ABSTRACT

Hand gesture-based Sign Language Recognition (SLR) serves as a crucial communication bridge between hard of hearing and non-deaf individuals. The absence of a universal sign language (SL) leads to diverse nationalities having various cultural SLs, such as Korean, American, and Japanese sign language. Existing SLR systems perform well for their cultural SL but may struggle with other or multicultural sign languages (McSL). To address these challenges, this paper introduces a novel end-to-end SLR system called GmTC, designed to translate McSL into equivalent text for enhanced understanding. Here, we employed a Graph and General deep-learning network as two stream modules to extract effective features. In the first stream, produce a graph-based feature by taking advantage of the super pixel values and the graph convolutional network (GCN), aiming to extract distance-based complex relationship features among the super pixel. In the second stream, we extracted long-range and short-range dependency features using attention-based contextual information that passes through multi-stage, multi-head self-attention (MHSA), and CNN modules. Combining these features generates final features that feed into the classification module. Extensive experiments with five culture SL datasets with high-performance accuracy compared to existing state-of-the-art models in individual domains affirming superiority and generalizability.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

CHAPTER-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 of 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 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 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.

• Analysing and manipulating the image.

• Output in which 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 2 digital techniques are pre- processing, 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, rely 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.

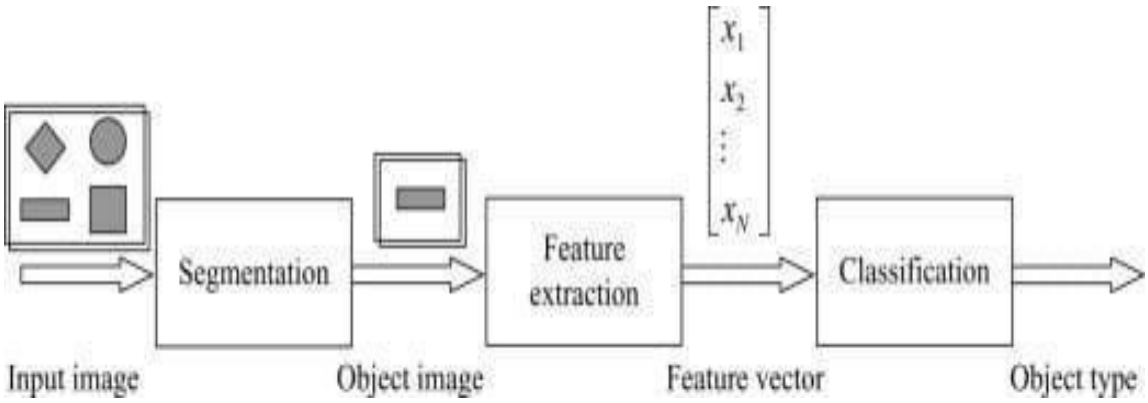


Fig 1.1 Phases of pattern recognition

The first phase includes the image segmentation and object separation. In this phase, different objects are detected and separate from other background. The second 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 third 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 structural analysis is a description of images in order to correctly understand and judge for the important information of images.

1.2 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 as, British, Indian 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 communicate 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.

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

1.3 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 is 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 so they have to depend on some sort of visual communication in 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 5 or the deaf community.

The importance of sign language is emphasized by the growing public approval and funds for international project. 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 a user-friendly human computer interface (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.

1.4 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 per cent of 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 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. 6 Normal people face difficulty in understanding their language. Hence there is a need of 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.

1.6 ORGANIZATION OF THESIS

The book is organised 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 the experimental analysis, the code involved and the results obtained.

Part 6: Concludes the project and provides the scope to which the project can be extended.

CHAPTER-2

LITERATURE SURVEY

2.1 SURVEY WALKTHROUGH:

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 Deep Dream.

2.1.2 OpenCV:

OpenCV (Opensource Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. 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 and 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 Neighbors (KNN) 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. CVIP-tools, 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 opensource 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 fat any pair of coordinates (x, y) is called the intensity or grey 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 colour it should be.

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

Image processing basically includes the following three steps:

Importing the image, Analysing and manipulating the image.

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 behaviours
- 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 Plaid ML. 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, and a 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 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 file so we have to download and extract the feature from ImageNet database. Some of the popular pretrained 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, multi-dimensional 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 plotting 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 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 it is 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's 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.

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. CPython and Pythran are static-compiling alternatives to these.

2.1.5 Neural Networks:

A neural network is a series of algorithms that endeavours 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), perceptions 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

Followings 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

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 − Multilayer networks Multilayer networks with recurrent connections 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.

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. 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 kind of applications.

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.

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 pre-processed. Then, the dimensionality of that image must be reduced. And, at last it must be classified using neural network training algorithm. Following neural networks are used for training purposes with pre-processed image – Fully-connected multilayer feed-forward neural network trained with the help of back- propagation algorithm. For dimensionality reduction, Principal Component Analysis PCA is used.

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 unlabelled, 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, unlabelled 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 scientist 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 unlabelled 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 examples, 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 labelled data.

A deep-learning network trained on labelled data can then be applied to unstructured data, giving it access to much more input than machine-learning nets.

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 search 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 baselevel. 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.

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 labelling as well. Analysing 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 it's complete testing is yet to be widely seen.

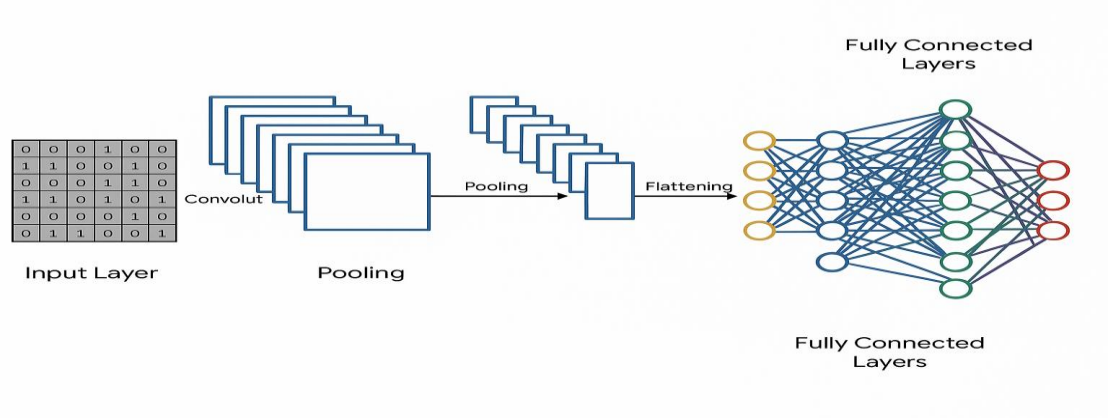


Fig 2.1 Layers involved in CNN

2.1.6 EXISTING SYSTEM

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,

A Survey of Hand Gesture Recognition Methods in Sign Language Recognition

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 analyse 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 method and Deep Learning. This paper focuses on the classification methods used in prior Sign Language Recognition system. 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 analyse 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. [5]

Communication between Deaf-Dumb People and Normal People

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 different people in different languages but there is not such a chat application that has facilitate to communicate with sign languages. The developed system is based on Sinhala Sign language. The system has included 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 has 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 is displayed by emoji. Those emoji and signs that are included in this system will bring the normal people more closed to the disabled people. This is a 2 ways communication system but it uses pattern of gesture recognition which is not very reliable in getting appropriate output. [40]

A System for Recognition of Indian Sign Language for Deaf People using Otsu's Algorithm

In this paper we proposed some methods, through which the recognition of the signs becomes easy for peoples while communication. And the result of those symbols signs will be converted into the text. In this project, we are capturing hand gestures through webcam and convert this image into gray scale image. The segmentation of gray scale image of a hand gesture is performed using Otsu thresholding algorithm. Total image level is divided into two classes one is hand and other is background. The optimal threshold value is determined by computing the ratio between class variance and total class variance. To find the boundary of hand gesture in image Canny edge detection technique is used. In Canny edge detection we used edge-based segmentation and threshold-based segmentation. Then Otsu's algorithm is used because of its simple calculation and stability. This algorithm fails, when the global distribution of the target and background vary widely. [40]

Intelligent Sign Language Recognition Using Image Processing

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 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's hand gestures and generate easily recognizable form 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]

Sign Language Recognition Using Image Processing

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 gesture 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 train a deaf and dumb. [9]

Sign Language Interpreter using Image Processing and Machine Learning

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 pre-processed 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. [4]

Hand Gesture Recognition based on Digital Image Processing using MATLAB

This research work presents a prototype system that helps to recognize hand gesture to normal people in order to communicate more effectively with the special people. Aforesaid research work focuses on the problem of gesture recognition in real time that sign language used by the community of deaf people. The problem addressed is based on Digital Image Processing using Colour Segmentation, Skin Detection, Image Segmentation, Image Filtering, and Template Matching techniques. This system recognizes gestures of ASL (American Sign Language) including the alphabet and a subset of its words. [9]

GESTURE RECOGNITION SYSTEM

Communication plays a crucial part in human life. It encourages a man to pass on his sentiments, feelings and messages by talking, composing or by utilizing some other medium. Gesture based communication is the main method for Communication for the discourse and hearing weakened individuals. Communication via gestures is a dialect that utilizations outwardly transmitted motions that consolidates hand signs and development of the hands, arms, lip designs, body developments and outward appearances, rather than utilizing discourse or content, to express the individual's musings. Gestures are the expressive and important body developments that speaks to some message or data. Gestures are the requirement for hearing and discourse hindered; they pass on their message to others just with the assistance of motions. Gesture Recognition System is the capacity of the computer interface to catch, track and perceive the motions and deliver the yield in light of the caught signals. It enables the clients to interface with machines (HMI) without the any need of mechanical gadgets. There are two sorts of sign recognition methods: image- based and sensor- based strategies. Image based approach is utilized as a part of this project that manages communication via gestures motions to distinguish and track the signs and change over them into the relating discourse and content. [20]

2.2 PROPOSED SYSTEM

Our proposed system is 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.

