CHAPTER 1
INTRODUCTION

1.1 GENERAL BACKGROUND

With the rapid development of science and technology, the way of human-computer interaction has also been greatly changed. Gestures are the most common in all kinds of body language, so it can be used as a simple and free means of human-computer interaction [6]. It has a very broad application prospect. Each individual utilizes language to communicate with others. The listening to disabled individuals likewise utilizes language to communicate among themselves.

Sign language is basically utilized by hearing impaired populace to communicate with others. Communication through signing is a very organized nonverbal language using both non-manual and manual correspondence. Non manual signals are essentially outward appearance, head movement, stance and orientation of the body. While manual signals incorporate movement and orientation of hand that passes on typical significance. On the other hand, communication with typical individuals is a major impediment for them since not every ordinary person comprehends their gesture-based communication. To beat this issue, a sign language recognition system is expected to assist the deaf and mute people to communicate with normal people. An important process for gesture based human computer interaction is to recognize gestures.

Gestures are expressive, meaningful body motions involving physical movements of the fingers, hands, arms, head, face, or body. When performing gesture recognition [4] the features of the gesture are first extracted, and then gesture recognition is performed according to the extracted features. There are many common gesture recognition methods. For example, the neural network-based recognition method has a strong ability to classify and identify.[8] However, if the number of neural network layers is generally shallow, it is easy to overfitting; the recognition method based on geometric features performs gesture recognition by extracting gesture structure, edge, contour and other features, it has good stability, but can’t improve the recognition rate by increasing the sample size. The recognition method based on the Hidden Markov Model has the ability to describe the time and space changes of gestures, but the recognition speed of the method is not satisfactory.

Some of the major problems faced by a person who are unable to speak is they cannot express their emotion as freely in this world. Utilizing voice recognition and voice search systems in

smartphone we can aim to resolve this issue. They are not able to utilize (Artificial Intelligence/personal Butler) like google assistance, or Apple's SIRI etc., because all those apps are based on voice controlling.

There is a need for such platforms for such kind of people. American Sign Language (ASL) is a complete, complex language that employs signs made by moving the hands combined with facial expressions and postures of the body. It is the go-to language of many North Americans who are not able to talk and is one of various communication alternatives used by people who are deaf or hard-of-hearing.

While sign language is very essential for deaf-mute people, to communicate both with normal people and with themselves, is still getting less attention from the normal people. The importance of sign language has been tending to ignored, unless there are areas of concern with individuals who are deaf-mute. One of the solutions to talk with the deaf-mute people is by using the mechanisms of sign language.

Hand gesture is one of the methods used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or talking disorders to communicate among themselves or with normal people. Various sign language systems have been developed by many manufacturers around the world but they are neither flexible nor cost-effective for the end users.

With the rapid development of machine learning and deep learning in computer vision, methods based on machine learning and deep learning have attracted more and more researchers' attention. Among them, the deep neural network has the characteristics of local connection, weight sharing, automatic feature extraction, etc., which brings new ideas to the task of gesture recognition. Therefore, based on the complexity of gesture changes, we have implemented a gesture recognition method based on deep Convolutional Neural Network (CNN) [5]. This aim behind this project was to design a system that enables the deaf and mute populace to communicate with others of society.

CHAPTER 2
LITERATURE SURVEY

2.1 THEORETICAL INVESTIGATIONS

Sign Language is the means of communication among the deaf and mute community. Sign Language emerges and evolves naturally within the hearing-impaired community. Sign Language communication involves manual and non-manual signals where manual signs involve fingers, hands, arms and non-manual signs involve face, head, eyes and body. Sign Language is a well-structured language with a phonology, morphology, syntax and grammar. Sign language is a complete natural language that uses different ways of expression for communication in everyday life. Sign Language recognition system transfers the communication from human-human to human-computer interaction.

The aim of the sign language recognition system is to present an efficient and accurate mechanism to transcribe text or speech, thus the "dialog communication" between the deaf and hearing person will be smooth. There is no standardized sign language for all deaf people across the world. However, sign languages are not universal, as with spoken languages, these differ from region to region. A person who can talk and hear properly (normal person) cannot communicate with deaf & dumb person unless he/she is familiar with sign language. Same case is applicable when a deaf & dumb person wants to communicate with a normal person or blind person. So, there are two main approaches used in the sign language recognition that is Sensor based and Vision based Approach.

2.1.1 Sensor Based Approach

This approach collects the data of gestures performed by using different sensors. The data is then analysed and conclusions are drawn in accordance with the recognition model [8]. In case of hand gesture recognition different types of sensors were used and placed on hand, when the hand performs any gesture, the data is recorded and is then further analysed.

The first sensor used was Data gloves then LED's came into existence. The invention of the first data glove was done in 1977. Sensor based approach damages the natural motion of hand because of use of external hardware. The major disadvantage is complex gestures cannot be performed using this method.

2.1.2 Vision Based Approach

This approach takes images from the camera as data of gesture. The vision-based method mainly concentrates on capturing images of gestures and extracting the main feature and recognizing it. The colour bands were used Gesture recognition was first proposed by Myron W. Krueger as a new form of interaction between human and computer in the middle of seventies [2]. It has become a very important research area with the rapid development of computer hardware and vision systems in recent years. Hand gesture recognition is a user- friendly and intuitive means for humans to interact with computers or intelligent machines (e.g. robot, vehicle etc.). The research has gained more and Gesture recognition was first proposed by Myron W. Krueger as a new form of interaction between human and computer in the middle of seventies [1]. The research has gained more and more attention in the field of human machine interaction. Currently, there are several available techniques that are applicable for hand gesture recognition, which are either based on sensing devices or computer vision. A typical widespread device-based example is a data glove, which was developed by Zimmerman in 1987 [3]. In this system, the user wears a data glove that is linked to the computer. The glove can measure the bending of fingers, the position and orientation of the hand in 3-D space. Data glove is able to capture the richness of a hand's gesture. Its successful example is real-time American Sign Language Recognition [4]. However, this approach does not meet the actual requirements of human-vehicle interaction under an outdoor environment. It is cumbersome to apply it to vehicles under public transportation context.

The vision-based gesture recognition has the advantages of being spontaneous, having no specialized hardware requirement and being hand size independence. In vision-based gesture recognition, hand shape segmentation is one of the toughest problems under

Gesture recognition was first proposed by Myron W. Krueger as a new form of interaction between human and computer in the middle of the seventies. It has become a very important research area with the rapid development of computer hardware and vision systems in recent years. Hand gesture recognition is a user- friendly and intuitive means for humans to interact with computers or intelligent machines (e.g. robot, vehicle etc.). The research has gained more and more attention in the field of human machine interaction. Currently, there are several available techniques that are applicable for hand gesture recognition, which are either based on sensing devices or computer vision. A typical widespread device-based example is a data glove, which was developed by Zimmerman in 1987. In this system, the user wears a data glove that is

linked to the computer. The glove can measure the bending of fingers, the position and orientation of the hand in 3-D space. Data glove is able to capture the richness of a hand's gesture. Its successful example is real-time American Sign Language Recognition.

In vision-based gesture recognition, hand shape segmentation is one of the toughest problems under a dynamic environment [8]. It can be simplified by using visual markings on the hands. Some researchers have implemented sign language and pointing gesture recognition based on different marking modes. For fingerspelling recognition, most proposed methods rely on instrumented gloves, due to the hard problem of discriminating complex hand configurations with vision-based methods. Lamar and Bhuiyant achieved letter recognition rates ranging from 70% to 93%, using coloured gloves and neural networks.

More recently, Rebollar. used a more sophisticated glove to classify 21 out of 26 letters with 100% accuracy. The worst case, letter 'U', achieved 78% accuracy. Shadows, the main cue used in our work, have already been exploited for gesture recognition and interactive applications. Segen and Kumar describe a system which uses shadow information to track the user's hand in 3D. They demonstrated applications in object manipulation and computer games. Leibe presented the concept of a perceptive workbench, where shadows are exploited to estimate 3D hand position and pointing direction. Their method used infrared lighting and was demonstrated in augmented reality gaming and terrain navigation applications.

CHAPTER 3
PROBLEM DEFINITION

Despite the fact that the deaf people can communicate without issues amongst themselves, there is a serious challenge for a hearing-impaired person trying to communicate with normal people. This is because not every single typical person can comprehend their gesture-based communication. The greater part of ordinary individuals knows not about sign language. As communication is imperative, this issue inevitably makes a limitation for the impaired individuals to correspond with the normal ones. Therefore, a sign language translator must be developed to tackle this issue. At present, there are numerous techniques for capturing and recognizing the hand gestures. However, every technique has their own favourable circumstances and inconveniences. In order to develop an efficient gesture recognizing software, the best method for capturing the hand posture needs to be contemplated and chosen.

The main objective of this project is to design a system that can assists the impaired people to communicate with normal people. Given a hand gesture, implementing such an application which detects predefined American Sign Language (ASL) in a real time through hand gestures and providing facility for the user to be able to store the result of the character detected in a text file, also allowing such users to build their customized gesture. So, that the problems faced by persons who aren’t able to talk vocally can be accommodated with technological assistance and the barrier of expressing can be overshadowed.

CHAPTER 4
EXISTING SYSTEM

To be able to recognize the signs, a set of measurable features of the body that make difference between signs is needed. The body characteristics that make the difference between the signs are the shape of the hand, the angle from each joint of the fingers and wrist, or arm position and trajectory [1]. To implement such a system, researches have been conducted in two main directions:

1. Systems that use specialized hardware devices for data acquisition: robotic glove to measure finger and hand joint angles, and various mechanical, optical, magnetic and acoustic devices to detect hand position and trajectory;
2. Systems that use image processing and computer vision techniques to detect the characteristics of the hand in images taken with a video or web camera.

If using imaging systems for detecting body characteristics, data pre-processing is required to obtain numerical features describing the characteristics of the body. This pre-processing step is quite difficult, so the best results were obtained using specialized hardware for data acquisition.

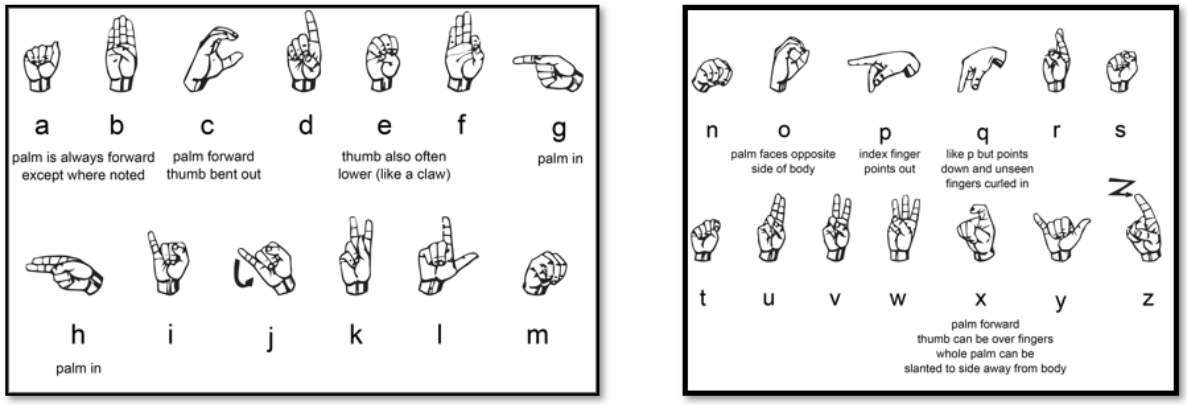


Figure: 4.1 American Sign Language [2][3]

Several existing sign language recognition systems are [1]:

1. Slarti : for data acquisition it uses a robotic glove and a system based on magnetic fields. For classifying data, a set of neural networks is used, each one trained to classify the sign according to a set of features.

