SIGN LANGUAGE RECOGNITION USING IMAGE PROCESSING

A Project done by students

Mr. Ashay Babu Naik

Ms. Christina Lucia Colaco

Mr. Hayden Abel Larry Gomes

Mr. Liam Savio Carvalho

Mr. Lyndon Manuel D'Souza

Ms. Nidhi Satyanarayan Haldankar

Ms. Valencia Merlin Salgaonkar

T.Y.B.Sc. (Electronics)

2022-2023

PROJECT SUPERVISOR

Dr. Caje Francis Pinto

St. Xavier's College, Mapusa, Goa.

Goa University

CERTIFICATE

This is to certify that the Project Entitled

SIGN LANGUAGE RECOGNITION USING IMAGE PROCESSING

Is a record work done by:

Mr. Ashay Babu Naik

Ms. Christina Lucia Colaco

Mr. Hayden Abel Larry Gomes

Mr. Liam Savio Carvalho

Mr. Lyndon Manuel D'Souza

Ms. Nidhi Satyanarayan Haldankar

Ms. Valencia Merlin Salgaonkar

T.Y.B.Sc. (Electronics)

2022-2023

H.O.D.	External Examiner	Project Guide

DECLARATION BY STUDENTS

We declare that this Project Report has been prepared by us, and to the best of our knowledge, it has not previously formed the basis for the award of the Diploma or Degree by this or any other University.

ROLL NO.	NAME OF THE STUDENTS	SIGNATURE
20-7403	Mr. Ashay Babu Naik	
20-7405	Ms. Christina Lucia Colaco	
20-7411	Mr. Liam Savio Carvalho	
20-7412	Mr. Lyndon Manuel D'Souza	
20-7413	Ms. Nidhi Satyanarayan Haldankar	
20-7420	Ms. Valencia Merlin Salgaonkar	
20-7421	Mr. Hayden Abel Larry Gomes	

CERTIFICATE BY SUPERVISOR

Certified that this Project Report is a record of work done by the candidate themselves under my guidance during the period of study and that to the best of my knowledge, it has not previously formed the basis of the award of any Degree or Diploma by this or any other University.

Dr. Caje Francis Pinto

Project Supervisor

ACKNOWLEDGEMENT

"The completion of a task with perfection requires, the guidance and contribution of many." As we complete our project, we take this opportunity to thank all those who played an important role in it with their valuable suggestions and guidance during the period of its preparation, testing and working.

We are grateful to our project guide Dr. Caje Francis Pinto from Department of Electronics, St. Xavier's College, Mapusa for his constant encouragement and guidance right from the beginning till the completion of our project.

A big thank you goes out to Ma'am Vilma Fernandes, Head of the Department of Electronics, St. Xavier's College, Mapusa for her constant encouragement and guidance.

We would also like to thank our lab attendant Ms. Santana Fernandes who most willingly helped us at our work. We would also like to thank our parents for helping and co-operating in doing this project.

To conclude we would like to thank our project-mates and our friends for their help and innovative ideas. We owe the success of our entire project to all the people mentioned above.

Table of Contents

SIGN LANGUAGE RECOGNITION USING IMAGE PROCESSING

List of Figures

List of Table

ABSTRACT

In Today's World sign language is an essential tool for communication among the deaf and hard-hearing community. However, many hearing individuals are not fluent in sign language, which creates a communication barrier. To help the community, a language translator is extensively utilized by the mute people for converting and giving shape to their thoughts. Therefore, we needed a system to recognize and translate sign language.

In this project, we propose a deep learning-based sign language recognition system using Python OpenCV and Keras modules. The proposed system consists of two main components: data acquisition and sign language recognition. The data acquisition component involves capturing real-time sign images using a webcam. The images are pre-processed using a Gaussian filter to make the image smooth and to eliminate all the irrelevant noise. The intensity is analysed and a Non-Maximum suppression is implemented to remove the chris false edges. Also, for better pre-processed image data, double thresholding is implemented to consider only the strong edges in the images. All the weak edges are finally removed and only the strong edges are considered for the further stages. This is then fed into a deep learning model for recognition such as Convolutional Neural Networks (CNNs). The CNNs are used to extract features from the real-time American Sign Language (ASL) data set for training and testing the model. The performance of the system was evaluated by using various metrics. Our model gave us 99.72% accuracy for Sign Language Detection after training the model for 60 minutes with 8100 images in the train dataset and 2700 images in the test dataset. Finally, a real-time sign language recognition system using a webcam is implemented to translate the sign language into text and speech. Our model is capable of predicting gestures from American sign language in real-time with high efficiency. These predicted alphabets are converted to form words and hence form sentences. These sentences are converted into voice modules by incorporating Google Text to Speech API. The model is efficient since we used a compact CNN-based architecture, it's also computationally efficient and thus makes it easier to deploy the model to embedded systems such as Raspberry Pi 3. This system can therefore be used in real-time applications which aim at bridging the gap in the process of communication between Deaf and Dumb people all over the world.

CHAPTER 1: INTRODUCTION

Communication is an important part of our lives. Deaf and dumb people being unable to speak and listen, experience a lot of problems while communicating with normal people. There are many other factors have affected today's generation of people such as Age-related hearing loss, Noise-induced hearing loss, Exposure to loud noise, Genetics, Ear infections, Trauma, Neurological conditions, Developmental disorders, Congenital disabilities, Structural abnormalities, etc. There are many ways by which people with these disabilities try to communicate. One of the most prominent ways is the use of sign language, i.e. hand gestures. Around 500,000 to 2,000,000 speech and hearing-impaired people express their thoughts through Sign Language in their daily communication as shown in figure 1.1. These numbers may diverge from other sources but it is most popular as mentioned that American Sign Language (ASL) is the 3rd most-used sign language in the world.

Figure 1.1: Sign Language in daily Communication.

ASL is a language for the hearing impaired and deaf alike people, in which manual communication with the help of hands, facial expression, and body language is used to convey thoughts to others without using sound. ASL is generally preferred as the communication tool for deaf and dumb people as shown in figure 1.2.

Figure 1.2: American Sign Language.

It is necessary to develop an application for recognizing gestures and actions of sign language so that deaf and dumb people can communicate easily with even those who don't understand sign language. There have been several advancements in technology and a lot of research has been done to help 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 deaf and dumb people in communicating with others as knowing sign language is not something common to all, moreover, this can be extended to creating automatic editors, where the person can easily write with just their hand gestures.

We the students of the Electronics Department at St Xavier's College Mapusa have decided to help one such category of people to help them live a better quality of life and improve their position in the digital front of society. In this project, we are recognizing gestures made by a person using the webcam and converting them into respective text and speech. Techniques like Image segmentation are used to crop the required part of the image. A gray algorithm is used for compensating the illumination in the picture. This project aims to develop a system that translates ASL to text using a webcam with

Raspberry Pi 3. The system is implemented using the Python programming language and will use deep learning techniques to recognize and translate ASL gestures captured by the webcam into text. The goal of the project is to create a practical and accessible solution for communication between individuals who use ASL and those who do not, making it easier for them to communicate with each other. It will also be a valuable resource for educators, healthcare professionals, and anyone else who may need to communicate with individuals who use ASL. The system will be able to translate sign language in real-time, facilitating natural and seamless communication. Overall, this project aims to bridge the communication gap between individuals who use ASL and those who do not and make it easier for them to connect and communicate with each other. The development of this system is an important step in making the world more inclusive and an accessible place for everyone.

CHAPTER 2: BACKGROUND

2.1: Literature Survey

A lot of research work is carried out in the field of Sign Language which is discussed below.

Ankit Ojha et. al [1] worked on Sign Language to Text and Speech Translation in Real Time by Creating a desktop application that uses a computer's webcam to capture a person's signing gestures for American sign language (ASL) and translate it into corresponding text and speech in real-time. The translated sign language gesture will be then acquired in the text which is further converted into audio. The database includes one thousand special gesture images. The technique used to enable the detection of gestures is a Convolutional neural network (CNN). CNN can be used to solve computer vision problems with an extremely high degree of accuracy. The result of this project leads to a success rate of 98% accuracy. Having used the machine learning algorithms, they have used YCbCr (Yellow Luminance (Y), Chroma blue (Cb), Chroma red (Cr) spacing of color CbCr plane to distribute the skin tone color.

