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.
- Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.
- Functionalities to be represented as use case, Actors Relationships among the use cases and actors.

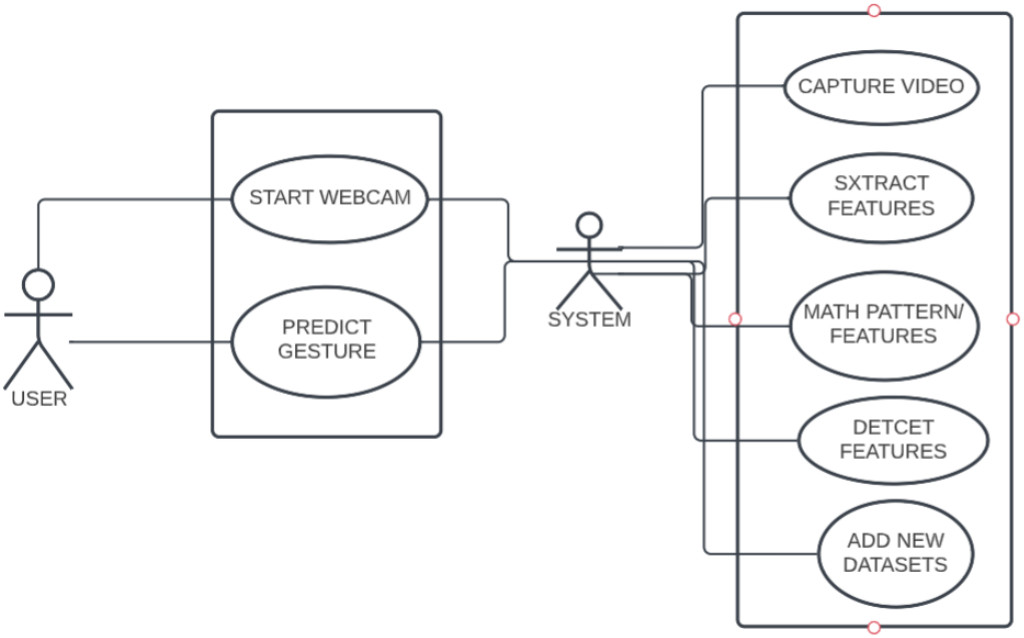


Fig 6..2.1.1 Use Case Diagram

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram.

6.2.2 CLASS DIAGRAM

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe the different perspectives when designing a system-conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagrams also display relationships such as containment, inheritance, association etc. The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes.

How to Draw a Class Diagram?

- Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagrams.
- Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.
- Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

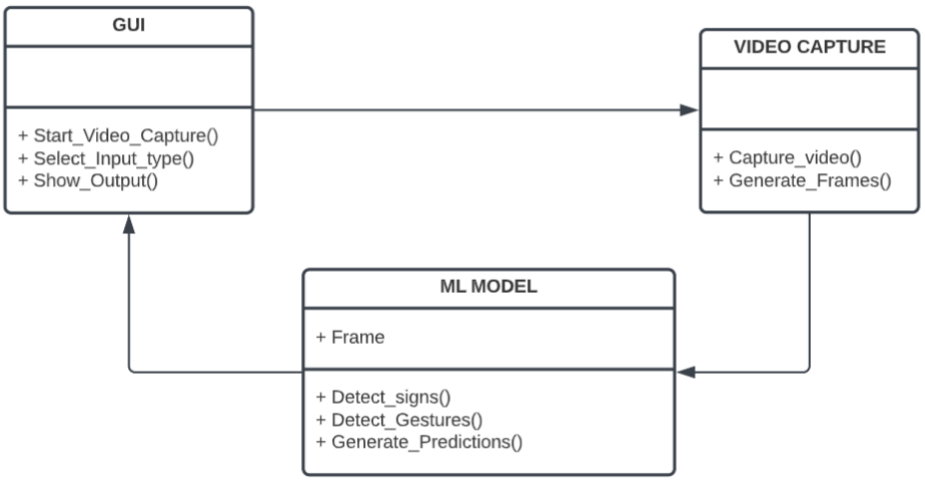


Fig 6.2.2.1 Class Diagram of the sign language recognition system

The following points should be remembered while drawing a class diagram

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance. Responsibility (attributes and methods) of each class should be clearly identified

- For each class, a minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

6.2.3 SEQUENCE DIAGRAM

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects).

Objects: Objects can be viewed as an entity at a particular point in time with specific value and as a holder of identity.

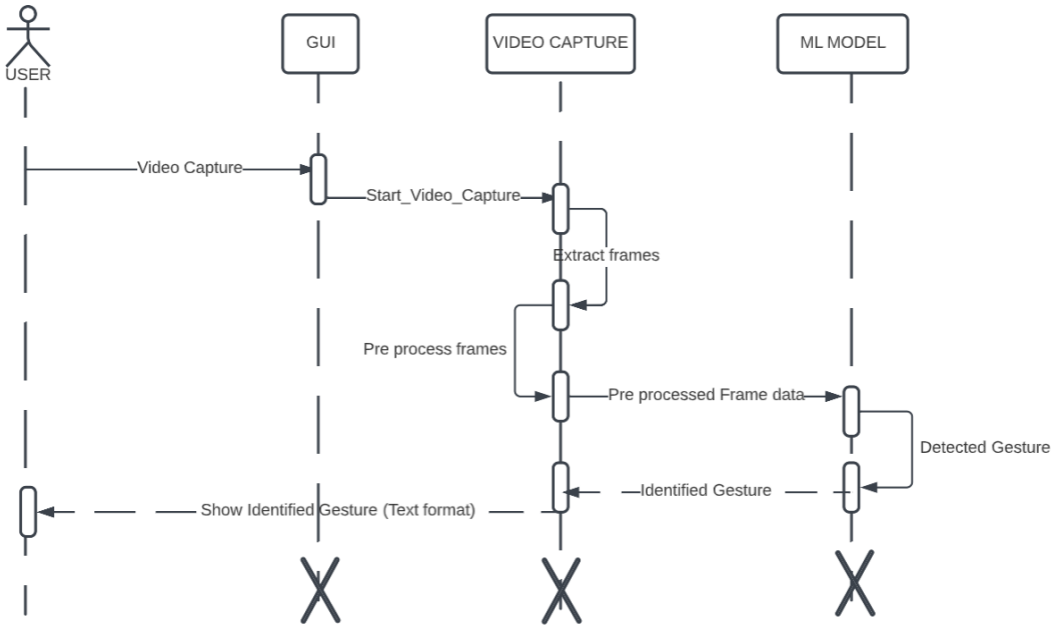


Fig 6.2.3.1 Sequence Diagram of Sign language recognition System

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

How to draw an use case diagram?