2. Glove-Talk: for data acquisition it uses a robotic glove and for classification uses multiple artificial neural networks.
3. Talking Glove: for data acquisition uses a robotic glove and for classification uses a neural network. This system achieved only finger spelling recognition.

4. University of Central Florida gesture recognition system: uses a webcam and computer vision techniques to collect the data and a neural network to classify shapes. For recognition, this system requires wearing a specially coloured glove to facilitate the imaging process.
5. ASLR: uses a webcam and computer vision techniques to collect field data and for classification of signs uses Bayesian learning.

4.1 DATA-GLOVE APPROACH

The data-glove approach utilizes a unique assembled electronic glove, which has fabricated sensors that are utilized to distinguish the hand stance. Most commercial sign language translation systems use the data-glove method, as it is simple to acquire data on the bending of fingers and 3D orientation of the hand using gloves. The framework requires less computational force, and continuous interpretation is much simpler to accomplish. The data glove is outlined with ten flex sensors, two on every finger .The flex sensors work as variable resistance sensors that change resistance as indicated by the sensor's flexing[8] .These sensors can recognize the bending point of every joint of the fingers and send the information to the microcontroller. It is mounted in the outer layer of the data glove, from the association joints of fingers and palm to fingertips. Furthermore, to expand the exactness in recognizing the hand pose, a 3-axis accelerometer is utilized to identify the change of acceleration of the hand's movement in distinctive bearings. The accelerometer is attached on the back of the data glove. The data glove is exceptionally suitable in perceiving both fingerspelling and sign motions, which include static and movement signs. However, these data gloves can be expensive. While it is conceivable to create less expensive data gloves, they are much more vulnerable to noise. If the amount of the sensors used is reduced, it will bring about loss of essential data about the hand stance. This will result in the loss of exactness in sign interpretation. The data glove also can be less comfortable to be worn by the signer.

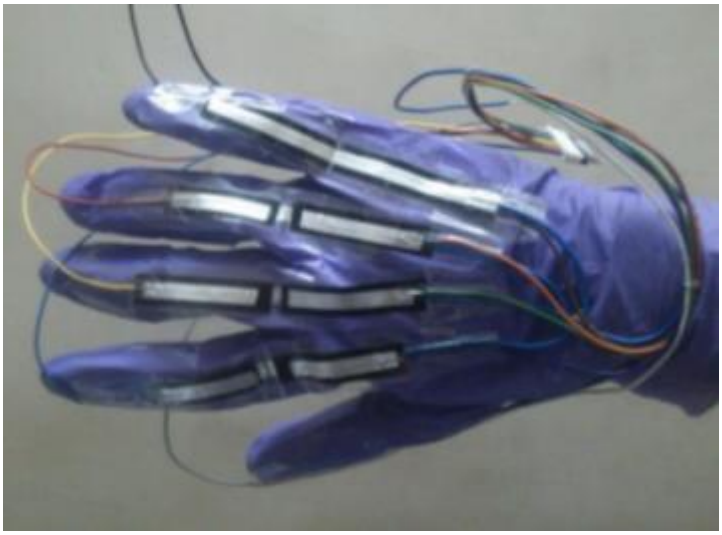


Figure: 4.2 Data Glove with Flex Sensors [4][15]

4.2 VISUAL-BASED APPROACH

With late progression in PC and data innovation, there has been an expanded regard for visual-based methodology. Images of the signer are captured by a camera and video processing is done to perform acknowledgment of the sign language. In Contrast with the data glove approach, the fundamental advantage of visual-based methodology is the adaptability of the framework. The recognition of facial expression and head movements additionally can be incorporated to the framework and perform lip-perusing. This system can be separated into two strategies, which are utilization of hand-crafted shading gloves and in light of skin-colour recognition. For the specially crafted glove, the signer is furnished with colour-coded gloves. The colour will give the extraction of information from the images of the signer through colour segmentation. These gloves are essentially a normal pair of gloves with particular shading on every fingertip and palm. Somehow, these gloves are less expensive as opposed to electronic data gloves.

This system uses insignificant equipment by utilizing just essential webcam and basic gloves. Webcam is used to acquire images from the signer in type of still images and video streams in RGB (red-green-blue) shading.

Figure: 4.3 Colour-coded Gloves [4][16]

For the recognition based on skin-colour, the framework requires just a camera to catch the pictures of the signer for the normal collaboration in the middle of human and computer and no additional gadgets are needed.

It turns out to be more common and helpful for constant applications. This system utilizes an uncovered hand to concentrate information required for recognition, and it is simple, and the user directly communicates with the system. In order to track the position of hand, the skin colour region will be fragmented utilizing colour threshold technique, then the region of interest can be determined. The image acquisition runs constantly until the signer demonstrates a stop sign. After the threshold, the segmented images are then analysed to obtain the unique features of each sign.

These visual-based approaches are fundamentally minimizing the equipment necessities and cost. However, these systems are just suitable and viable for deciphering alphabets and numbers, as opposed to perceiving sign gestures. Signs with comparable stance to another sign can be confused, along these lines diminishing the precision of the system. Moreover, the image acquisition process is subjected to numerous ecological concerns, for example, position of the camera, background condition and lighting affectability. The diverse tallness of the signer likewise must be considered. Adequate lighting additionally required to have enough brightness to be seen and analysed.

CHAPTER 5
PROPOSED SYSTEM

The purpose of this project is to implement a system that recognizes a component of a sign language, namely: finger-spelling in English. Here This project is introducing a system that can recognize American Sign Language using a Convolutional Neural Network which will help mute people to communicate. The images are processed to extract the characteristics necessary for recognition, which are then used as inputs for an artificial neural network that will recognize the sign. The project steps are described in the Figure below.

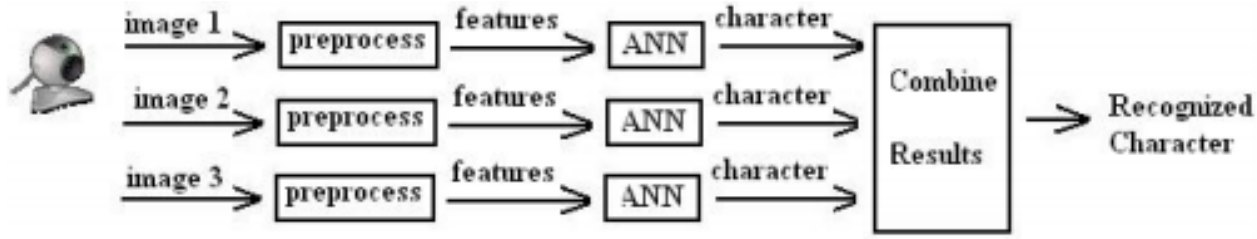


Figure: 5.1 Proposed Model

To overcome the errors that may occur because of noise from the images, Usage of three consecutive images taken from the webcam to recognize a single character. Each image is individually processed and recognized, and then the results are combined using a heuristic method to obtain the final result.

Features:

- Real time (ASL) detection based on gestures made by the user.
- Customized gesture generation.
- TTS assistance mechanisms concerning the illiterate people.

As shown in above figure, after taking the image from web camera, three steps are required for recognition of one sign:

5.1 PRE-PROCESSING:

Uses digital image processing techniques to extract important features from the image that will be used for recognition, after the initial image processing, a set of numerical features that uniquely describe a sign. To implement this detecting the hand shape from the image is sufficient for the recognition of a sign. For this reason, the problem becomes a matter of recognizing objects in an image based on their shape.

The pre-processing is done according to the following diagram:

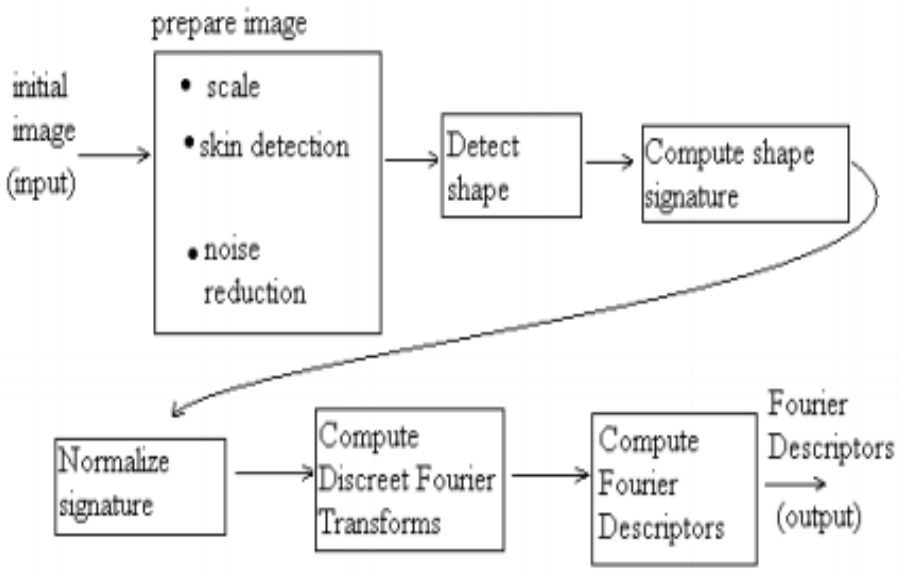


Figure: 5.2 Pre-processing Stage [14][15]

5.2 IMAGE PREPARATION:

For this step it is proposed to use the following sub steps:

- Image scaling is done to reduce the computational effort needed for image processing.
- Skin detection processing involves classification of each pixel of the image as part of a human skin or not. A pixel is categorized as belonging to the skin if the colour components (R, G, and B) meet set relations.
- Noise reduction is meant to overcome these errors, colouring the pixels according to the colours of neighbouring pixels.

5.3 SHAPE DETECTION:

After image processing, the outline of a hand is detected, as a vector of (x, y) coordinates, being the coordinates of the pixels that define the contour of the hand.

5.4 SHAPE SIGNATURE:

Shape signature is a one-dimensional vector characterizing the outline of a two-dimensional shape.

5.5 SIGN RECOGNITION:

The data obtained at the first step are used as inputs for a neural network that recognizes the sign.

5.6 RESULTS COMBINATION:

The results obtained from these three images are then combined using a heuristic to produce the final result.