Sawant Pramada et. al. [2] worked on Intelligent Sign Language Recognition Using Image Processing. This project introduces an efficient and fast algorithm for the identification of the number of fingers opened in a gesture representing an alphabet of the Binary Sign Language. The system does not require hand to be perfectly aligned with the webcam. The project uses an image processing system to identify, especially English alphabetic sign language used by deaf people to communicate. The basic objective of this project is to develop a computer-based intelligence system that will enable dumb people significantly to communicate with all other people using their natural hand gestures. The system's overall performance was 90.9%. The algorithm used here is image processing, machine learning, and artificial intelligence concept that take visual inputs of sign language hand gestures and generate an easily recognizable form of outputs. The data size used for this project is 31 gestures using the fingers of a single hand, and 1023 gestures if both hands are used. And for each pixel typically two colors are used black and white though any two colors can be used.

Mahesh Kumar N B [3] worked on the Conversion of Sign Language into Text. This paper shows the sign language recognition of 26 hand gestures in Indian sign language using MATLAB. The proposed system contains four modules such as pre-processing and hand segmentation, feature extraction, sign recognition, and sign-to-text. By using image processing the segmentation can be done. Some of the features are extracted such as Eigen values and Eigen vectors which are used in recognition. The Linear Discriminant Analysis (LDA) algorithm was used for gesture recognition and the recognized gesture is converted into text and voice format. The recognition of various hand gestures was done by vision-based approaches, data glove-based approaches, soft computing approaches like Artificial Neural Networks, Fuzzy logic, Genetic Algorithms, and others like PCA, Canonical Analysis, etc. The recognition techniques are divided into three broad categories as Hand segmentation approach, the Feature extraction approach, and the Gesture recognition approach. By using the LDA algorithm for sign recognition operation the dimensionality will be reduced. Due to dimensionality reduction, the noise will be reduced with high accuracy.

Amrita Thakur et. al. [4] worked on Real Time Sign Language Recognition and Speech Generation. This paper works on computer vision-based hand gesture recognition using a convolutional neural network in Python. In computer vision-based gesture recognition, the webcam is used for input, and image processing of input gestures is done before recognition. The processed gestures then are recognized using various algorithms like Hidden Markov Model and Neural network techniques. Neural Network techniques and Hidden Markov Model are used together with sensor data for more accuracy. The images of various alphabets of American Sign Language were collected using different webcams from different laptops. A total of 2500 images for each alphabet till now were collected. Then the collected data set was shuffled for each category to generate a random data set for each category and used data augmentation processes to get another 2500 images by horizontally flipping 1000 images, adding Gaussian noise to 600 images, and adding Gamma contrast to 900 images using image libraries in Python The model proposed by Simonyan had the training accuracy and training loss were obtained as 99.65 % and 0.0259 respectively. And the test accuracy was 99.62%. The total number of images used to train the model was 32,000 and the test data set had 8,000 images. Parallel Hidden Markov models (PaHMMs) with 22 signs resulted showing an 87.88% accuracy rate. This paper aims to build a user-friendly and accurate sign language recognition

system trained by a neural network thereby generating text and speech of the input gesture. This paper also presents text to sign language generation model that enables a way to establish two-way communication without the need for a translator.

Rajaganapathy. S et. al [5] worked on the Conversation of Sign Language to Speech with Human Gestures. The motive of this paper is to convert human sign language to Voice with human gesture understanding and motion capture. This is achieved with the help of Microsoft Kinect a motion capture device from Microsoft. Research in this sign language system has two well-known approaches i.e., Image processing and Data glove. The system is stably designed to identify 20 human joints (head, hand right, hand left, and so on). In a test done for a sample of 100 spells for different signs. Accuracy of up to 90 percent has been achieved. There are a few systems available for sign language to speech conversion but none of them provide a natural user interface. For consideration, if a person cannot speak and to perform human gestures and the system converts the human gestures into speech and plays it loud so that the person can communicate to a mass crowd gathering. Also, the system is planned in bringing high efficiency for the users for improved communication.

Chhaya Narvekar et. al [6] worked on Sign language to speech conversion using image processing and machine learning. This project consists of image processing and machine learning methods for this purpose. The aim is to design a human-computer interface system that can recognize the language of the deaf and dumb accurately. In this paper, a vision-based hand gesture recognition system has been discussed as the hand plays a vital communication mode, considering various techniques available for hand tracking, segmentation, feature extraction, and classification are referred. Implementation of the project is as; images are captured using a webcam and are processed using image processing techniques such as the OTSU method and classification of the captured gesture is done by using the linear classification method. Here, the captured gestures are stored in folders consisting of 120 replicas of the same gesture. Image gesture is captured in the form of a histogram. The algorithm used in this project is the SIFT algorithm. In this project, they have used the Naïve Bayes Classification method that provides maximum accuracy by less noise distortion and creating multiple replicas of each gesture. We have seen from the above-related work that many have designed a Sign Language Translator

using machine Convolutional Neural Networks, Image Processing, and convolutional neural network on Python, Matlab, etc.

All of the literature papers above deal with the day-to-day problems occurring communications amongst dumb and deaf individuals and thus a solution have been proposed regarding the same. Hand gestures are a powerful way of human communication, with lots of potential applications in the area of human-computer interaction. Vision-based hand gesture recognition techniques have many proven advantages compared with traditional devices. Many breakthroughs have been made in the field of artificial intelligence, machine learning and computer vision. However, most of them require extra computing power. On the other hand. In our project, we proposed to normalise and rescale our images to 32 pixels to extract features and make the system more robust. We propose to use CNN architecture to classify the 26 alphabetical American sign gestures to achieve an accuracy of above 99% which will be better.

2.2: Machine and Deep learning

Machine Learning is all about machines learning automatically without being explicitly programmed or learning without any direct human intervention. This machine learning process starts with feeding them good quality data and then training the machines by building various machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data we have and what kind of task we are trying to automate.

Different types of Machine Learning

1. **Supervised Machine Learning**

   The algorithm learns from a training dataset and makes predictions that are compared with the actual output values. If the predictions are not correct, then the algorithm is modified until it is satisfactory. This learning process continues until the algorithm achieves the required level of performance. Then it can provide the desired output values for any new inputs.

2. **Unsupervised Machine Learning**

   The algorithm doesn't figure out any output for input but it explores the data. The algorithm is left unsupervised to find the underlying structure in the data to learn more and more about the data itself.

3. **Reinforcement Machine Learning**

   The algorithms learn optimal actions through trial and error. This means that the algorithm decides the next action by learning behaviors that are based on its current state and that will maximize the reward in the future. This is done using reward feedback that allows the Reinforcement Algorithm to learn which are the best behaviors that lead to maximum reward. This reward feedback is known as a reinforcement signal.

Deep Learning is a subset of Machine Learning. It is based on learning by example, just like humans do, using Artificial Neural Networks. These Artificial Neural Networks are created to mimic the neurons in the human brain so that Deep Learning algorithms can learn much more efficiently. Deep Learning is so popular now because of its wide range of applications in modern technology. From self-driving cars to image, speech

recognition, and natural language processing, Deep Learning is used to achieve results that were not possible before. Figure 2.1 shows the pictorial representation of Machine learning v/s Deep learning.

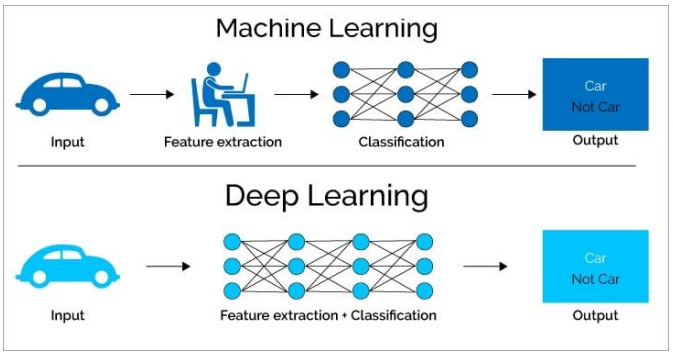


Figure 2.1: Machine learning v/s Deep learning.