- A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.
- If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances.
- Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrowheads represent synchronous calls, open arrowheads represent asynchronous messages, and dashed lines represent reply messages. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response. Asynchronous calls are present in multithreaded applications, event-driven applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (ExecutionSpecifications in UML).
- Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. If an object is destroyed (removed from memory), an X is drawn on the bottom of the lifeline, and the dashed line ceases to be drawn below it. It should be the result of a message, either from the object itself, or another.
- A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found message in UML) or from a border of the sequence diagram (gate in UML).

UML has introduced significant improvements to the capabilities of sequence diagrams. Most of these improvements are based on the idea of interaction fragments which represent smaller pieces of an enclosing interaction. Multiple interaction fragments are combined to create a variety of combined fragments, which are then used to model interactions that include parallelism, conditional branches, optional interactions.\

6.2.4 STATE CHART

A state chart diagram describes a state machine which shows the behavior of classes. It shows the actual changes in state, not processes or commands that create those changes and is the dynamic behavior of objects over time by modeling the life cycle of objects of each class.

It describes how an object is changing from one state to another state. There are mainly two states in the State Chart Diagram:1. Initial State 2. Final-State. Some of the

components of Statechart Diagram are:

State: It is a condition or situation in the life cycle of an object during which it satisfies the same condition or performs some activity or waits for some event.

Transition: It is a relationship between two states indicating that the object in the first state performs some actions and enters into the next state or event.

Event: An event is a specification of a significant occurrence that has a location in time and space.

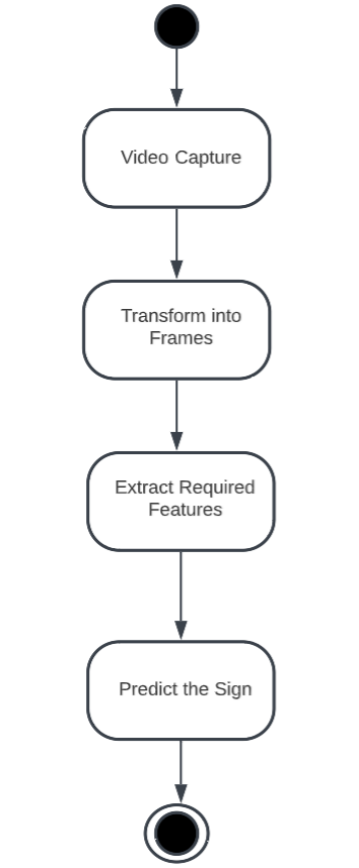


Fig 6.2.4.1 State Chart Diagram of sign language recognition System

7. IMPLEMENTATION

7.1 PREPROCESSING

Preprocessing of the inputs before feeding it to the algorithm either for prediction or for training is termed as preprocessing. Preprocessing allows the user to remove noise from the input or to change the input into a particular format that is easy to learn from or easy to understand by the machine. The preprocessing techniques used in this system are

1. Image resizing
2. Augmentation
3. Extracting landmarks for training

7.1.1 RESIZING THE IMAGES

The images in the dataset are (128 X 128) that makes the image size used for training and validation as (128 X 128) but when we capture the video / image from the webcam of the computer its resolution is that of the camera that captured the image. So, in order to be detected by the model the images for prediction must be in the same size as the images that the model has been trained on so we need to resize the images that are captured into (128 X 128).

7.1.2 FLIPPING THE IMAGES IN DATASET

The images in the dataset contain the signs which are shown by a right-handed person but in the real world there is a possibility of showing the signs in reverse order because the user may be left-handed or may have some other reason. We noticed that the signs shown by the other hand are being wrongly predicted so we decided to add the signs shown by left-handed people by simply flipping the images in the dataset. We also augmented the images on the factors of brightness and rotation.

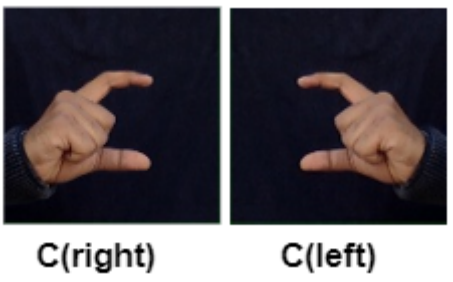


Fig 7.1.2.1 Augmentation of the dataset

7.1.3 EXTRACTION OF LANDMARKS

We have chosen the LSTM model to identify the patterns in the video and deduce the sign in the video by the patterns that has been already learnt but inorder to train the

LSTM we must first establish the area that has to be learnt. In this case to identify the signs we must learn the relative movement of the hands and pose of the body to be able to learn and detect a sign. So, we used a module that can identify the landmarks on the hands, pose and face landmarks accurately and return a list containing the coordinates of the landmarks in a frame. If we apply the same process for a set of frames containing movement and transform them into a list then the relative movement in the video formed by these frames will be preserved.

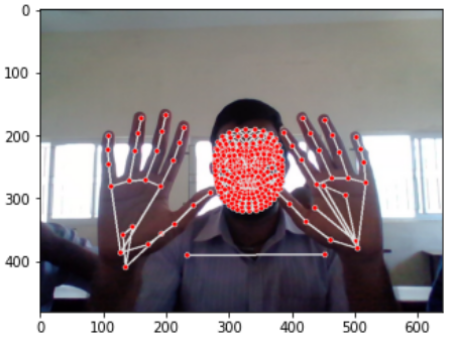


Fig 7.1.3.1 Landmarks in a single frame

7.2 ALGORITHM ARCHITECTURE

Instead of using pre-trained models for recognizing the Indian Sign language we use our own custom models and trained them with the selected datasets.