2.2.1 System Architecture

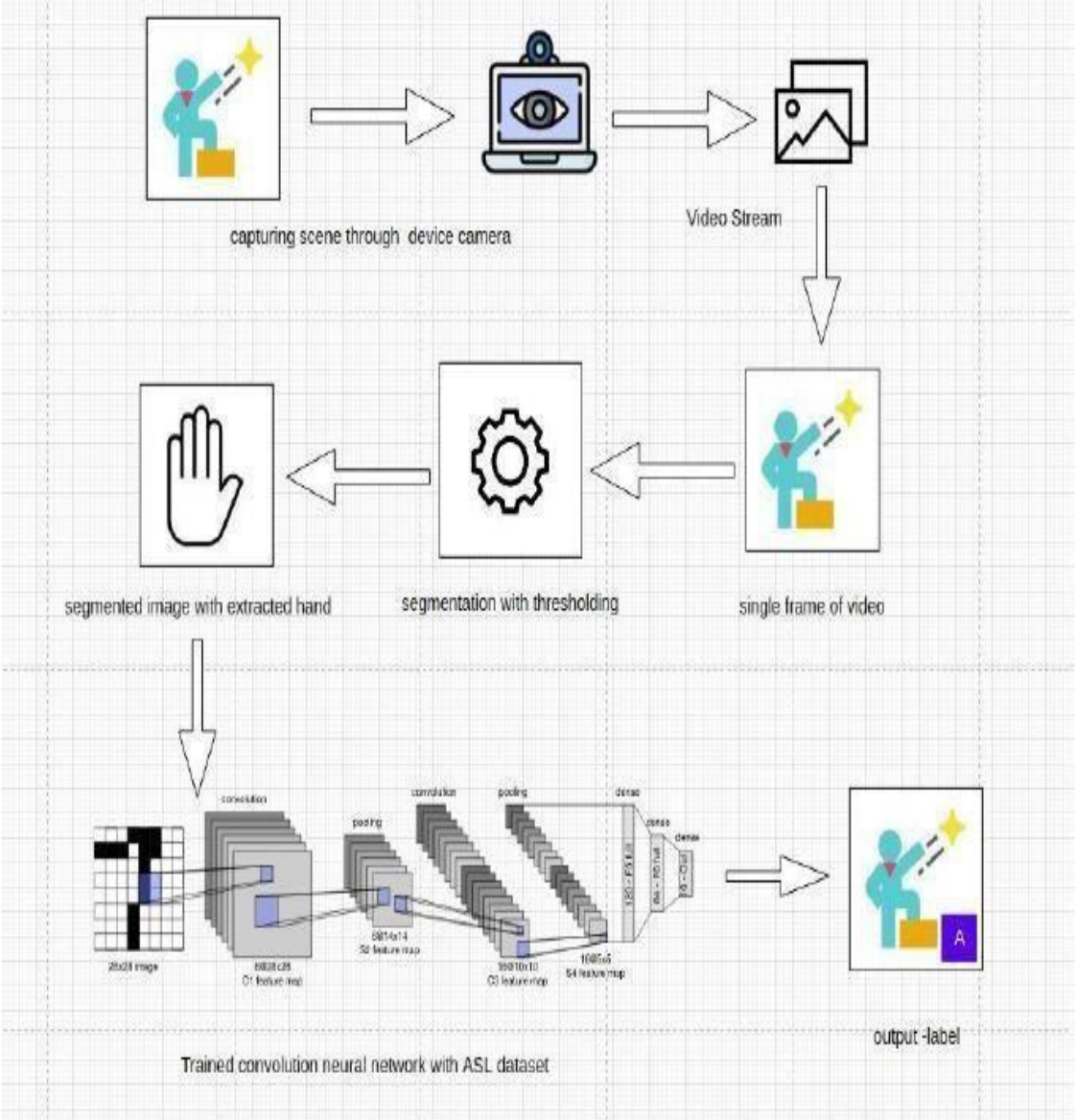


Fig 2.2 Architecture of Sign Language recognition System

CHAPTER – 3

METHODOLOGY

3.1 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 projects case, it was neural networks.

Such an algorithm looks like:

1. Begin with its object:
   model = Sequential ()
2. Add layers with specific 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_optimizer_algorithm')

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's looked 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 it's 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.

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 same aspect ratio, they'll all crop the same way on your site. 7

Option 1 - Crop to a pre-set 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 a visible quality loss. From the standpoint of digital signal processing, the scaling of raster graphics is a two-dimensional example of sample-rate conversion.

DATASETS USED FOR TRAINING AND TESTING

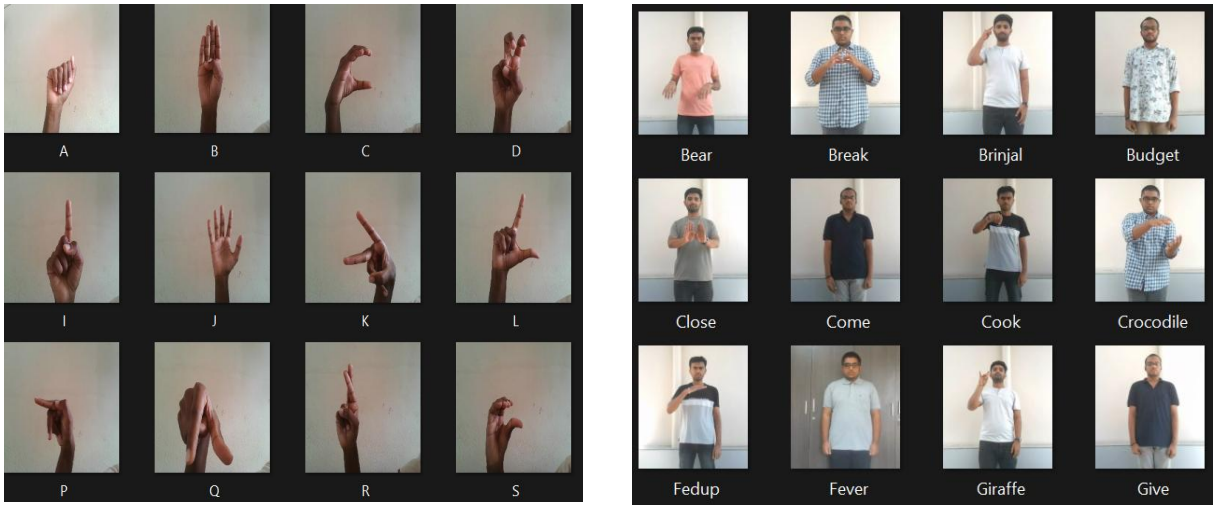


Fig 3.1 Sample dataset from train set

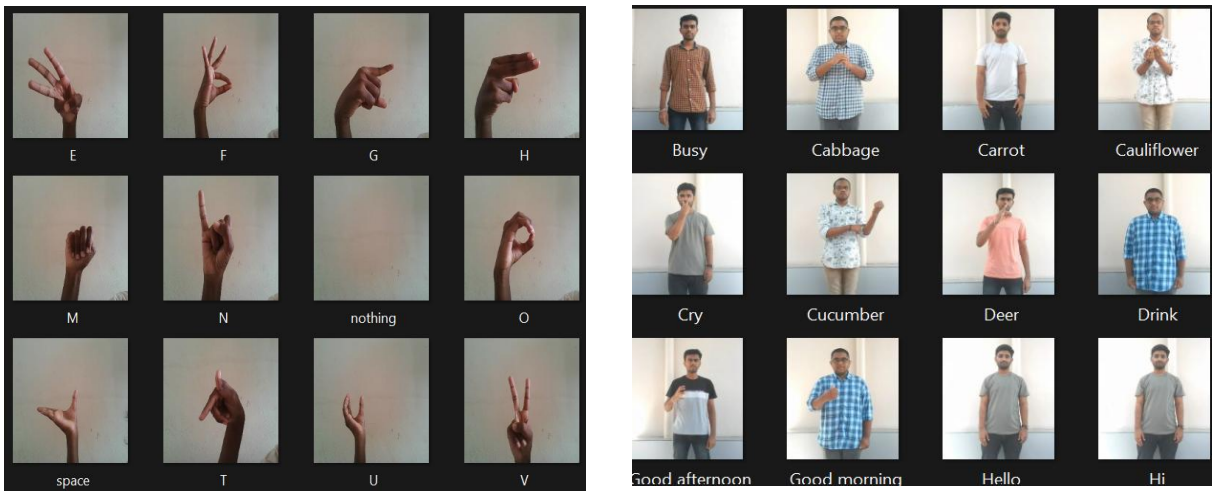


Fig 3.2 Sample dataset from test set

3.2 ALGORITHM

HISTOGRAM CALCULATION:

Histograms are collected counts of data organized into a set of predefined bins When we say data we are not restricting it to be intensity value. The data collected can be whatever feature you find useful to describe your image. What happens if we want to count this data in an organized way? Since we know that the range of information value for this case is 256 values, we can segment our range in subparts (called bins) like: [0,255] = [0,15] ∪ [16,31] ∪ .... ∪ [240, 255] range=bin1∪bin2∪. ∪binn=15 and we can keep count of the number of pixels that fall in the range of each bini.