The basic Modules in proposed model:

1. **Scanne**r: The Module is used to capture real time images from the camera, OpenCV is used for capturing the images

2. **Data Pre-Processing:** The Module that adjusts the parameters of the captured image to get a clear image. Here, the captured image is converted to HSV file format. HSV is an alternative representation of RGB, HSV values are adjusted to get a sharper image due to environment factors affecting clarity.

3. **Gesture Processing**: This module acts as the classifier, From the pre-processed image the gestures are identified by the NN. The proposed system uses a CNN with 3 convolution layers.

4. **Custom Gesture Generation**: This Module allows users to create custom gestures.

5. **Output Prediction:** The final text output is predicted, Evaluation Results 90% of accuracy with 9% loss. This module returns an alphabet corresponding to the sign.

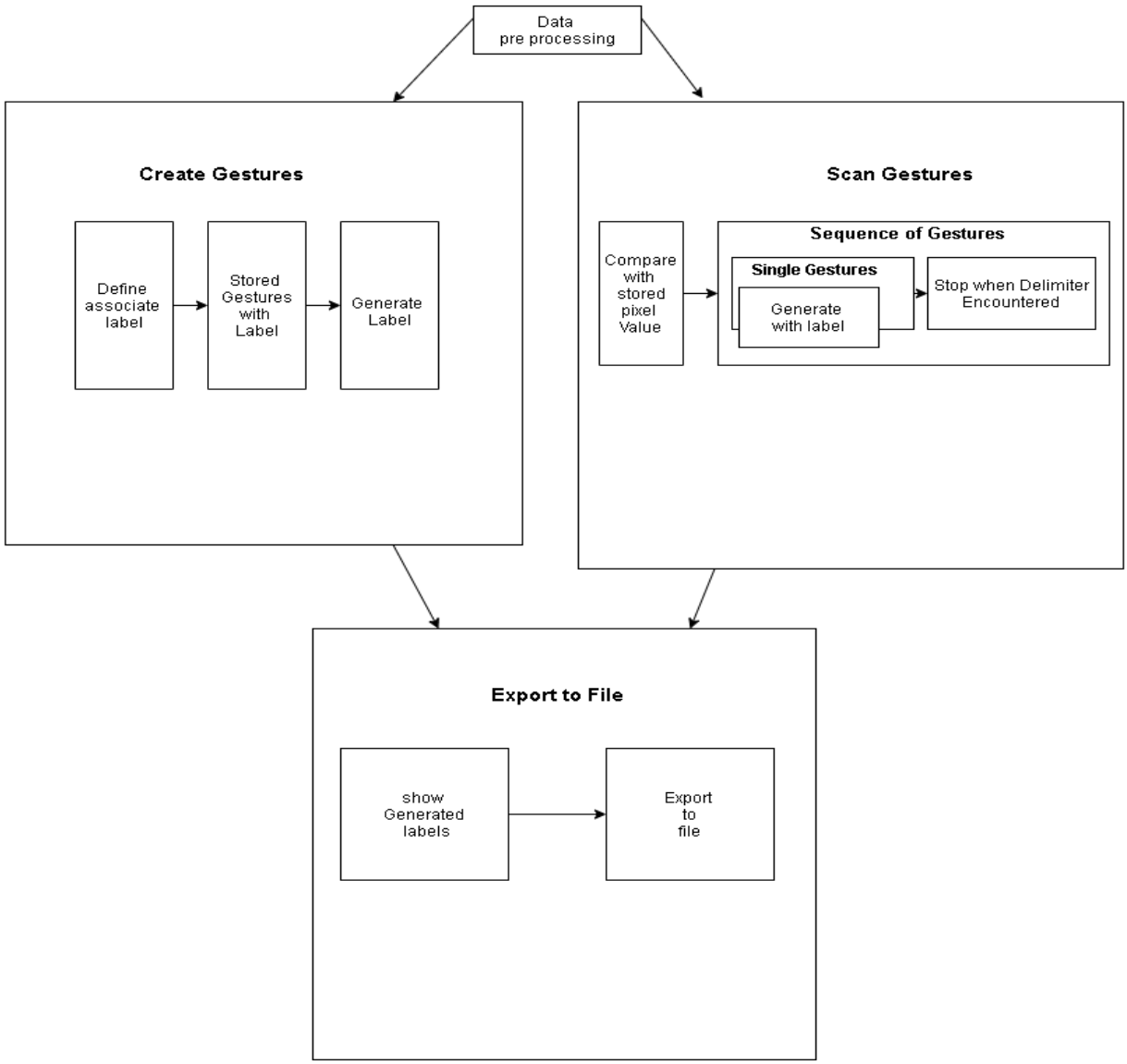


Figure: 5.3 Modules in Proposed System

The scanner module captures real images using OpenCV for capturing the images and pre-processing it into HSV format. This is done to get sharper and clearer images for gesture recognition that follows.

In Gesture Processing the image is processed using a previously trained model that employs Convolutional Neural Network to process the gesture captured. The Convolutional Neural Network used in the model has the parameters as shown:

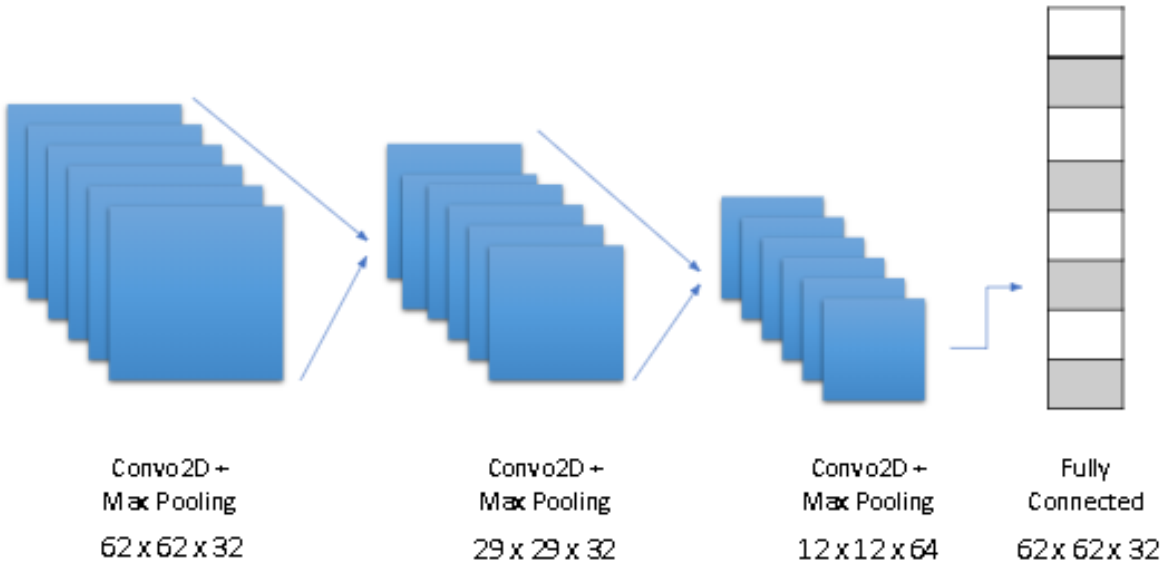


Figure: 5.4 Convolutional Neural Network Model

The final output per gesture will be a corresponding letter which is then formed with other letters to form a word, which is then displayed together.

In this system initial Training of the model is done with cnn_model.py where a Classifier model of the file format h5 is created in correspondence to the input ASL data set. Testing is carried out using the file test.py where the trained model is assessed for its accuracy and consistency. The Program is initiated with recognise.py which uses the pre trained model as its classifier to process the captured images and to output the resulting letter as response. capture.py is used for custom gesture inclusion into the system, through which expressions like emoticons and such can be effectively incorporated to the system.

The initial GUI Prototype for the system is designed as given in the figure below:

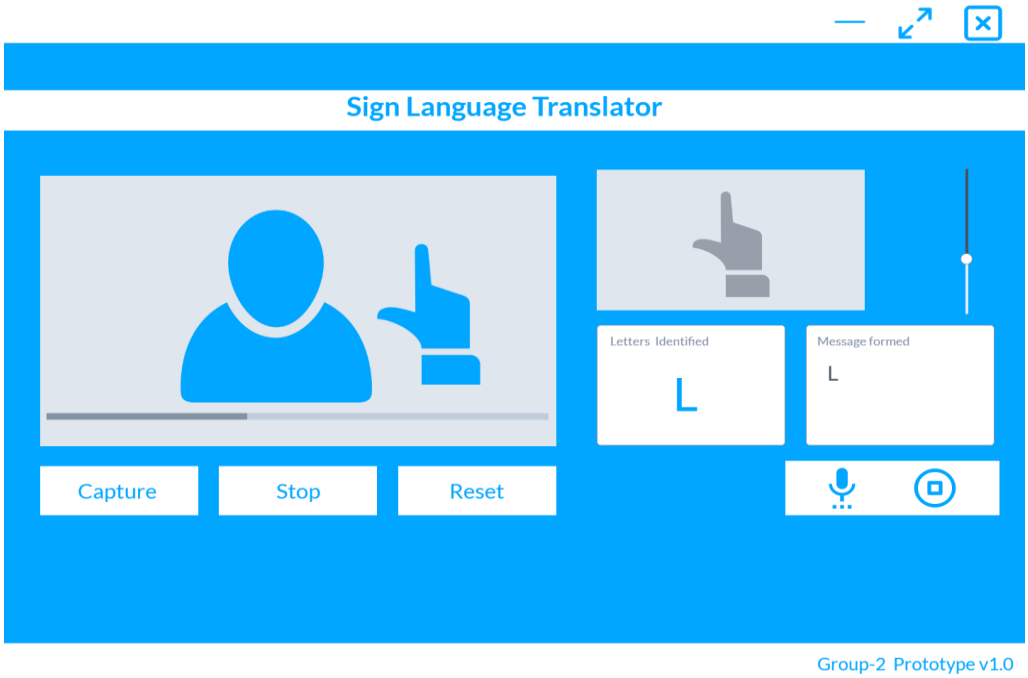


Figure: 5.5 GUI Prototype Design

In the initial GUI Prototype, all of the various functionalities were made available on the same screen. Here the live feed from the camera is taken and projected on the GUI feed window, where a box shaped parameter will be displayed where the user should do their gestures. This done for the ease of processing the feed from the user effectively and efficiently. The selected feed from user will be displayed over a second window in HSV format for the ease of gesture recognition, the values of the feed can be adjusted using the slider to get more clearer and sharper HSV images. This is done as various environments will have varying disturbances which will affect the image captured. If a gesture is successfully recognized then its corresponding letter will be shown in the letter window, with continuous gestures when words are formed it will display the word. The letter window will only show the currently processed gestures, no preceding gestures will be shown together with it. There are options given for stopping , capturing and creating features given via buttons below the live feed window. To the right of the screen there are options for text to voice translation for better and effective communication from the user. There is also an option for pausing the voice translation through a button next to it. Through text to speech translation the user can have a more fulfilling communication with another person.