7.2.1 CNN-Architecture

We used the CNN algorithm to recognize the gestures (Alphabets and numbers) in image format. There are already many pre-trained models like Lenet,Dense,Alexnet etc .,with which we can train the model a lot faster but we decided to use our own model to have more flexibility. The image shows the architecture diagram of the CNN model we used to extract features from the images. All the images in the data set are 128 X 128 So the input layer is 128 X 128 and we used only Gray scale images to reduce computational time. After all the features are extracted we use a dense network to classify the images into 35 classes (A-Z(26) + 1-9(9)).

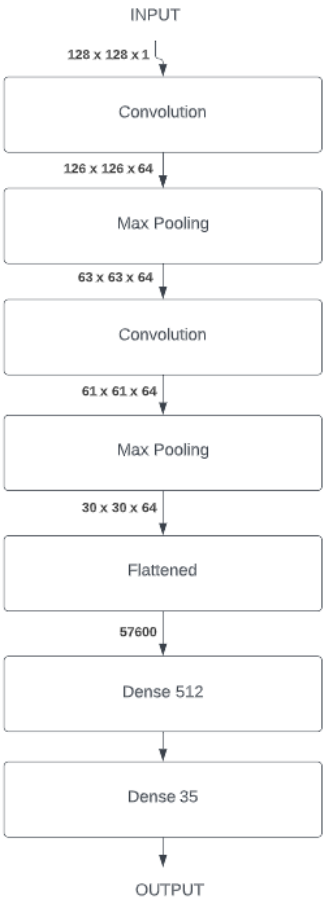


Fig 7.2.1.1 CNN Architecture

7.2.2 LSTM-Architecture

LSTM is mainly used to recognize patterns from sequential data like Audio, Text data and also video. The problem at hand is Recognizing the Sings form the Video so LSTM is the best option to use here.

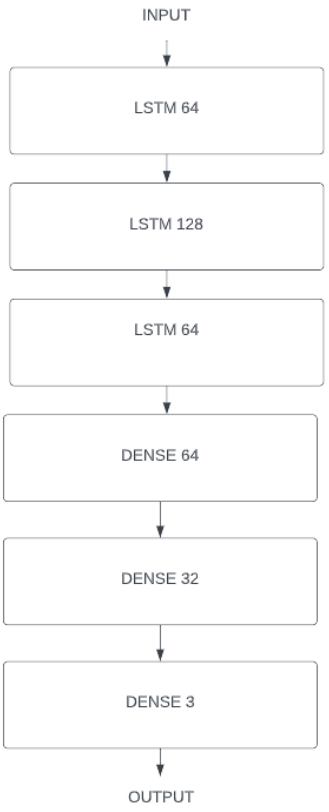


Fig 7.2.2.1 LSTM Architecture

We Start by inserting multiple layers of LSTM units for pattern recognition followed by the DENSE for classification.

7.3 TRAINING AND RECOGNITION

Alphabet Recognition

With the help of modern technology and computer vision we are able to take images and videos as input and use the supervised models to recognise the alphabets. With the well-known methods of convolutional neural networks (CNN), we train using the thousands of image dataset consisting of various use-case scenarios, the alphabet recognition is one of the phase were we predict the alphabets in Indian sign language by reorganizing the hand signs in the images, this module is modular making it a reusable in the further development of the system. The alphabet recognition takes frames or images as input, the future scope of the system is where frames are extracted from the video footage, all the frames will have essential information in-order to recognise the pattern which further help the model to predict the sign into an English language alphabet. The alphabet recognition module brilliantly detects the hands in the image apart from the background, making it easier for the system to recognize the hand, the hand outline module extracts the hand features from the image and converts the image into a black and white image. This processed image is then sent as input feed to the supervised model

which can take advantage of the processed form of the image and predict the alphabet more accurately. Hand outline detection will be discussed further in the paper as it is one of the essential features for the system to accurately predict the sign given by the user.

Training Phase (Alphabet)

In the training phase, we gather thousands of images for our various classifications which include alphabets and numbers with different possible scenarios of the use cases. The training phase takes the training dataset and trains the model by recursively going through the supervised methodology and with each epoch the accuracy of the model alters, this accuracy changes significantly with each epoch, so the accuracy can sometimes be increasing or decreasing. This accuracy is the major and essential part of the training as it is what determines the accuracy of the supervised trained model. All the thousands of images in the dataset are iterated and accuracy is varied throughout, as we have chosen a supervised method, the dataset not only hold the training data but also each data solution in-order for the system to validate its recognition of the sign or gesture on each iteration of the data in the dataset. The training phase is really essential for our system as we are going to do the accuracy analysis of the machine learning models and then choose the best model which has satisfactory and higher accuracy rate when compared to other models, this considered model is the final model chosen for the system to use.

Recognition Phase(Alphabet)

Recognition Phase uses the trained supervised model to recognise the gestures in the images/frames of the video footage. This phase takes the live video as the input where the length of the video can vary according to the Indian sign language, the system makes sures to extract the frames which are needed to recognise the sign language from the footage. These frames are then sent to the a module which detects the hands in the frames and then differentiates the hands from the background, this feature is essential and gives an advantageous gain as the module processed image/ frame makes sure to get a higher recognition rate of the input, the images which are processed by the Hand Outline Detection module are then sent to the supervised model where the trained model takes the processed image as input and uses the training data from the training phase and checks for patterns and other features which are relevant to the recognition from the input and then predicts/ recognises the sign/ gesture. The recognised gestures are then shown in the Graphical User Interface (GUI) which are quality of life features making the system easy user- friendly. The Recognised gestures are not only shown live but also stored under the history panel where the user can revisit the past recognised words, the words recognised are stored in English language and are stored either in sentence or single word format. The recognised words are stored temporarily until the session ends and the user can either give input as image format or video format, where the system can take both types of inputs and can recognise the sign language in both cases.

Training Phase (ISL SIGNS)

In this Training phase we train the LSTM using the video data of the selected signs (Multiple instances of each sign) and train it on all the recorded signs. We used the mediapipe module to extract anchor points like face landmarks, hand landmarks and pose landmarks. We make a array containing all these points and train the model on the change of the coordinates of each of these landmarks for each frame.

