

APPENDIX 1

SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING

FINAL REPORT

Submitted by

**SAHIL KUMAR – 20BCS9238
SHRUTI SHREYA – 20BCS9229**

in partial fulfillment for the award of the degree of

BACHELOR'S OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
CHANDIGARH UNIVERSITY, GHARUAN**

June 2022

APPENDIX 2



BONAFIDE CERTIFICATE

Certified that this project report “**SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING**” is the Bonafide work of “**SAHIL KUMAR & SHRUTI SHREYA**” who carried out the project work under our supervision.



SIGNATURE

Er. Parwinder Kaur
SUPERVISOR

SIGNATURE

**HEAD OF THE
DEPARTMENT**

COMPUTER SCIENCE & ENGINEERING

Submitted for the project viva-voce examination held on 21st May, 2022, Saturday

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide Er. Parwinder Kaur, Associate Professor, Department of Computer Science and Engineering, CHANDIGARH UNIVERSITY, for their guidance with unsurpassed knowledge and immense encouragement. We are grateful to Dr. Puneet Soni, Coordinator of the Department, of Computer Science and Engineering, for providing us with the required facilities for the completion of the project work. We are grateful to Dr. _____, Head of the Department, of Computer Science and Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the Principal and Management, CHANDIGARH UNIVERSITY, for their encouragement and cooperation to carry out this work. We express our thanks to all teaching faculty of the Department of CSE, whose suggestions during reviews helped us in the accomplishment of our project. We would like to thank all non-teaching staff of the Department of CSE, CHANDIGARH UNIVERSITY for providing great assistance in the accomplishment of our project. We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. Last but not least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

ABSTRACT

There have been several advancements in technology and a lot of research has been done to help the people who are deaf and dumb. Aiding the cause, Deep learning, and computer vision can be used too to make an impact on this cause.

This can be very helpful for the deaf and dumb people in communicating with others as knowing sign language is not something that is common to all, moreover, this can be extended to creating automatic editors, where the person can easily write by just their hand gestures.

In this sign language recognition project, we create a sign detector, which detects numbers from 1 to 10 that can very easily be extended to cover a vast multitude of other signs and hand gestures including the alphabets.

We have developed this project using OpenCV and Keras modules of python.

DECLARATION

We, **SHRUTI SHREYA & SAHIL KUMAR** of 4TH semester B.E., in the Department of Computer Science and Engineering from CHANDIGARH UNIVERSITY, GHARUAN, hereby declare that the project work entitled **SIGN LANGUAGE RECOGNITION SYSTEM USING MACHINE LEARNING** is carried out by us and submitted in partial fulfillment of the requirements for the award of **Bachelor of Engineering in Computer Science Engineering**, under **CHANDIGARH UNIVERSITY** during the academic year 2020-2024 and has not been submitted to any other university for the award of any kind of degree.

CONTENTS

| TITLE | Page no. |
|---|-----------------|
| LIST OF FIGURES | vi |
| LIST OF TABLES | vii |
| ABSTRACT | viii |
| 1.INTRODUCTION | 1 |
| 1.1 IMAGE PROCESSING | 1 |
| 1.2 SIGN LANGUAGE | 3 |
| 1.3 SIGN LANGUAGE AND HAND GESTURE RECOGNITION | 3 |
| 1.4 MOTIVATION | 4 |
| 1.5 PROBLEM STATEMENT | 5 |
| 1.6 ORGANISATION OF THESIS | 5 |
| 2.LITERATURE SURVEY | 6 |
| 2.1 INTRODUCTION | 6 |
| 2.1.1 TENSOR FLOW | 6 |
| 2.1.2 OPENCV | 6 |
| 2.1.3 KERAS | 10 |
| 2.1.4 NUMPY | 11 |
| 2.1.5 NEURAL NETWORKS | 13 |
| 2.2 EXISTING MODELS | 20 |
| 2.3 PROPOSED SYSTEM | 24 |
| 2.3.1 ARCHITECTURE | 25 |
| 3.METHODOLOGY | 26 |
| 3.1 TRAINING MODULE | 26 |

| | |
|---|-----------|
| 3.1.1 PRE-PROCESSING | 27 |
| 3.2 ALGORITHM | 29 |
| 3.3 SEGMENTATION | 30 |
| 3.4 CONVOLUTION NEURAL NETWORKS | 31 |
| 3.5 TESTING MODULE | 34 |
| 4.DESIGN | 38 |
| 4.1 DATAFLOW DIAGRAM | 38 |
| 4.2 UML | 40 |
| 4.2.1 USE CASE DIAGRAM | 42 |
| 4.2.2 CLASS DIAGRAM | 46 |
| 4.2.3 SEQUENCE DIAGRAM | 45 |
| 4.2.4 STATECHART DIAGRAM | 50 |
| 5. EXPERIMENTAL ANALYSIS AND RESULTS | 51 |
| 5.1 SYSTEM CONFIGURATION | 51 |
| 5.1.1 SOFTWARE REQUIREMENTS | 51 |
| 5.1.2 HARDWARE REQUIREMENTS | 51 |
| 5.2 CODE | 51 |
| 5.3 SCREENSHOTS AND RESULTS | 68 |
| 6. CONCLUSION AND FUTURE SCOPE | 70 |
| REFERENCES | 71 |

LIST OF FIGURES

| Figure no. | Name of the Figure | Page no. |
|-------------------|--|-----------------|
| 1.1 | Phases of pattern recognition | 2 |
| 2.1 | Layers involved in CNN | 19 |
| 2.2 | Architecture of Sign Language recognition System | 25 |
| 3.1 | Dataset used for training the model | 28 |
| 3.2 | Sample pictures of training data | 28 |
| 3.3 | Training data given for Letter A | 29 |
| 4.1 | Data flow Diagram | 39 |
| 4.2. | Use Case Diagram | 43 |
| 4.3 | Class Diagram | 46 |
| 4.4 | Sequence Diagram | 49 |
| 4.5 | State Chart Diagram | 56 |
| 5.1 | Screenshot of the result obtained for letter A | 68 |
| 5.2 | Screenshot of the result obtained for letter W | 68 |
| 5.3 | Screenshot of result obtained for letter L | 69 |
| 5.4 | Screenshot of result obtained for letter B | 69 |

LIST OF TABLES

| Table no. | Name of the Table | Page no. |
|------------------|---|-----------------|
| 3.1 | Verification of testcases | 37 |
| 4.1 | Usecase Scenario for Sign language recognition system | 44 |

ABSTRACT

Sign language is the only tool of communication for the person who is not able to speak and hear anything. Sign language is a boon for the physically challenged people to express their thoughts and emotion. In this work, a novel scheme of sign language recognition has been proposed for identifying the alphabets and gestures in sign language. With the help of computer vision and neural networks we can detect the signs and give the respective text output.

KeyWord: Sign LanguageRecognition1, Convolution Neural Network2, Image Processing3, Edge Detection4, Hand Gesture Recogniton5.

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 IMAGEPROCESSING

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 digital technique 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.

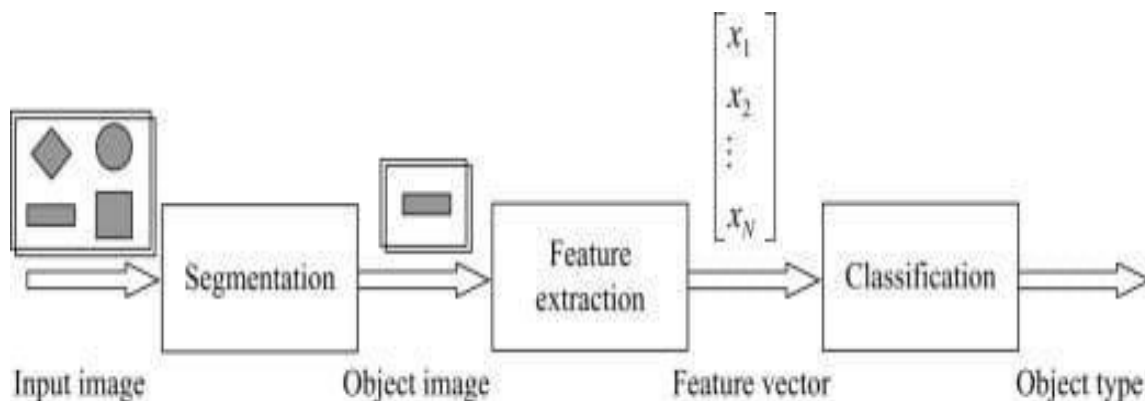


Fig1.1: Phases of pattern recognition

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 is 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 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 interfaces (HCI). Making a computer understand speech, facial expressions and human gestures are some steps towards it. Gestures are the non-verbally exchanged information. A person can perform innumerable gestures at a time. Since humangestures are perceived through vision, it is a subject of great interest forcomputer 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 “hearingimpairment”. 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 PROBLEMSTATEMENT

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.