CHAPTER 6
SYSTEM REQUIREMENTS

6.1 SOFTWARE REQUIREMENTS

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

The necessary components for the system are:

- Python 3.6.
- TensorFlow framework, Keras API
- Real-time computer vision using OpenCV
- Industrial standard GUI application (PyQT5), Tkinter
- Offline TTS assistance for python

6.1.1 Python 3.6

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License.

Figure 6.1: Python Logo [11]

Python 3.0 was released in 2008. Although this version is supposed to be backward incompatible, later on many of its important features have been backported to be compatible with version 2.7. This tutorial gives enough understanding on Python 3 version programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactical constructions than other languages.

6.1.2 TensorFlow framework

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.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015

Figure 6.2: TensorFlow Logo [11]

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs. TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

TensorFlow is cross-platform. It runs on nearly everything: GPUs and CPUs—including mobile and embedded platforms—and even tensor processing units (TPUs), which are specialized hardware to do tensor math on.

The TensorFlow distributed execution engine abstracts away the many supported devices and provides a high performance-core implemented in C++ for the TensorFlow platform.

On top of that sit the Python and C++ frontends (with more to come). The Layers API provides a simpler interface for commonly used layers in deep learning models. On top of that sit higher-level APIs, including Keras (more on the Keras.io site) and the Estimator API, which makes training and evaluating distributed models easier.

6.1.2.1 TensorFlow execution model

Machine learning can get complex quickly, and deep learning models can become large. For many model graphs, you need distributed training to be able to iterate within a reasonable time frame. And, you'll typically want the models you develop to deploy to multiple platforms.

With the current version of TensorFlow, you write code to build a computation graph, then execute it. The graph is a data structure that fully describes the computation you want to perform. This has lots of advantages:

- It's portable, as the graph can be executed immediately or saved to use later, and it can run on multiple platforms: CPUs, GPUs, TPUs, mobile, embedded. Also, it can be deployed to production without having to depend on any of the code that built the graph, only the runtime necessary to execute it.
- It's transformable and optimizable, as the graph can be transformed to produce a more optimal version for a given platform. Also, memory or compute optimizations can be performed and trade-offs made between them. This is useful, for example, in supporting faster mobile inference after training on larger machines.
- Support for distributed execution

TensorFlow's high-level APIs, in conjunction with computation graphs, enable a rich and flexible development environment and powerful production capabilities in the same framework.

6.1.3 Keras API

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

Figure 6.3: Keras Logo [11]

“Being able to go from idea to result with the least possible delay is key to doing good research.” is the core concept on its realisation

It is an open-source neural-network library written in Python. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible

Keras is often used where there is a need for a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Keras is compatible with Python version 2.7 to 3.6.

Features

- User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and centre. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

- Modularity. A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.

- Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

- Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

6.1.4 OpenCV

OpenCV is an open source computer vision and machine learning software library. Originally developed by Intel, it was later supported by Willow Garage then Itseez, which was later acquired by Intel.

Figure 6.4: OpenCV Logo [11]

The library is cross-platform and free for use under the open-source BSD license. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.

OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch and Caffe according to a defined list of supported layers. It promotes OpenVisionCapsules, which is a portable format, compatible with all other formats.

The goals behind OpenCV was described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available Open Source with a license that did not require code to be open or free itself.

6.1.5 PyQt

PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt is free software developed by the British firm Riverbank Computing. It is available under similar terms to Qt versions older than 4.5; this means a variety of licenses including GNU General Public License and commercial license, but not the GNU Lesser General Public License. PyQt supports Microsoft Windows as well as various flavours of UNIX, including Linux and MacOS.
PyQt implements around 440 classes and over 6,000 functions and methods including:

- a substantial set of GUI widgets
- classes for accessing SQL databases (ODBC, MySQL, PostgreSQL, Oracle, SQLite)
- QScintilla, Scintilla-based rich text editor widget
- data aware widgets that are automatically populated from a database
- an XML parser
- SVG support
- classes for embedding ActiveX controls on Windows (only in commercial version)

To automatically generate these bindings, Phil Thompson developed the tool SIP, which is also used in other projects.

In August 2009, Nokia, the then owners of the Qt toolkit, released PySide, providing similar functionality, but under the LGPL, after failing to reach an agreement with Riverbank Computing to change its licensing terms to include LGPL as an alternative license.

PyQt4 contains the following Python modules:

- The QtCore module contains the core non-GUI classes, including the event loop and Qt's signal and slot mechanism. It also includes platform independent abstractions for Unicode, threads, mapped files, shared memory, regular expressions, and user and application settings.

- The QtGui module contains the majority of the GUI classes. These include a number of table, tree and list classes based on the model–view–controller design pattern. Also provided is a sophisticated 2D canvas widget capable of storing thousands of items including ordinary widgets.

- The QtNetwork module contains classes for writing UDP and TCP clients and servers. It includes classes that implement FTP and HTTP clients and support DNS lookups. Network events are integrated with the event loop making it very easy to develop networked applications.
- The QtOpenGL module contains classes that enable the use of OpenGL in rendering 3D graphics in PyQt applications.

- The QtSql module contains classes that integrate with open-source and proprietary SQL databases. It includes editable data models for database tables that can be used with GUI classes. It also includes an implementation of SQLite.

- The QtSvg module contains classes for displaying the contents of SVG files. It supports the static features of SVG 1.2 Tiny.

- The QtXml module implements SAX and DOM interfaces to Qt's XML parser.

- The QtMultimedia module implements low-level multimedia functionality. Application developers would normally use the phonon module.

- The QtDesigner module contains classes that allow Qt Designer to be extended using PyQt.

- The Qt module consolidates the classes contained in all of the modules described above into a single module. This has the advantage that you don't have to worry about which underlying module contains a particular class. It has the disadvantage that it loads the whole of the Qt framework, thereby increasing the memory footprint of an application. Whether you use this consolidated module, or the individual component modules is down to personal taste.

- The uic module implements support for handling the XML files created by Qt Designer that describe the whole or part of a graphical user interface. It includes classes that load an XML file and render it directly, and classes that generate Python code from an XML file for later execution.

PyQt5 contains the following Python modules:

- QtQml Module
- QtQtuick Module
- QtCore Module
- QtGui Module
- QtPrintSupport Module
- QtWidgets Module
- QGLContext Module
- QGLFormat Module
- QGLWidget Module
- QtWebKit Module
- QtWebKitWidgets Module

6.1.6 Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.

The name Tkinter comes from the Tk interface. Tkinter was written by Fredrik Lundh.
Tkinter is free software released under a Python license.

As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

There are several popular GUI library alternatives available, such as wxPython, PyQt , Pygame, Pyglet, and PyGTK.

Some definitions:

Window: This term has different meanings in different contexts, but in general it refers to a rectangular area somewhere on the user's display screen.

Top Level Window: A window that exists independently on the screen. It will be decorated with the standard frame and controls for the desktop manager. It can be moved around the desktop, and can usually be resized.

Widget: The generic term for any of the building blocks that make up an application in a graphical user interface.

Core widgets: The containers: frame, toplevel, paned window. The buttons: button, radiobutton, checkbutton, menubutton.

The text widgets: label, labelframe, message, text. The entry widgets: scale, scroll, listbox, slider, spinbox, entry (singleline), text (multiline), and canvas (vector and pixel graphics).

There are the extension widgets: tk_optionMenu, tk_dialog, tk_messageBox, tk_getOpenFile, tk_getSaveFile, tk_chooseColor, tk_chooseDirectory.

Python 2.7 and Python 3.1 incorporate the "themed Tk" functionality of Tk 8.5.

This allows Tk widgets to be easily themed to look like the native desktop environment in which the application is running, thereby addressing a long-standing criticism of Tk (and hence of Tkinter).

Frame: In Tkinter, the Frame widget is the basic unit of organization for complex layouts. A frame is a rectangular area that can contain other widgets.

Child and parent: When any widget is created, a parent-child relationship is created. For example, if you place a text label inside a frame, the frame is the parent of the label.

6.1.7 Offline TTS assistance for python

Speech Recognition in Python One can make the computer speak with Python. Given a text string, it will speak the written words in the English language. This process is called Text To Speech.

Text to Speech library for Python 2 and 3. Works without internet connection or delay. Supports multiple TTS engines, including Sapi5, nsss, and espeak. pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline, and is compatible with both Python 2 and 3.

6.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list, especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application.

The following are the various requirements for the project:

- Intel Pentium Dual Core E6600 3.06GHz / AMD Athlon II X2 260
- 512 MB disk space
- 2 GB memory
- AMD Radeon HD 6450 or NVIDIA GeForce GT 520
- USB Keyboard, Mouse & Speaker
- 2MP VGA webcam & Monitor

6.2.1 Intel Pentium Dual Core E6600 3.06GHz / AMD Athlon II X2 260Processor

A central processing unit, also called a central processor or main processor, is the electronic circuitry within a computer that executes instructions that make up a computer program. The CPU performs basic arithmetic, logic, controlling, and input/output operations specified by the instructions. The computer industry used the term "central processing unit" as early as 1955.Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit, distinguishing these core elements of a computer from external components such as main memory and I/O circuitry.

The Core 2 brand refers to Intel's x86/x86-64 microprocessors with the Core microarchitecture targeted at the consumer and business markets above Pentium. The Core 2 solo branch covered single-core CPUs for notebook computers, Core 2 Duo – dual-core CPUs for both desktop and notebook computers, Core 2 Quad – quad-core CPUs for both desktop and notebook computers, and Core 2 Extreme – dual-core and quad-core CPUs for both desktop and notebook computer.

6.2.2 Hard Disk

A hard disk drive, hard disk, hard drive, or fixed disk, is an electro-mechanical data storage device that uses magnetic storage to store and retrieve digital information using one or more rigid rapidly rotating disks coated with magnetic material. The platters are paired with magnetic heads, usually arranged on a moving actuator arm, which read and write data to the platter surfaces. Data is accessed in a random-access manner, meaning that individual blocks of data can be stored or retrieved in any order and not only sequentially. HDDs are a type of non-volatile storage, retaining stored data even when powered off. The primary characteristics of an HDD are its capacity and performance. Capacity is specified in unit prefixes corresponding to powers of 1000: a 1-terabyte drive has a capacity of 1,000 gigabytes . Typically, some of an HDD's capacity is unavailable to the user because it is used by the file system and the computer operating system, and possibly inbuilt redundancy for error correction and recovery. The two most common form factors for modern HDDs are 3.5-inch, for desktop computers, and 2.5-inch, primarily for laptops. HDDs are connected to systems by standard interface cables such as PATA, SATA , USB or SAS cables.

6.2.3 RAM

Random-access memory (RAM) is a form of computer data storage that stores data and machine code currently being used. A random-access memory device allows data items to be read or

written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older magnetic tapes and drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement. RAM is normally associated with volatile types of memory, where stored information is lost if power is removed, although non-volatile RAM has also been developed.25 Other types of non-volatile memories exist that allow random access for read operations, but either do not allow write operations or have other kinds of limitations on them.

6.2.4 Graphics Card

A graphics card is an expansion card which generates a feed of output images to a display device. Frequently, these are advertised as discrete or dedicated graphics cards, emphasizing the distinction between these and integrated graphics. At the core of both is the graphics processing unit , which is the main part that does the actual computations, but should not be confused with the video card as a whole, although "GPU" is often used to refer to video cards.

