INTERNSHIP REPORT

A report submitted in partial fulfilment of the requirements for the Award of Degree of

INTERNSHIP UNDER IBM SKILLSBUILD PROGRAM ON AI & ML

On

AMERICAN SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING ALGORITHM

By,

ABHISHEK CHAKRABORTY

Student Id: **STU61507abd9adb81632664253**

AICTE Email: **1da19ml001.ml@drait.edu.in**

Personal Email: **ronnieabhishek98@gmail.com**

Institute: **Dr. Ambedkar Institute of Technology, Bengaluru**

ACKNOWLEDGEMENT

I would like to thank mentors **Mr. Bassar Patel, Edunet Foundation** and **Mr. Utkarsh Sharma, Edunet Foundation** for mentoring and providing constant support and help in the completion of internship.

I would like to thank **Mrs. Khyati Chugh, Edunet Foundation** for giving me the opportunity to do an internship within the organization and for her constant support and help in the completion of internship.

I would like to thank **Mr. Thirukumaran Nagarajan, VP HR- India and South Asia, IBM** for accepting my application of internship and giving me the opportunity to do an internship within the organization

I also would like to thank all the people that worked along with me for their patience and openness, they created an enjoyable working environment. It is indeed with a great sense of pleasure and immense sense of gratitude that I acknowledge the help of these individuals.

I am extremely grateful to the **Medical Electronics Engineering department Professors and staff members** of **Dr. Ambedkar Institute of Technology, Bengaluru** and **friends** who helped me in successful completion of this internship.

ABHISHEK CHAKRABORTY

CONTENTS

CHAPTER 1

ABSTRACT

One approach to communicate with the deaf is through sign language. There is no universally recognised sign language. There are many variations of sign languages because, like spoken languages, they evolved naturally via interaction between various groups of people. Between 138 and 300 different sign languages are currently in use all over the world. One of them is American Sign Language. The main obstacles that have prevented more research on American Sign Language from being done in this body of work are incorporated features and local language variance. To communicate with them, sign language should be learned. Peer groups are typically where learning happens. There aren't many study resources accessible for learning signs. The process of learning sign language is therefore a very challenging undertaking. Finger spelling is the first stage of sign learning, and it is also used when there is no corresponding sign or when the signer is unaware of it. The majority of the currently available sign language learning systems rely on pricey external sensors. By gathering a dataset and using various feature extraction techniques to extract relevant data, our study intends to advance this subject. This data is then used as input into machine learning techniques. Currently, we have published all of the validated results for the techniques, and the improvement over earlier work can be attributed to the results' consistently high accuracy. In upcoming studies, the validation set will correspond to pictures of people who aren't the same as the people in the training set.

CHAPTER 2

INTRODUCTION

The sign language is frequently used by deaf-dumb individuals as a means of communication. Few people are familiar with sign language. A sign language is nothing more than a collection of diverse gestures made by varied hand gestures, including movements, orientations, and facial expressions. Despite what many people think, it is not a universal tongue. There are approximately 466 million persons with hearing loss in the globe, 34 million of them are children. Obviously, this makes it more difficult for the Deaf community to communicate with the hearing majority. Because the Deaf people is typically less adept at writing a spoken language, written communication is a laborious alternative. For communication, they employ sign language. Around the world, people utilise a variety of sign languages. They are significantly less in number than spoken languages. Additionally, this form of communication is slow and impersonal in face-to-face interactions. For instance, it is frequently vital to speak with the emergency doctor right away after an accident because textual communication is not always an option.

The purpose of this work is to contribute to the field of automatic sign language recognition. We focus on the recognition of the signs or gestures. An automated system for human activity recognition in spatiotemporal data can be built in two primary steps. The initial step is to take the frame sequences' features and extract them. A representation made up of one or more feature vectors, also known as descriptors, will be the outcome of this. The computer will be helped by this representation in its ability to discern between various action classes. The categorisation of the action is the second step. Now these signs will be marked using Mediapipe Hand to track the movements of the fingers and hand. Accordingly, the images will be trained using a machine learning algorithm which will easily differentiate between different signs. And when camera is switched on to test later, then it easily shows which letters are there for the signs and their accuracy.