Back-Propagation: 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 respects to all the weights in the network.

Optimizer (Adam): Adam can be looked at as a combination of RMSprop 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 moving average of the gradient instead of 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 to adapt the learning rate for each weight of the neural network. Now, what is moment? Nth 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.

3.3 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 all 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 labelled 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.

3.4 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 encoded 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 best on 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 labelled 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 featured 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. Convolution formula:

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

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 time, 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 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 grey and white colours). The essential difference between the non-rectified version of the image and the rectified one is the progression of colours. After we rectify the image, you will find the colours 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 from 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.

3.5 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, subassemblies, 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 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:

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.

Integration Testing:

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 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 entire system is tested at 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 into frame	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

Table 3.1 Verification of testcases

3.6 DESIGN

Dataflow Diagram

The DFD is also known as 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 analyse 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:

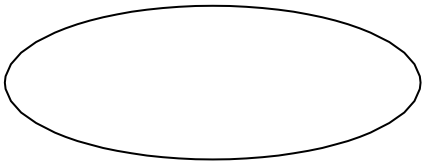
It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators, sources and 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 for sort data based on logic or direct the data flow based on business rules.

Representation:



3) Data Flow:

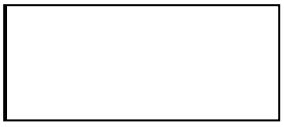
A dataflow represents a package of information flowing between two objects in the data-flow diagram, Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

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:



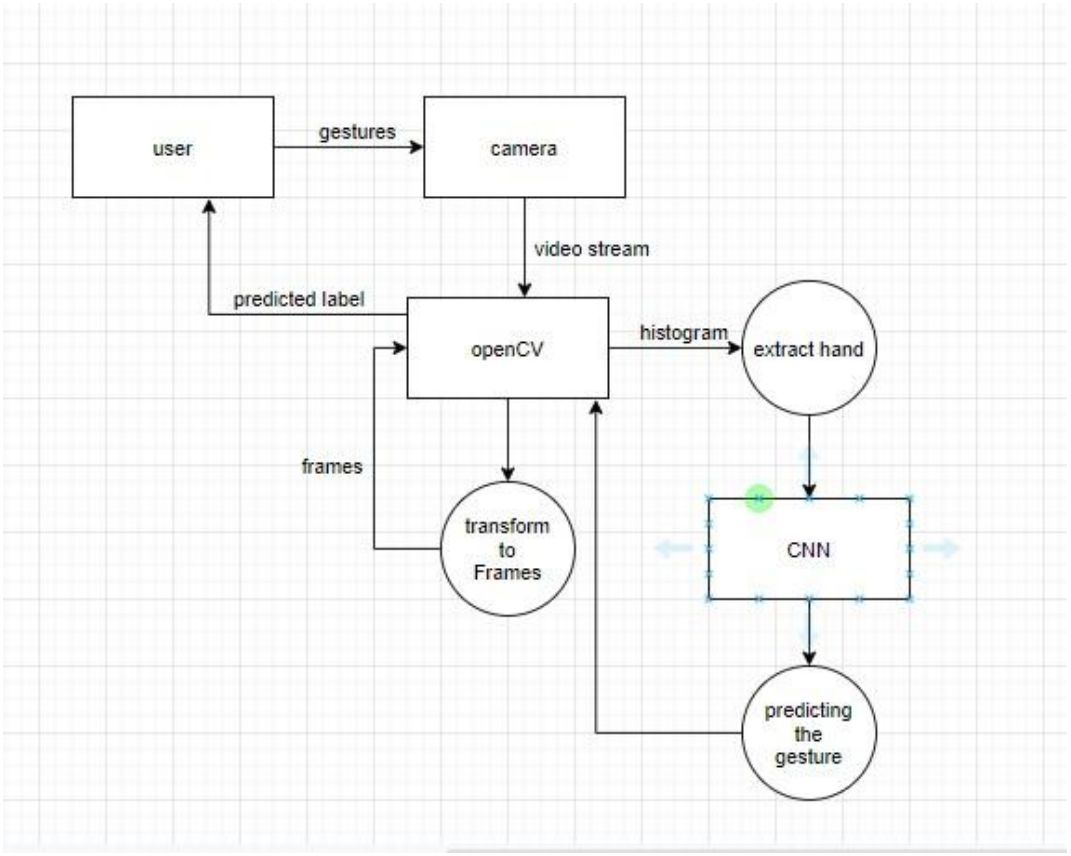


Fig 3.3 Dataflow Diagram for Sign Language Recognition

3.6.1 UML DIAGRAMS

UML stands for Unified Modelling Language. Taking SRS document of analysis as input to the design phase drawn UML diagrams. The UML is only language so 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 language for visualizing, specifying, constructing, documenting the articles in a software- intensive system. It is based on diagrammatic representations of software components.

A modelling language is a language whose vocabulary and rules focus on the conceptual and physical representation of the system. A modelling 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.

3.6.2 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 behaviour 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 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 State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from 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 analysed to gather its functionalities, use cases are prepared and actors are identified. When the initial task is complete, use case diagrams are modelled 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 analysed, 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.

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.

The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.

Give a suitable name for actors.

Show relationships and dependencies clearly in the diagram.

Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.

Use notes whenever required to clarify some important points.