Most video cards are not limited to simple display output. Their integrated graphics processor can perform additional processing, removing this task from the central processor of the computer. For example, Nvidia and AMD produced cards rendering the graphics pipeline OpenGL and DirectX on the hardware level. In the later 2010s, there has also been a tendency to use the computing capabilities of the graphics processor to solve non-graphic tasks, which can be done through the use of OpenCL and CUDA. Video cards can also be used for AI training.

Usually the graphics card is made in the form of a printed circuit board (expansion board) and inserted into an expansion slot, universal or specialized (AGP, PCI Express). Some have been made using dedicated enclosures, which are connected to the computer via a docking station or a cable. These are known as eGPUs.

CHAPTER 7
IMPLEMENTATION

This section talks about the implementation and working of the system in a real world scenario. The purpose of this project is to implement a system that recognizes a component of a sign language, to understand the gestures of the user and give the corresponding message. Here we are introducing a system that can recognize American Sign Language using a Convolutional Neural Network which will help mute people to communicate. The images are processed to extract the characteristics necessary for recognition, which are then used as inputs for an artificial neural network that will recognize the sign.

The project is implemented in python adopting Convolutional Neural Network using TensorFlow backend.

The program implemented models are:

1. Predictor
2. Single Scan
3. Sentence Scan
4. Export File
5. Create Gesture

7.1 CONVOLUTIONAL NEURAL NETWORK

In deep learning, a Convolutional Neural Network is a class of deep neural networks, most commonly applied to analysing visual imagery.[7] They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics [14]. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

7.1.1 Design

A Convolutional Neural Network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product.[7] The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

Though the layers are colloquially referred to as convolutions, this is only by convention. Mathematically, it is technically a sliding dot product or cross-correlation. This has significance for the indices in the matrix, in that it affects how weight is determined at a specific index point.

7.1.1.1 Convolutional

When programming a CNN, the input is a tensor with shape number of images x image width x image height x image depth. Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape number of images x feature map width x feature map height x feature map channels.
A convolutional layer within a neural network should have the following attributes:

- Convolutional kernels defined by a width and height.
- The number of input channels and output channels.
- The depth of the Convolution filter must be equal to the number channels of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neural processes data only for its receptive field.[7] Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images.

A very high number of neurons would be necessary, even in a shallow architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for an image of size 100 x 100 has 10,000 weights for each neuron in the second layer.

The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.

7.1.1.2 Pooling

Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average.[7] Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

7.1.1.3 Fully connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional MLP. The flattened matrix goes through a fully connected layer to classify the images.

7.1.1.4 Receptive field

In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically, the subarea is of a square shape. The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.

7.1.1.5 Weights

Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias. Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

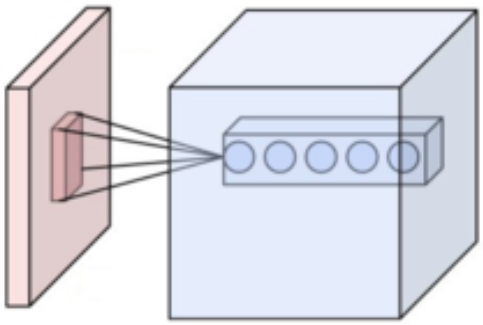
The vector of weights and the bias are called filters and represent particular features of the input. A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting

7.1.2 Building blocks

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume through a differentiable function. A few distinct types of layers are commonly used. These are further discussed below.

7.1.2.1 Convolutional layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters, which have a small[7] receptive field, but extend through the full depth of the input volume.



.

Figure 7.1: Neurons of A Convolutional Layer [8]

During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

7.1.2.2 Local connectivity

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume.

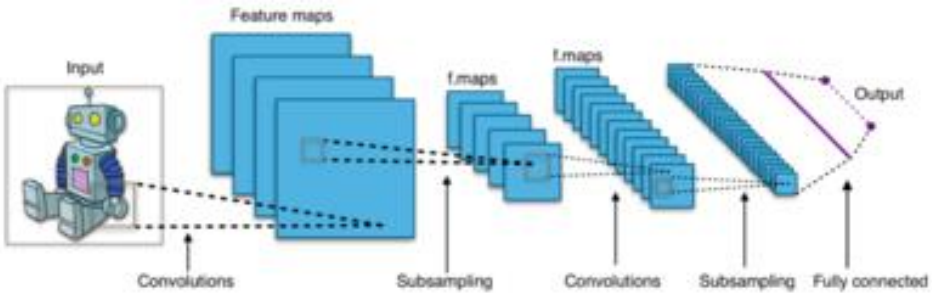


Figure 7.2: Typical CNN Architecture [10][8][7]

The extent of this connectivity is a hyperparameter called the receptive field of the neuron. The connections are local in space along width and height, but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern

7.1.2.3 Spatial arrangement

Three hyperparameters control the size of the output volume of the convolutional layer: the depth, stride and zero-padding.

- The depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input. For example, if the first convolutional layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of colour.

- Stride controls how depth columns around the spatial dimensions are allocated. When the stride is 1 then one can move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. When the stride is 2 then the filters jump 2 pixels at a time as they slide around. Similarly,

for any integer S>0a stride of S causes the filter to be translated by S units at a time per output. In practice, stride lengths of S > 3 are rare. The receptive fields overlap less and the resulting output volume has smaller spatial dimensions when stride length is increased.

- Sometimes it is convenient to pad the input with zeros on the border of the input volume. The size of this padding is a third hyperparameter. Padding provides control of the output volume spatial size. In particular, sometimes it is desirable to exactly preserve the spatial size of the input volume.

7.1.2.4 Parameter sharing

A parameter sharing scheme is used in convolutional layers to control the number of free parameters. It relies on the assumption that if a patch feature is useful to compute at some spatial position, then it should also be useful to compute at other positions. Denoting a single 2-dimensional slice of depth as a depth slice, the neurons in each depth slice are constrained to use the same weights and bias.

Since all neurons in a single depth slice share the same parameters, the forward pass in each depth slice of the convolutional layer can be computed as a convolution of the neuron's weights with the input volume. Therefore, it is common to refer to the sets of weights as a filter, which is convolved with the input. The result of this convolution is an activation map, and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume. Parameter sharing contributes to the translation invariance of the CNN architecture.

7.1.2.5 Pooling layer

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several nonlinear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the

number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.

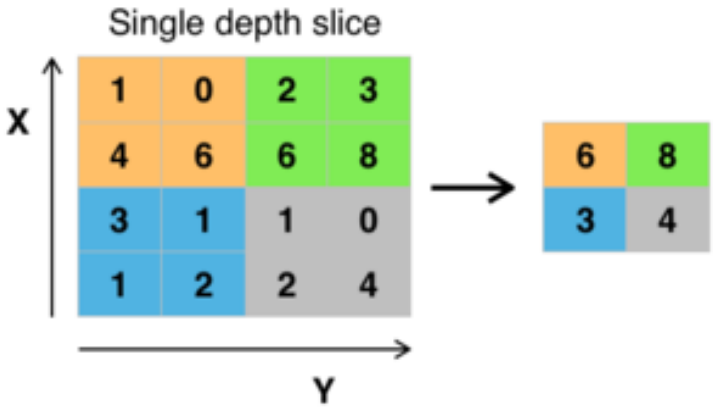


Figure 7.3: Max Pooling with A 2x2 Filter And Stride = 2 [10][7]

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 down sampled at every depth slice in the input by 2 along both width and height, discarding 75% of the activations:

7.1.2.6 ReLU layer

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function f(x) = max(0,x). It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent f(x)=tanh(x) , f(x)=|tanh(x)|, and the sigmoid function . ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

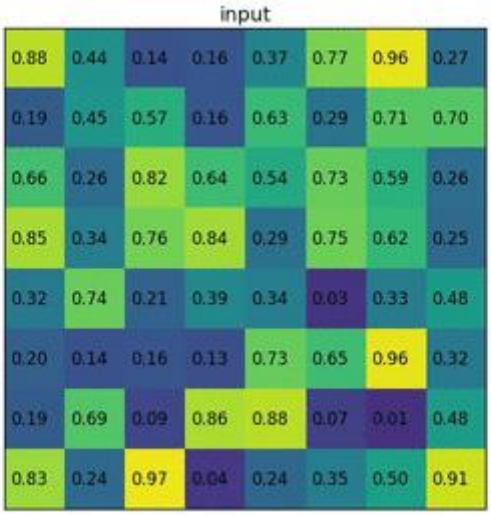


Figure 7.4: Roi Pooling to Size 2x2, above Region Proposal Has Size 7x5 [10][7]

7.1.2.7 Fully connected layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

7.1.2.8 Loss layer

The "loss layer" specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions appropriate for different tasks may be used.

Softmax loss is used for predicting a single class of K mutually exclusive classes. [nb 3] Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1]. Euclidean loss is used for regressing to real-valued labels (-infinity, infinity).

7.2 IMPLEMENTED MODULES

In the project a predesigned CNN model is used in the main program to process the gestures real-time from the user and converts the image captured to specific size mentioned in the code and convert the picture into HSV format. In the basic modules this is covered by scanner and Data Pre-Processing modules, here in the actual program implementation this is also done through them. Sliders are provided in the GUI for adjusting the values between high and low for HSV parameters the image is then captured and stored as “1.png”.

This is done to gain an optimally clear image from the camera input. This image is then passed to the predictor function which then forwards it to the classifier function which converts the image into a 2D array, in this array there are spaces for storing the value of the letter.

For example, suppose a gesture for “A” is given, it will enter value of 1 in the space for ‘A’ and enter value of 0 in all others. Hence there will be value spaces for all letters in ASL.

Predictor uses 2 different algorithms for gesture processing:

1. The traditional 2D array for recognizing the gestures with which the mode was trained.
2. SIFT algorithm for recognizing the custom gestures added to the system.

To create new gestures in this system run the capture program which will re-initiate the gesture recognition process resulting in a HSV image which is saved in a new folder named in accordance to the user. Feedbacks given c for correct predictions.

Sentence formation is formed using multiple letters recognised by the system by temporarily appending the letter to a temporary file.

Export function is used to store the temporary for usage by voice assistance for converting the word to speech. i.e, for making the software speak the gestured word.

Single scan is used for recognizing the image captured frame by frame to return its value, captured images are passed to the predictor after conversion into HSV.

Sentence scan is used when there is the need for continuous prediction of images even in the delay. Usage of Tkinter in GUI is only for gesture creation, for all others PyQt is used. TTS is implemented using Google TTS assistance preinstalled in the system which converts the temporary file content into mp3 audio file which is played with assistance of OS in the browser.

The actual functional modules of the system are:

1. Predictor: processes the gestures and passes it through one of the two compatible algorithms for classification.

2. Single Scan: Image captured is recognized frame by frame to return its value, captured images are passed to the predictor after conversion into HSV.