Recognition Phase (ISL SIGNS)

The LSTM model trained on the Signs will be able to identify the signs shown to the camera. The user can show the signs to the camera and the captured video frames are send to the model in a sequential manner so, the model will output the probabilities of the recorded sign being one of the trained sign. From this data we can decide what is the sign shown by the user.

8. EXPERIMENTAL ANALYSIS AND RESULTS

8.1 CODE

Phase 1 - Alphabet recognition

dataset.ipynb

```
import cv2
import os
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.image as mpimg

frompath=r"imagedataset/train/X"
dir1=os.listdir(frompath)
filename=1001
for img in dir1:
    imgpath=os.path.join(frompath+'/',img)
    realimg=cv2.imread(imgpath)
    flipimg=cv2.flip(realimg,1)
    filerealname=str(filename)+".jpg"
    cv2.imwrite(filerealname,flipimg)
    filename+=1
```

FaceRecogniser.ipynb

```
from imutils import paths
import face_recognition
import pickle
import cv2
import os
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
img = cv2.imread('test1.jpg')
imS = cv2. resize(img, (960, 540))
gray = cv2.cvtColor(imS, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x, y, w, h) in faces:
    cv2.rectangle(imS, (x, y), (x+w, y+h), (255, 0, 0), 2)
cv2.imshow('img', imS)
cv2.waitKey()
```

Modeltesting.ipynb

```
import tensorflow as tf
import os
import cv2
import numpy as np

mymodel1=tf.keras.models.load_model('saved_model/stage1_flip_01')
mymodel2=tf.keras.models.load_model('saved_model/stage1')


predictdata=[]
path=r"C:\Users\azhar\Desktop\project 3c\src\stage1\images\dataset"
os.listdir(path)
imagpath=""

for i in os.listdir(path):
    imgpath=os.path.join(path,i)
    print(imgpath)
    imgarr=cv2.imread(imgpath,0)
    imgarr=cv2.resize(imgarr,(128,128))
    predictdata.append(imgarr)

x=np.array(predictdata)

lookup = {0:'1',1:'2',2:'3',3:'4',4:'5',5:'6',
          6:'7',7:'8',8:'9',9:'A',10:'B',11:'C',12:'D',13:'E',14:'F',15:'G',
          16:'H',17:'I',18:'J',19:'K',20:'L',21:'M',22:'N',23:'O',24:'P',
          25:'Q',26:'R',27:'S',28:'T',29:'U',30:'V',31:'W',32:'X',33:'Y',34:'Z'}

y=mymodel1.predict(x)
z=mymodel2.predict(x)
# y1=mymodel3.predict(x)

for i in range(len(y)):
    ls=y[i]
    ls1=z[i]

    ls11=[ele*1000 for ele in ls]
    ls12=[ele*1000 for ele in ls1]

    maxalgo=[]
    a=[]
    a.append(lookup[ls11.index(max(ls11))])
    maxalgo.append(max(ls11))
    a.append(lookup[ls12.index(max(ls12))])
    maxalgo.append(max(ls12))
```

```
    print(maxalgo," ",a)
```

normalandflipmodel.ipynb

```
import os
import cv2
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import pandas as pd
import tensorflow as tf
from tensorflow import keras
import numpy as np

img_height = 128
img_width = 128
batch_size = 100

import matplotlib.pyplot as plt

ds_train = tf.keras.preprocessing.image_dataset_from_directory(
    "imagedataset/train/",
    labels="inferred",
    label_mode="int",
    color_mode="grayscale",
    batch_size=batch_size,
    image_size=(img_height, img_width),  # reshape if not in this size
    shuffle=True,
    seed=123,
    validation_split=0.01,
    subset="training",)

ds_validation = tf.keras.preprocessing.image_dataset_from_directory(
    "imagedataset/test/",
    labels="inferred",
    label_mode="int",
    color_mode="grayscale",
    batch_size=1
    ,
    image_size=(img_height, img_width),  # reshape if not in this size
    shuffle=True,
    seed=123,
    validation_split=0.95,
    subset="validation",)
```

```
img=cv2.imread("imagedataset/test/6/1001.jpg",0)
#cv2.imshow("test",img)
print(img.shape)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(128, 128, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(35)
])

print(model.summary())

model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=[tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),],
    metrics=["accuracy"],
)

history = model.fit(ds_train, epochs=10, verbose=1)

print(history.history.keys())
print(history.history.values())
print(history.history)

loss = history.history['loss']
accuracy = history.history['accuracy']

epochs = range(1,11)
plt.plot(epochs, loss, 'g', label='Training loss')
plt.plot(epochs, accuracy, 'b', label='accuracy')
plt.title('Training loss and accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

def augment(x, y):
    image = tf.image.random_brightness(x, max_delta=0.05)
    return image, y


ds_train = ds_validation.map(augment)
```

```
new_model = tf.keras.models.load_model('saved_model/stage1_flip')

# Check its architecture
new_model.summary()

new_history=new_model.evaluate(ds_validation)

print(new_history)

new_model.predict()
```