Deep Learning is a Machine Learning method that takes in an input of X and uses it to predict an output of Y. As an example, given the image of the fundus of the eye, the Deep Learning algorithm will try to predict whether the person is likely to go blind or not. Given a large data set of input and output pairs, a Deep Learning algorithm will try to minimize the difference between its predicted and expected output. By doing this, it tries to learn the association/pattern between the given inputs and outputs, this in turn allows a Deep Learning model to generalize to inputs that it hasn't seen before.

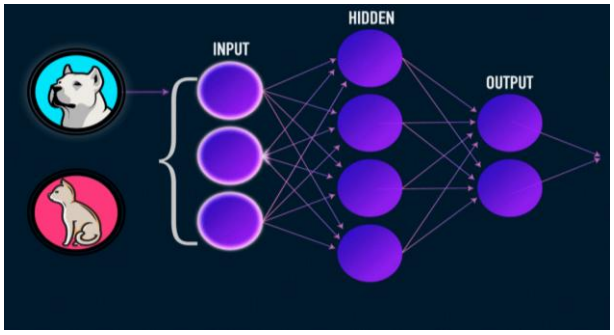


Figure 2.2: Neural Network using cat and dog images as inputs.

As another example figure 2.2, let's say that inputs are images of dogs and cats, and outputs are labels for those images (i.e. is the input picture a dog or a cat). If an input has a label of a dog, but the Deep Learning algorithm predicts a cat, then the Deep Learning algorithm will learn that the features of the given image (e.g. sharp teeth, facial features) which are associated with a dog.

2.3: Neural networks

Deep Learning Algorithms use Neural Network to find associations between a set of inputs and outputs. A Neural Network is composed of an input layer, a hidden layer, and an output layer, all of which are composed of nodes. Input layers take in a numerical representation of data (e.g. images with pixel specs), output layers make predictions, while hidden layers are correlated with most of the computation. The information is passed between network layers through the function shown in figure 2.3.

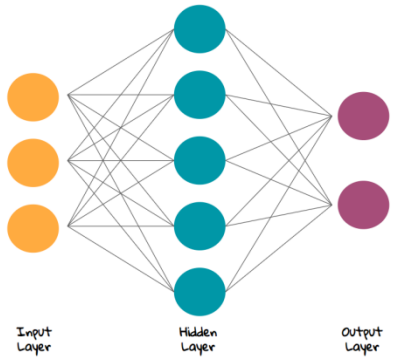


Figure 2.3: Simplified Neural Network.

The tuneable weight and bias parameters represented by w and b respectively in the function as shown in figure 2.4. These are essential to the actual Learning process of a Deep Learning algorithm. After the Neural Network passes its inputs to its outputs, the network evaluates how good its prediction was and it is called a loss function.

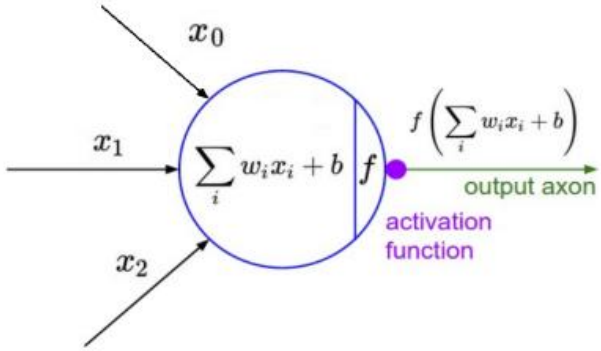


Figure 2.4: Primary Function in Neural Networks.

The goal of a network is ultimately to minimize this loss by adjusting the weights and biases of the network. In using backpropagation through gradient descent, the network backtracks through all its layers to update the weights and biases of every node in the opposite direction of the loss function in other words, every iteration of backpropagation

should result in a smaller loss function than before. The continuous updates of the weights and biases of the network ultimately turn it into a precise function approximator one that models the relationship between inputs and expected outputs.

Activation Functions

There are several activation functions and we need to choose a proper one depending on the problems. Activation functions are mathematical equations that determine the output of a Neural Network. The function is attached to each neuron in the network and determines whether it should be activated or not, based on whether each neuron's input is relevant to the model's prediction. To activate the real power of Neural Networks, we need to apply an activation function that helps the model to capture non-linearities within the data. The different activation functions are shown in figure 2.5.

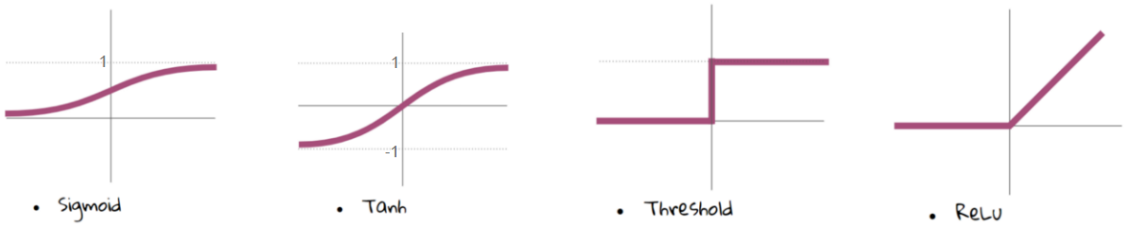


Figure 2.5: Examples of activation functions.

The sigmoid function is appropriate for the case of binary classification. It transforms the values only between 0 and 1. The higher the input values are, the closer it goes to 1. The smaller the input values are, the closer it goes to 0.

Tanh function (Tangent Hyperbolic) is similar to the sigmoid function, but its lower limit goes to -1. As it sets the center of data at 0, Tanh is preferred over the sigmoid function.

The threshold function and ReLu (Rectified Linear Units) have a certain point from which the value changes. Because of the slope, ReLu is efficient to use in most cases.

Gradient Descent:

Gradient Descent is a process that occurs in the backpropagation phase where the goal is to continuously resample the gradient of the model's parameter in the opposite direction based on the weight w, updating consistently until we reach the global minimum of a cost function.

2.4 Convolutional Neural Network

In the last few years of the IT industry, there has been a huge demand for one particular skill set known as Deep Learning. Deep Learning is a subset of Machine Learning which consists of algorithms that are inspired by the functioning of the human brain or the neural networks. A convolutional neural network (CNN) is a network architecture for deep learning that learns directly from data. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. They can also be quite effective for classifying audio, time-series, and signal data. CNN can be very useful to minimize human effort by automatically detecting the features. For example, for apples and mangoes, it would automatically detect the distinct features of each class on its own.

A convolutional neural network can have tens or hundreds of layers that each learn to detect different features of an image. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters can start as very simple features, such as brightness and edges, and increase in complexity to features that uniquely define the object.

There are 2 primary parts to the CNN architecture:

- A convolution tool that separates and identifies the different features of the image for review in a process called Extraction Functionality.
- The functionality extraction network consists of several convolutional or pooled layer pairs.
- A fully connected layer that makes use of the convolutional process's output and determines the class of the image using the features that were previously extracted.
- This CNN feature extraction model seeks to minimize the number of features in a data set. It generates new features that compile an initial set of features' existing features into a single new feature. As depicted in figure 2.5 in the CNN architectural diagram, there are numerous CNN levels.

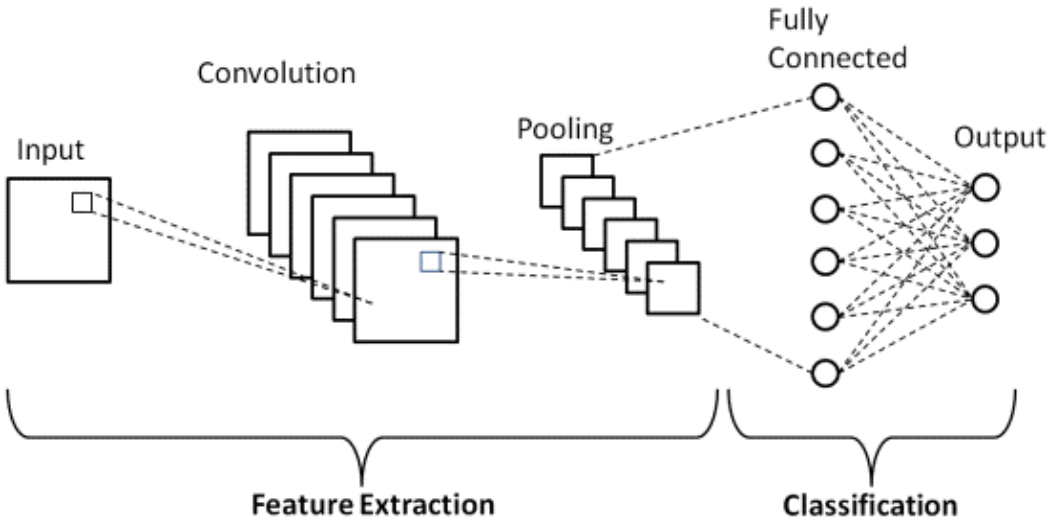


Figure 2.5: CNN architectural diagram.