3. Sentence Scan: used when there is the need for continuous prediction of images even in the delay

4. Export File: used to store the temporary for usage by voice assistance for converting the word to speech.

5. Create Gesture: used for creating and storing custom gestures from the user, initialized by running the capture program which will re-initiate the gesture recognition process resulting in a HSV image which is saved in a new folder named in accordance to the user.

The reason for branching the basic modules into differing function modules were for the ease of access and programming in a more efficient and effective manner. Predictor acts as a classification point for determining the control flow to the next stage of operations which can be from arrange of differing scenarios like single gesturing, continuous gesturing or custom gesture inclusion. In the implementation to better facilitate needs of user the GUI was prepared with an option menu to various other function screen from the current one. That is to say one can change between the function screens a from other screens after entering via Main GUI Screen. The flow of operations from Predictor to single gestures will be result in two possible branches being either single gesture recognition or custom gesture recognition. The user can set custom gestures to convey emotions which is not possible through traditional ASL by a single gesture or be simply gesture a letter in their daily life.

7.2.1 Predictor module code:

```
def predictor():
        import numpy as np
        from keras.preprocessing import image
        test_image = image.load_img('1.png', target_size=(64, 64))
        test_image = image.img_to_array(test_image)
        test_image = np.expand_dims(test_image, axis = 0)
        result = classifier.predict(test_image)                    #using cnn model *array checking
        gesname=' '
        fileEntry=fileSearch()
        for i in range(len(fileEntry)):                #custom gesture is verified using sift algorithm
                image_to_compare = cv2.imread("./SampleGestures/"+fileEntry[i])
                original = cv2.imread("1.png")
                sift = cv2.xfeatures2d.SIFT_create()
                kp_1, desc_1 = sift.detectAndCompute(original, None)
                kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)
                index_params = dict(algorithm=0, trees=5)
                search_params = dict()
                flann = cv2.FlannBasedMatcher(index_params, search_params)
                matches = flann.knnMatch(desc_1, desc_2, k=2)
                good_points = []
                ratio = 0.6
                for m, n in matches:
                    if m.distance < ratio*n.distance:
                        good_points.append(m)
                if(abs(len(good_points)+len(matches))>20):
                        gesname=fileEntry[i]
                        gesname=gesname.replace('.png','')
                        if(gesname=='sp'):                                  #sp is replaced with <space>
                                gesname=' '
                        return gesname
        if result[0][0] == 1:
                return 'A'
        elif result[0][1] == 1:
                return 'B'
        elif result[0][2] == 1:
                return 'C'
        elif result[0][3] == 1:
                return 'D'
        elif result[0][4] == 1:
                return 'E'
        elif result[0][5] == 1:
                return 'F'
        elif result[0][6] == 1:
                return 'G'
        elif result[0][7] == 1:
                return 'H'
        elif result[0][8] == 1:
                return 'I'
```

```
        elif result[0][9] == 1:
                return 'J'
        elif result[0][10] == 1:
                return 'K'
        elif result[0][11] == 1:
                return 'L'
        elif result[0][12] == 1:
                return 'M'
        elif result[0][13] == 1:
                return 'N'
        elif result[0][14] == 1:
                return 'O'
        elif result[0][15] == 1:
                return 'P'
        elif result[0][16] == 1:
                return 'Q'
        elif result[0][17] == 1:
                return 'R'
        elif result[0][18] == 1:
                return 'S'
        elif result[0][19] == 1:
                return 'T'
        elif result[0][20] == 1:
                return 'U'
        elif result[0][21] == 1:
                return 'V'
        elif result[0][22] == 1:
                return 'W'
        elif result[0][23] == 1:
                return 'X'
        elif result[0][24] == 1:
                return 'Y'
        elif result[0][25] == 1:
                return 'Z'
```

7.2.2 Single Scan module code:

```
        def scanSingle(self):                                   #Single gesture scanner
                        img_text = ''
                        while True:
                                ret, frame = self.cam.read()
                                frame = cv2.flip(frame,1)
                                frame=cv2.resize(frame,(321,270))
                                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
img1 = cv2.rectangle(frame, (150,50),(300,200), (0,255,0), thickness=2, lineType=8, shift=0)
                                height1, width1, channel1 = img1.shape
                                step1 = channel1 * width1
                                                        # create QImage from image
```

```
qImg1 = QImage(img1.data, width1, height1, step1, QImage.Format_RGB888)

                                                    # show image in img_label
                        try:
                                self.label_3.setPixmap(QPixmap.fromImage(qImg1))
                                slider1=self.trackbar.value()
                        lower_blue = np.array([0, 0, 0])
                        upper_blue = np.array([179, 255, slider1])
                        imcrop = img1[52:198, 152:298]
                        hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                        mask = cv2.inRange(hsv, lower_blue, upper_blue)
                        cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                        cv2.imshow("mask", mask)

            cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WIND
    OW_FULLSCREEN)
                        cv2.resizeWindow("mask",118,108)
                        cv2.moveWindow("mask", 894,271)
                        try:
                                self.textBrowser.setText("\n\n\t"+str(img_text))
                        img_name = "1.png"
                        save_img = cv2.resize(mask, (image_x, image_y))
                        cv2.imwrite(img_name, save_img)
                        img_text = predictor()
                        if cv2.waitKey(1) == 27:
                           break
                self.cam.release()
                cv2.destroyAllWindows()
```

7.2.3 Sentence Scan module code:

```
    def scanSent(self):                                    #sentence formation module
                img_text = ''
                append_text=''
                new_text=''
                finalBuffer=[]
                counts=0
                while True:
                        ret, frame =self.cam.read()
                        frame = cv2.flip(frame,1)
                        frame=cv2.resize(frame,(331,310))
     frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                        img = cv2.rectangle(frame, (150,50),(300,200), (0,255,0),
    thickness=2, lineType=8, shift=0)
                        height, width, channel = img.shape
                        step = channel * width                 # create QImage from image

qImg = QImage(img.data, width, height, step, QImage.Format_RGB888)
```

```
                                                    # show image in img_label
                        self.label_3.setPixmap(QPixmap.fromImage(qImg))
                        slider=self.trackbar.value()
                        lower_blue = np.array([0, 0, 0])
                        upper_blue = np.array([179, 255, slider])
                        imcrop = img[52:198, 152:298]
                        hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                        mask1 = cv2.inRange(hsv, lower_blue, upper_blue)
                        cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                        cv2.imshow("mask", mask1)
        cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WIND
OW_FULLSCREEN)
                        cv2.resizeWindow("mask",118,108)
                        cv2.moveWindow("mask", 1180,415)
                        try:
                                self.textBrowser.setText("\n        "+str(img_text))
                        img_name = "1.png"
                        save_img = cv2.resize(mask1, (image_x, image_y))
                        cv2.imwrite(img_name, save_img)
                        img_text=predictor()
if cv2.waitKey(1)== ord('c'):
                                        #wait till c is pressed , if  c is pressed letter is appended
                try:
                        counts+=1
                        append_text+=img_text
                        new_text+=img_text
                        if not os.path.exists('./TempGest'):
                            os.mkdir('./TempGest')
img_names = "./TempGest/"+"{}{}.png".format(str(counts),str(img_text))
                        save_imgs = cv2.resize(mask1, (image_x, image_y))
                                                cv2.imwrite(img_names, save_imgs)
                                                self.textBrowser_4.setText(new_text)
                except:
                        append_text+="

                        if(len(append_text)>1):
                                finalBuffer.append(append_text)
                                append_text="
                        else:
                                finalBuffer.append(append_text)
                                append_text="
                try:

        self.pushButton.clicked.connect(lambda:saveBuff(self,self.cam,finalBuffer))
        except:
                        pass
                if cv2.waitKey(1) == 27:
                        break
        if keyboard.is_pressed('shift+s'):
                                                    #string is saved to temp file
```

```
                    if(len(finalBuffer)>=1):
                        f=open("temp.txt","w")
                            for i in finalBuffer:
                        f.write(i)
                        f.close()
                        break
                    self.cam.release()
                    cv2.destroyAllWindows()
```

7.2.4 Export File module code:

```
        def exportFile(self):
                def tts(mytext):                                    #text to speech
                mytext.lower()
                language = 'en'
                myobj = gTTS(text=mytext, lang=language, slow=False)
                myobj.save("text.mp3")
                import webbrowser
                webbrowser.open("text.mp3")


        checkfile=os.path.isfile('temp.txt')
        if(checkfile==True):
            fr=open("temp.txt","r")
            content=fr.read()
            fr.close()
        else:
            content="No Content Available"
        tts(content)
```

7.2.5 Create Gesture module code:

```
    def create_gest():
            img_text = ''
            saveimg=[]
            while True:
                ret, frame = self.cam.read()
                frame = cv2.flip(frame,1)
                frame=cv2.resize(frame,(321,270))
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
img2 = cv2.rectangle(frame, (150,50),(300,200), (0,255,0), thickness=2, lineType=8, shift=0)
                height2, width2, channel2 = img2.shape
                step2 = channel2 * width2
                                                    # create QImage from image

qImg2 = QImage(img2.data, width2, height2, step2, QImage.Format_RGB888)
                                                    # show image in img_label
```

```
                    self.label_3.setPixmap(QPixmap.fromImage(qImg2))
                    slider2=self.trackbar.value()
        lower_blue = np.array([0, 0, 0])
                    upper_blue = np.array([179, 255, slider2])
                    imcrop = img2[52:198, 152:298]
                    hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                    mask = cv2.inRange(hsv, lower_blue, upper_blue)
                    cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                    cv2.imshow("mask", mask)

        cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FU
LLSCREEN)
                    cv2.resizeWindow("mask",170,160)
                    cv2.moveWindow("mask", 766,271)
                    ges_name = input("enter the gesture name")
                    if(len(ges_name)>=1):
                            saveimg.append(ges_name)
                    else:
                            saveimg.append(ges_name)
                            ges_name="
        cam.release()
        cv2.destroyAllWindows()
        if not os.path.exists('./SampleGestures'):
           os.mkdir('./SampleGestures')
        gesname=saveimg[-1]
        if(len(gesname)>=1):
        img_name = "./SampleGestures/"+"{}.png".format(str(gesname))
        save_img = cv2.resize(mask, (image_x, image_y))
        cv2.imwrite(img_name, save_img)
        gesname=saveimg[-1]
                    if keyboard.is_pressed('shift+s'):
                            if not os.path.exists('./SampleGestures'):
                                    os.mkdir('./SampleGestures')
                            if(len(gesname)>=1):
        img_name = "./SampleGestures/"+"{}.png".format(str(gesname))
                                    save_img = cv2.resize(mask, (image_x, image_y))
                                    cv2.imwrite(img_name, save_img)
                            break

                    if cv2.waitKey(1) == 27:
                            break
                self.cam.release()
                cv2.destroyAllWindows()
                if os.path.exists("./SampleGestures/"+str(gesname)+".png"):
                    print("Gesture Saved Successfully!")
```

CHAPTER 8
ADVANTAGES

- Greater processing speeds
- Real-time results
- Lesser resource requirements
- Portability to multiple implementations
- User necessity adaptable
- Lightweight and efficient
- Lesser resource wastage
- Custom gesture inclusion facility
- Eliminates the requirement for a translator

CHAPTER 9
FUTURE SCOPE

In the future scope, the system can be deployed in a smaller scale implementation in embedded systems for the ease of mobility and access for the mute populace. Deeper integration with more components are to be added into the system to improve functionality and user convenience. For example, a feature that allows for chatting through the internet. A more complex and complete algorithm is to be developed for greater efficiency and for better recognition filters to be used for improving the accuracy of gestures scanned. In the future, one can even use gestures to play games, chat and email with others. Although the accuracy obtained by the experiment has been very high, we feel that it is necessary to further improve for the application to real life.