1.6 ORGANISATION OF THESIS

The book is organised as follows:

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

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

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

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.

2. LITERATURE SURVEY

2.1 INTRODUCTION:

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

2.1.1 TensorFlow:

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

Features: TensorFlow provides stable Python (for version 3.7 across all platforms) and C APIs; and without API backwards compatibility guarantee: C++, Go, Java, JavaScript and Swift (early release). Third-party packages are available for C#, Haskell Julia, MATLAB, R, Scala, Rust, OCaml, and Crystal. "New language support should be built on top of the C API. However, not all functionality is available in C yet." Some more functionality is provided by the Python API.

Application: Among the applications for which TensorFlow is the foundation, are automated image-captioning software, such as DeepDream.

2.1.2 Opencv:

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

OpenCV's application areas include:

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

Stereopsis stereo vision: depth perception from 2 cameras

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

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

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

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

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

VXL, an alternative library written in C++.

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

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

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

of free and open source software packages

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

Image-Processing:

Image processing is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it.

If we talk about the basic definition of image processing then “Image processing is the analysis and manipulation of a digitized image, especially in order to improve its quality”.

Digital-Image :

An image may be defined as a two-dimensional function $f(x, y)$, where x and y are spatial(plane) coordinates, and the amplitude of f at 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 tumordetection)
- 2D/3D segmentation
- 3D human organ reconstruction (MRI or ultrasound)
- Vision-guided robotics surgery
- Industrial Automation Application
- Industrial inspection (defect detection)
- Assembly
- Barcode and package label reading
- Object sorting
- Document understanding (e.g. OCR)
- Security Application
- Biometrics (iris, finger print, face recognition)
- Surveillance – Detecting certain suspicious activities or behaviors
- Transportation Application
- Autonomous vehicle
- Safety, e.g., driver vigilance monitoring

2.1.3Keras:

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

Features: Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, 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 pre-trained models are listed below,

- ResNet
- VGG16
- MobileNet
- InceptionResNetV2
- InceptionV3

2.1.4Numpy:

NumPy (pronounced /'nʌmpaɪ/ (NUM-py) or sometimes /'nʌmpi/ (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 a 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 a universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Limitations: Inserting or appending entries to an array is not as trivially possible as 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 as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.

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

2.1.5 Neural Networks:

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

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

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

In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map. Hidden layers fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.

Areas of Application

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 preprocessed. 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 preprocessed 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 unlabeled, unstructured data, which is the vast majority of data in the world. Another word for unstructured data is raw media; i.e. pictures, texts, video and audio recordings. Therefore, one of the problems deep learning solves best is in processing and clustering the world's raw, unlabeled media, discerning similarities and anomalies in data that no human has organized in a relational database or ever put a name to.

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

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

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

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

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

Applications of convolution neural network:

Decoding Facial Recognition:

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

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

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

A similar process is followed for scene labeling as well. Analyzing

Documents:

Convolutional neural networks can also be used for document analysis. This is not just useful for handwriting analysis, but also has a major stake in recognizers. For a machine to be able to scan an individual's writing, and then compare that to the wide database it has, it must execute almost a million commands a minute. It is said with the use of CNNs and newer models and algorithms, the error rate has been brought down to a minimum of 0.4% at a character level, though its complete testing is yet to be widely seen.

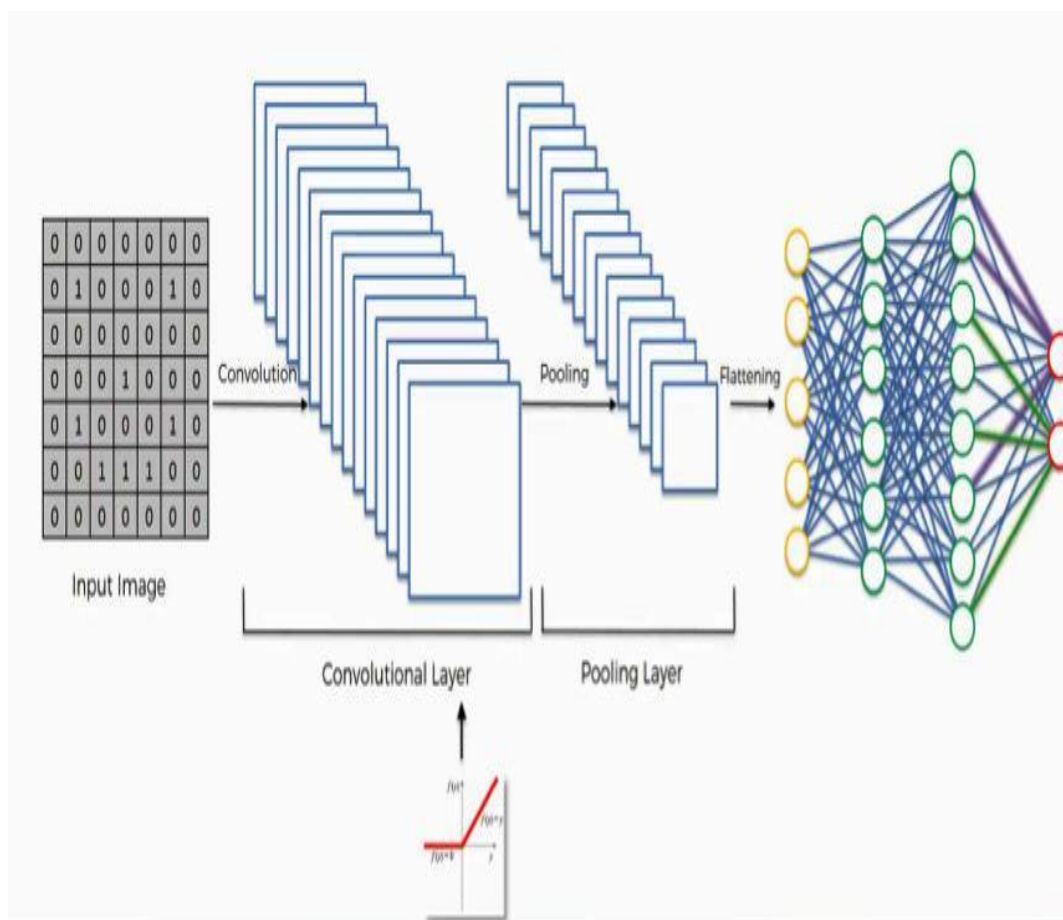


Fig2.1: Layers involved in CNN

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

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 by different people in different languages but there are not such a chat application that has facilitated 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 been 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 close to the disabled people. This is a 2 way communication system but it uses pattern of gesture recognition which is not very reliable in getting appropriate output.

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 people 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.

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

Sign Language Recognition Using Image Processing

One of the major drawback 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.

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 preprocessed using a combinational algorithm and recognition is done using template matching. The translation in the form of text is then converted to audio. The database used for this system includes 6000 images of English alphabets. We used 4800 images for training and 1200 images for testing. The system produces 88% accuracy.

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 Color 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.

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 utilizes outwardly transmitted motions that consolidate 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 speak 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.