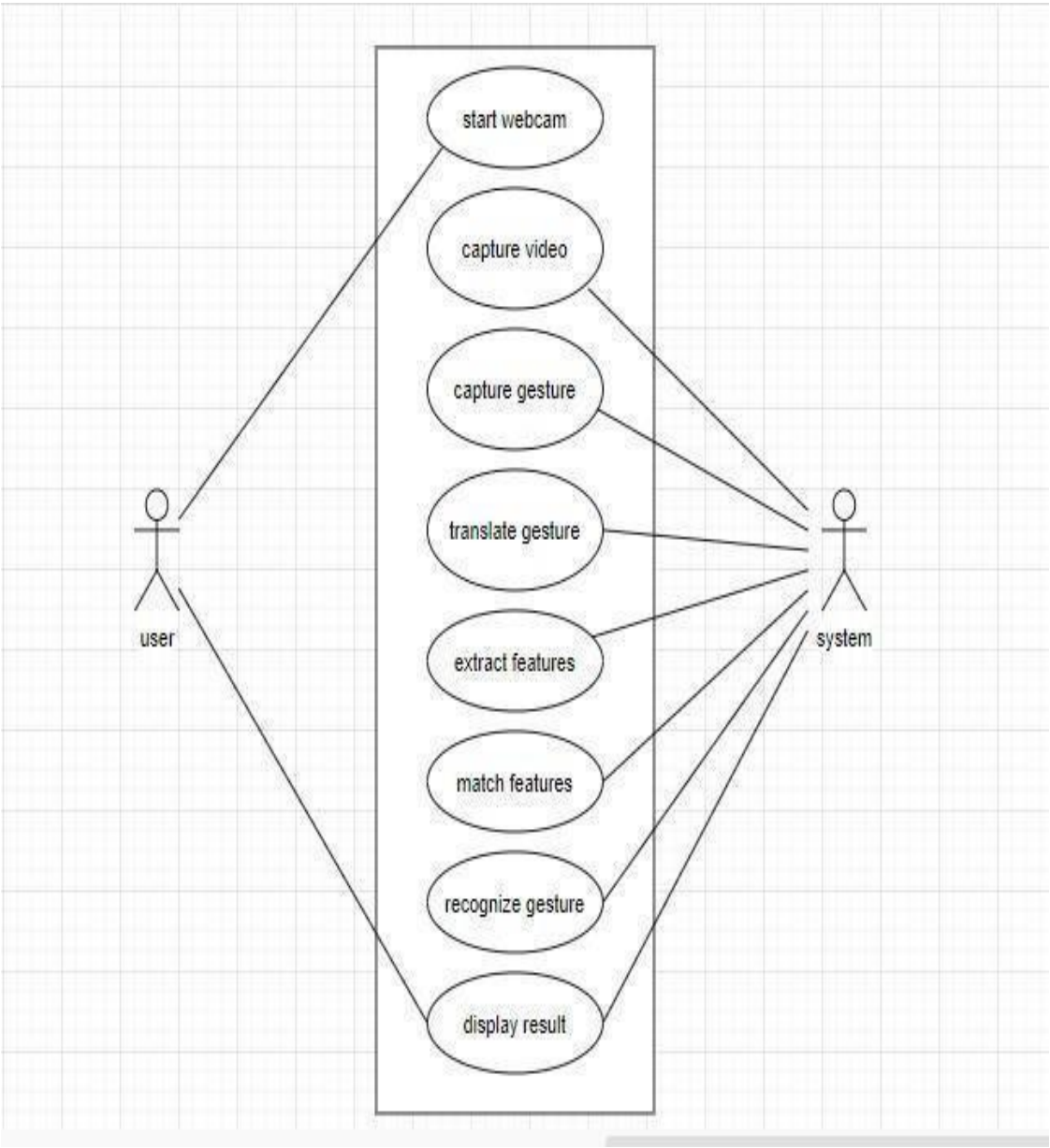


Fig 3.4 Use case diagram of sign language recognition System

Use case element	Details
Use Case Name	Sign Language Recognition
Participating Actors	User, System
Brief Description	This use case describes the process of recognizing and translating sign language gestures using a webcam and machine learning techniques.
Preconditions	- The system is installed and running. - The webcam is functional and accessible. - User is within the camera's view.
Postconditions	- Recognized gesture is displayed as text. - System ready for next gesture input.
Main Flow of Events	1. User starts the system. 2. System activates the webcam. 3. System captures live video. 4. System detects hand gesture. 5. System extracts gesture features. 6. System matches features with known gestures. 7. System recognizes the gesture. 8. System translates the gesture. 9. System displays the result.
Alternative Flows	- **Gesture not recognized:** System displays "Unrecognized gesture" - **Webcam error:** System shows an error and prompts to fix the camera.
Exceptions	- Poor lighting or occluded hands may cause recognition failure. - Invalid input or noise may cause false detection.
Priority	High
Frequency of Use	Frequently used during communication sessions by hearing/speech-impaired users.

Table 3.2 Use case Scenario for sign language recognition system

3.6.3 Class Diagram

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram described the different perspective when designing a system conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also displays relationships such as containment, inheritance, association etc. The association relationship is 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 diagram. 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 represents the whole 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, 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 drawings. 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.

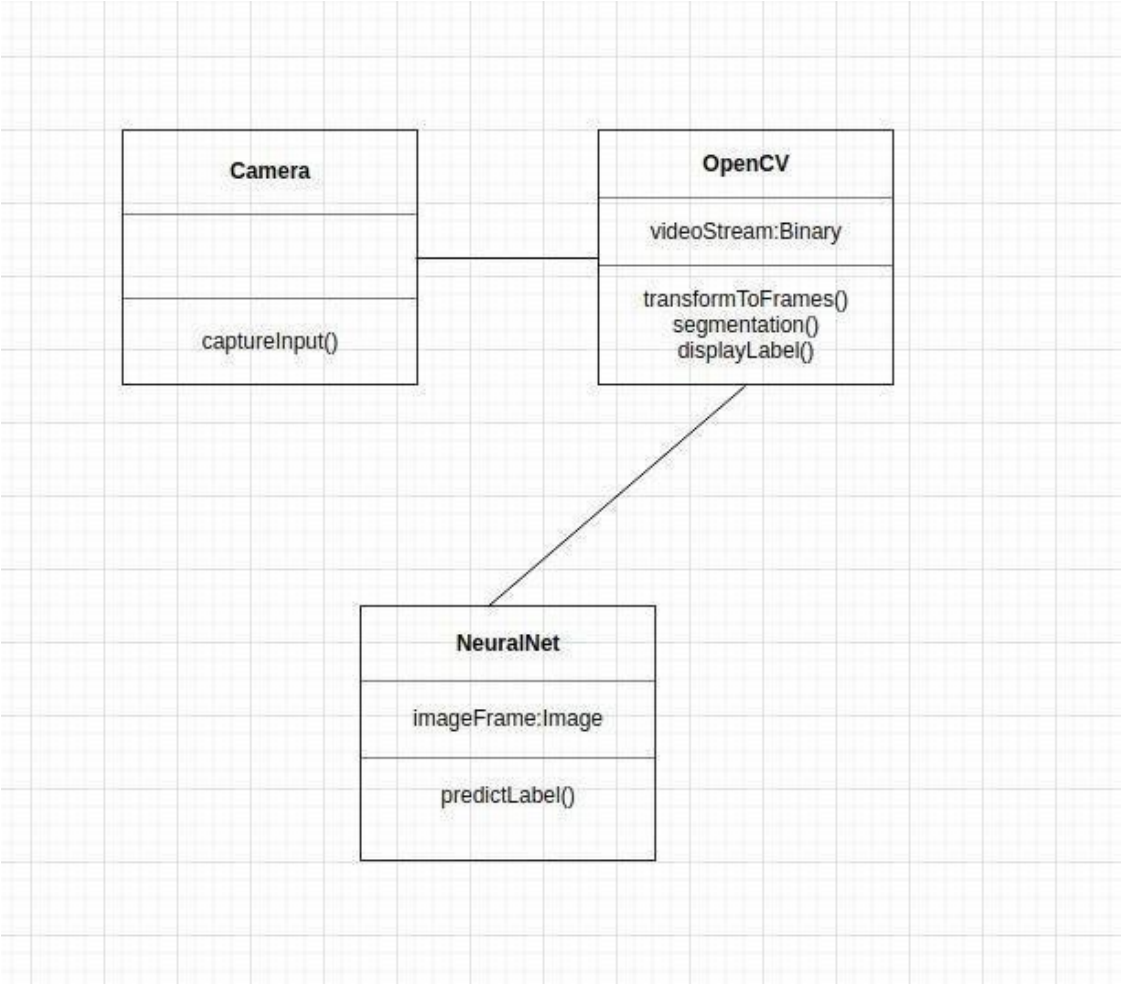


Fig 3.5 Class diagram of sign language recognition system

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

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.

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 arrow heads represent synchronous calls, open arrow heads 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 (Execution Specifications 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 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.

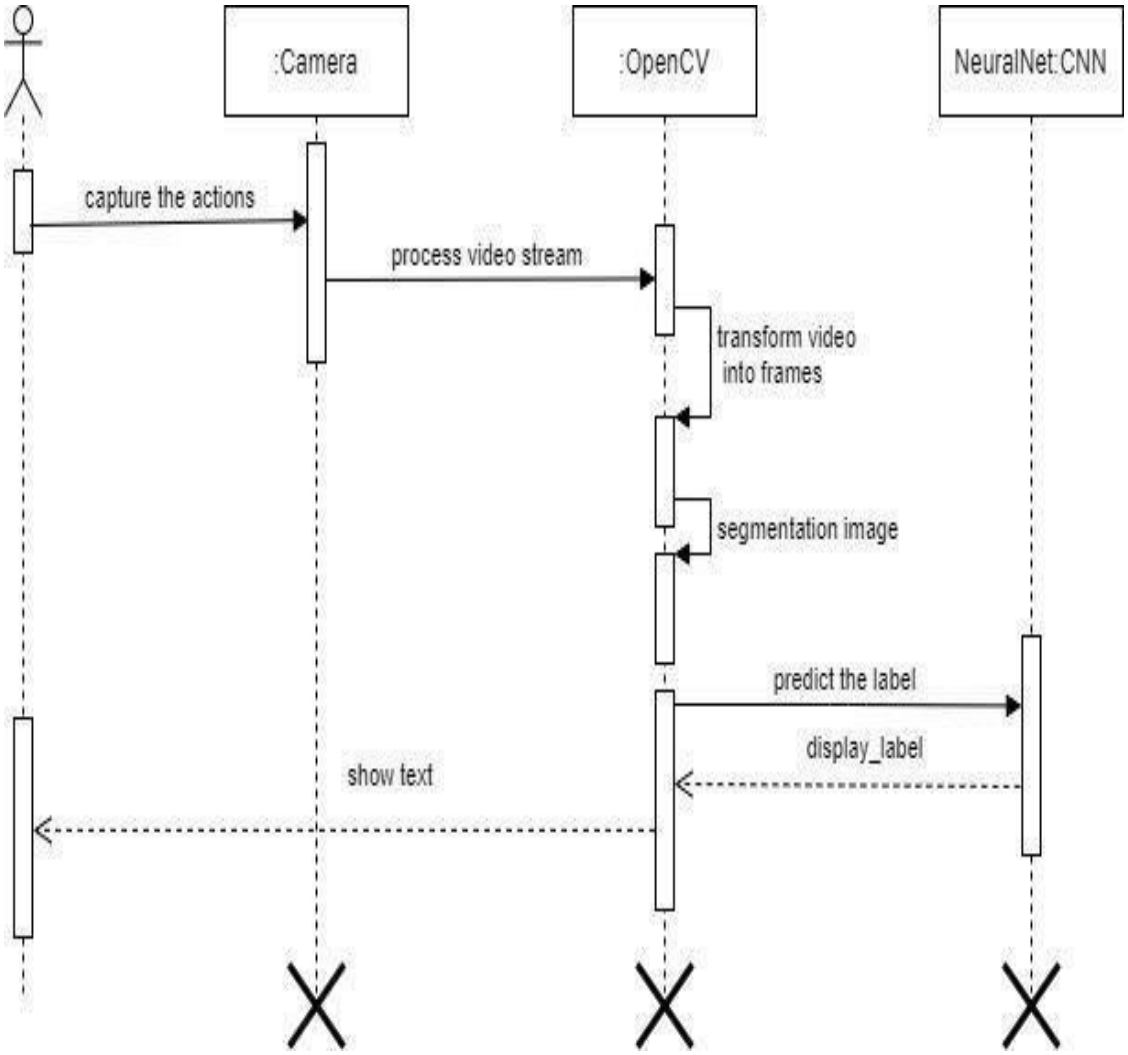


Fig 3.6 Sequence diagram of sign language recognition system

3.6.5 State Chart

A state chart diagram describes a state machine which shows the behaviour of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behaviour of objects over time by model1ing 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 State Chart Diagram:1. Initial State 2. Final-State. Some of the components of State Chart Diagram are:

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

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

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

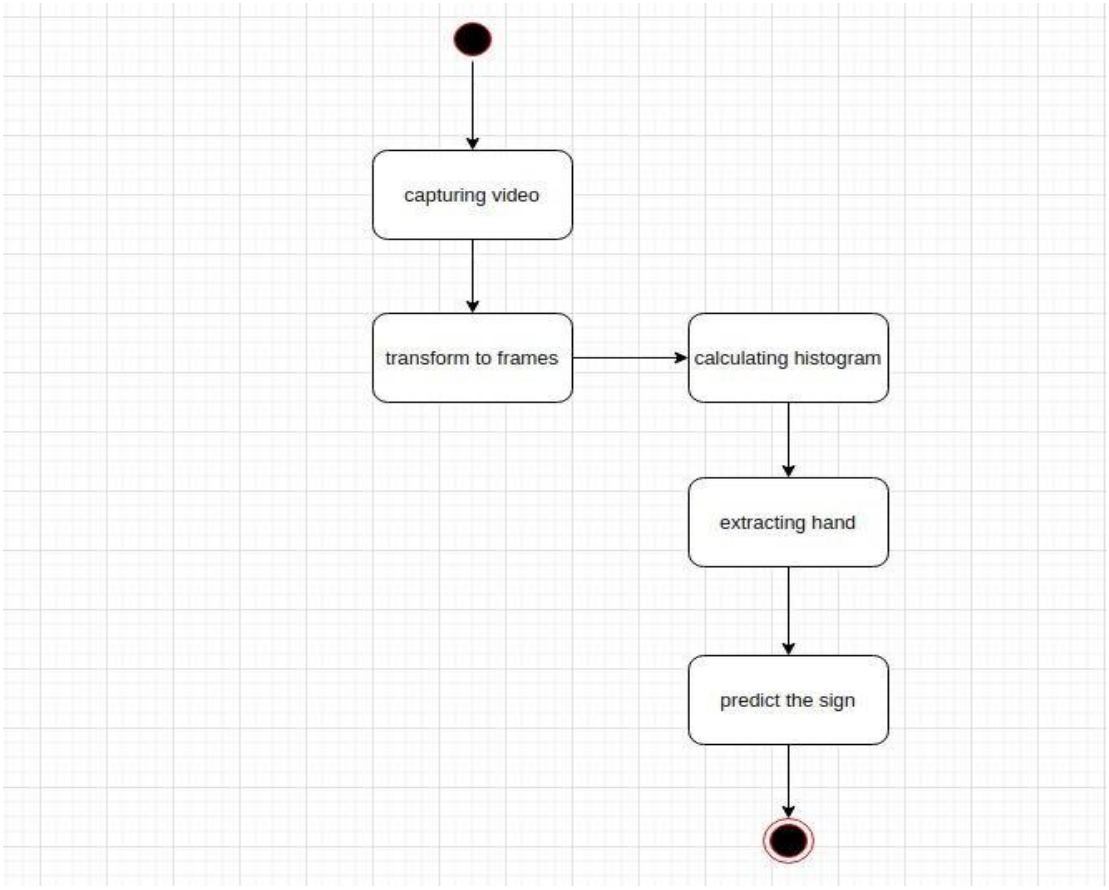


Fig 3.7 State Chart diagram of sign language recognition system

3.7 SYSTEM REQUIREMENTS

3.7.1 Software Requirements

There are several software requirements that must be met for software to function on a computer, including resources requirements and prerequisites. The minimal requirements are as follows,

- Raspbian OS
- Anaconda with Spyder

3.7.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. The minimal hardware requirements are as follows,

- Raspberry Pi B+
- Camera Module
- 8 GB SD Card

3.8 PROCESSING MODULE

Raspberry Pi is a small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation. The organization behind the Raspberry Pi consists of two arms. The first two models were developed by the Raspberry Pi Foundation. The Raspberry Pi hardware has evolved through several versions that feature variations in memory capacity and peripheral- device support.

The Raspberry Pi device looks like a motherboard, with the mounted chips and ports exposed (something you'd expect to see only if you opened up your computer and looked at its internal boards), but it has all the components you need to connect input, output, and storage devices and start computing.

Raspberry Pi is a low-cost, basic computer that was originally intended to help spur interest in computing among school-aged children.

The Raspberry Pi is contained on a single circuit board and features ports for:

- HDMI
- USB 2.0
- Composite video
- Analog audio
- Internet
- SD Card

The computer runs entirely on open-source software and gives students the ability to mix and match software according to the work they wish to do.

The Raspberry Pi debuted in February 2012. The group behind the computer's development - the Raspberry Pi Foundation - started the project to make computing fun for students, while also creating interest in how computers work at a basic level. Unlike using an encased computer from a manufacturer, the Raspberry Pi shows the essential guts behind the plastic. Even the software, by virtue of being open- source, offers an opportunity for students to explore the underlying code-if they wish.

The Raspberry Pi is believed to be an ideal learning tool, in that it is cheap to make, easy to replace and needs only a keyboard and a TV to run. These same strengths also make it an ideal product to jumpstart computing in the developing world. The quad-core Raspberry Pi 3 is both faster and more capable than its predecessor, the Raspberry Pi 2. For those interested in benchmarks, the Pi 3's CPU--the board's main processor--has roughly 50-60 percent better performance in 32-bit mode than that of the Pi 2, and is 10x faster than the original single-core Raspberry Pi.

Fig 3.8 Processing Module

Compared to the original PI, real-world applications will see performance increase of between 2.5x for single threaded applications and more than 20x when video playback is accelerated by the chip's NEON engine. Unlike its predecessor, the new board is capable of playing 1080p MP4 video at 60 frames per second (with a bit rate of about 5400Kbps), boosting the Pi's media centre credentials. That's not to say, however, that all video will playback this smoothly, with performance dependent on the source video, the player used and bitrate. The Pi 3 also supports wireless internet out of the box, with built- in Wi-Fi and Bluetooth. The latest board can also boot directly from a USB-attached hard drive or pen drive, as well as supporting booting from a network-attached file system, using PXE, which is useful for remotely updating a Pi and for sharing an operating system image between multiple machines.

3.9 STREAMING MODULE

An USB camera is a video camera that feeds or streams its image in real time to or through a computer to a computer network. When "captured" by the computer, the video stream may be saved, viewed or sent on to other networks travelling through systems such as the internet, and e-mailed as an attachment. When sent 56 to a remote location, the video stream may be saved, viewed or on sent there. Unlike an IP camera (which connects using Ethernet or Wi-Fi), a webcam is generally connected by a USB cable, or similar cable, or built into computer hardware, such as laptops. The term “USB” camera (a clipped compound) may also be used in its original sense of a video camera connected to the USB continuously for an indefinite time, rather than for a particular session, generally supplying a view for anyone who visits its web page over the Internet. Some of them, for example, those used as online cameras, are expensive, rugged professional video cameras.

Fig 3.9 USB Camera

3.10 PERFORMANCE MEASURE

Performance measures are used to evaluate the network performance of the proposed model. This work uses accuracy, precision, recall and f1-score as performance measure, which are formulated.

3.10.1 PRECISION

Precision, used in document retrievals, may be defined as the number of correct documents returned by our ML model. We can easily calculate it by confusion matrix with the help of following formula

$$Precision = \frac{TP}{TP + FP}$$

3.10.2 RECALL

Recall may be defined as the number of positives returned by our ML model. We can easily calculate it by confusion matrix with the help of following formula.

$$Recall = \frac{TP}{TP + FN}$$

3.10.3 SUPPORT

Support may be defined as the number of samples of the true response that lies in each class of target values.

3.10.4 F1 SCORE

This score will give us the harmonic mean of precision and recall. Mathematically, F1 score is the weighted average of the precision and recall. The best value of F1 would be 1 and worst would be 0. We can calculate F1 score with the help of following formula

F1 = 2 * (precision * recall) / (precision + recall)

F1 score is having equal relative contribution of precision and recall.

We can use classification_report function of sklearn.metrics to get the classification report of our classification model.

CHAPTER – 4

RESULT

4.1 RESULTS

Our proposed methodology's execution has been examined on the test data which was distinct from the training data set. The testing process involves 43,200 image samples of different hand signals. All these images were simultaneously updated into our proposed model to come up with accurate results. Our expected result is to obtain a text which is the translation of the sign language given as an input. Our model will anticipate all the hand gestures of the Indian Sign Language. To achieve an efficient result. The estimated accuracy of our proposed system is more than 95% even in a multiplex lighting environment which is considered an adequate result as of now for real-time interpretation.