CHAPTER 10
RESULT

This section discusses the results of the project. The pretrained model used in the main program has great accuracy and efficiency. This can be inferred from the accuracy of 97.54% and loss of 7.26%. made clear from the results of trained model and accuracy rate and loss rate graphs.

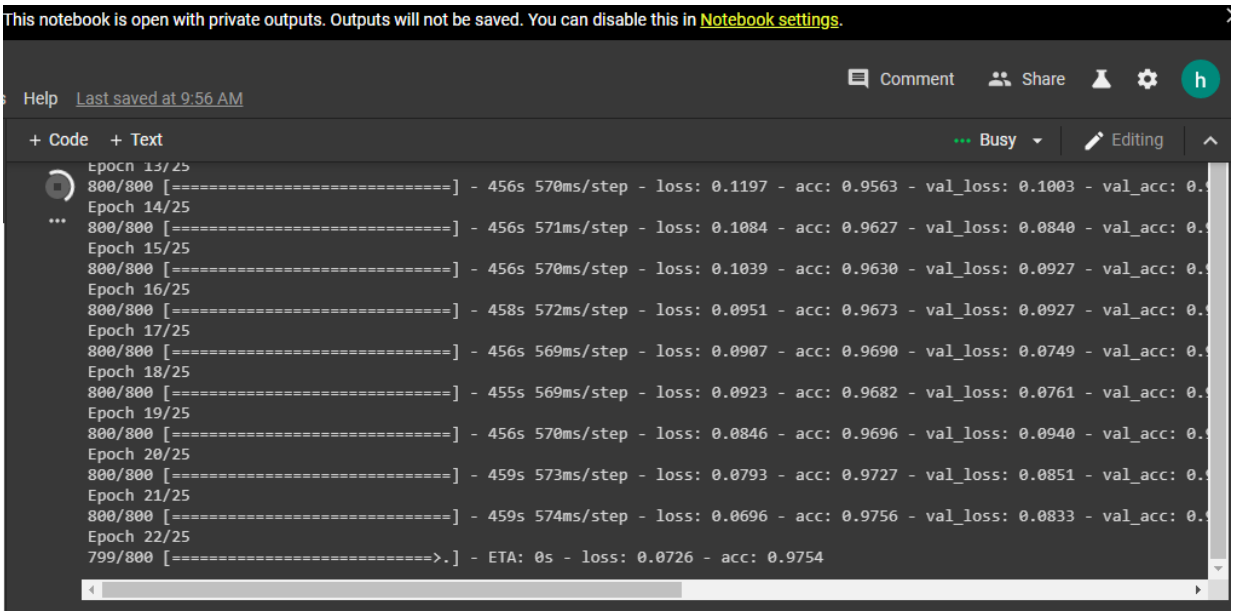


Figure 10.1: Accuracy and Loss Rates of Trained Model

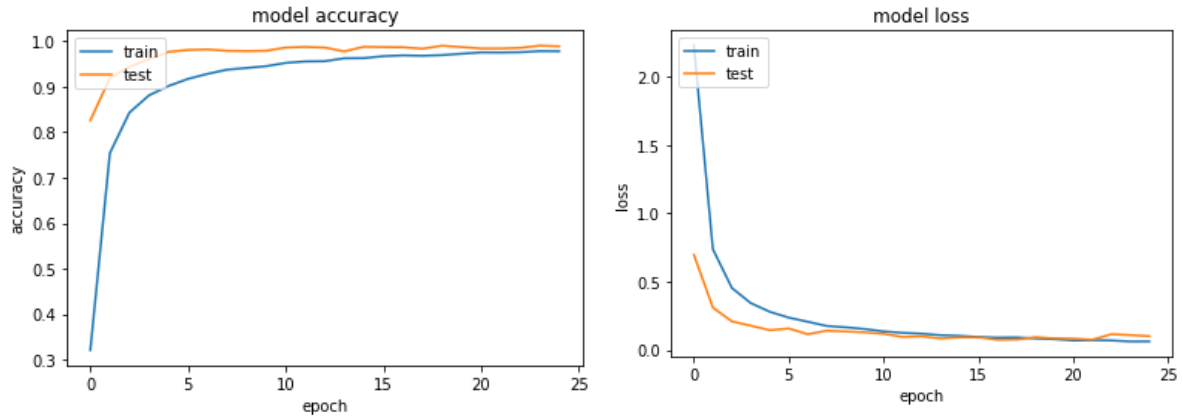


Figure 10.2: Accuracy And Loss Rate Graphs

The interface for the system was designed in PyQt for ease of usage and tkinter for custom gesture creation.

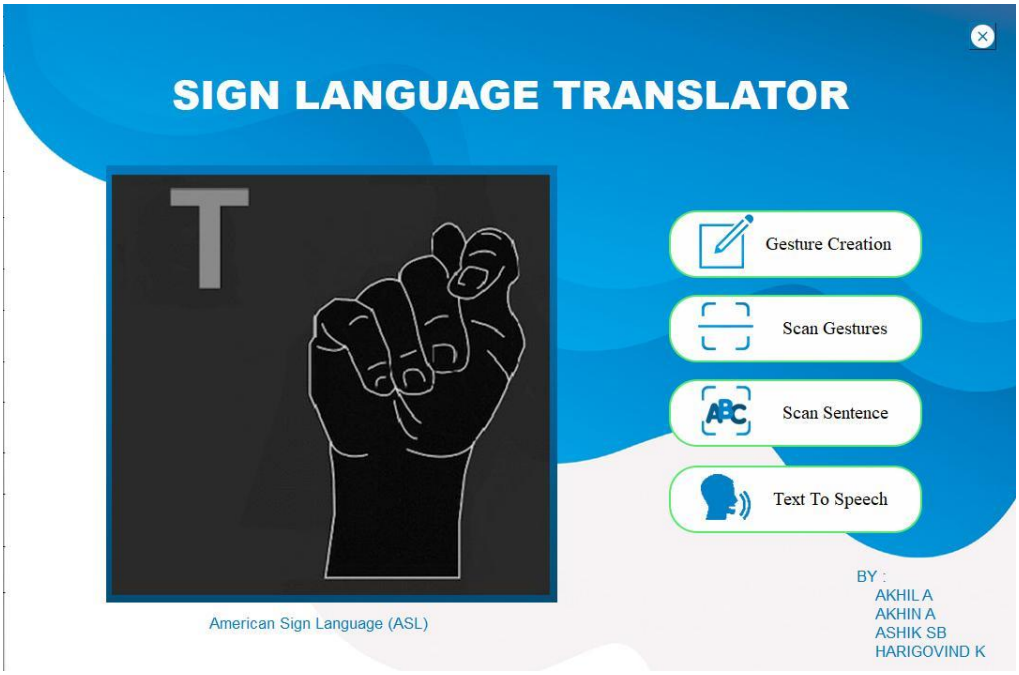


Figure 10.3: GUI Main Screen

This is the welcome screen of the GUI on which four buttons and a camera interface are given. The Four options include gesture creation which is used for creating new gestures into the system, scan gesture is to scan and identify the gesture, scan sentence button is used for constructing sentences from gestures, and the last button converts the sentence of gestures from text to speech. In all other screens links to other screens are provided for the ease of access and usability.

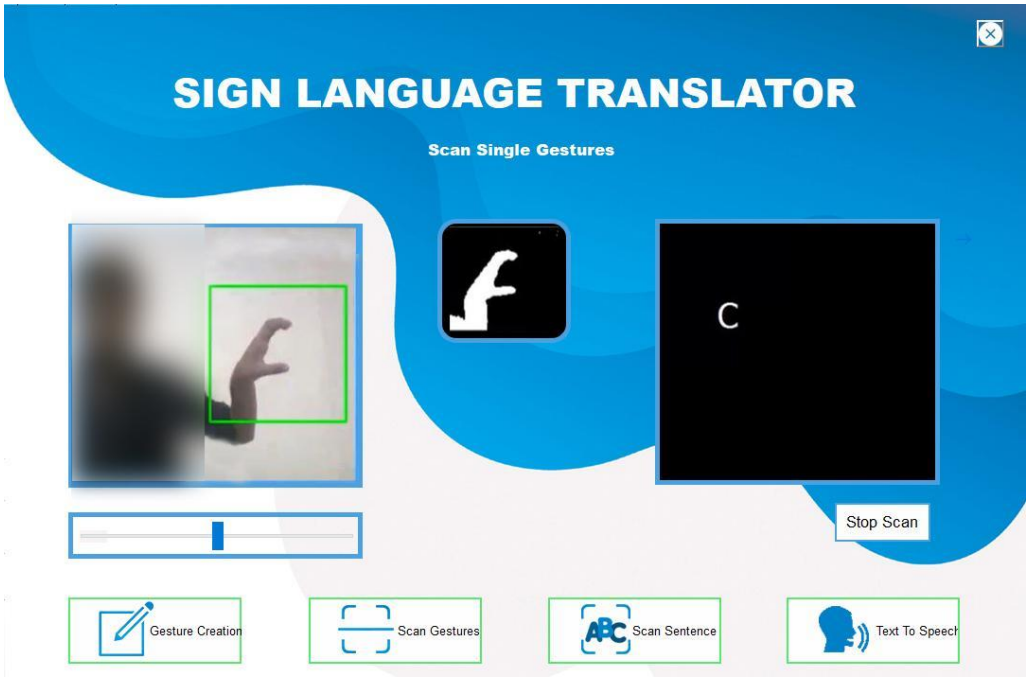


Figure 10.4: Scan Single Gesture Screen

This screen is the scan single gestures which is used for the scanning of independent gestures and letters. There is a slider provided for adjusting the HSV values and a button for stopping the scan.