Layers of CNN

The CNN is made up of three different kinds of layers: fully connected (FC), pooling, and convolution layers. A CNN architecture is created when these layers are stacked. The dropout layer and the activation function, which are detailed below, are two additional crucial parameters in addition to these three layers.

1) Convolution Layer

The first layer utilized to extract the different features from the input photos is this one. Convolution is a mathematical process that is carried out at this layer between the input image and a filter of a specific size, MxM. The dot product is taken between the filter and the input image's components about the filter's size by sliding the filter over the input image (MxM). The output is called a feature map, which gives us information about the image, such as corners and edges. Later, this feature map is passed to other layers to learn several other features from the input image. The CNN convolution layer passes the result to the next layer when a convolution function is applied to the input. CNN convolutional layers are very useful because they ensure that the spatial relationship between pixels is preserved.

Rectified linear unit (ReLU) allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is called as an activation, because only the activated features are carried forward into the next layer.

2) Pooling Layer

Pooling layers almost always follow the convolutional layers. The main purpose of this layer is to reduce the size of convolutional feature maps to reduce computational costs. This is done by reducing connections between layers and works independently on each feature map. There are several types of pooling operations depending on the method used. It summarizes the features produced by the convolutional layers. Max pooling gets the largest element from the feature map. Average pooling calculates the average of elements within a defined image section. Total pooling calculates the total items within a predefined section. A pooling layer usually acts as a bridge between convolutional and FC layers. This CNN model generalizes the features extracted by the convolutional layers, allowing the network to recognize the features individually. With its help, computations are also reduced in the network by reducing the number of parameters that the network needs to learn.

3) Fully Connected Layer

The fully connected (FC) layer includes weights and biases as well as neurons and is used to connect neurons between two different layers. These layers are usually placed before the output layer and form the final layers of the CNN architecture. In this case, the input image from the previous layers is flattened and converted to the FC layer. The flattened vector then goes through a few more FC layers, where the mathematical function operations normally take place. At this point, the sorting process begins. The reason two layers are connected is that two fully connected layers will perform better than one connected layer.

4) Dropout

Usually, when all the features are connected to the Fully Connected layer, that can lead to overfitting in the training data set. Overfitting occurs when a particular model performs too well on the training data, which negatively affects the model's performance when used on new data. To overcome this problem, a suppression layer is used where some neurons are removed from the neural network during training, which reduces the size of the model. When passing the dropout rate of 0.3, 30% of the nodes are randomly dropped from the neural network. Dropping improves the performance of the machine

learning model because it prevents overfitting by simplifying the network. It removes neurons from the neural network during training.

5) Activation Function

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and estimate any kind of continuous and complex relationship between network variables. Simply put, it decides which information from the model should be activated in the forward direction and which information should not be activated at the end of the network. It adds nonlinearity to the network. There are several commonly used activation functions such as ReLU, Softmax, tanH, and Sigmoid. Each of these functions has a specific use. For binary classification, the sigmoid and softmax functions of the CNN model are preferred, and for multi-class classification we usually use softmax. Simply put, the activation functions in the CNN model determine whether a neuron should be activated or not. It decides whether the work input is important or not predicted using mathematical operations.

2.5 Python and it's Packages

Python is an object-oriented programming language. It is a lightweight language and an open-source coding language. Python is very similar to other popular programming languages. The Prerequisites for sign language recognition using image processing are:

1. Numpy
2. Pandas
3. Keras
4. TensorFlow
5. Gtts
6. Sklearn / scikit learn
7. Playsound
8. Matplotlib
9. model-checkpoint
10. Opencv-python

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices. It is an open-source project and you can use it freely. NumPy stands for Numerical Python. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory, unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also, it is optimized to work with the latest CPU architectures.

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name Pandas has a reference to both Panel Data, and Python Data Analysis. Pandas allow us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them

readable and relevant. Pandas are also able to delete rows that are not relevant or contain wrong values, like empty or NULL values. This is called cleaning the data.

Keras is a high-level, deep-learning API developed by Google for implementing neural networks. Keras is the best when working with small data sets, rapid prototyping, and multiple back-end support. Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly. Prototyping time in Keras is less. This means that your ideas can be implemented and deployed in a shorter time. Keras also provides a variety of deployment options depending on user needs. Keras is also deeply integrated with TensorFlow, so one can create customized workflows with ease.

The tensorFlow platform helps us implement best practices for data automation, model tracking, performance monitoring, and model retraining. Using production-level tools to automate and track model training over the lifetime of a product, service, or business process is critical to success. TensorFlow has adopted Keras as its official high-level API. Keras is embedded in TensorFlow and can be used to perform deep learning fast as it provides inbuilt modules for all neural network computations. TensorFlow allows developers to create dataflow graphs structures that describe how data moves through a graph or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array or tensor. TensorFlow applications can be run on almost any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs, or GPUs. The resulting models created by TensorFlow, though, can be deployed on almost any device where they will be used to serve predictions.

gTTS (Google Text-to-Speech), a Python library and CLI tool to interface with Google Translate's text-to-speech API. It can be used for speech translation and is a very easy-to-use tool that converts the text entered, into audio which can be saved as an mp3 file. The gTTS API supports several languages including English, Hindi, Tamil, French, German, and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow.

PlaySound function plays a sound specified by the given file name, resource, or system event. The playsound module contains only one thing - the function (also named) play sound. It requires one argument i.e. the path to the file with the sound you'd like to play. This may be a local file or a URL. WAVE and MP3 have been tested and are known to work. Other file formats may work as well. The playsound module is a cross platform module that can play audio files.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering, and dimensionality reduction via a consistency interface in Python.

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing a feature to control line styles, font properties, formatting axes, etc. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

OS module provides the facility to establish the interaction between the user and the operating system. It offers many useful OS functions that are used to perform OS-based tasks and get related information about the operating system. The OS comes under Python's standard utility modules. The os module is a part of the standard library within Python 3. This means that it comes with your Python installation, but you still must import it. The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

ModelCheckpoint callback is used in conjunction with training using the model. fit() to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved. The checkpoint may be used directly or as starting point for a new run, picking up where it left off. When training deep learning models, the checkpoint is at the weight of the model. These weights can be used to make predictions as iron as the basis for ongoing training.

OpenCV stands for Open Source Computer Vision. To put it simply, it is a library used for image processing. It is a huge open-source library used for computer vision applications, in areas powered by Artificial Intelligence or Machine Learning algorithms, and for completing tasks that need image processing. As a result, it assumes significance today in real-time operations in today's systems. Using OpenCV, one can process images and videos to identify objects, faces, or even the handwriting of a human. The OpenCV Library is compatible with Windows, Linux, and Mac.

When an image is read using OpenCV, it is represented using a numpy array.