CHAPTER – 5

CONCLUSIONS

5.1 CONCLUSIONS

The proposed a prototype model can recognize and classify the Indian Sign Language using the deep structured learning techniques called CNN. We observe that the CNN model gives the highest accuracy due to the advanced techniques. From the process, we can conclude that CNN is an efficient technique to categorize hand gestures with high degree of accuracy. In the future work, we would like to expand the dialects for a few more sign languages and our response time can also be improved.

APPENDIES

Appendix A: Source Code

data_collect.py

```
import os
import cv2
DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
number_of_classes = 28
dataset_size = 100
# Try using camera index 0, 1, or 2
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("❌ Error: Could not access the webcam. Try changing the camera index.")
    exit()
for j in range(number_of_classes):
    class_dir = os.path.join(DATA_DIR, str(j))
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)
    print(f"\n📸 Collecting data for class {j}")
    print("👉 Press 'Q' to start capturing images...")
    while True:
        ret, frame = cap.read()
        if not ret:
            print("❌ Failed to grab frame. Is your camera free?")
            break
        cv2.putText(frame, 'Ready? Press "Q" ! :)', (100, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE_AA)
        cv2.imshow('frame', frame)
```

```
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    print("✅ Starting capture...")
    counter = 0
    while counter < dataset_size:
        ret, frame = cap.read()
        if not ret:
            print("❌ Failed to grab frame.")
            break
        resized_frame = cv2.resize(frame, (224, 224))  # ✅ Resize to 224x224
        cv2.imshow('frame', resized_frame)
        cv2.imwrite(os.path.join(class_dir, f'{counter}.jpg'), resized_frame)
        counter += 1
        if cv2.waitKey(25) & 0xFF == ord('q'):
            print("🟦 Capture interrupted.")
            break
    print(f"✅ Collected {counter} images for class {j}")
cap.release()
cv2.destroyAllWindows()
```

train.py

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
import os
# Paths
DATASET_PATH = "./data"
```

```
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 30
# Data Generators
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)
train_gen = datagen.flow_from_directory(
    DATASET_PATH,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)
val_gen = datagen.flow_from_directory(
    DATASET_PATH,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)
print("DataLoaded")
# Load pretrained MobileNetV2
base_model = MobileNetV2(include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3), weights='imagenet')
base_model.trainable = False
print("Model Loaded")
```

```
# Add custom head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
preds = Dense(train_gen.num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=preds)
# Compile
print("Compiling..")
model.compile(optimizer=Adam(learning_rate=1e-4),  loss='categorical_crossentropy', metrics=['accuracy'])
# Train
print("Training..")
model.fit(train_gen, validation_data=val_gen, epochs=EPOCHS)
# Save
print("Saving....")
model.save("asl_model_mobilenetv2.h5")
print("Saved..")
```

main.py

```
import cv2
import numpy as np
import mediapipe as mp
from tensorflow.keras.models import load_model
import pyttsx3
import time
import os
model = load_model('./asl_model_mobilenetv2.h5')
class_names=[
   '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', 'nothing','space'
          ]
```

```
engine = pyttsx3.init()
last_spoken = ""
identified=[]
last_prediction_time = 0
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False,
                       max_num_hands=1,
                       min_detection_confidence=0.7,
                       min_tracking_confidence=0.6)
mp_draw = mp.solutions.drawing_utils
cap = cv2.VideoCapture(0)
prediction_interval = 10
frame_count = 0
while True:
    ret, frame = cap.read()
    frame = cv2.flip(frame, 1)
    if not ret:
        break
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(frame_rgb)
    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            # Get bounding box coordinates
            h, w, _ = frame.shape
            landmark_array = np.array([[lm.x * w, lm.y * h] for lm in
hand_landmarks.landmark])
            x_min, y_min = np.min(landmark_array, axis=0).astype(int)
            x_max, y_max = np.max(landmark_array, axis=0).astype(int)
            padding = 30
            x_min = max(x_min - padding, 0)
            y_min = max(y_min - padding, 0)
```

```
            x_max = min(x_max + padding, w)
            y_max = min(y_max + padding, h)
            roi = frame[y_min:y_max, x_min:x_max]
            # Only predict every N frames
            if frame_count % prediction_interval == 0 and roi.size > 0:
                img = cv2.resize(roi, (224, 224))
                img = img.astype("float32") / 255.0
                img = np.expand_dims(img, axis=0)
                pred = model.predict(img, verbose=0)
                class_idx = np.argmax(pred)
                predicted_class = class_names[class_idx]
                # Speak only if new prediction and enough time has passed
                if predicted_class != last_spoken and time.time() - last_prediction_time > 1.5:
                    identified.append(predicted_class)
                    engine.say(f"The sign is {predicted_class}")
                    engine.runAndWait()
                    last_spoken = predicted_class
                    last_prediction_time = time.time()
                    print(identified)
            mp_draw.draw_landmarks(frame,hand_landmarks,
mp_hands.HAND_CONNECTIONS)
            cv2.rectangle(frame, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
        cv2.putText(frame, f"Prediction: {last_spoken}", (10, 40),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        cv2.imshow("ASL with Mediapipe", frame)
        frame_count += 1
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
cap.release()
cv2.destroyAllWindows()
```

voiceto sign.py

```
import os
import cv2
import time
import speech_recognition as sr
import subprocess
#  Speech Recognition Function
def speech_to_text(filename=None):
    recognizer = sr.Recognizer()
    try:
        if filename:
            with sr.AudioFile(filename) as source:
                audio = recognizer.record(source)
        else:
            with sr.Microphone() as source:
                print("🎤 Speak something...")
                audio = recognizer.listen(source)
        text = recognizer.recognize_google(audio)
        print(f"✅ Recognized Text: {text}")
        return text
    except sr.UnknownValueError:
        print("❌ Could not understand the audio.")
    except sr.RequestError:
        print("❌ Could not request results from speech recognition service.")
    except Exception as e:
        print(f"❌ Unexpected error: {e}")
    return ""
#  Cartoon effect for a frame
def cartoonize_frame(frame):
    color = cv2.bilateralFilter(frame, 9, 75, 75)
```

```
    for _ in range(2):
        color = cv2.bilateralFilter(color, 9, 75, 75)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    edges = cv2.adaptiveThreshold(gray, 255,
                                  cv2.ADAPTIVE_THRESH_MEAN_C,
                                  cv2.THRESH_BINARY, 9, 9)
    cartoon = cv2.bitwise_and(color, color, mask=edges)
    return cartoon
# Play cartoonized video
def play_cartoon_video(path):
    cap = cv2.VideoCapture(path)
    if not cap.isOpened():
        print(f"❌ Error opening video: {path}")
        return
    print(f"🎬 Playing: {os.path.basename(path)}")
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        cartoon_frame = cartoonize_frame(frame)
        cv2.imshow("Cartoon Sign Language", cartoon_frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
# Text to Cartoon Sign Language Video
def text_to_cartoon_sign(text,video_dir=r'D:\Final year project\project\speechto sign\project\isl'):
    print(f"📝 Processing Text: {text}")
    custom_phrases = {
        "what is your name": "What is your Name",
```

```
        "good afternoon": "Good afternoon",
        "good morning": "Good morning",
        "thank": "Thank you",
        "take care":"take care"
    }
    text = text.lower()
    for phrase, filename in custom_phrases.items():
        if phrase in text:
            path = os.path.join(video_dir, f"{filename}.mp4")
            if os.path.exists(path):
                play_cartoon_video(path)
            else:
                print(f"[ ⚠️ ] Currently training on: {filename}")
            text = text.replace(phrase, "")
    for word in text.split():
        filename = f"{word.capitalize()}.mp4"
        path = os.path.join(video_dir, filename)
        if os.path.exists(path):
            play_cartoon_video(path)
        else:
            print(f"[ ⚠️ ] Currently training on : {word}")
        time.sleep(0.5)
# Menu Loop
def main_menu():
    while True:
        print("\n================================")
        print(" Sign Language Recognition System")
        print("================================")
        print("1. Sign Language to Voice")
        print("2. Voice to Sign Language")
```