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

2.3.1 System Architecture

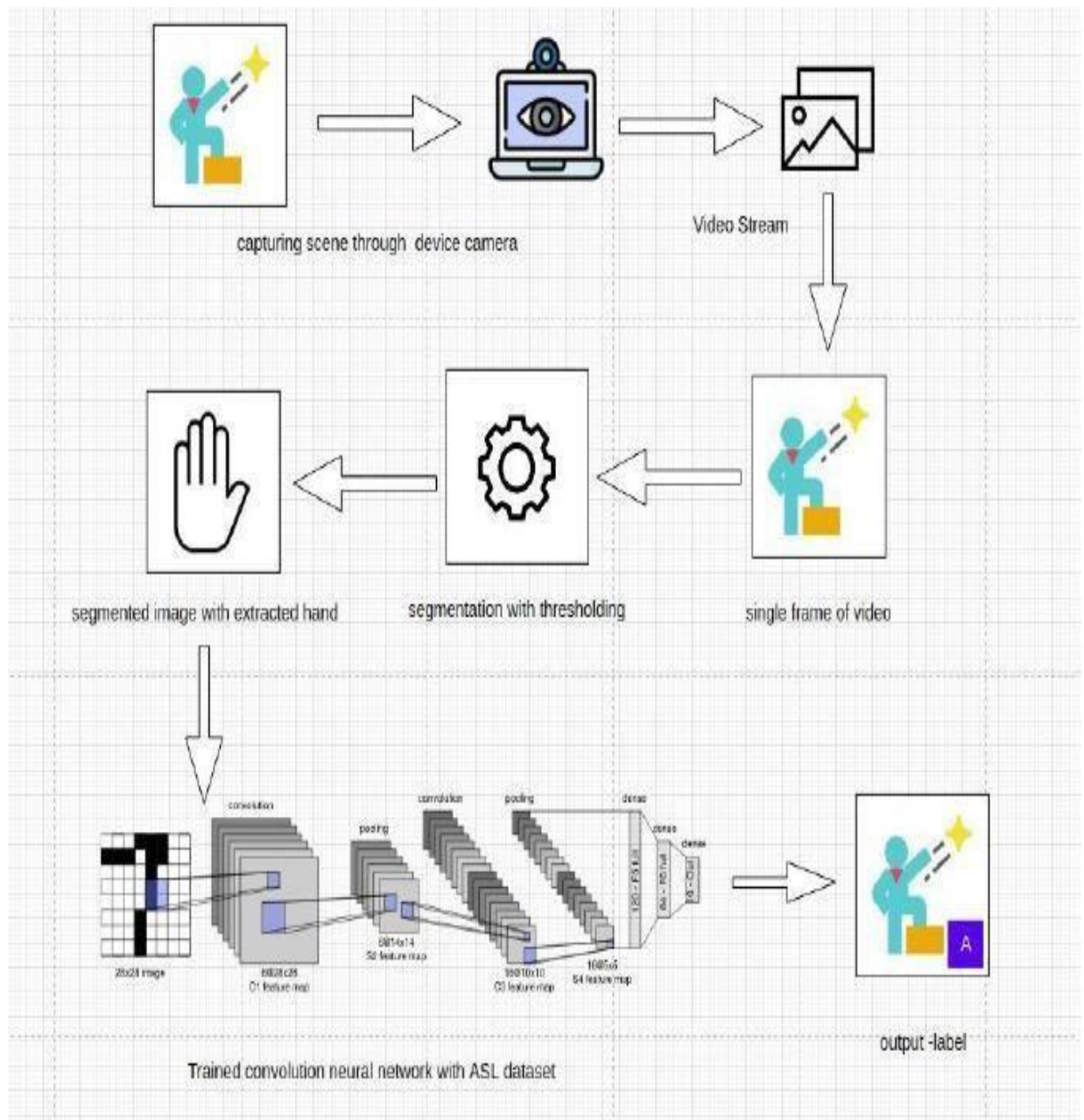


Fig2.2:Architecture of Sign Language recognition System

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 project case, it was neural networks. Such an algorithm looks like:

1. begin with its object: `model = Sequential()`
2. then consist of layers with their types: `model.add(type_of_layer())`
3. after adding a sufficient number of layers the model is compiled. At this moment Keras communicates with TensorFlow for construction of the model. During model compilation it is important to write a loss function and an optimizer algorithm. It looks like:
`model.compile(loss= 'name_of_loss_function', optimizer= 'name_of_optimizer_alg')` The loss function shows the accuracy of each prediction made by the model.

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

Model training:

After model construction it is time for model training. In this phase, the model is trained using training data and expected output for this data. It's look 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.

3.1.1 Preprocessing:

Uniform aspect ratio

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

Cropping to an aspect ratio

Aside from using built-in site style options, you may want to manually crop an image to a certain aspect ratio. For example, if you use product images that have the same aspect ratio, they'll all crop the same way on your site. 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 visible quality loss. From the standpoint of digital signal processing, the scaling of raster graphics is a two-dimensional example of sample-rate conversion, the conversion of a discrete signal from a sampling rate (in this case the local sampling rate) to another.

DATASETS USED FOR TRAINING

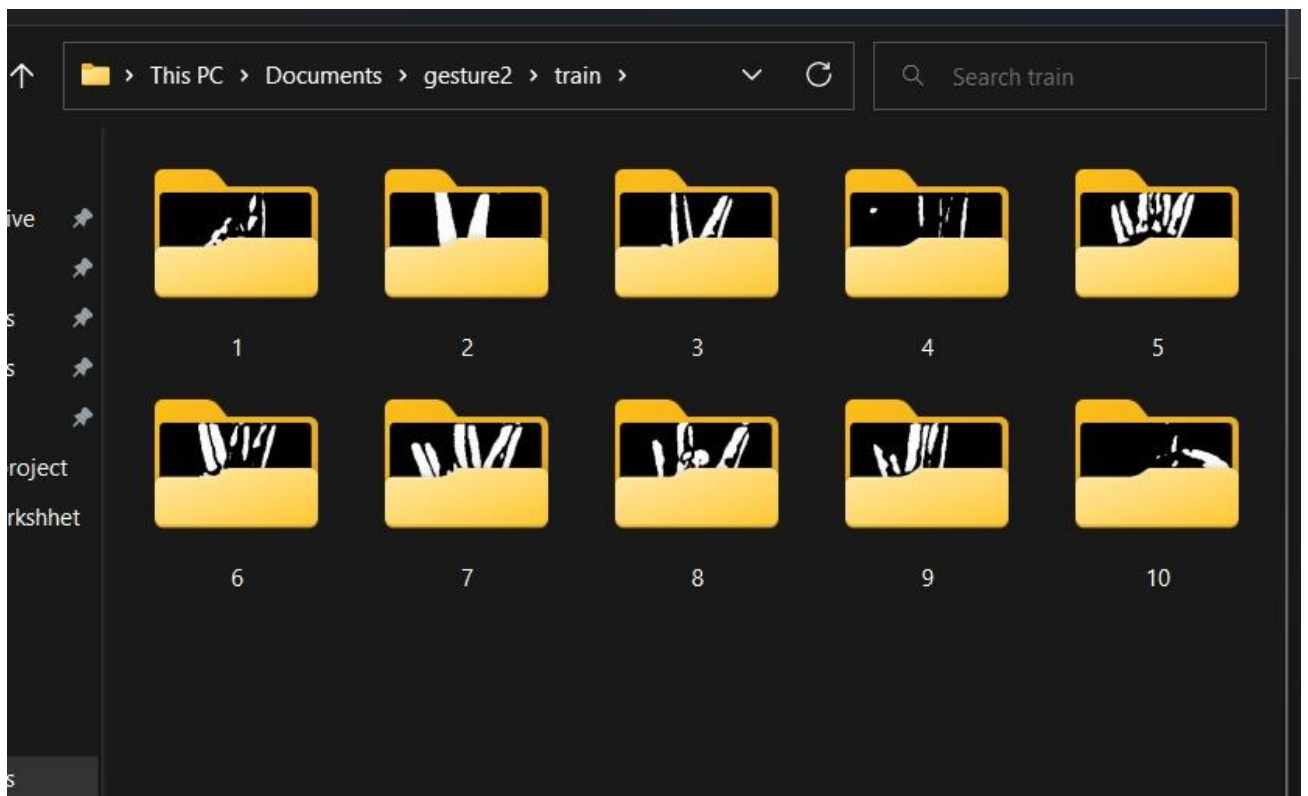


Fig3.1:Dataset used for training the model



Fig3.2:Sample pictures of training data



Fig3.3: Training data given for number 4

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.