Syntax: pip install opencv-python

pip install numpy

Capture and read images/videos from different sources

1. Read and show an image:

```
import cv2
img= cv2.imread('path-of-the-image')
print (img)
cv2.imshow(''title-of-the-OpenCV-window",img)
cv2.waitKey()
cv2.destroyAllWindows()
```

2. Capture Read Videos:

Part 1: Capture Videos:

a. **Capture video frame from an external video file or live video stream using the laptop's webcam:**

```
import cv2
captureobject=cv2.VideoCapture('path-to-video')
captureobject=cv2.VideoCapture(0)
```

b. **Capture video frame from external webcam connected to desktop / laptop:**

```
captureobject=cv2.VideoCapture(n)
```

c. **Capture video frame from phone video stream:**

```
captureobject=cv2.VideoCapture(<URL>)
```

Part 2: Read Video frames captured:

```
ret, frame = captureobject.read()
```

Basic Transformations using OpenCV

1. Resizing the images:

```
resized_img = cv2.resize(img, (new_width, new_height))
```

2. Read Images with different colors:

a. Read the image with BGR color:

```
img= cv2.imread('path-to-image')
```

b. Read image as grayscale:

```
img= cv2.imread('path-to-image')
gray_img = cv2.cvtColor (img, cv2.COLOR_BGR2GRAY)
```

3. Flipping an image:

```
flip_img= cv2.flip(img, 0)
```

4. Inverting an image:

```
invert_img=cv2.bitwise_not(img)
```

5. Blur an image:

```
blur_img=cv2.blur(img, (kernel_width, kernel_height))
```

Advanced Transformations using OpenCV

1. Dilation of an image:

```
dilated_img= cv2.dilate(img, (kernel_width, kernel_height))
```

2. Erosion of an image:

```
eroded_img= cv2.erode(img, (kernel_width, kernel_height))
```

3. Thresholding:

```
threshold_image = cv2.threshold(gray_img, threshold_value,
maximum_pixel_value, cv2.THRESH_BINARY)
```

4. Edge Detection:

```
img_edge = cv2.Canny(img, threshold_value1, threshold_value2)
```

Saving Images on various devices

1.Writing images on to any device:

```
img_written - cv2.imwrite(path-to-save-the-image, img)
```

2.Saving the frames from videos:

```
import cv2
videoObject = cv2.VideoCapture(0)
while True:
    var, frame = videoObject.read()
     cv2.imshow('Video', frame)
    cv2.imwrite("path-to-the-image", frame)
    if cv2.waitkey(1) == and 0xFF == ord('q'):
        break
    videoObject.release()
    cv2.destroyAllWindows()
```

CHAPTER 3: METHODOLOGY

3.1 Objectives

Our aim is to develop an accurate device for Sign Language Recognition using image processing. Our objectives may be stated as follows:

1. To implement pre-processing algorithms for images such as grayscale image, Gaussian Filter, image segmentation, noise reduction, edge detection techniques.
2. To create a model for training sign languages using CNN architecture.
3. To achieve an accuracy of more than 99% in predicting sign language in real-time.

3.2 Overview of the Block diagram of Sign Language Recognition system

The proposed system as its core is a Sign Language Recognition system as shown in figure 3.1. The system consists of Raspberry Pi 3 with a webcam, speaker, and screen. In this project, the webcam captures the person's sign language gesture and sends it to the Raspberry Pi 3. Raspberry Pi 3 is a key element in the processing module, performing functions such as image processing, image segmentation, grayscale processing, data training, and testing. A sign language translator is created that recognizes the alphabet from A to Z. The project was developed using Python's OpenCV, TensorFlow, and Keras modules to train our data sets. The live gestures with sign language are translated into text and speech.

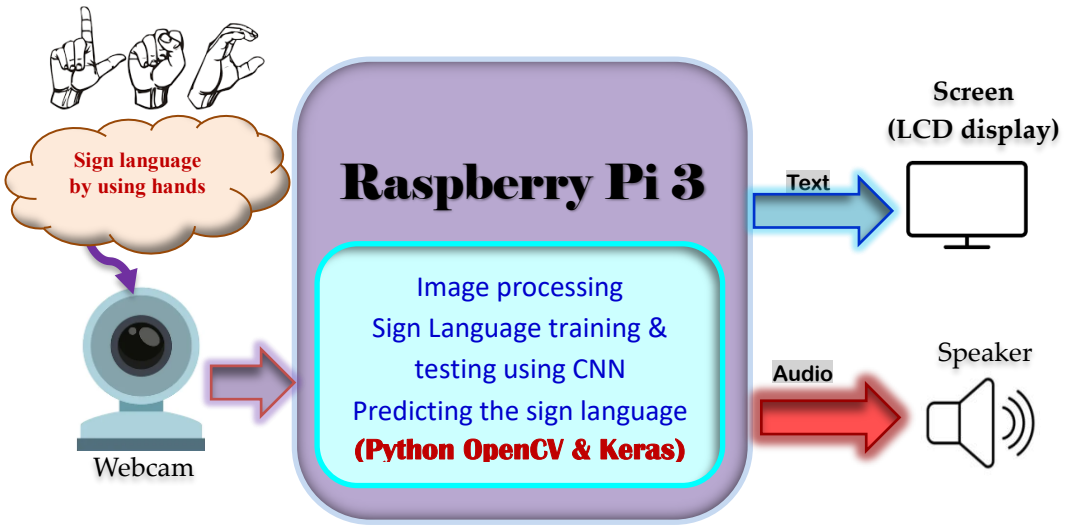


Figure 3.1: Block diagram of Sign Language Recognition System.

3.2 Software Implementation

The flowchart of Sign Language Training using CNN and Real-time Sign Language Prediction are shown in figure 3.2 and figure 3.3 respectively. The various steps in training the dataset and predicting sign language using CNN model in real-time are also explained further in details.

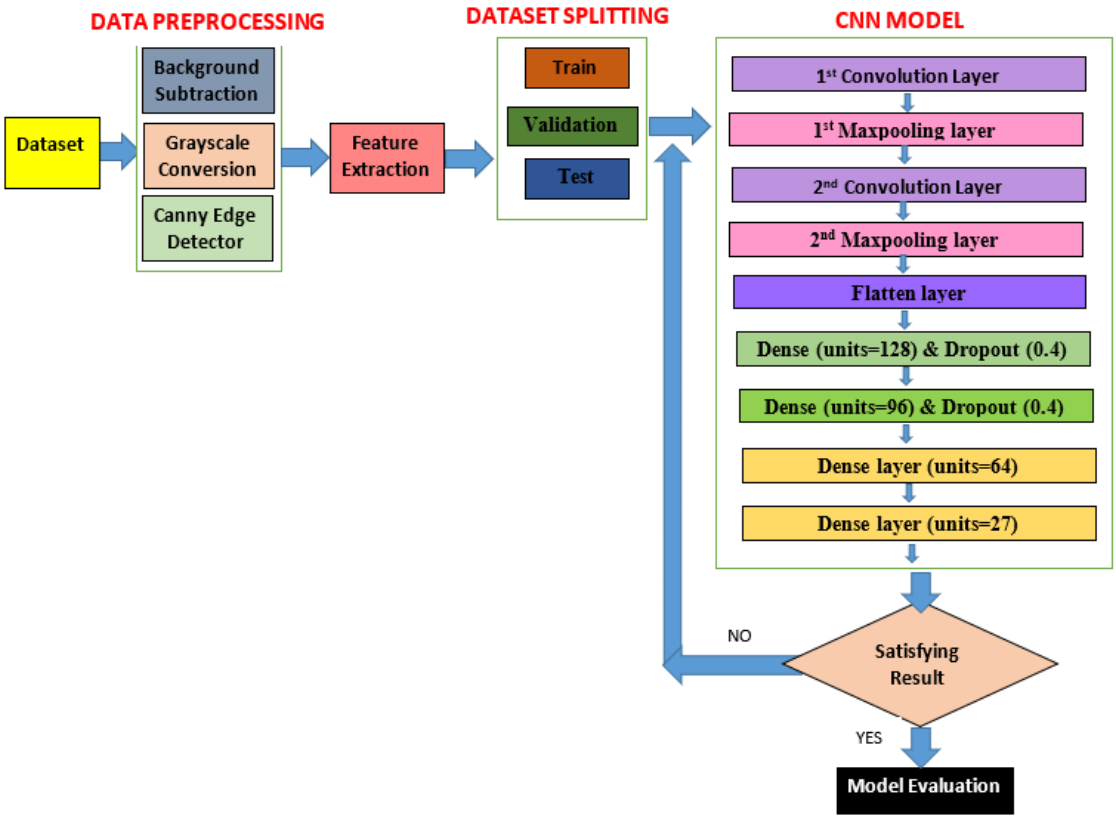


Figure 3.2: Flowchart of Sign Language Training using CNN.