```
        print("3. Exit")
        choice = input("Enter your choice (1-3): ")
        if choice == '1':
            print("🔧 Sign Language to Voice.")
            os.system("python main.py")
            subprocess.run(["python", "main.py"])
        elif choice == '2':
            text = speech_to_text()
            if not text:
                text = input("📝 Or type the sentence manually: ")
            if text:
                text_to_cartoon_sign(text)
        elif choice == '3':
            print("👋 Exiting program.")
            break
        else:
            print("❌ Invalid choice. Please try again.")
# 🚀 Entry Point
if __name__ == "__main__":
    main_menu()
```

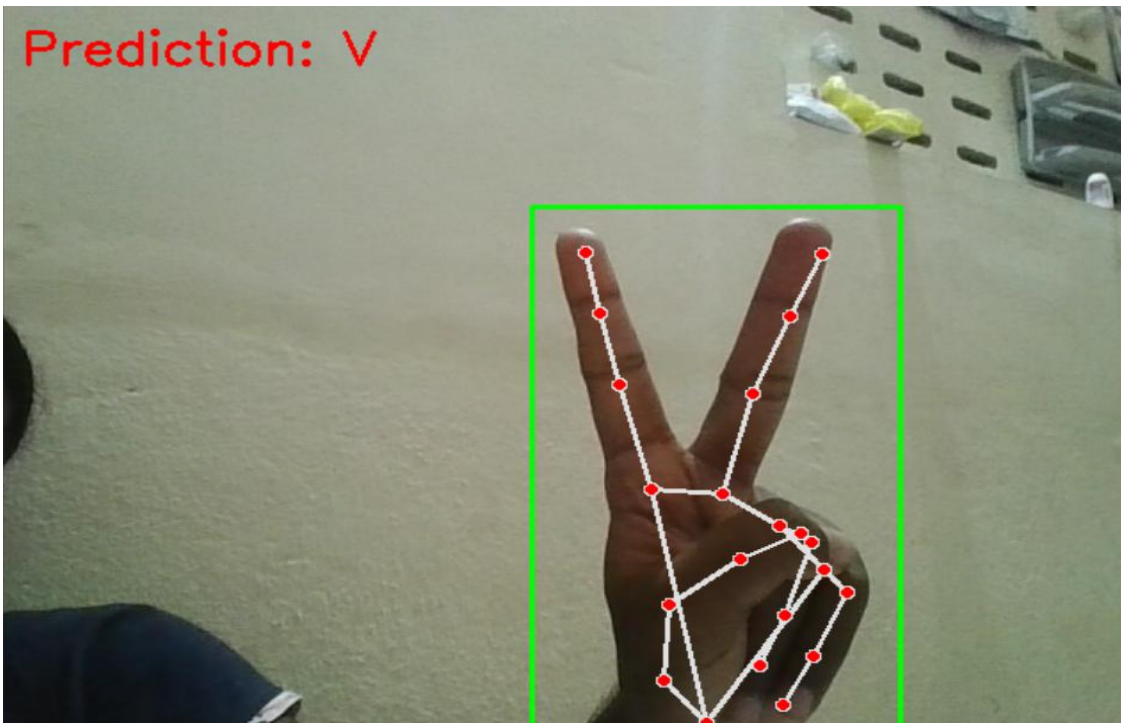
Appendix B: Screenshots

train.py

```
tor.  **kwargs  can include  workers ,  use_multiprocessing ,  max_queue_size . Do not pass these arguments to  fit() , as they will be ign
ored.
  self._warn_if_super_not_called()
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.0254 - loss: 4.3584 - val_accuracy: 0.0179 - val_loss: 3.9320
Epoch 2/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 618ms/step - accuracy: 0.0313 - loss: 4.0408 - val_accuracy: 0.0179 - val_loss: 3.6405
Epoch 3/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 600ms/step - accuracy: 0.0127 - loss: 3.9226 - val_accuracy: 0.0179 - val_loss: 3.4552
Epoch 4/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 643ms/step - accuracy: 0.0510 - loss: 3.6233 - val_accuracy: 0.0357 - val_loss: 3.4083
Epoch 5/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 584ms/step - accuracy: 0.0582 - loss: 3.4764 - val_accuracy: 0.1071 - val_loss: 3.2085
Epoch 6/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 594ms/step - accuracy: 0.0385 - loss: 3.4824 - val_accuracy: 0.0000e+00 - val_loss: 3.2146
Epoch 7/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 5s 667ms/step - accuracy: 0.0613 - loss: 3.3561 - val_accuracy: 0.1071 - val_loss: 2.9093
Epoch 8/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 653ms/step - accuracy: 0.1459 - loss: 3.1553 - val_accuracy: 0.1250 - val_loss: 2.9357
Epoch 9/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 641ms/step - accuracy: 0.0955 - loss: 3.2391 - val_accuracy: 0.1429 - val_loss: 2.8519
Epoch 10/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 621ms/step - accuracy: 0.1128 - loss: 3.0337 - val_accuracy: 0.2857 - val_loss: 2.6500
Saving....
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consider
ed legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_m
odel.keras')`.
Saved..
```

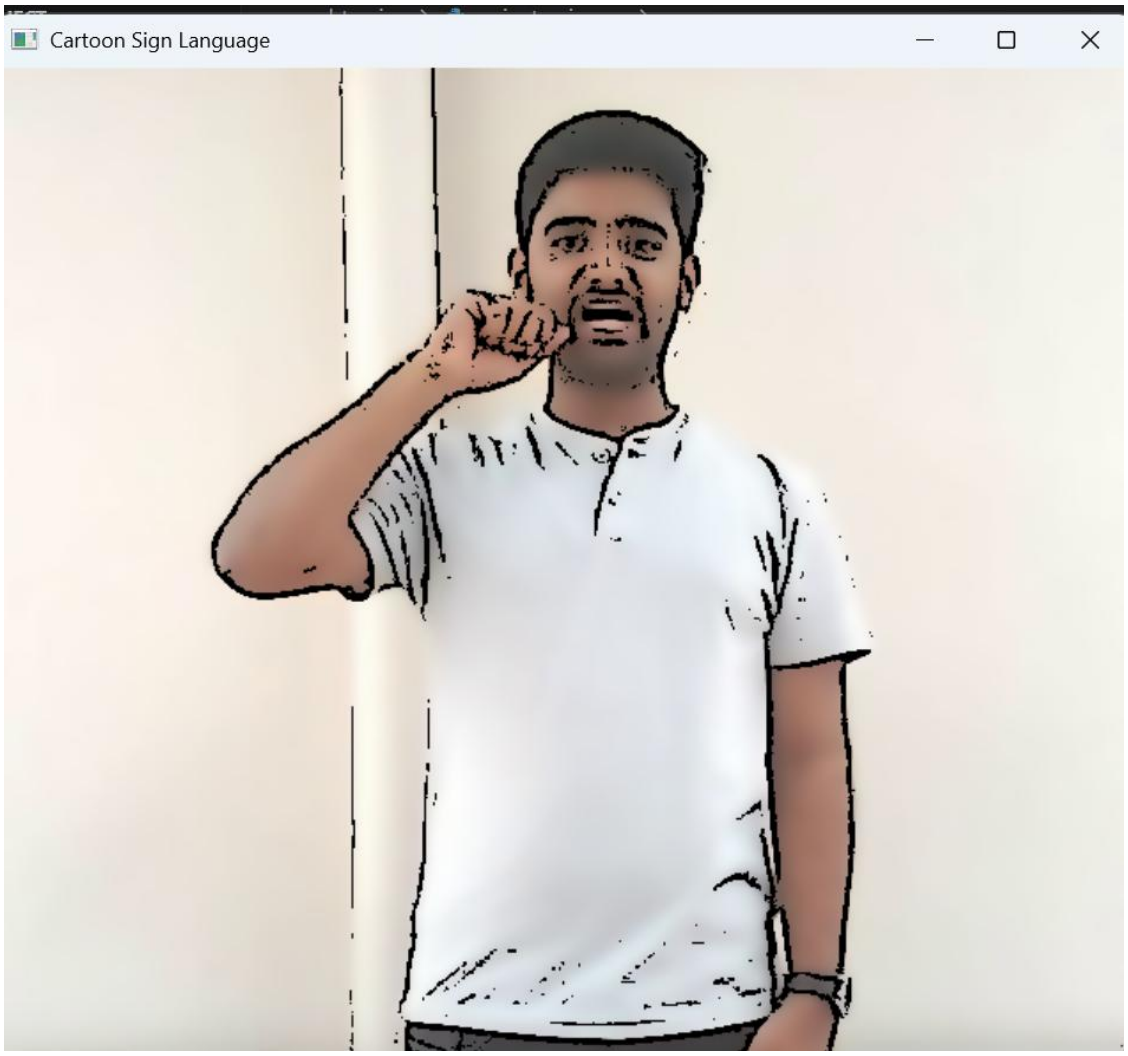
main.py

```
e. Disabling support for feedback tensors.
W0000 00:00:1747940038.360845    6028 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signature inferenc
e. Disabling support for feedback tensors.
W0000 00:00:1747940041.400333    9072 landmark_projection_calculator.cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only supported for
 the square ROI. Provide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.
['N']
['N', 'G']
['N', 'G', 'F']
['N', 'G', 'F', 'J']
['N', 'G', 'F', 'J', 'E']
['N', 'G', 'F', 'J', 'E', 'F']
['N', 'G', 'F', 'J', 'E', 'F', 'S']
['N', 'G', 'F', 'J', 'E', 'F', 'S', 'Y']
['N', 'G', 'F', 'J', 'E', 'F', 'S', 'Y', 'I']
```



voiceto sign.py