Let's see an example. Imagine that a Matrix contains information of an image (i.e. intensity in the range 0–255):

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 into subparts (called **bins**) like:

$[0,255]=[0,15]\cup[16,31]\cup\dots\cup[240,255]$ range=bin1 U bin2 U U bin15
and we can keep count of the number of pixels that fall in the range of each bini

BackPropogation: 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 thereason 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 ? N-th moment of a random variable is defined as the expected value of that variable to the power of n. More formally:

Loss Function(categorical cross entropy): Categorical crossentropy 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 are several deep learning architectures used for segmentation:

Why does Image Segmentation even matter?

If we take an example of Autonomous Vehicles, they need sensory input devices like cameras, radar, and lasers to allow the car to perceive the world around it, creating a digital map. Autonomous driving is not even possible without object detection which itself involves image classification/segmentation.

How Image Segmentation works

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

3.44 CLASSIFICATION: CONVOLUTION NEURAL NETWORK

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

- 1) One hot encode the data: A one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.
- 2) Define the model: A model said in a very simplified form is nothing but a function that is used to take in certain input, perform certain operation to its 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 labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization.

5) Test the model

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

Convolution Operation:

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

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 stride and since we are moving the feature detector one cell at a 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 on an image when we put it through the convolution operation. What the rectifier function does to an image like this is remove all the black elements from it, keeping only those carrying a positive value (the grey and white colors). The essential difference between the non-rectified version of the image and the rectified one is the progression of colors. After we rectify the image, you will find the colors changing more abruptly. The gradual change is no longer there. That indicates that the linearity has been disposed of.

Pooling Layer:

The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input. This process is what provides the convolutional neural network with the “spatial variance” capability. In addition to that, pooling serves to minimize the size of the images as well as the number of parameters which, in turn, prevents an issue of “overfitting” from coming up. Overfitting in a nutshell is when you create an excessively complex model in order to account for the idiosyncracies 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.

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, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner.

Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well planned thorough 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 as 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 the entire system is tested at the client's site on real world data to find errors.

User Acceptance Test (UAT)

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

Requirements traceability:

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

| id | Test case | Input description | Expected output | Test status |
|----|----------------------------|--|---------------------------------------|-------------|
| 1 | Loadind model | Initializing trained model and load it into ON | Loaded model without errors | pass |
| 2 | Converting video to frames | Capturing video and converting it into frames | Image frames of captured video stream | pass |
| 3 | Recognize hand gesture | Image frame that contains hand object | label | Pass |

Table3.1: verification of testcases

4. DESIGN

4.1 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 dataflow based on business rules.

Representation:



3) Data Flow:

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

Representation:



4) Data Store:

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

Representation:

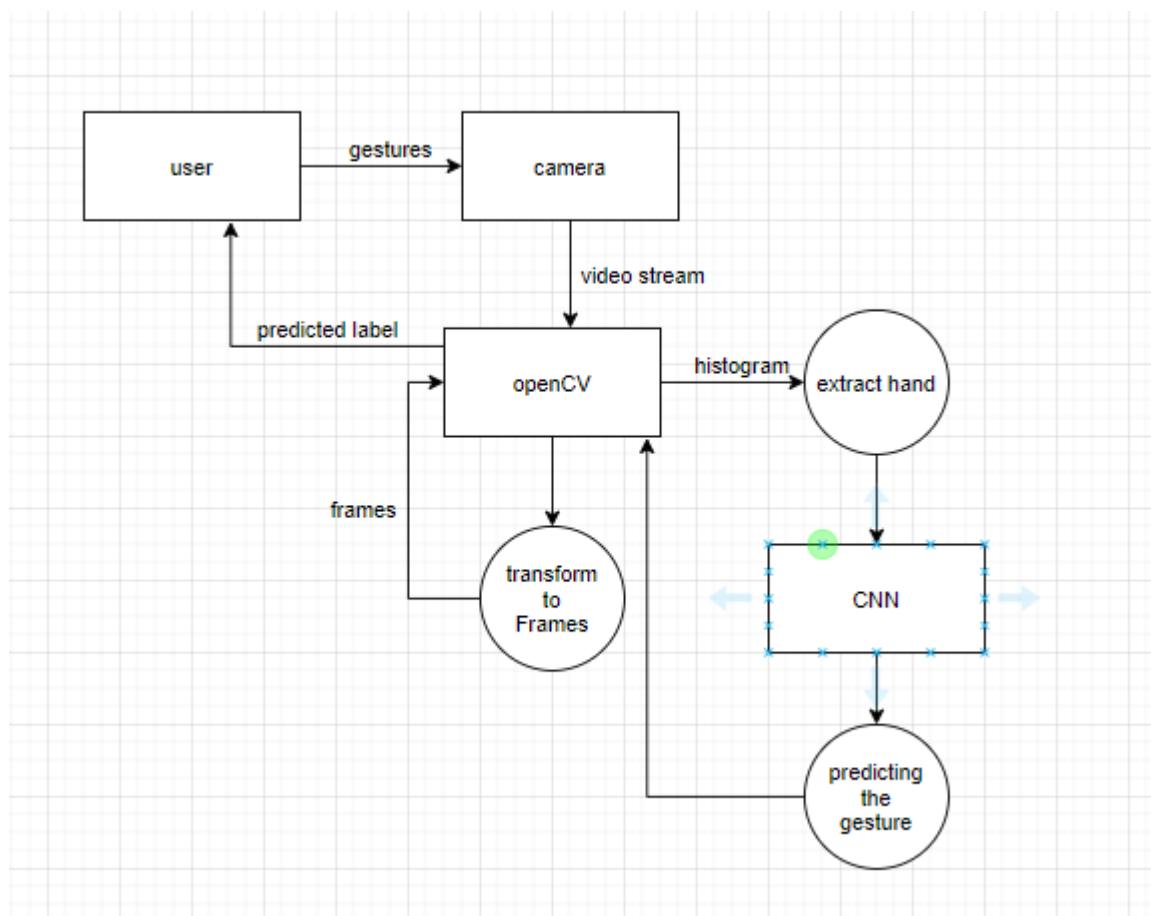
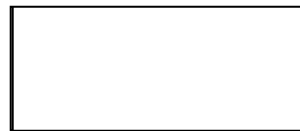


Fig4.1:Dataflow Diagram for Sign Language Recognition

4.2 UML DIAGRAMS

UML stands for Unified Modeling 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 modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of the system. A modeling language such as the UML is thus a standard language for software blueprints.

The UML is a graphical language, which consists of all interesting systems. There are also different structures that can transcend what can be represented in a programming language.

These are different diagrams in UML.

4.2.1 Use Case Diagram

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system. Use case describes the 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 Statechart) 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 analyzed 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 analyzed, the functionalities are captured in use cases.

We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

Functionalities to be represented as use case

Actors

Relationships among the use cases and actors.

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.