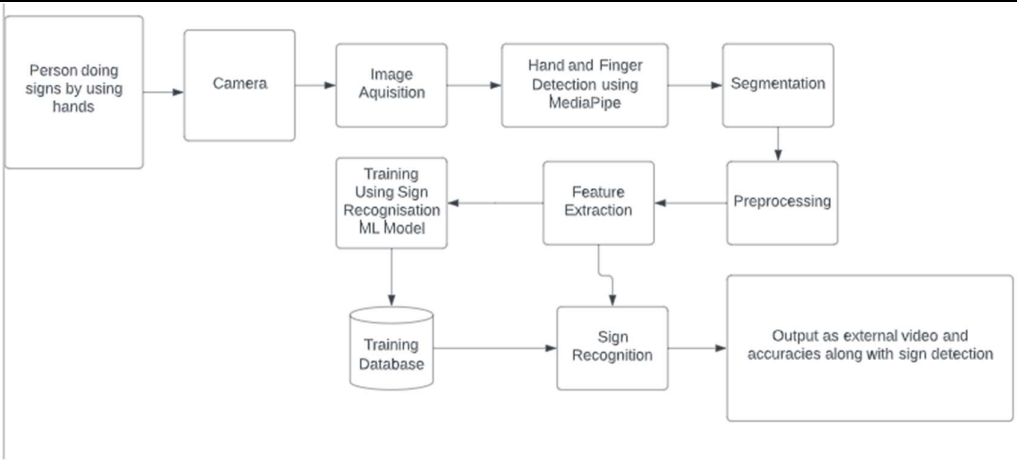

Fig2.1: Block Diagram

2.1 User / Customer:

It is most commonly used by deaf & dumb people who have hearing or talking disorders to communicate among themselves or with normal people.

2.2 Problem Statement:

Given a hand gesture, implementing such an application that detects sign language in real time through hand gestures and allowing the user to get the accurate values of the result of the character detected in a live video take as well as allowing such users to build their customised gesture so that the problems faced by people who are unable to talk vocally can be accommodated with technological assistance and the barrier of expression can be overcome. This motivated me to work for the project.

CHAPTER 3

SOFTWARE REQUIREMENTS SPECIFICATIONS

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behaviour, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

3.1 Software Requirements:

Operating System: Windows 10 (64-bit operating system, x64-based processor)

Coding Language: Python version 3.9.5

IDE: Visual Studio Code (VS Code)

Project on: Machine Learning

3.2 Hardware Requirements:

System: Intel(R) Core (TM) i5-8265U CPU @ 1.60GHz 1.80 GHz

Hard Disk: 1TB HDD, 256GB SSD

RAM: 8GB

CHAPTER 4

TECHNOLOGIES

4.1 TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

TensorFlow was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019.

TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

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 most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

TensorFlow 2.0, released in October 2019, revamped the framework in many ways based on user feedback, to make it easier to work with (as an example, by using the relatively simple Keras API for model training) and more performant. Distributed training is easier to run thanks to a new API, and support for TensorFlow Lite makes it possible to deploy models on a greater variety of platforms. However, code written for earlier versions of TensorFlow must be rewritten—sometimes only slightly, sometimes significantly—to take maximum advantage of new TensorFlow 2.0 features.

4.2 MediaPipe Hands

The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. For example, it can form the basis for sign language understanding and hand gesture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality. While coming naturally to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other (e.g. finger/palm occlusions and handshakes) and lack high contrast patterns.

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. We hope that providing this hand perception functionality to the wider research and development community will result in an emergence of creative use cases, stimulating new applications and new research avenues.

MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints. This strategy is similar to that employed in our MediaPipe Face Mesh solution, which uses a face detector together with a face landmark model.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in our pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand.

The pipeline is implemented as a MediaPipe graph that uses a hand landmark tracking subgraph from the hand landmark module, and renders using a dedicated hand renderer subgraph. The hand landmark tracking subgraph internally uses a hand landmark

subgraph from the same module and a palm detection subgraph from the palm detection module.

4.3 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Vectorization describes the absence of any explicit looping, indexing, etc., in the code - these things are taking place, of course, just "behind the scenes" in optimized, pre-compiled C code. Vectorized code has many advantages, among which are:

- Vectorized code is more concise and easier to read
- Fewer lines of code generally mean fewer bugs
- The code more closely resembles standard mathematical notation (making it easier, typically, to correctly code mathematical constructs)

Vectorization results in more "Pythonic" code. Without vectorization, our code would be littered with inefficient and difficult to read for loops.