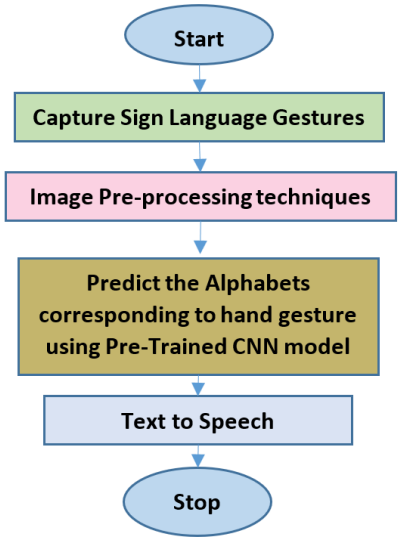


Figure 3.3: Flowchart of Real-time Sign Language Prediction.

There are two codes for this project namely (1) ASL Train and (2) Real-time ASL. We shall briefly explain the code below.

(1) ASL Train

a) The Python script builds and trains a Convolutional Neural Network (CNN) model to classify American Sign Language (ASL) images.

b) The script reads the dataset from a given path, pre-processes the images by converting them to grayscale, resizing them, and applying various filters to enhance the image quality.

c) The preprocessed data is then split into training and testing sets, and the CNN model is built using the Keras library.

d) The model consists of multiple layers of convolutional and pooling layers, followed by fully connected layers with dropout regularization to prevent overfitting.

e) The CNN model is trained using the Adam optimizer and categorical cross-entropy loss function.

f) The script also saves the model to a file and generates plots for the training and validation loss and accuracy metrics.

(2) Real time ASL

a) The Python script for real-time American Sign Language (ASL) recognition using a pre-trained convolutional neural network (CNN) model.

b) The model was trained on a dataset of hand gestures corresponding to each letter of the alphabet and can recognize the gesture in real-time video input from a webcam.

c) The script uses OpenCV to capture video input from the webcam and processes the video to isolate the hand gesture using various image processing techniques such as thresholding and contour detection.

d) The isolated gesture is then resized and normalized before being fed into the pre-trained CNN model.

e) The output of the model is a probability distribution over the 27 classes (26 letters and a space) which is used to determine the most likely letter corresponding to the hand gesture.

f) The recognized letter is displayed on the video feed, along with a running string of recognized letters which is updated every 200 frames (approximately every 10 seconds).

g) The script also generates an audio output of the recognized string using the Google Text-to-Speech (gTTS) API, which is played back using the playsound library.

h) Overall, the script provides a functional ASL recognition system that can recognize hand gestures in real-time and output the corresponding letter in both text and audio format.

Code for Training American Sign Language (ASL)

```
import cv2,os
data_path='DATASET'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]
label_dict=dict(zip(categories,labels)) #empty dictionary
print(label_dict)
print(categories)
print(labels)
data_path='DATASET/train'
classes_path=os.listdir(data_path)
classesf=os.listdir(data_path)
print(classesf)
labels_classes=[i for i in range(len(classesf))]
print(labels_classes)
data_path='DATASET'
label_classes_dict=dict(zip(classesf,labels_classes))
print(label_classes_dict)
import numpy as np img_size=128 data=[] target=[] c=0 minValue = 70 for
category in categories:
    cat_path=os.path.join(data_path,category)
    print(cat_path)
    cat_names=os.listdir(cat_path)
    print(cat_names)
    for classes in cat_names:
        folder_path=os.path.join(data_path,category,classes)
        print(folder_path)
```

```
    img_names=os.listdir(folder_path)
    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)
        try:
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray,(5,5),2)
            th3 =cv2.adaptiveThreshold
            (blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)
            ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.
            THRESH_OTSU)
            #Converting the image into gray scale
            resized=cv2.resize(res,(img_size,img_size))
            #resizing the gray scale into 50x50, since we need a fixed common size for all
             #the images in the dataset
            data.append(resized)
            target.append(label_classes_dict[classes])
        except Exception as e:
            print('Exception:',e)

datanp=np.array(data)
datanp.shape
targetnp=np.array(target)
targetnp.shape
data=np.array(data)/255.0
data=np.reshape(data,(data.shape[0],img_size,img_size,1))
target=np.array(target)
from keras.utils import np_utils
new_target=np_utils.to_categorical(target)
new_target.shape
np.save('data_img',data)
np.save('target',new_target)
data=np.load('data_img.npy')
target=np.load('target.npy')
```

```
from sklearn.model_selection import train_test_split
train_data,test_data,train_target,test_target=train_test_split(data,new_target,test_size=0.2)
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense , Dropout
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
sz = 128
# Step 1 - Building the CNN
# Initializing the CNN
classifier = Sequential()
# First convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 1), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Flattening the layers
classifier.add(Flatten())
# Adding a fully connected layer
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dropout(0.40))
classifier.add(Dense(units=96, activation='relu'))
classifier.add(Dropout(0.40))
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dense(units=27, activation='softmax')) # softmax for more than 2
# Compiling the CNN
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# categorical_crossentropy for more than 2
# Step 2 - Preparing the train/test data and training the model
```

```
classifier.summary()
from keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint('model-{epoch:03d}.model',monitor='val_loss',verbose=0,
save_best_only=True,mode='auto')history=classifier.fit(train_data,train_target,shuffle=True,
epochs=20,callbacks=[checkpoint],validation_split=0.3)
print(classifier.evaluate(test_data,test_target))
import matplotlib.pyplot as plt
N = 20
H=history
plt.style.use("ggplot")
print("[INFO] saving mask detector model...") # serialize the model to disk
classifier.save('asl_classifier.h5')
print("Done !")
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epochs')
plt.ylabel('Loss')
plt.legend(['train_loss','val_loss'], loc=0)
plt.show()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epochs')
plt.ylabel('Accuracy')
plt.legend(['train_accuracy','val_accuracy'], loc=0)
plt.show()
```

Code for Realtime American Sign Language

```
import cv2
from keras.models import load_model
from tensorflow.keras.utils import img_to_array
import numpy as np
import tensorflow as tf
import keras
model = keras.models.load_model("asl_classifier.h5")
labels_dict = {0:'0',
               1:'A',
               2:'B',
               3:'C',
               4:'D',
               5:'E',
               6:'F',
               7:'G',
               8:'H',
               9:'I',
               10:'J',
               11:'K',
               12:'L',
               13:'M',
               14:'N',
               15:'O',
               16:'P',
               17:"Q",
               18:'R',
               19:'S',
               20:'T',
               21:'U',
               22:'V',
               23:'W',
               24:'X',
               25:'Y',
               26:'Z'}
color_dict=(0,255,0)
x=0
y=0
w=64
h=64
img_size=128
minValue = 70
source=cv2.VideoCapture(0)
count = 0
string = " "
prev = " "
prev_val = 0
```

```
while(True):
    ret,img=source.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    cv2.rectangle(img,(24,24),(250 , 250),color_dict,2)
    crop_img=gray[24:250,24:250]
    count = count + 1
    if(count % 100 == 0):
        prev_val = count
    cv2.putText(img, str(prev_val//100), (300,
150),cv2.FONT_HERSHEY_SIMPLEX,1.5,(255,255,255),2)
    blur = cv2.GaussianBlur(crop_img,(5,5),2)
    th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRES
H_BINARY_INV,11,2)
    ret, res = cv2.threshold(th3, minValue, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    resized=cv2.resize(res,(img_size,img_size))
    normalized=resized/255.0
    reshaped=np.reshape(normalized,(1,img_size,img_size,1))
    result = model.predict(reshaped)
    label=np.argmax(result,axis=1)[0]
    if(count == 200):
        count = 0
        prev= labels_dict[label]
        if(label == 0):
              string = string + " "

        else:
              string = string + prev

    cv2.putText(img, prev, (24,
14),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)
    cv2.putText(img, string, (275,
50),cv2.FONT_HERSHEY_SIMPLEX,0.8,(200,200,200),2)
    cv2.imshow("Gray",res)
    cv2.imshow('Live',img)
    key=cv2.waitKey(1)


    if(key==27):#press Esc. to exit
        break
print(string)
cv2.destroyAllWindows()
source.release()
cv2.destroyAllWindows()
```

```
from gtts import gTTS
# This module is imported so that we can  play the converted audio
import os
# The text that you want to convert to audio Language in which you want to convert
language = 'en'
# Passing the text and language to the engine, here we have marked slow=False.
# Which tells the module that the converted audio should have a high speed
myobj = gTTS(text=string, lang=language, slow=False)

# Saving the converted audio in a mp3 file named sign
myobj.save("sign.mp3")

# Playing the converted file
os.system("sign.mp3")

from playsound import playsound
playsound('sign.mp3')
```