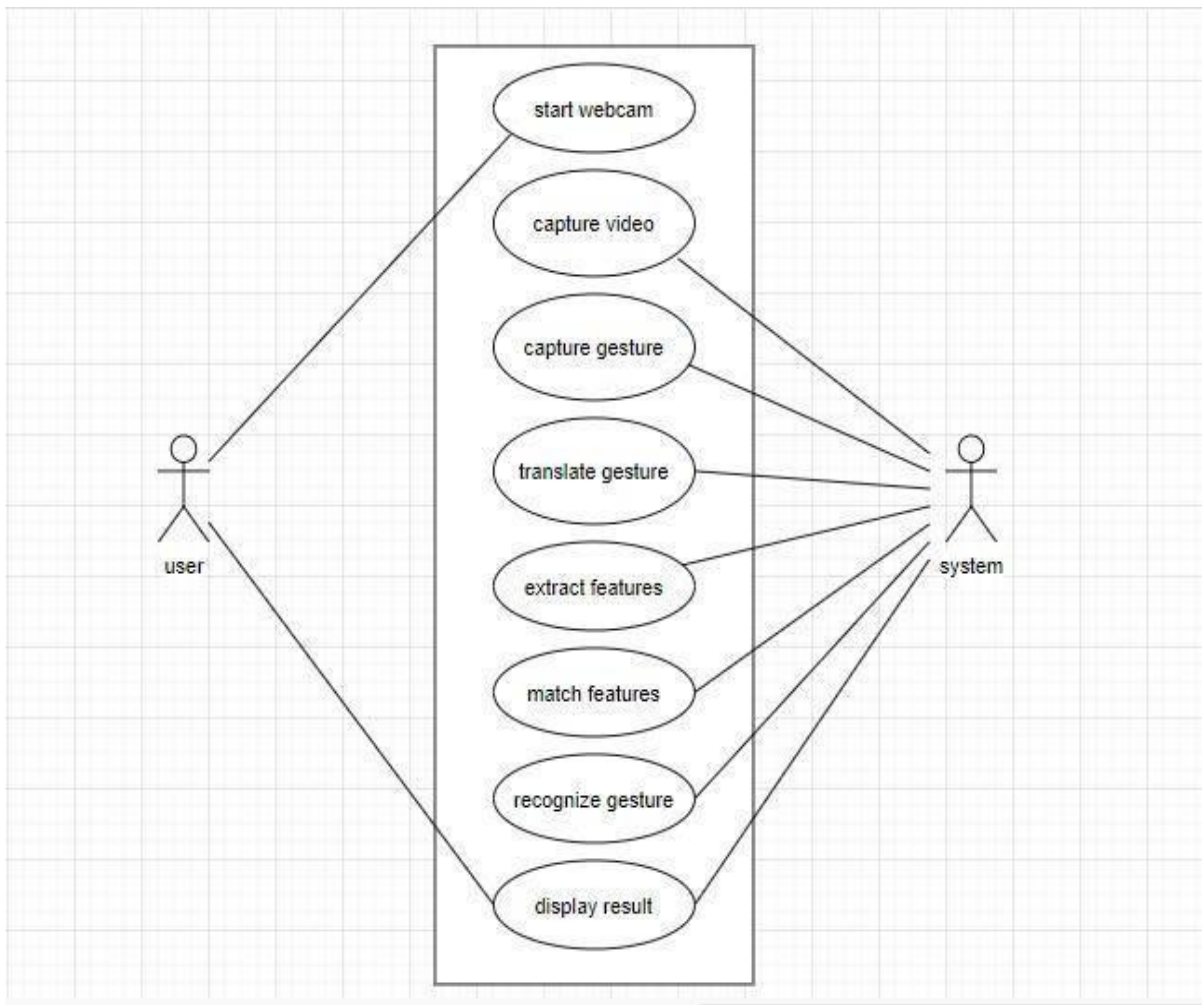


Fig 4.2: Usecase diagram of sign language recognition System

| | |
|----------------------|---|
| Usecase name | Sign language recognition |
| Participating actors | User, System |
| Flow of events | Start the system(u) Capturing video(s) Capture gesture(s) Translate gesture(s) Extract features(s) Match features(s) Recognizing gesture(s) Display result |
| Entry condition | Run the code |
| Exit condition | Displaying the label |
| Quality requirements | Cam pixels clarity , good light condition |

Table 4.1: Usecase Scenario for sign language recognition system

4.2.2 Class Diagram

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram describe the different perspective when designing a system-conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also display 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 represent 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 drawing it should be understandable to the developer/coder.

Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

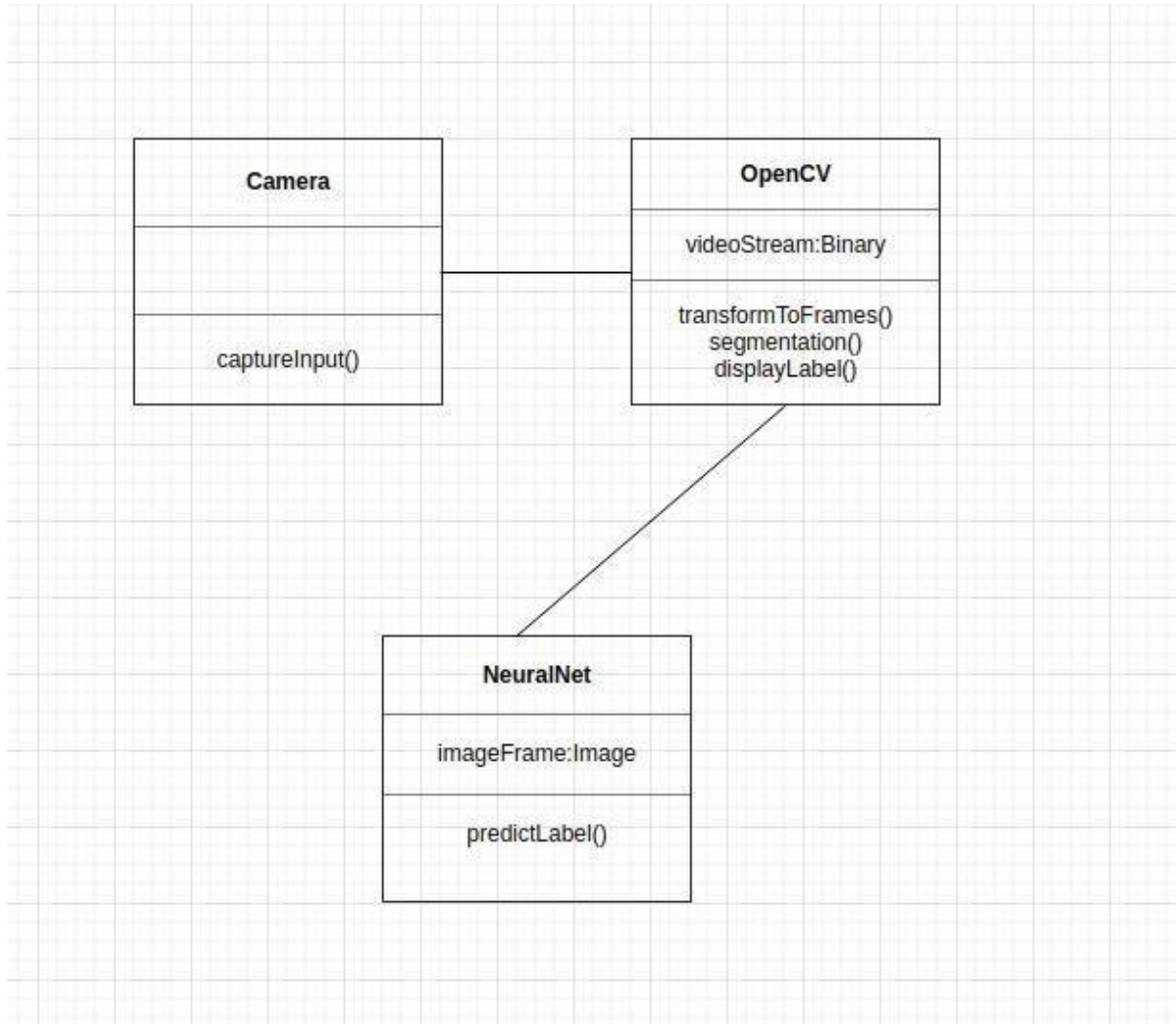


Fig4.3: Class diagram of sign language recognition system

4.2.3 Sequence Diagram

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

Objects: 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 simpler 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