Broadcasting is the term used to describe the implicit element-by-element behavior of operations; generally speaking, in NumPy all operations, not just arithmetic operations, but logical, bit-wise, functional, etc., behave in this implicit element-by-element fashion, i.e., they broadcast. Moreover, in the example above, a and b could be multidimensional arrays of the same shape, or a scalar and an array, or even two arrays of with different shapes, provided that the smaller array is "expandable" to the shape of the larger in such a way that the resulting broadcast is unambiguous.

NumPy fully supports an object-oriented approach, starting, once again, with ndarray. For example, ndarray is a class, possessing numerous methods and attributes. Many of its methods are mirrored by functions in the outer-most NumPy namespace, allowing the programmer to code in whichever paradigm they prefer. This flexibility has allowed the NumPy array dialect and NumPy ndarray class to become the de-facto language of multi-dimensional data interchange used in Python.

4.4 Scikit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Scikit-learn integrates well with many other Python libraries, such as Matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.

4.5 Pandas

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the threeclause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself. Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

4.6 Keras

Keras is the most used deep learning framework among top-5 winning teams on Kaggle. Because Keras makes it easier to run new experiments, it empowers you to try more ideas than your competition, faster. Built on top of TensorFlow 2, Keras is an industry-strength framework that can scale to large clusters of GPUs or an entire TPU pod. Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

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.

CHAPTER 5

DATASET AND INPUT

Lots of datasets are available on net. But I have created my own dataset. I have used American Sign Language signs for this project and have created images of signs of “A” to “Z” using OpenCV module by taking pictures frame by frame through computer camera.

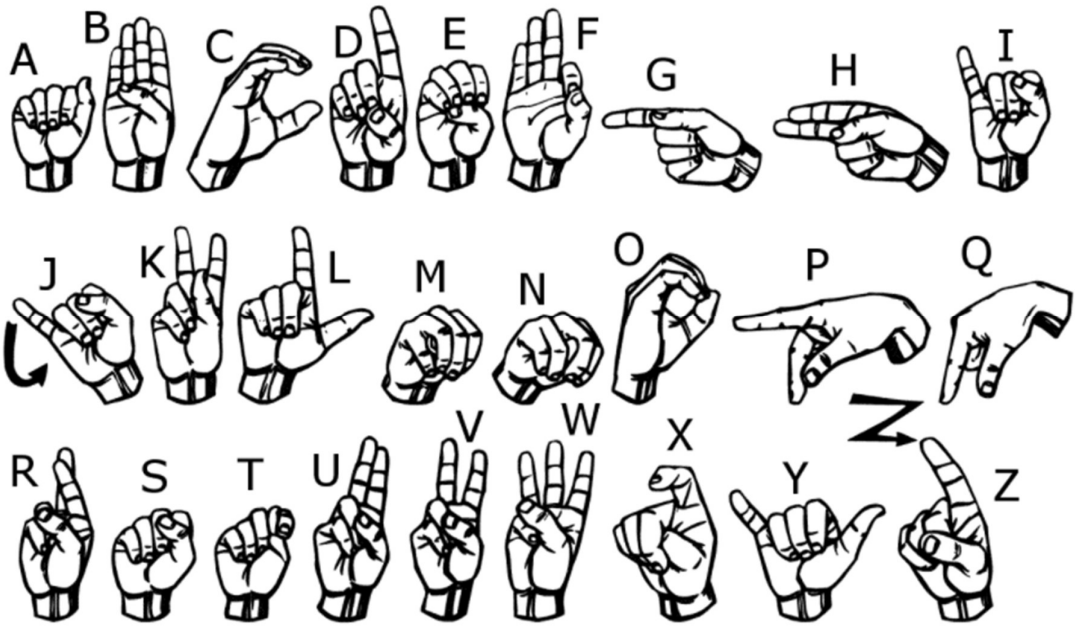

Fig5.1: American Sign Language

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

To get access of the full dataset used for this project, please click on the link:-

https://drive.google.com/drive/folders/1kkSK-MFVe8OL_UJVek_xZSZNbqGcj9md?usp=sharing

CHAPTER 6

PROCESSING OF DATASET IMAGES

Since MediaPipe is used, so fingers were tracked and accordingly the signs were saved in the process, which were later trained using sign detection ML model.