CHAPTER 4: RESULTS AND DISCUSSIONS

4.1 Results and Discussions

In this project, we have used the American Sign Language (ASL) data set that is provided by beingaryan/Sign-To-Speech-Conversion. This dataset contains total of 17113 images of which 12845 are training images and 4368 are test images all with a shape of 128 x 128 pixels. These images belong to the 27 classes of English alphabet starting from A to Z including a null character. The model has been trained on a python-based environment on the IDLE platform. The model was iterated for a total epoch of 20. We used the training dataset to train the model to detect the signs and the test dataset to test the model that is pre-trained with a full dataset. It was noted that the training time was 135 mins and the model had attained an accuracy of 99.91%. We reduced the training dataset to 8100 images and 4368 images in the test dataset and found that the accuracy was 99.80% with training time reduced to 70 mins. We further reduced 2700 images in the test dataset and kept the training dataset the same and found that the accuracy was 99.72% with training time reduced to 60 mins. We further reduced the training dataset to 2700 images and 4368 images in the test dataset and obtained an accuracy of 99.58% to a reduced training time of 30 mins. At last, we reduced the train dataset and test dataset to 1350 images. The accuracy was reduced to 98.30% with training time further reduced to 10 mins. From table 4.1, we can deduce that the training time can be reduced by reducing the number of images in the dataset at the cost of losing accuracy. We got the following accuracy/loss training curve plot. Figure 4.1 and Figure 4.2 shows the Loss plot and Accuracy plot of the model throughout its training journey respectively for the dataset highligthed in blue in table 4.1.

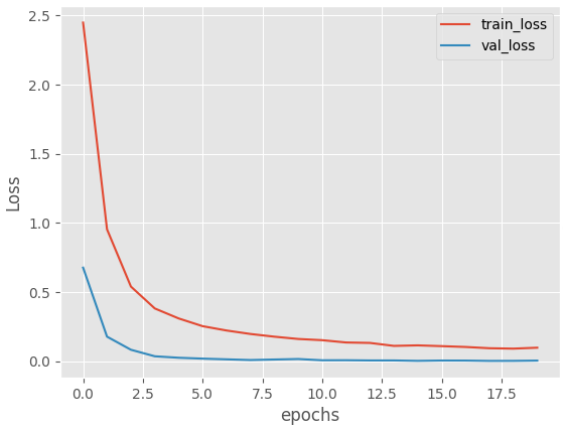


Figure 4.1: Loss v/s Epochs.

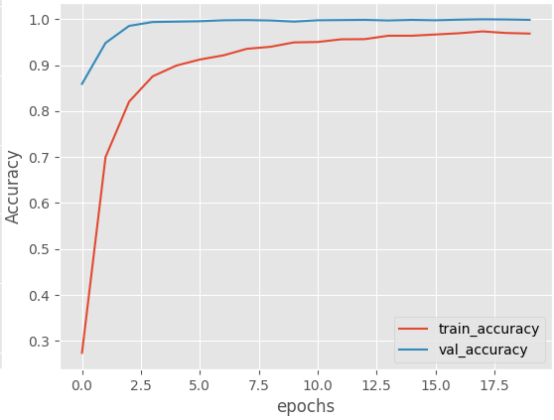


Figure 4.2: Accuracy v/s Epochs.

Table 4.1: Training Performance for different sizes of Datasets consisting of images

Sr. No	**Train dataset** (images)	**Test Dataset** (images)	**Total Dataset** (images)	**Validation Accuracy**	**Training Time (mins)**
1	456 *27=12845	155*27 =4368	17113	99.91	135
2	300*27 =8100	155*27 =4368	12468	99.80	70
3	**300*27 =8100**	**100*27 =2700**	**10800**	**99.72**	**60**
4	200*27 =5400	155*27 =4185	9585	99.70	45
5	100*27=2700	155*27=4185	6885	99.58	30
6	100*27 =2700	100*27 =2700	5400	99.20	20
7	100*27 =2700	50*27 =1350	4050	99.01	14
8	50*27 =1350	50*27 =1350	2700	98.30	10

. We obtained real-time sign language predictions as shown in figure 4.3. Sign languages are very broad and differ from country to country in terms of gestures, body language and face expressions. The grammars and structure of a sentence also varies a lot. In our study, learning and capturing the gestures was quite a challenge for us since the movement of hands had to be precise and on point. Some gestures are difficult to reproduce. And it was hard to keep our hands in exact same position when testing it in real time.

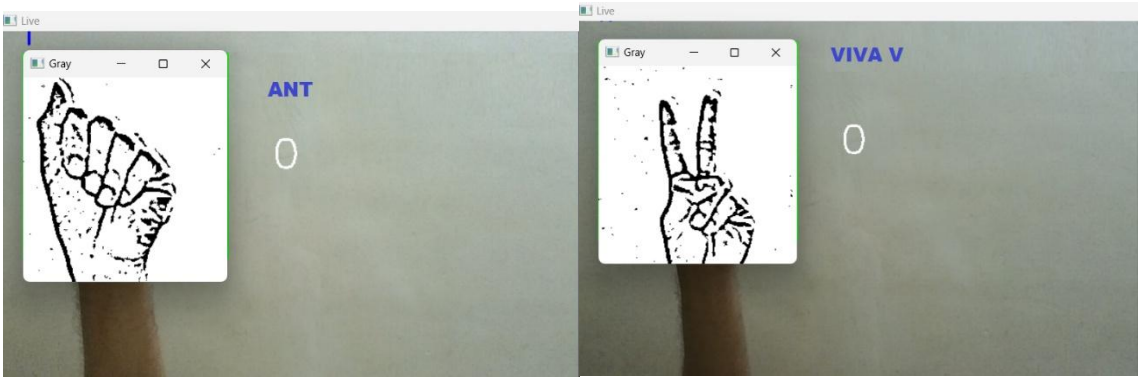


Figure 4.3: Real-time Prediction of Sign Language.

4.2 Applications

1) **Communication:** Sign language recognition can help bridge the communication gap between hearing-impaired individuals and those who do not know sign language. Machine learning can be used to recognize and translate sign language into spoken language or text, making it easier for people to communicate.
2) **Education:** Sign language recognition can be used in schools and universities to make education more accessible for hearing-impaired students. It can be used to automatically transcribe lectures or provide real-time translations of spoken language into sign language.
3) **Accessibility:** Sign language recognition can make digital content more accessible for hearing-impaired individuals. For example, it can be used to automatically generate captions for videos or to provide sign language translations for websites and online resources.
4) **Healthcare:** Sign language recognition can be used in healthcare settings to improve communication between healthcare providers and hearing-impaired patients. It can help ensure that patients receive accurate and appropriate medical care, and can help reduce the risk of medical errors.
5) **Employment:** Sign language recognition can help hearing-impaired individuals access job opportunities that may otherwise be inaccessible. For example, it can be used to provide real-time sign language translations during job interviews or to make training materials more accessible.
6) **Customer service:** Sign language recognition can be used in customer service settings to improve the experience for hearing-impaired customers. For example, it can be used to provide real-time sign language translations during phone or video calls with customer service representatives.
7) **Emergency services:** Sign language recognition can be used by emergency services such as police, fire, and ambulance to communicate with hearing-impaired individuals during emergencies. It can help ensure that critical information is conveyed accurately and quickly, potentially saving lives.
8) **Social media:** Sign language recognition can be used to make social media platforms more inclusive for hearing-impaired individuals. For example, it can be used to automatically generate sign language translations for live streams or video content shared on social media.

9) **Gaming:** Sign language recognition can be used to make video games more accessible for hearing-impaired players. It can be used to provide in-game translations of spoken language into sign language or to allow players to communicate with each other using sign language.
10) **Research:** Sign language recognition can be used in research settings to collect data from hearing-impaired participants. For example, it can be used to automatically transcribe sign language during interviews or to provide real-time translations of spoken language into sign language during experiments

Overall, sign language recognition using machine learning has the potential to improve the lives of hearing-impaired individuals and to promote greater inclusivity and accessibility in various domains.