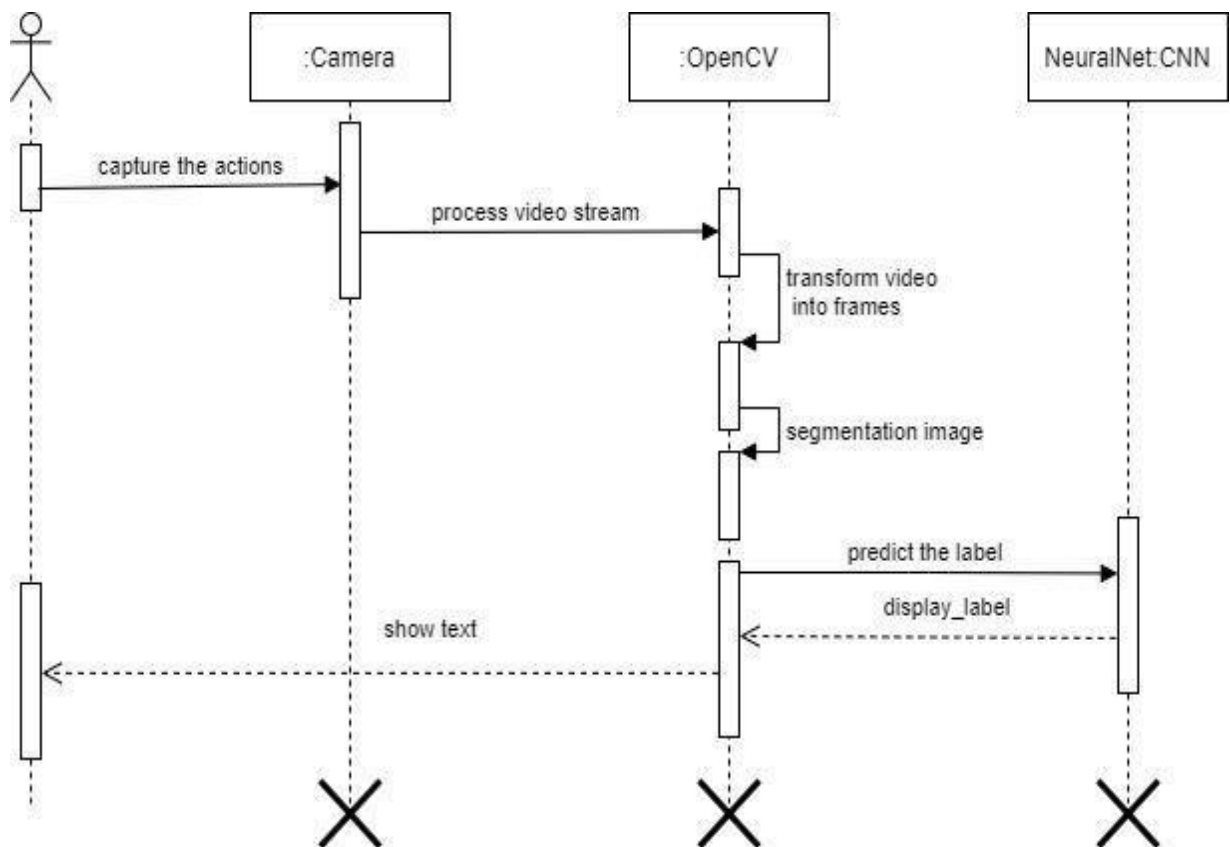


Fig4.4: Sequence diagram of sign language recognition system

4.2.4 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 modelling 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 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.

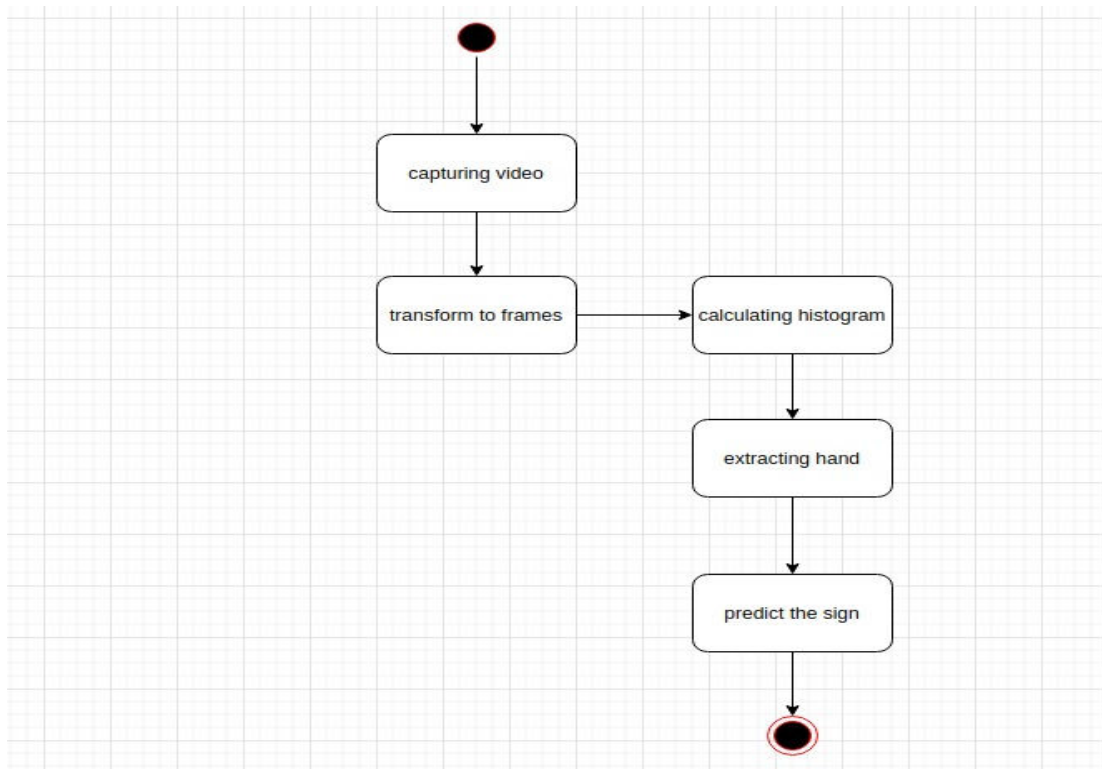


Fig:4.5:State Chart diagram of sign language recognition system

5.EXPERIMENTAL ANALYSIS AND RESULTS

5.1 SYSTEM CONFIGURATION

5.1.1 Software requirements

Operating System: Windows, Mac, Linux **SDK:**
OpenCV, TensorFlow, Keros, Numpy

5.1.2. Hardware Requirements

The Hardware Interfaces Required are:

Camera: Good quality, 3MP

Ram: Minimum 8GB or higher

GPU: 4GB dedicated **Processor:**

Intel Pentium 4 or higher **HDD:** 10GB or
higher

Monitor: 15” or 17” colour monitor

Mouse: Scroll or Optical Mouse or Touch Pad

Keyboard: Standard 110 keys keyboard

5.2 CODE:

Create_gesture_data

```
In [1]: import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Activation, Dense, Flatten, BatchNormalization, Conv2D, MaxPool2D, Dropout
from keras.optimizers import adam_v2
from tensorflow.keras.optimizers import SGD
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator

import warnings
import numpy as np
import cv2
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint, EarlyStopping
warnings.simplefilter(action='ignore', category=FutureWarning)

background = None
accumulated_weight = 0.5

#creating the dimensions for the ROI...
ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350

In [2]: def cal_accum_avg(frame, accumulated_weight):

    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)

In [3]: def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)

    _, thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

    # Grab the external contours for the image
    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        return None
    else:

        hand_segment_max_cont = max(contours, key=cv2.contourArea)

        return (thresholded, hand_segment_max_cont)

In [4]: cam = cv2.VideoCapture(0)

num_frames = 0
element = 10
num_imgs_taken = 0

while True:
    ret, frame = cam.read()

    # flipping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(frame, 1)

    frame_copy = frame.copy()

    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)

    if num_frames < 60:
        cal_accum_avg(gray_frame, accumulated_weight)
        if num_frames <= 59:

            cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT", (80, 400), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,0,255), 2)

    #Time to configure the hand specifically into the ROI...
    elif num_frames <= 300:

        hand = segment_hand(gray_frame)

        cv2.putText(frame_copy, "Adjust hand...Gesture for" + str(element), (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
```

```

# Checking if the hand is actually detected by counting the number of contours detected...
if hand is not None:

    thresholded, hand_segment = hand

    # Draw contours around hand segment
    cv2.drawContours(frame_copy, [hand_segment + (ROI_right,
ROI_top)], -1, (255, 0, 0), 1)

    cv2.putText(frame_copy, str(num_frames)+"For" + str(element),
(70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

    # Also display the thresholded image
    cv2.imshow("Thresholded Hand Image", thresholded)

else:

    # Segmenting the hand region...
    hand = segment_hand(gray_frame)

    # Checking if we are able to detect the hand...
    if hand is not None:

        # unpack the thresholded img and the max_contour...
        thresholded, hand_segment = hand

        # Drawing contours around hand segment
        cv2.drawContours(frame_copy, [hand_segment + (ROI_right, ROI_top)], -1, (255, 0, 0), 1)

        cv2.putText(frame_copy, str(num_frames), (70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

        cv2.putText(frame_copy, str(num_imgs_taken) + 'images' + "For" + str(element), (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

        # Displaying the thresholded image
        cv2.imshow("Thresholded Hand Image", thresholded)
        if num_imgs_taken <= 300:
            cv2.imwrite(r"C:\\Users\\MYPC\\Documents\\gesture2\\train\\"+str(element)+"\\" + str(num_imgs_taken+300) + '.jpg')

        else:
            break
        num_imgs_taken +=1
    else:
        cv2.putText(frame_copy, 'No hand detected...', (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

# Drawing ROI on frame copy
cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right,ROI_bottom), (255,128,0), 3)

cv2.putText(frame_copy, "DataFlair hand sign recognition _ _ _", (10, 20), cv2.FONT_ITALIC, 0.5, (51,255,51), 1)

# increment the number of frames for tracking
num_frames += 1

# Display the frame with segmented hand
cv2.imshow("Sign Detection", frame_copy)

# Closing windows with Esc key...(any other key with ord can be used too.)
k = cv2.waitKey(1) & 0xFF

if k == 27:
    break

# Releasing the camera & destroying all the windows...

cv2.destroyAllWindows()
cam.release()

```


DataFlair_trainCNN:

```
In [1]: import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation, Dense, Flatten, BatchNormalization, Conv2D, MaxPool2D, Dropout
from keras.optimizers import adam_v2
from tensorflow.keras.optimizers import SGD
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator

import warnings
import numpy as np
import cv2
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint, EarlyStopping
warnings.simplefilter(action='ignore', category=FutureWarning)

In [2]: train_path = r'C:\Users\MVPC\Documents\gesture2\train'
test_path = r'C:\Users\MVPC\Documents\gesture2\test'


train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input).flow_from_directory(directory=train_path)
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input).flow_from_directory(directory=test_path)

In [3]: from matplotlib import pyplot as plt
imgs, labels = next(train_batches)

#Plotting the images...
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(30,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

plotImages(imgs)
print(imgs.shape)
print(labels)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



(10, 64, 64, 3)
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]

In [4]: model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(64,64,3)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation = "relu"))
model.add(Dense(128,activation = "relu"))
#model.add(Dropout(0.2))
model.add(Dense(128,activation = "relu"))
#model.add(Dropout(0.3))
model.add(Dense(10,activation = "softmax"))
```

```
In [5]: model.compile(keras.optimizers.Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
        reduce_lr = ReduceLRonPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0001)
        early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0, mode='auto')

        model.compile(optimizer=SGD(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
        reduce_lr = ReduceLRonPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0005)
        early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0, mode='auto')
```

```
In [6]: history2 = model.fit(train_batches, epochs=10, callbacks=[reduce_lr, early_stop], validation_data = test_batches)

Epoch 1/10
301/301 [=====] - 19s 61ms/step - loss: 0.6281 - accuracy: 0.9236 - val_loss: 1.7148 - val_accuracy:
0.4621 - lr: 0.0010
Epoch 2/10
301/301 [=====] - 14s 47ms/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 1.7270 - val_accuracy:
0.4701 - lr: 0.0010
Epoch 3/10
301/301 [=====] - 14s 47ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.7346 - val_accuracy:
0.4784 - lr: 5.0000e-04
```

```
In [7]: # For getting next batch of testing imgs...
        imgs, labels = next(test_batches)

        scores = model.evaluate(imgs, labels, verbose=0)
        print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')

        #Once the model is fitted we save the model using model.save() function.

        model.save('best_model_dataflair3.h5')

        loss of 1.4288480281829834; accuracy of 50.0%
```

```
In [8]: print(history2.history)

        imgs, labels = next(test_batches)

        model = keras.models.load_model(r"best_model_dataflair3.h5")

        scores = model.evaluate(imgs, labels, verbose=0)
        print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')

        model.summary()

        word_dict = {'One':1,'Ten':2,'Two':3,'Three':4,'Four':5,'Five':6,'Six':7,'Seven':8,'Eight':9,'Nine':}

        predictions = model.predict(imgs, verbose=0)
        print("predictions on a small set of test data--")
        print("")
        for ind, i in enumerate(predictions):
            print(word_dict[np.argmax(i)], end=" ")

        plotImages(imgs)
        print('Actual labels')
        for i in labels:
            print(word_dict[np.argmax(i)], end=" ")

('loss': [0.6280882954597473, 0.002735637594014406, 0.00143634423147887], 'accuracy': [0.9235880374908447, 1.0, 1.0], 'val_loss': [1.714832067489624, 1.7270175218582153, 1.7346190214157104], 'val_accuracy': [0.4621262550354004, 0.47009965777397156, 0.47840532660484314], 'lr': [0.001, 0.001, 0.0005])
loss of 2.7075629234313965; accuracy of 20.000000298023224%
Model: "sequential"
```

| Layer (type) | Output Shape | Param # |
|--------------------------------|---------------------|---------|
| conv2d (Conv2D) | (None, 62, 62, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 31, 31, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 31, 31, 64) | 18496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 15, 15, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 13, 13, 128) | 73856 |
| max_pooling2d_2 (MaxPooling2D) | (None, 6, 6, 128) | 0 |


```
conv2d_2 (Conv2D)          (None, 13, 13, 128)    73856
max_pooling2d_2 (MaxPooling (None, 6, 6, 128)    0
2D)
flatten (Flatten)          (None, 4608)            0
dense (Dense)              (None, 64)              294976
dense_1 (Dense)            (None, 128)             8320
dense_2 (Dense)            (None, 128)             16512
dense_3 (Dense)            (None, 10)              1290
```

```
*****
Total params: 414,346
Trainable params: 414,346
Non-trainable params: 0
```

predictions on a small set of test data--

Eight Four Four Nine Three One Four Four Eight Four

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Actual labels
Eight Five Five Three Two Seven Ten Ten Eight Five