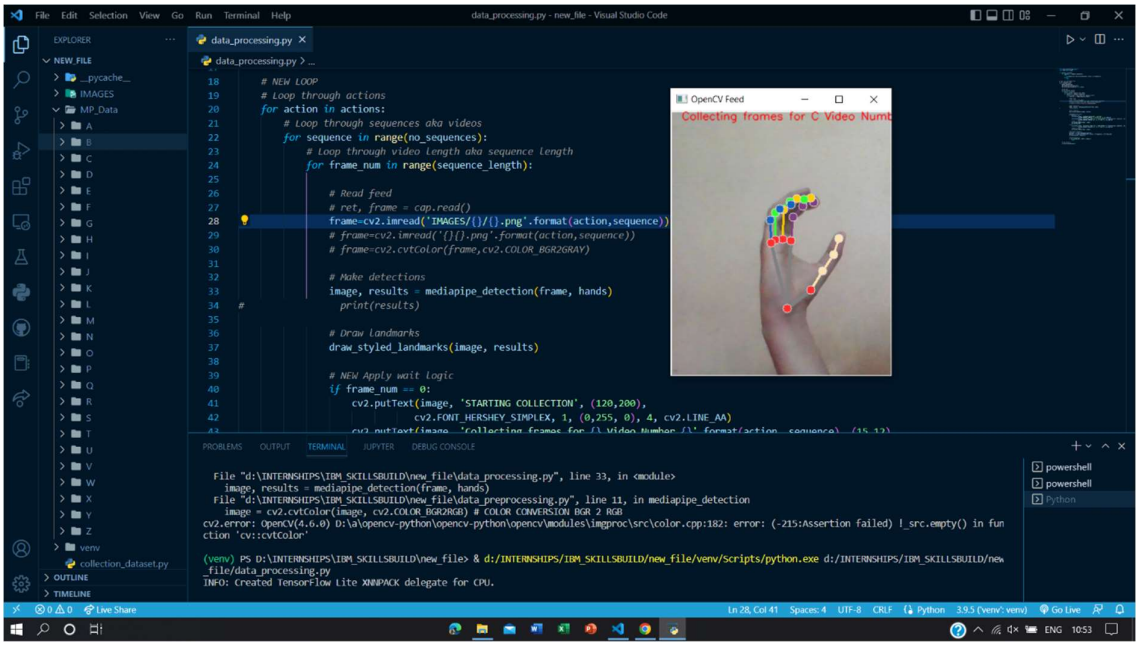

Fig6.1: Detection of finger movement during letter C recognition

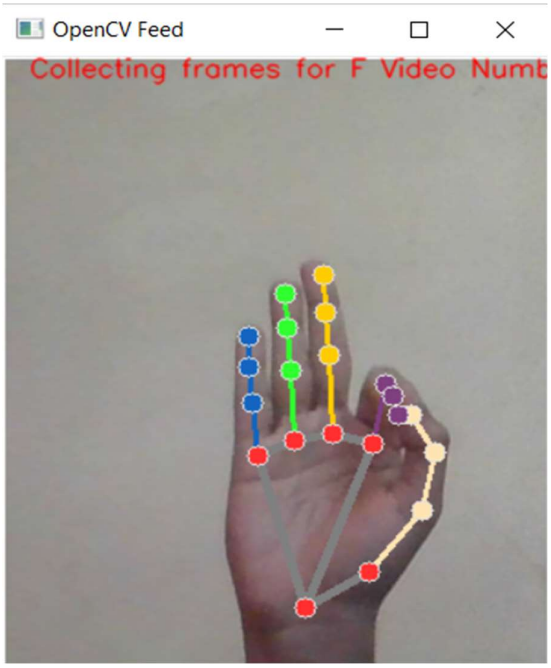

Fig6.2: MediaPipe Hand tracking the parts of finger for F sign

CHAPTER 7

MODEL USED AND MODEL TRAINING

The model code used for training the images is:

```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,63)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

Fig7.1: Sign Detection ML Model

After training the model, the epochs values received are:

```
Epoch 1/200
24/24 [==============================] - 12s 95ms/step - loss: 3.2575 - categorical_accuracy: 0.0418
Epoch 2/200
24/24 [==============================] - 2s 72ms/step - loss: 3.1472 - categorical_accuracy: 0.0783
Epoch 3/200
24/24 [==============================] - 2s 96ms/step - loss: 3.0928 - categorical_accuracy: 0.0972
Epoch 4/200
24/24 [==============================] - 2s 77ms/step - loss: 3.1831 - categorical_accuracy: 0.1282
Epoch 5/200
24/24 [==============================] - 2s 98ms/step - loss: 2.9050 - categorical_accuracy: 0.1296
Epoch 6/200
24/24 [==============================] - 2s 94ms/step - loss: 3.0471 - categorical_accuracy: 0.0864
Epoch 7/200
24/24 [==============================] - 2s 63ms/step - loss: 2.9882 - categorical_accuracy: 0.1107
Epoch 8/200
24/24 [==============================] - 2s 69ms/step - loss: 2.7761 - categorical_accuracy: 0.1404
Epoch 9/200
24/24 [==============================] - 2s 79ms/step - loss: 2.6368 - categorical_accuracy: 0.1741
Epoch 10/200
24/24 [==============================] - 2s 71ms/step - loss: 2.1229 - categorical_accuracy: 0.2780
Epoch 11/200
24/24 [==============================] - 2s 84ms/step - loss: 2.3945 - categorical_accuracy: 0.2794
Epoch 12/200
24/24 [==============================] - 2s 83ms/step - loss: 1.8931 - categorical_accuracy: 0.3252
Epoch 13/200
24/24 [==============================] - 2s 70ms/step - loss: 1.8749 - categorical_accuracy: 0.3765
Epoch 14/200
24/24 [==============================] - 2s 81ms/step - loss: 1.4892 - categorical_accuracy: 0.4926
Epoch 15/200
24/24 [==============================] - 2s 98ms/step - loss: 1.3145 - categorical_accuracy: 0.5924
Epoch 16/200
24/24 [==============================] - 2s 72ms/step - loss: 1.1653 - categorical_accuracy: 0.6140
Epoch 17/200
24/24 [==============================] - 2s 77ms/step - loss: 0.6896 - categorical_accuracy: 0.7841
Epoch 18/200
24/24 [==============================] - 2s 84ms/step - loss: 1.3701 - categorical_accuracy: 0.5803
```

```
Epoch 19/200
24/24 [==============================] - 2s 63ms/step - loss: 0.7326 - categorical_accuracy: 0.8030
Epoch 20/200
24/24 [==============================] - 2s 71ms/step - loss: 1.4594 - categorical_accuracy: 0.5614
Epoch 21/200
24/24 [==============================] - 2s 88ms/step - loss: 0.7423 - categorical_accuracy: 0.8057
Epoch 22/200
24/24 [==============================] - 2s 93ms/step - loss: 0.2957 - categorical_accuracy: 0.9258
Epoch 23/200
24/24 [==============================] - 3s 106ms/step - loss: 0.2533 - categorical_accuracy: 0.9271
Epoch 24/200
24/24 [==============================] - 2s 84ms/step - loss: 0.7998 - categorical_accuracy: 0.7449
Epoch 25/200
24/24 [==============================] - 2s 80ms/step - loss: 0.2843 - categorical_accuracy: 0.9244
Epoch 26/200
24/24 [==============================] - 2s 77ms/step - loss: 0.3866 - categorical_accuracy: 0.8826
Epoch 27/200
24/24 [==============================] - 2s 78ms/step - loss: 0.6902 - categorical_accuracy: 0.7760
Epoch 28/200
24/24 [==============================] - 2s 81ms/step - loss: 0.2863 - categorical_accuracy: 0.9325
Epoch 29/200
24/24 [==============================] - 2s 77ms/step - loss: 0.1543 - categorical_accuracy: 0.9514
Epoch 30/200
24/24 [==============================] - 2s 78ms/step - loss: 0.1564 - categorical_accuracy: 0.9433
Epoch 31/200
24/24 [==============================] - 2s 82ms/step - loss: 0.0453 - categorical_accuracy: 0.9933
Epoch 32/200
24/24 [==============================] - 2s 74ms/step - loss: 0.0086 - categorical_accuracy: 1.0000
Epoch 33/200
24/24 [==============================] - 2s 78ms/step - loss: 0.0037 - categorical_accuracy: 1.0000
Epoch 34/200
24/24 [==============================] - 2s 85ms/step - loss: 0.0027 - categorical_accuracy: 1.0000
Epoch 35/200
24/24 [==============================] - 2s 84ms/step - loss: 0.0025 - categorical_accuracy: 1.0000
Epoch 36/200
24/24 [==============================] - 2s 94ms/step - loss: 0.0016 - categorical_accuracy: 1.0000
Epoch 37/200
24/24 [==============================] - 3s 103ms/step - loss: 0.0011 - categorical_accuracy: 1.0000
Epoch 38/200
24/24 [==============================] - 2s 96ms/step - loss: 9.3640e-04 - categorical_accuracy: 1.0000
```