```
============================
 Sign Language Recognition System
============================
1. Sign Language to Voice
2. Voice to Sign Language
3. Exit
Enter your choice (1-3): 2
🎤 Speak something...
✅ Recognized Text: carrot
📝 Processing Text: carrot
🎬 Playing: Carrot.mp4
```



REFERENCES

[1] Vijayalakshmi, P., & Aarthi, M. (2016, April). Sign language to speech conversion. In 2016 International Conference on Recent Trends in Information Technology (ICRTIT) (pp. 1-6). IEEE.

[2] NB, M. K. (2018). Conversion of sign language into text. International Journal of Applied Engineering Research, 13(9), 7154-7161.

[3] Masood, S., Srivastava, A., Thuwal, H. C., & Ahmad, M. (2018). Real-time sign language gesture (word) recognition from video sequences using CNN and RNN. In Intelligent Engineering Informatics (pp. 623-632). Springer, Singapore.

[4] Apoorv, S., Bhowmick, S. K., & Prabha, R. S. (2020, June). Indian sign language interpreter using image processing and machine learning. In IOP Conference Series: Materials Science and Engineering (Vol. 872, No. 1, p. 012026). IOP Publishing.

[5] Kaushik, N., Rahul, V., & Kumar, K. S. (2020). A Survey of Approaches for Sign Language Recognition System. International Journal of Psychosocial Rehabilitation, 24(01).

[6] Kishore, P. V. V., & Kumar, P. R. (2012). A video-based Indian sign language recognition system (INSLR) using wavelet transform and fuzzy logic. International Journal of Engineering and Technology, 4(5), 537.

[7] Dixit, K., & Jalal, A. S. (2013, February). Automatic Indian sign language recognition system. In 2013 3rd IEEE International Advance Computing Conference (IACC) (pp. 883-887). IEEE.

[8] Das, A., Yadav, L., Singhal, M., Sachan, R., Goyal, H., Taparia, K., ... & Trivedi, G. (2016, December). Smart glove for Sign Language communications. In 2016 International Conference on Accessibility to Digital World (ICADW) (pp. 27-31). IEEE.

[9] Sruthi, R., Rao, B. V., Nagapravallika, P., Harikrishna, G., & Babu, K. N. (2018). Vision-Based Sign Language by Using MATLAB. International Research Journal of Engineering and Technology (IRJET), 5(3).

[10] Kumar, A., & Kumar, R. (2021). A novel approach for ISL alphabet recognition using Extreme Learning Machine. International Journal of Information Technology, 13(1), 349-357.

[11] Maraqa, M., & Abu-Zaiter, R. (2008, August). Recognition of Arabic Sign Language (Arce) using recurrent neural networks. In 2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT) (pp. 478-481). IEEE.

[12] Masood, S., Srivastava, A., Thuwal, H. C., & Ahmad, M. (2018). Real-time sign language gesture (word) recognition from video sequences using CNN and RNN. In Intelligent Engineering Informatics (pp. 623-632). Springer, Singapore.

[13] Camgoz, N. C., Hadfield, S., Koller, O., Ney, H., & Bowden, R. (2018). Neural sign language translation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 7784-7793).

[14] Cheng, K. L., Yang, Z., Chen, Q., & Tai, Y. W. (2020, August). Fully convolutional networks for continuous sign language recognition. In European Conference on Computer Vision (pp. 697-714). Springer, Cham.

[15] Dabre, K., & Dholay, S. (2014, April). The machine learning model for sign language interpretation using webcam images. In 2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA) (pp. 317-321). IEEE.

[16] Taskiran, M., Killioglu, M., & Kahraman, N. (2018, July). A real-time system for recognition of American sign language by using deep learning. In 2018 41st International Conference on Telecommunications and Signal Processing (TSP) (pp. 15). IEEE.

[17] Khan, S. A., Joy, A. D., Asaduzzaman, S. M., & Hossain, M. (2019, April). An efficient sign language translator device using convolutional neural network and customized ROI segmentation. In 2019 2nd International Conference on Communication Engineering and Technology (ICCET) (pp. 152-156). IEEE.

[18] Nair, A. V., & Bindu, V. (2013). A review on Indian sign language recognition. International journal of computer applications, 73(22).

[19] Kumar, D. M., Bavanraj, K., Thavananthan, S., Bastiansz, G. M. A. S., Harshanath, S. M. B., & Alosious, J. (2020, December). EasyTalk: A Translator for Sri Lankan Sign Language using Machine Learning and Artificial Intelligence. In 2020 2nd International Conference on Advancements in Computing (ICAC) (Vol. 1, pp. 506-511). IEEE.

[20] Kumar, A., Madaan, M., Kumar, S., Saha, A., & Yadav, S. (2021, August). Indian Sign Language Gesture Recognition in Real-Time using Convolutional Neural Networks. In 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN) (pp. 562-568). IEEE.

[21] Manikandan, K., Patidar, A., Walia, P., & Roy, A. B. (2018). Hand gesture detection and conversion to speech and text. arXiv preprint arXiv:1811.11997.

[22] Misra, S., Singha, J., & Laskar, R. H. (2018). Vision-based hand gesture recognition of alphabets, numbers, arithmetic operators and ASCII characters to develop a virtual text-entry interface system. Neural Computing and Applications, 29(8), 117-135.

[23] Hoste, L., Dumas, B., & Signer, B. (2012, May). SpeeG: a multimodal speech-and gesture-based text input solution. In Proceedings of the International working conference on advanced visual interfaces (pp. 156-163).

[24] Buxton, W., Fiume, E., Hill, R., Lee, A., & Woo, C. (1983). Continuous hand-gesture driven input. In Graphics Interface (Vol. 83, pp. 191-195).

[25] Kunjumon, J., & Megalingam, R. K. (2019, November). Hand gesture recognition system for translating indian sign language into text and speech. In 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT) (pp. 14-18). IEEE.

[26] Dardas, N. H., & Georganas, N. D. (2011). Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques. IEEE Transactions on Instrumentation and measurement, 60(11), 3592-3607.

[27] Köpüklü, O., Gunduz, A., Kose, N., & Rigoll, G. (2019, May). Real-time hand gesture detection and classification using convolutional neural networks. In 2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019) (pp. 1-8). IEEE.

[28] Francke, H., Ruiz-del-Solar, J., & Verschae, R. (2007, December). Real-time hand gesture detection and recognition using boosted classifiers and active learning. In Pacific-Rim Symposium on Image and Video Technology (pp. 533-547). Springer, Berlin, Heidelberg.

[29] Zhang, Q., Chen, F., & Liu, X. (2008, July). Hand gesture detection and segmentation based on difference background image with complex background. In 2008 International Conference on Embedded Software and Systems (pp. 338-343). IEEE.

[30] Mazhar, O., Navarro, B., Ramdani, S., Passama, R., & Cherubini, A. (2019). A real-time humanrobot interaction framework with robust background invariant hand gesture detection. Robotics and Computer-Integrated Manufacturing, 60, 34-48.

[31] Liu, W., Li, X., Jia, Z., Yan, H., & Ma, X. (2017). A three-dimensional triangular vision-based contouring error detection system and method for machine tools. Precision Engineering, 50, 85-98.

[32] Cohen, C. J., Beach, G., & Foulk, G. (2001, October). A basic hand gesture control system for PC applications. In Proceedings 30th Applied Imagery Pattern Recognition Workshop (AIPR 2001). Analysis and Understanding of Time Varying Imagery (pp. 74 79). IEEE.

[33] Reifinger, S., Wallhoff, F., Ablassmeier, M., Poitschke, T., & Rigoll, G. (2007, July). Static and dynamic hand-gesture recognition for augmented reality applications. In International Conference on Human-Computer Interaction (pp. 728-737). Springer, Berlin, Heidelberg

[34] Kurakin, A., Zhang, Z., & Liu, Z. (2012, August). A real time system for dynamic hand gesture recognition with a depth sensor. In 2012 Proceedings of the 20th European signal processing conference (EUSIPCO) (pp. 1975-1979). IEEE.

[35] Plouffe, G., & Cretu, A. M. (2015). Static and dynamic hand gesture recognition in depth data using dynamic time warping. IEEE transactions on instrumentation and measurement, 65(2), 305- 316.

[36] Ghotkar, A. S., Khatal, R., Khupase, S., Asati, S., & Hadap, M. (2012, January). Hand gesture recognition for indian sign language. In 2012 International Conference on Computer Communication and Informatics (pp. 1-4). IEEE.

[37] Dutta, K. K., & GS, A. K. (2015, December). Double handed Indian Sign Language to speech and text. In 2015 Third International Conference on Image Information Processing (ICIIP) (pp. 374-377). IEEE.

[38] Dixit, K., & Jalal, A. S. (2013, February). Automatic Indian sign language recognition system. In 2013 3rd IEEE International Advance Computing Conference (IACC) (pp. 883-887). IEEE.

[39] Nair, A. V., & Bindu, V. (2013). A review on Indian sign language recognition. International journal of computer applications, 73(22).

[40] Rajam, P. S., & Balakrishnan, G. (2011, September). Real time Indian sign language recognition system to aid deaf-dumb people. In 2011 IEEE 13th international conference on communication technology (pp. 737-742). IEEE.

PLAGIARISM REPORT

CSE111 Proposal.doc

ORIGINALITY REPORT

8%	7%	7%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Mcclintock High School Student Paper	1%
2	export.arxiv.org Internet Source	1%
3	scholarworks.gsu.edu Internet Source	1%
4	patents.justia.com Internet Source	1%
5	www.scholarsresearchlibrary.com Internet Source	1%
6	www.psychosocial.com Internet Source	1%
7	Submitted to University of East London Student Paper	<1%
8	repository.tudelft.nl Internet Source	<1%
9	www.sciencepublishinggroup.com Internet Source	<1%