Model_for_gesture:

```
In [1]: import numpy as np
import cv2
import keras
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
```

```
In [2]: model = keras.models.load_model(r"C:\Users\MYPC\Documents\anaconda\python project\best_model_dataflair3.h5")

background = None
accumulated_weight = 0.5

ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350
```

```
In [3]: def cal_accum_avg(frame, accumulated_weight):

    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)
```

```
In [4]: def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)

    _, thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

    # Grab the external contours for the image
    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        return None
    else:

        hand_segment_max_cont = max(contours, key=cv2.contourArea)

        return (thresholded, hand_segment_max_cont)

In [5]: cam = cv2.VideoCapture(0)
num_frames = 0
while True:
    ret, frame = cam.read()

    # flipping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(frame, 1)
    frame_copy = frame.copy()

    # ROI from the frame
    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)

    if num_frames < 70:

        cal_accum_avg(gray_frame, accumulated_weight)

        cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT",
            (80, 400), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,0,255), 2)

    else:
        # segmenting the hand region
        hand = segment_hand(gray_frame)

        # Checking if we are able to detect the hand...
        if hand is not None:

            thresholded, hand_segment = hand

            # Drawing contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (ROI_right,
                ROI_top)], -1, (255, 0, 0), 1)

            cv2.imshow("Thresholded Hand Image", thresholded)

            thresholded = cv2.resize(thresholded, (64, 64))
            thresholded = cv2.cvtColor(thresholded, cv2.COLOR_GRAY2RGB)
            thresholded = np.reshape(thresholded, (1, thresholded.shape[0], thresholded.shape[1], 3))

            pred = model.predict(thresholded)
            word_dict = {'0':'One', 1:'Ten', 2:'Two', 3:'Three', 4:'Four', 5:'Five', 6:'Six', 7:'Seven', 8:'Eight', 9:'Nine'}

            cv2.putText(frame_copy, word_dict[np.argmax(pred)],
                (170, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

            # Draw ROI on frame_copy
            cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right,
                ROI_bottom), (255, 128, 0), 3)

            # incrementing the number of frames for tracking
            num_frames += 1

            # Display the frame with segmented hand
            cv2.putText(frame_copy, "DataFlair hand sign recognition _ _",
                (10, 20), cv2.FONT_ITALIC, 0.5, (51, 255, 51), 1)
            cv2.imshow("Sign Detection", frame_copy)

            # Close windows with Esc
            k = cv2.waitKey(1) & 0xFF

            if k == 27:
                break

# Release the camera and destroy all the windows
cam.release()
cv2.destroyAllWindows()
```

5.3 SCREENSHOTS OF RESULTS

Sign Language Recognition Output

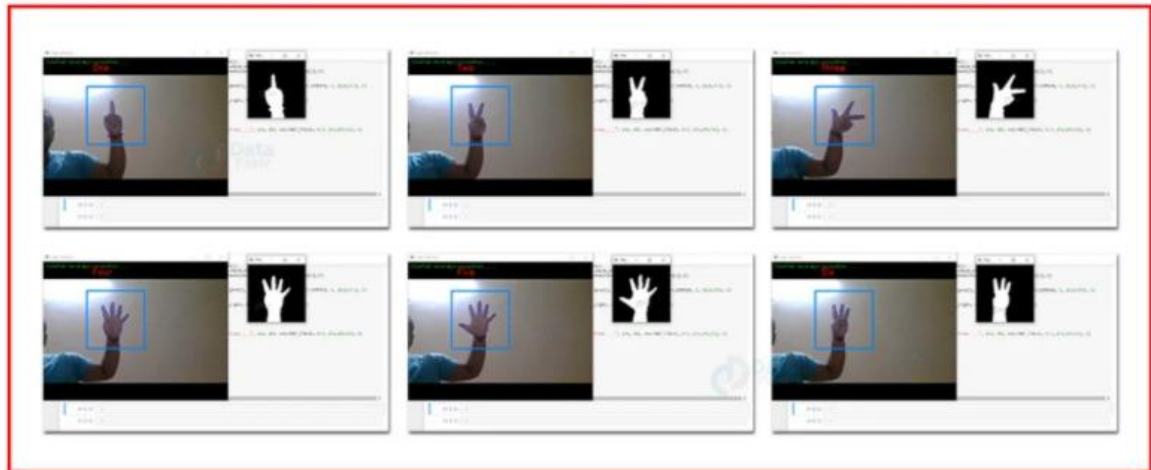


Fig5.1: Screenshot of the result obtained for digits -1,2,3,4,5 & 6

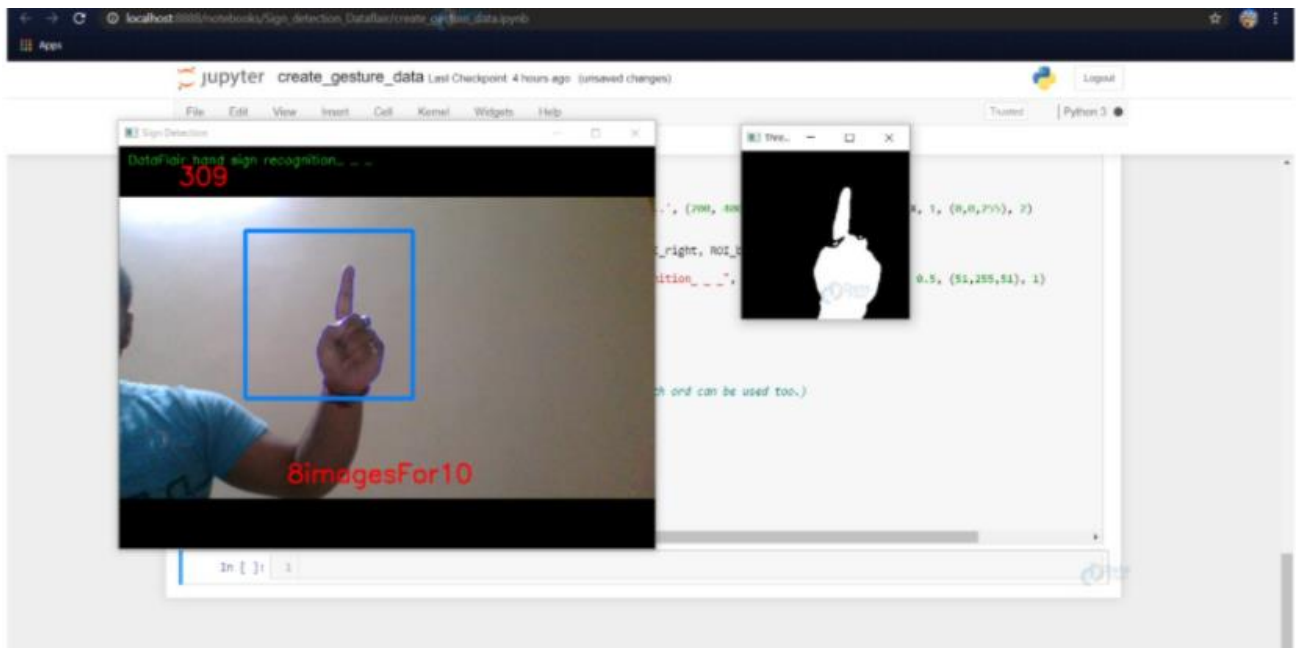


Fig5.2: Screenshot of the result obtained

6. CONCLUSION AND FUTURE SCOPE

Nowadays, applications need several kinds of images as sources of information for elucidation and analysis. Several features are to be extracted so as to perform various applications. When an image is transformed from one form to another such as digitizing, scanning, and communicating, storing, etc. degradation occurs. Therefore, the output image has to undertake a process called image enhancement, which contains a group of methods that seek to develop the visual presence of an image. Image enhancement is fundamentally enlightening the interpretability or awareness of information in images for human listeners and providing better input for other automatic image processing systems. Image then undergoes feature extraction using various methods to make the image more readable by the computer. Sign language recognition system is a powerful tool to prepare an expert knowledge, edge detect and the combination of inaccurate information from different sources. The intent of convolution neural network is to get the appropriate classification

Future work

The proposed sign language recognition system used to recognize sign language letters can be further extended to recognize gestures facial expressions. Instead of displaying letter labels it will be more appropriate to display sentences as more appropriate translation of language. This also increases readability. The scope of different sign languages can be increased. More training data can be added to detect the letter with more accuracy. This project can further be extended to convert the signs to speech.

REFERENCES

Base Paper:

- http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf
- [http://www.iosrjen.org/Papers/vol3_issue2%20\(part-2\)/H03224551.pdf](http://www.iosrjen.org/Papers/vol3_issue2%20(part-2)/H03224551.pdf)

Other References:

- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.734.8389&rep=rep1&type=pdf>
- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.734.8389&rep=rep1&type=pdf>
- <https://ieeexplore.ieee.org/document/7507939>
- <https://ieeexplore.ieee.org/document/7916786>
- <https://www.sciencedirect.com/science/article/pii/S1877050917320720>
- <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>
- <https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>