Pickle_model.ipynb

```
import pickle

import tensorflow as tf
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense
from tensorflow.keras.models import Sequential

X_test=pickle.load(open('testingimages.pkl', 'rb'))
Y_test=pickle.load(open('testinglabels.pkl', 'rb'))
X_train=pickle.load(open('trainingimages.pkl', 'rb'))
Y_train=pickle.load(open('traininglabels.pkl', 'rb'))

from tensorflow.keras.utils import to_categorical as toc
Y_train_one=toc(Y_train)
Y_test_one=toc(Y_test)

X_test=pickle.load(open('testingimages.pkl', 'rb'))

def model():
    model=Sequential()
    model.add(Conv2D(36, (3, 3), activation='relu', input_shape=X_train.shape[1:]))
    model.add(MaxPooling2D(2, 2))
    model.add(Conv2D(36, (3, 3), activation='relu'))
    model.add(MaxPooling2D(2,2))
    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(Dense(36,activation='softmax'))

    return model

mymodel=model()

print(mymodel.summary())
mymodel.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
h=mymodel.fit(X_train,Y_train_one,epochs=10,validation_data=(X_test,Y_test_one),bat
ch_size=30)

mymodel.save('saved_model/stage1_03')

history=h

import matplotlib.pyplot as plt

loss_train = history.history['loss']
loss_val = history.history['val_loss']
epochs = range(1,11)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

loss = history.history['val_accuracy']
accuracy = history.history['accuracy']

plt.plot(loss)
plt.plot(accuracy)

mymodel1=tf.keras.models.load_model('saved_model/stage1')

import os
import cv2
import numpy as np

predictdata=[]
path=r"C:\Users\azhar\Desktop\project 3c\src\stage1\images"
os.listdir(path)
imagpath=""
for i in os.listdir(path):
    imgpath=os.path.join(path,i)
    imgarr=cv2.imread(imgpath,0)
    imgarr=cv2.resize(imgarr,(128,128))
    predictdata.append(imgarr)

x=np.array(predictdata)
lookup = {0:'1',1:'2',2:'3',3:'4',4:'5',5:'6',
6:'7',7:'8',8:'9',9:'A',10:'B',11:'C',12:'D',13:'E',14:'F',15:'G',16:'H',17:'I',18:'J',19:'K',20:'L',2
1:'M',22:'N',23:'O',24:'P',25:'Q',26:'R',27:'S',28:'T',29:'U',30:'V',31:'W',32:'X',33:'Y',34:'Z'
}
```

```
y=mymodel1.predict(x)
for i in y:
    ls=list(i)
    ls1=[ele*1000 for ele in ls]
    print(lookup[ls1.index(max(ls1))])
```

Pickledatasetpreparation.ipynb

```
import os
import cv2
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np

data=r"C:\Users\azhar\Desktop\project 3c\src\stage1\imagedataset\train"
labels=['1','2','3','4','5','6','7','8','9',
    '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','Z']
target={'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,
    'A':10,'B':11,'C':12,'D':13,'E':14,'F':15,'G':16,'H':17,'I':18,'J':19,'K':20,

'L':21,'M':22,'N':23,'O':24,'P':25,'Q':26,'R':27,'S':28,'T':29,'U':30,'V':31,'W':32,'X':33,'Y':3
4,'Z':35}

dataset=[]
for i in labels:
    c=0
    path=os.path.join(data,i)
    for img in os.listdir(path):
        imgpath=os.path.join(path,img)
        imgarr=cv2.imread(imgpath)
        imgarr=cv2.resize(imgarr,(128,128))
        print(i,c)
        dataset.append([imgarr,target[i]])
        c+=1

import random
random.shuffle(dataset)

X=[]
Y=[]
for a,b in dataset:
    X.append(a)
    Y.append(b)
```

```
X=np.array(X)
Y=np.array(Y)

import pickle

pickle.dump(X,open('trainingimages.pkl', 'wb'))
pickle.dump(Y,open('traininglabels.pkl', 'wb'))

data=r"C:\Users\azhar\Desktop\project 3c\src\stage1\imagedataset\test"
labels=['1','2','3','4','5','6','7','8','9',
    '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','Z']

testingdataset=[]
for i in labels:
   c=0
   path=os.path.join(data,i)
   for img in os.listdir(path):
      imgpath=os.path.join(path,img)
      imgarr=cv2.imread(imgpath)
      imgarr=cv2.resize(imgarr,(128,128))
      print(i,c)
      testingdataset.append([imgarr,target[i]])
      c+=1

import random
random.shuffle(testingdataset)

X_test=[]
Y_test=[]
for a,b in testingdataset:
   X_test.append(a)
   Y_test.append(b)

X_test=np.array(X_test)
Y_test=np.array(Y_test)

pickle.dump(X_test,open('testingimages.pkl', 'wb'))
pickle.dump(Y_test,open('testinglabels.pkl', 'wb'))
```

svmmodel.ipynb

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense

from tensorflow.keras.regularizers import l2

img_height = 128
img_width = 128
batch_size = 100

ds_train = tf.keras.preprocessing.image_dataset_from_directory(
    "imagedataset/train/",
    labels="inferred",
    label_mode="int",
    color_mode="grayscale",
    batch_size=batch_size,
    image_size=(img_height, img_width),  # reshape if not in this size
    shuffle=True,
    seed=123,
    validation_split=0.01,
    subset="training",
)

ds_validation = tf.keras.preprocessing.image_dataset_from_directory(
    "imagedataset/test/",
    labels="inferred",
    label_mode="int",
    color_mode="grayscale",
    batch_size=1
    ,
    image_size=(img_height, img_width),  # reshape if not in this size
    shuffle=True,
    seed=123,
    validation_split=0.95,
    subset="validation",
)

# Part 2 - Building the CNN
# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Step 1 - Convolution
cnn.add(tf.keras.layers.Conv2D(filters=64,padding="same",kernel_size=3,
activation='relu', strides=2, input_shape=[128, 128, 1]))

# Step 2 - Pooling
```

```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=64,padding='same',kernel_size=3,
activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

# Step 4 - Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

cnn.add(Dense(35, kernel_regularizer=tf.keras.regularizers.l2(0.01),activation
          ='softmax'))
cnn.compile(optimizer = 'adam', loss = 'squared_hinge', metrics = ['accuracy'])

cnn.summary()

r=cnn.fit(x = ds_train, validation_data = ds_validation, epochs = 10)
```

Testing.ipynb

```
import tensorflow as tf
import os
import cv2
import numpy as np

mymodel1=tf.keras.models.load_model('saved_model/stage1_flip_01')
mymodel1.summary()

lookup = {0:'1',1:'2',2:'3',3:'4',4:'5',5:'6',
        6:'7',7:'8',8:'9',9:'A',10:'B',11:'C',12:'D',13:'E',14:'F',15:'G',
        16:'H',17:'I',18:'J',19:'K',20:'L',21:'M',22:'N',23:'O',24:'P',
        25:'Q',26:'R',27:'S',28:'T',29:'U',30:'V',31:'W',32:'X',33:'Y',34:'Z'}

def getresult(image):
   image = np.array(image)
   z=mymodel1.predict(image)
   dic={}
   for i in range(30):
      f=list(z[i])
      t=lookup[f.index(max(f))]
      if t in dic:
         dic[t]+=1
      else:
         dic[t]=1
   dic=dict(sorted(dic.items(), key=lambda item: item[1], reverse=True))
```