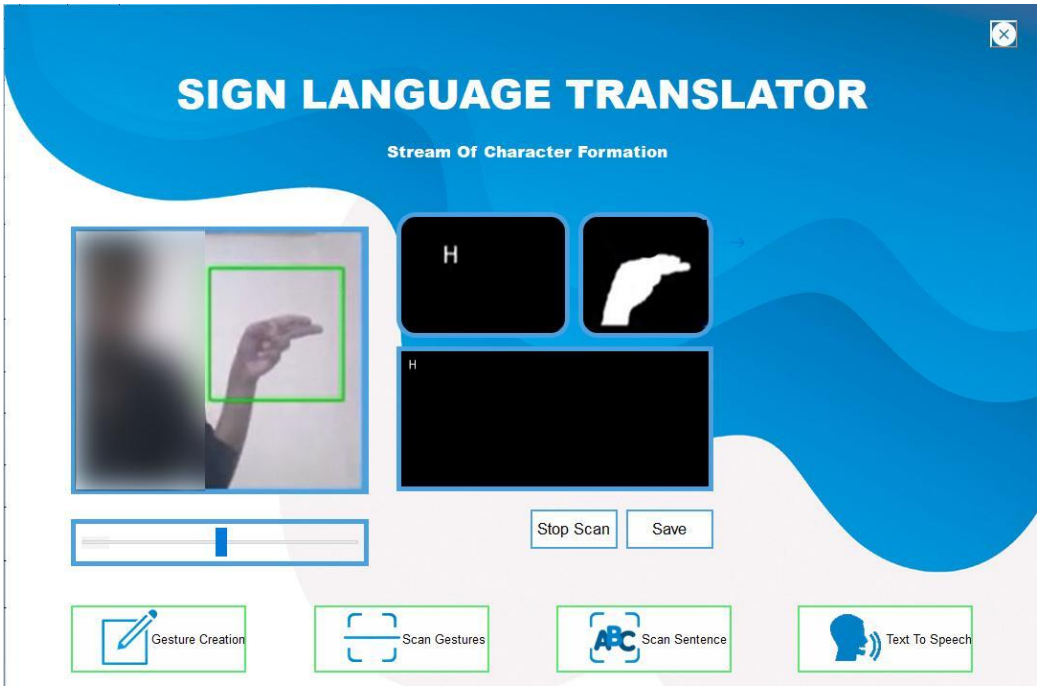


Figure 10.5: Stream of Character Formation Screen

This is the character steam formation screen where the streams or sentences are analysed from continuous gestures are inputted. Sliders are provided for adjusting HSV values and buttons are provided for saving and stopping the scan.

Figure 10.6: Conversion Screen

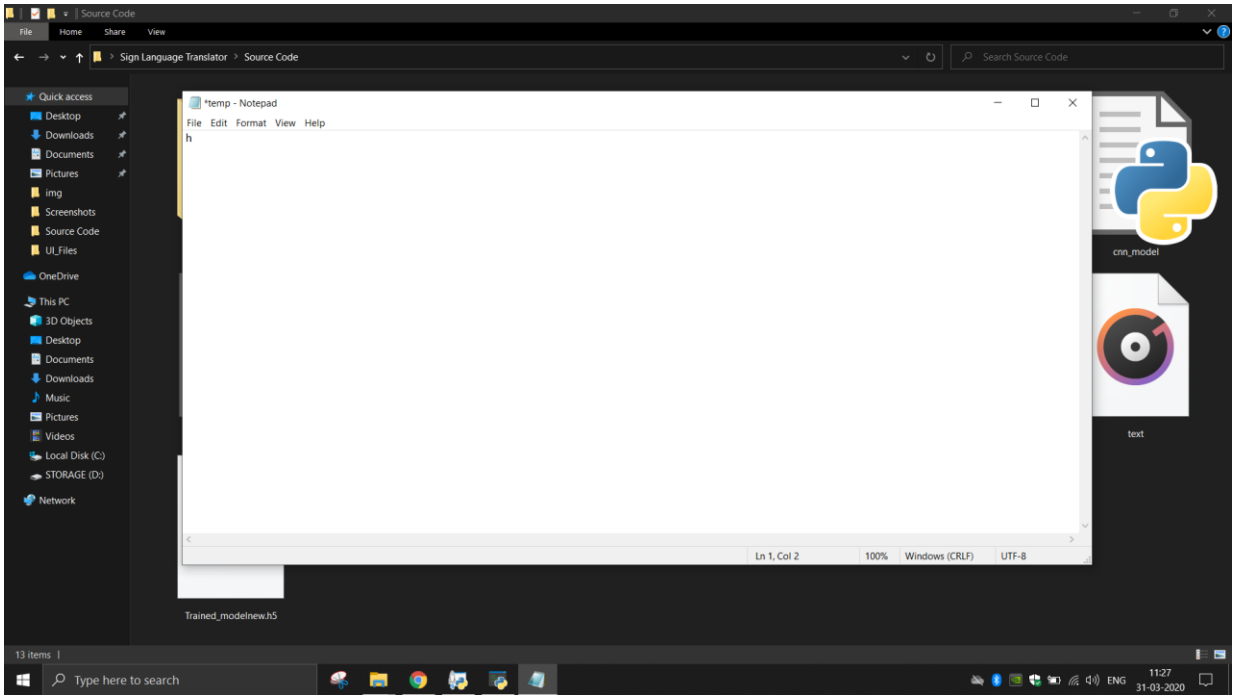


Figure 10.7: Exported file after conversion

This is the conversion screen , used for converting the sentences formed from gesture recognition into speech use google assistance

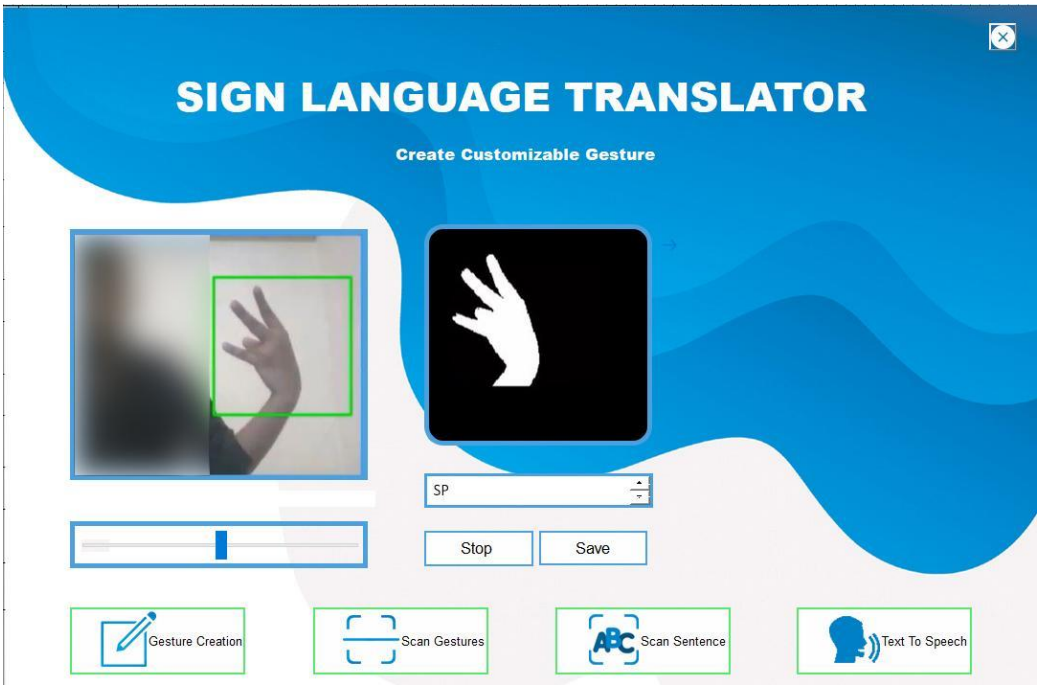


Figure 10.8: Custom Gesture Screen

This is the custom gesture screen here the user can add custom gestures for expressing special or unique meanings. The name for the gesture is provided by the user and the same name will be used for the folder creation where its data is stored.

CHAPTER 11
CONCLUSION

Here a method of gesture recognition based on CNN is introduced and evaluation of the model in a real-world environment. The experimental results show that our model can achieve good results. First, for a specific gesture, by using the recognition model, it can effectively recognize the actual meaning of the gesture. The model can also achieve full automation, and its accuracy can reach a high level. Moreover, in the state where the illumination conditions are not particularly good, the recognition accuracy can be effectively improved by our pre-processing. Next, we will further test and improve our model. We have some preliminary thoughts on how to improve the results. The network also supports the addition of more gestures. In the future, we can even use gestures to play games, chat and email with others. Although the accuracy obtained by the experiment has been very high, we feel that it is necessary to further improve for the application to real life. We plan to further optimize our model by adding training data and changing the network structure.

REFERENCES

[1] Shobhit Agarwal, “What are some problems faced by deaf and dumb people while using today’s common tech like phones and PCs”, 2017 [Online].

[2] NIDCD, “American sign language”, 2017 [Online]. Available: https://www.nidcd.nih.gov/health/american-sign-language, [Accessed April 06, 2019].

[3] NAD, “American sign language-community and culture frequently asked questions”, 2017 [Online].

[4] J. Lee, Y. Lee, E. Lee, and S. Hong, ``Hand region extraction and gesture recognition from video stream with complex background through entropy analysis," in *Proc. Conf. IEEE Eng. Med. Biol. Soc.*, Jan. 2004, vol. 2, no. 2, pp. 1513_1516.

[5] Wei Fang1,2, Yewen Ding 1, Feihong Zhang1, And Jack Sheng3, ``Gesture Recognition Based on CNN and DCGAN for Calculation and Text Output", February 27, 2019,

[6] T. Takahashi and F. Kishino, ``Hand gesture coding based on experiments using a hand gesture interface device," ACM Sigchi Bull., vol. 23, no. 2, 1991.

[7] Wikipedia, “Convolutional neural network”, 2017 [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network, [Accessed April 08, 2019].

[8] Ricky Anderson ,FannyWiryana ,Meita Chandra Ariesta, ``Sign Language Recognition Application Systems for Deaf-Mute People: A Review Based on Input Process-Output,"2017 [Online].

[9] R. Meng, S. G. Rice, J. Wang, and X. Sun, ‘‘A fusion steganographic algorithm based on faster R-CNN,’’ CMC, Comput., Mater. Continua, vol. 55, no. 1, pp. 1–16, 2018.

[10] C. Szegedy et al., ‘‘Going deeper with convolutions,’’ presented at the CVPR, Boston, MA, USA, Jun. 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun, ‘‘Deep residual learning for image recognition,’’ in Proc. CVPR Las Vegas, NV, USA, 2016, pp. 770–778.

[12] I. J. Goodfellow et al., ‘‘Generative adversarial networks,’’ in Proc. Adv. Neural Inf. Process. Syst., vol. 3, Jun. 2014, pp. 2672–2680.

[13] A. Radford, L. Metz, and S. Chintala, ‘‘Unsupervised representation learning with deep convolutional generative adversarial networks,’’ presented at the ICLR, San Juan, PR, USA, May 2016.

[14] W. Fang, F. Zhang, V. S. Sheng, and Y. Ding, ‘‘A method for improving CNN-based image recognition using DCGAN,’’ CMC, Comput., Mater. Continua, vol. 57, no. 1, pp. 167–178, 2018.

[15] Wikipedia,“Wiredglove”,[Online]. Available: https://en.wikipedia.org/wiki/Wired_gove, [Accessed December 08, 2019].

[16] GloveNation,“ColourcodedGloves”,[Online].Available:https://glovenation.com/blogs/default-aultblog/blog-color-coding-gloves, [Accessed December 08, 2019].