4.2 Conclusion

The Sign Language Recognition project using image processing was an innovative technology that helped to bridge the communication gap between hearing and deaf individuals. Through the use of advanced image processing algorithms, this project aimed to accurately recognize and translate sign language gestures into text and speech. The development of this project required a significant amount of research and expertise in the areas of computer vision, machine learning, and signal processing. The use of deep learning techniques, such as convolutional neural networks, proved to be effective in improving the accuracy and reliability of sign language recognition. Our model gave 99.72% accuracy for Sign Language Detection after training for 60 minutes with 8100 images in the training dataset and 2700 images in the test dataset.

The Sign Language Recognition project using image processing was a powerful and meaningful application of technology that had the potential to improve the lives of many individuals. The development of this project requires a multidisciplinary approach and a commitment to accessibility and inclusivity. With further research and development, this project has the potential to transform the way we communicate and interact with each other.

4.4 Scope for Future Development

The Sign Language Recognition project using image processing has the potential for a wide range of future applications and improvements. Here are some of the potential future scopes for the project:

(1) **Expansion to different sign languages:** The project could be expanded to recognize different sign languages from around the world, thus increasing its accessibility to a wider range of users.

(2) **Integration with mobile devices:** The project could be integrated into mobile devices such as smartphones and tablets, making it more accessible and convenient for users.

(3) **Improved accuracy:** The project could be improved to achieve even higher accuracy rates by exploring newer and more advanced image processing algorithms and machine learning techniques.

(4) **Gesture detection in noisy environments:** The project could be enhanced to recognize sign language gestures even in noisy or cluttered environments, such as in crowded public spaces.

(5) **Gesture recognition in video streams:** The project could be extended to recognize sign language gestures from video streams, such as video calls or live streams, which would enable real-time communication between hearing and deaf individuals.

(6) **Gesture database creation:** The project could be used to create a database of sign language gestures, which would enable researchers and developers to build more advanced applications and services based on sign language recognition.

In summary, the future scope for the Sign Language Recognition project using image processing is vast and exciting. With continued research and development, the project has the potential to transform the way we communicate and interact with individuals who use sign language as their primary means of communication.

CHAPTER 5: REFERENCES

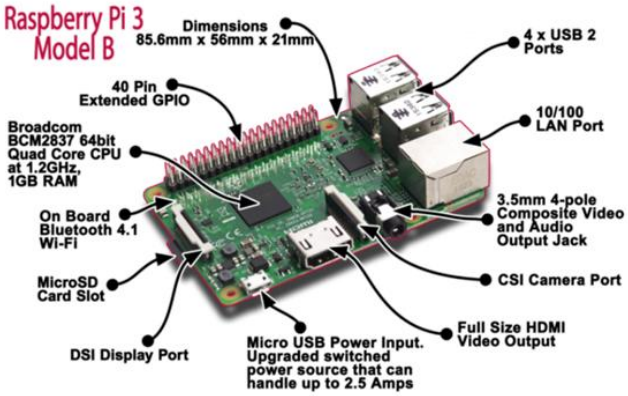
5.1 Bibliography

1. Ankit Ojha, Ayush Pandey, Shubham Maurya, Abhishek Thakur, Dr. Dayananda, (2020), Sign Language to Text and Speech Translation in Real Time Using Convolutional Neural Network, International Journal of Engineering Research & Technology, pp.191-196.
2. Sawant Pramada, Deshpande Saylee, Nale Pranita, Nerkar Samiksha, Mrs.Archana S. Vaidya, (2013), Intelligent Sign Language Recognition Using Image Processing, IOSR Journal of Engineering, pp. 45-51.
3. Mahesh Kumar N B, (2018), Conversion of Sign Language into Text, International Journal of Applied Engineering Research, Volume 13, Number 9, pp. 7154-7161.
4. Rajaganapathy. S, Aravind. B, Keerthana. B, Sivagami. M., (2015), Conversation of Sign Language to Speech with Human Gestures, 2nd International Symposium on Big Data and Cloud Computing, pp.10-15.
5. Amrita Thakur, Pujan Budhathoki, Sarmila Upreti, Shirish Shrestha, Subarna Shakya, (2020), Real Time Sign Language Recognition and Speech Generation, Journal of Innovative Image Processing, pp. 65-76.
6. Shreyas Rajan, Rahul Nagarajan, Akash Kumar Sahoo, M. Gowtham Sethupati, (2019), Interpretation and Translation of American Sign Language for Hearing Impaired Individuals using Image Processing, International Journal of Recent Technology and Engineering, pp. 415-420.
7. Omkar Vedak, Prasad Zavre, Abhijeet Todkar, Manoj Patil, (2019), Sign Language Interpreter using Image Processing and Machine Learning, pp.1907-1909.
8. Mangesh B, Mayur K, Rujali P, (2020), Sign Language Text to Speech Converter using Image Processing and CNN, pp. 609-613.
9. Chhaya Narvekar, Sayukta Mungekar, Arpita Pandey, (2020), Sign Language to Speech Conversion Using Image Processing and Machine Learning, pp. 7229-7223.

5.2 Websites

1. CNN, https://in.mathworks.com/discovery/convolutional-neural-network-matlab.html
2. Basics of CNN in Deep Learning, https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/
3. Sign Language Recognition Using Python and OpenCV, https://data-flair.training/blogs/sign-language-recognition-python-ml-opencv/
4. Sign-To-Speech-Conversion, https://github.com/beingaryan/Sign-To-Speech-Conversion#-analysis

5. 3 Appendix A: Raspberry Pi 3



Technical Specifications of Raspberry Pi 3:

Raspberry Pi 3 is the third generation Raspberry Pi model. Its features are as follows:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- Video Core IV 3D graphics core, 400MHz
- 802.11n Wireless LAN & Bluetooth 4.1
- 1GB RAM LPDDR2(900MHz)
- 4 USB ports & Ethernet port
- 40 GPIO Pins
- Full HDMI port & Combined 3.5mm audio jack and composite video
- Camera interface (CSI)&Display interface (DSI)
- Micro SD card slot

Functions of the Raspberry Pi 3:

- **ARM CPU/GPU**: This is a Broadcom BCM2837 System on Chip (SoC) that's made up of an ARM central processing unit (CPU) and Video Core IV graphics processing unit (GPU). The CPU handles all the computations that make a computer work (taking input, doing calculations, and producing output), and the GPU handles graphics output.
- **GPIO**: These are general-purpose input/output connection points that will allow real hardware hobbyists the opportunity to tinker.
- **Audio out:** This is a standard 3.55-millimeter jack for connection of audio output devices such as headphones or speakers.
- **LEDs**: In light-emitting diodes, the indicator lights according to Raspberry Pi's functions.

- **CSI (Webcam Serial Interface):** This will allow connection to add-on webcam modules (when available). The GPU is potentially able to handle up to 40Mp stills and 1080p 30fps (frames per second) video capture.
- **DSI(Display Serial Interface):** This will support direct connection to add-on screens
- **USB**: This is a common connection port for peripheral devices of all types (including your mouse and keyboard).
- **HDMI:** This connector allows you to hook up a high-definition television or other compatible device using an HDMI cable.
- **Power**: This is a 5V Micro USB power connector into which you can plug your compatible power supply.
- **SD card slot**: This is a full-sized SD card slot. A SD card with an operating system (OS) installed is required for booting the device.
- **Ethernet**: This connector allows for wired network access.
- **Wireless radio:** The Broadcom BCM43438 chip provides 2.4GHz 802.11n wireless LAN, Bluetooth Low Energy, and Bluetooth 4.1 Classic radio support. Cleverly built directly onto the board to keep costs down, rather than the more common fully qualified module approach, its only unused feature is a disconnected FM radio receiver.
- **Antenna:** There's no need to connect an external antenna to the Raspberry Pi 3. Its radios are connected to chips antenna soldered directly to the board, to keep the size of the device to a minimum. Despite its diminutive stature, this antenna should be more than capable of Picking up wireless LAN and Bluetooth signals – even through walls.

5.4 Appendix B: Sign Language for Alphabets