```
    for i in dic.keys():
        print(i)
        break

import numpy as np

vid = cv2.VideoCapture(0)
counter=0
arr=[]
while(True):
    ret, frame = vid.read()
    cv2.imshow('frame', frame)
    image=cv2.resize(frame,(128,128))
    image=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    arr.append(image)
    counter+=1
    if counter==60:
        getresult(arr)
        counter=0
        arr=[]
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
vid.release()
cv2.destroyAllWindows()
```

Phase 2 - Sign recognition

Indian sign recognition.ipynb

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp

mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities

def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

```
    return image, results

def draw_landmarks(image, results):
mp_drawing.draw_landmarks(image,results.face_landmarks,mp_holistic.FACEMESH_T
ESSELATION)
mp_drawing.draw_landmarks(image,results.pose_landmarks,mp_holistic.POSE_CONNE
CTIONS)
mp_drawing.draw_landmarks(image,results.left_hand_landmarks,mp_holistic.HAND_C
ONNECTIONS)
mp_drawing.draw_landmarks(image,results.right_hand_landmarks,mp_holistic.HAND_
CONNECTIONS)

def draw_styled_landmarks(image, results):
    # Draw face connections
mp_drawing.draw_landmarks(image,results.face_landmarks,mp_holistic.FACEMESH_T
ESSELATION,mp_drawing.DrawingSpec(color=(80,110,10),thickness=1,
circle_radius=1),          mp_drawing.DrawingSpec(color=(80,256,121),          thickness=1,
circle_radius=1))
    # Draw pose connections
mp_drawing.draw_landmarks(image,results.pose_landmarks,mp_holistic.POSE_CONNE
CTIONS,mp_drawing.DrawingSpec(color=(80,22,10),    thickness=1,    circle_radius=1),
mp_drawing.DrawingSpec(color=(80,44,121), thickness=1, circle_radius=1))
    # Draw left hand connections
mp_drawing.draw_landmarks(image,results.left_hand_landmarks,mp_holistic.HAND_C
ONNECTIONS,mp_drawing.DrawingSpec(color=(121,22,76),thickness=1,circle_radius
=1),mp_drawing.DrawingSpec(color=(121,44,250), thickness=1, circle_radius=1))
    # Draw right hand connections
mp_drawing.draw_landmarks(image,results.right_hand_landmarks,mp_holistic.HAND_
CONNECTIONS,mp_drawing.DrawingSpec(color=(245,117,66),thickness=1,circle_radi
us=1),mp_drawing.DrawingSpec(color=(245,66,230), thickness=1, circle_radius=1))

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)
as holistic:
    while cap.isOpened():
        # Read feed
        ret, frame = cap.read()
        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)
        # Draw landmarks
        draw_styled_landmarks(image, results)
        # Show to screen
        cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
            break
    cap.release()
    cv2.destroyAllWindows()
draw_landmarks(frame, results)

plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
len(results.left_hand_landmarks.landmark)
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
Pose = np.array([[res.x,res.y,res.z,res.visibility]forres in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(132)
face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten()
if results.face_landmarks else np.zeros(1404)
lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else
np.zeros(21*3)
rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else
np.zeros(21*3)

face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten()
if results.face_landmarks else np.zeros(1404)

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else
np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks else
np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else
np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else
np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])

result_test = extract_keypoints(results)

Result_test

np.save('0', result_test)

np.load('0.npy')
```

```
DATA_PATH = os.path.join(r'C:\Users\azhar\Desktop\project 3c\src\stage2\Sign_Data')

# actions = np.array(['bye', 'good', 'hello' , 'iam' , 'name' , 'thanks'])
actions = ["hello",'thanks','good']

no_sequences = 30

sequence_length = 30

start_folder = 0
print(DATA_PATH)

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5)
as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(start_folder, start_folder+no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)

                # Draw landmarks
                draw_styled_landmarks(image, results)

                # NEW Apply wait logic
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                                cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(1000)
                else:
                                cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
```

```
cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)

            # NEW Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
            np.save(npy_path, keypoints)

            # Break gracefully
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break

    cap.release()
    cv2.destroyAllWindows()

import os
npy_path = os.path.join(DATA_PATH, 'hello', str(1), str(0))
print(npy_path)

from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

label_map = {label:num for num, label in enumerate(actions)}
Label_map

sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence),
"{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])

np.array(sequences).shape
np.array(labels).shape
X = np.array(sequences)
X.shape
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
Y_test.shape

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
```

```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)

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(3, activation='softmax'))

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

model.summary()

model.fit(X_train, y_train, epochs=200, callbacks=[tb_callback])
res = model.predict(X_test)

actions[np.argmax(res[3])]
actions[np.argmax(y_test[3])]
model.save('threeword3.h5')
model.load_weights('threeword.h5')
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
yhat = model.predict(X_test)
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
print(ytrue,yhat)
multilabel_confusion_matrix(ytrue, yhat)
accuracy_score(ytrue, yhat)
from scipy import stats

colors = [(245,117,16), (117,245,16), (16,117,245)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40),
colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)
```

```
    return output_frame

plt.figure(figsize=(18,18))
plt.imshow(prob_viz(res, actions, image, colors))

import pyttsx3
engine = pyttsx3.init()
engine.setProperty("rate", 150)
def play_aud(text):
    engine.say(text)
    engine.runAndWait()

# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    words={}
    while cap.isOpened():
        # Read feed
        ret, frame = cap.read()
        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)
        # Draw landmarks
        draw_styled_landmarks(image, results)
        # 2. Prediction logic
        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-30:]
        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            predictions.append(np.argmax(res))

            if np.unique(predictions[-10:])[0]==np.argmax(res):
                if res[np.argmax(res)] > threshold:
                    if len(sentence) > 0:
                        if actions[np.argmax(res)] != sentence[-1]:
```

```
                    sentence.append(actions[np.argmax(res)])
                    play_aud(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])
                    play_aud(actions[np.argmax(res)])
            if len(sentence) > 5:
                sentence = sentence[-5:]
            image = prob_viz(res, actions, image, colors)
        cv2.imshow('OpenCV Feed', image)
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
cap.release()
cv2.destroyAllWindows()
```

Phase 3 - Sign recognition

Faceemotionrecognisation.py

```
from fer import FER
import matplotlib.pyplot as plt

test_image_one = plt.imread("test.jpg")
emo_detector = FER(mtcnn=True)
captured_emotions = emo_detector.detect_emotions(test_image_one)
print(captured_emotions)
plt.imshow(test_image_one)

dominant_emotion, emotion_score = emo_detector.top_emotion(test_image_one)
print(dominant_emotion, emotion_score)
```

8.2 SCREENSHOTS OF RESULTS

Fig 8.2.1 Test images to test the model

```
[26913.053512573242, 20663.593292236328]   ['1', '1']
[50278.160095214844, 23984.33494567871]   ['2', '2']
[32546.08154296875, 32486.282348632812]   ['3', '3']
[29165.180206298828, 35007.36618041992]   ['4', '4']
[34480.35430908203, 30196.508407592773]   ['5', '5']
[21971.454620361328, 27989.97688293457]   ['6', '6']
[57980.93032836914, 29609.506607055664]   ['7', '7']
[25168.964385986328, 24111.034393310547]   ['8', '8']
[28849.733352661133, 28793.991088867188]   ['9', '9']
[37121.14715576172, 41535.50720214844]   ['A', 'A']
```

Fig 8.2.2 Results corresponding to test images

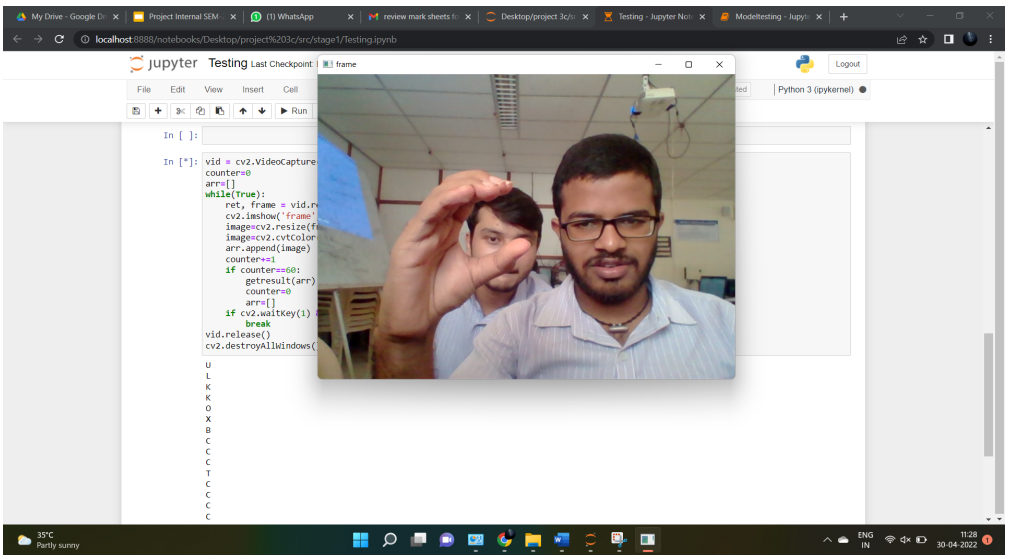


Fig 8.2.3 Realtime alphabet recognition from video

Fig 8.2.4 Landmarks detected by mediapipe and making of dataset

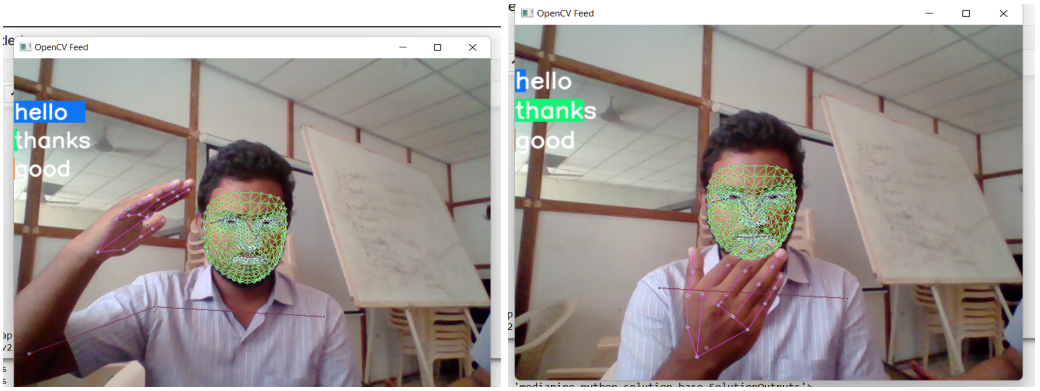
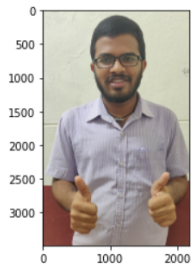


Fig 8.2.5 Real Time Sign recognition with accuracy

```
[{'box': [756, 425, 725, 883], 'emotions': {'angry': 0.0, 'disgust': 0.0, 'fear': 0.0, 'happy': 0.97, 'sad': 0.0, 'surprise':
0.0, 'neutral': 0.02}}, {'box': [1152, 2261, 41, 47], 'emotions': {'angry': 0.16, 'disgust': 0.0, 'fear': 0.08, 'happy': 0.18,
'sad': 0.15, 'surprise': 0.0, 'neutral': 0.43}}]
happy 0.97
```



2

Fig 8.2.6 Face emotion recognition results

```
multilabel_confusion_matrix(ytrue, yhat)
```

```
array([[[2, 0],
        [0, 3]],

       [[3, 2],
        [0, 0]],

       [[3, 0],
        [2, 0]]], dtype=int64)
```

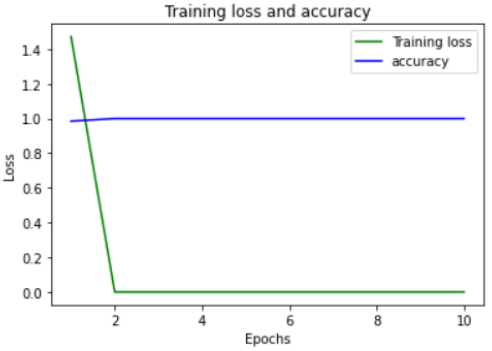
```
accuracy_score(ytrue, yhat)
```

```
0.6
```

Fig 8.2.7 Confusion matrix for LSTM

```
loss = history.history['loss']
accuracy = history.history['accuracy']

epochs = range(1,11)
plt.plot(epochs, loss, 'g', label='Training loss')
plt.plot(epochs, accuracy, 'b', label='accuracy')
plt.title('Training loss and accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
loss = history.history['val_accuracy']
accuracy = history.history['accuracy']

plt.plot(loss)
plt.plot(accuracy)
```

```
[<matplotlib.lines.Line2D at 0x1ce112b6100>]
```

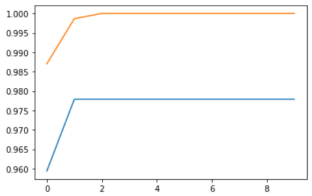


Fig 8.2.8 Accuracy graphs obtained on training (Dense)

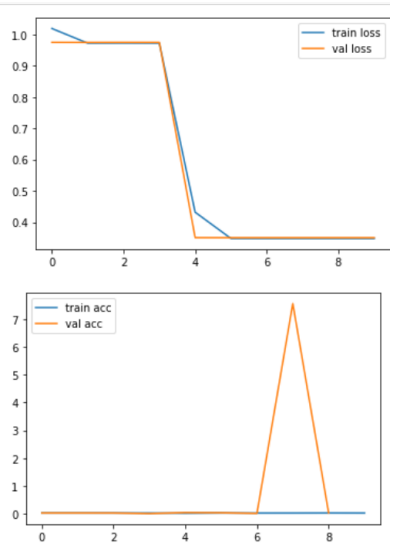


Fig 8.2.9 Accuracy graphs (SVM)

9. TESTING

The purpose of testing is to discover errors. Testing is a process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner.

Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well planned through testing.

Testing Objectives

There are several rules that can serve as testing objectives they are:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is the one that has a high probability of finding an undiscovered error.

Types of Testing

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at different phases of software development are :

- Unit testing
- Black box testing
- White box testing
- Integration testing
- System testing
- Acceptance testing

Unit Testing

Unit testing is done on individual models as they are completed and becomes executable. It is confined only to the designer's requirements. Unit testing is different from and should be preceded by other techniques, including: Inform Debugging Code Inspection

Black Box testing

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been used to find error in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures are external database access
- Performance error
- Initialisation and termination of errors
- In this testing only the output is checked for correctness
- The logical flow of data is not checked

White Box testing

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been used to generate the test cases in the following cases:

- Guarantee that all independent paths have been executed
- Execute all loops at their boundaries and within their operational bounds.
- Execute internal data structures to ensure their validity.

IntegrationTesting

Integration testing ensures that software and subsystems work together a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together. It is typically performed by developers, especially at the lower, module to module level. Testers become involved in higher levels

System Testing

Involves in house testing of the entire system before delivery to the user. The aim is to satisfy the user that the system meets all requirements of the client's specifications. It is conducted by the testing organization if a company has one. Test data may range from and generated to production.
Requires test scheduling to plan and organize:

- Inclusion of changes/fixes.
- Test data to use

One common approach is graduated testing: as system testing progresses and (hopefully) fewer and fewer defects are found, the code is frozen for testing for increasingly longer time periods.

Acceptance Testing

It is a pre-delivery testing in which the entire system is tested at the client's site on real world data to find errors.

User Acceptance Test (UAT)

“Beta testing”: Acceptance testing in the customer environment.

Requirements Traceability:

- Match requirements to test cases.
- Every requirement has to be cleared by at least one test case.
- Display in a matrix of requirements vs. test cases.

ID	**Test case**	**Input Description**	**Expected Output**	**Test Status**
1	Loading model	Initializing trained model and load it into ON	Loaded model without errors	pass
2	Converting video to frames	Capturing video and converting it into frames	Image frames of captured video stream	pass
3	Recognize hand gesture	Image frame that contains hand object	Label	pass
4	Recognize Sign	Recognize the trained Sigs Shown by the us	Audio of detected sig	pass
5	Recognize all signs	Recognize all The alphabet Signs successfully	Label	fail

Table 9.1: verification of test cases

In test case 5 there is low accuracy due to the consideration of the whole frame for the extraction of the features. Instead if we just give the hand after extracting it from the image we can improve the accuracy.

10. CONCLUSION AND FUTURE SCOPE

The Sign Language Translation system can change the lives of many people across the world. It can be used to talk to the mute people and understand them better. This will help to communicate with children at the younger ages well and therefore help them to get out of their loneliness and depression. In Adults this translation system opens up more opportunities to those who are backward until now due to the communication gap.

The development of this system helps the people using Indian sign language and also using this model we can change the training dataset and implement translators for any sign language. Also we can use transfer learning to increase the sign space for recognition. Adding sign generation for English to this system will lead to an even more decrease in communication gap.

With all the advancements in the area of machine learning and artificial intelligence it is high time that we overcame this communication barrier. Our initiative is to make this gap decrease even by a small amount.

Future Scope

We can use the same process to identify signs of any sign language just by changing the dataset used in training and other features to make the system more feasible. The words generated can be passed to a sentence generator to frame a proper English sentence using the generated words. Using text to sign language generator api in combination with our system we can completely build an end-to-end sign language translation system.

11. REFERENCES

Base Paper:

[1] https://www.sciencedirect.com/science/article/pii/S1877050917320720

[2] https://www.sciencedirect.com/science/article/pii/S1877050918321331

[3] Sign Language Translator, Divyanshu Mishra, Medhavi Tyagi, and Ankur Verma, Gaurav Dubey, ABES Engineering College, Ghaziabad, U.P., 201009, India

Other references:

[4]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[5] https://en.wikipedia.org/wiki/Long_short-term_memory

[6] https://indiansignlanguage.org/

[7] http://www.islrtc.nic.in/

[8] https://ieeexplore.ieee.org/document/7507939

[9] https://ieeexplore.ieee.org/document/7916786

[10] https://www.sciencedirect.com/science/article/pii/S1877050917320720

[11]https://medium.com/@RaghavPrabhu/understanding-of-convolutia-neural network-cnn-deep-learning-99760835f148