```
Epoch 39/200
24/24 [==============================] - 2s 91ms/step - loss: 7.8609e-04 - categorical_accuracy: 1.0000
Epoch 40/200
24/24 [==============================] - 2s 93ms/step - loss: 7.1311e-04 - categorical_accuracy: 1.0000
Epoch 41/200
24/24 [==============================] - 2s 95ms/step - loss: 6.3186e-04 - categorical_accuracy: 1.0000
Epoch 42/200
24/24 [==============================] - 2s 94ms/step - loss: 5.7471e-04 - categorical_accuracy: 1.0000
Epoch 43/200
24/24 [==============================] - 2s 92ms/step - loss: 4.8548e-04 - categorical_accuracy: 1.0000
Epoch 44/200
24/24 [==============================] - 2s 95ms/step - loss: 4.2266e-04 - categorical_accuracy: 1.0000
Epoch 45/200
24/24 [==============================] - 2s 84ms/step - loss: 3.7876e-04 - categorical_accuracy: 1.0000
Epoch 46/200
24/24 [==============================] - 2s 74ms/step - loss: 3.5209e-04 - categorical_accuracy: 1.0000
Epoch 47/200
24/24 [==============================] - 2s 82ms/step - loss: 3.2171e-04 - categorical_accuracy: 1.0000
Epoch 48/200
24/24 [==============================] - 2s 69ms/step - loss: 3.0849e-04 - categorical_accuracy: 1.0000
Epoch 49/200
24/24 [==============================] - 2s 95ms/step - loss: 2.7325e-04 - categorical_accuracy: 1.0000
Epoch 50/200
24/24 [==============================] - 2s 96ms/step - loss: 2.4556e-04 - categorical_accuracy: 1.0000
Epoch 51/200
24/24 [==============================] - 2s 76ms/step - loss: 2.3267e-04 - categorical_accuracy: 1.0000
Epoch 52/200
24/24 [==============================] - 2s 91ms/step - loss: 2.0814e-04 - categorical_accuracy: 1.0000
Epoch 53/200
24/24 [==============================] - 2s 86ms/step - loss: 1.9368e-04 - categorical_accuracy: 1.0000
Epoch 54/200
24/24 [==============================] - 2s 79ms/step - loss: 1.8545e-04 - categorical_accuracy: 1.0000
Epoch 55/200
24/24 [==============================] - 2s 73ms/step - loss: 1.8806e-04 - categorical_accuracy: 1.0000
Epoch 56/200
24/24 [==============================] - 3s 109ms/step - loss: 1.5602e-04 - categorical_accuracy: 1.0000
Epoch 57/200
24/24 [==============================] - 2s 90ms/step - loss: 1.4798e-04 - categorical_accuracy: 1.0000
Epoch 58/200
24/24 [==============================] - 2s 74ms/step - loss: 1.3744e-04 - categorical_accuracy: 1.0000
```

```
Epoch 59/200
24/24 [==============================] - 2s 66ms/step - loss: 1.2998e-04 - categorical_accuracy: 1.0000
Epoch 60/200
24/24 [==============================] - 2s 85ms/step - loss: 1.2334e-04 - categorical_accuracy: 1.0000
Epoch 61/200
24/24 [==============================] - 2s 89ms/step - loss: 1.1553e-04 - categorical_accuracy: 1.0000
Epoch 62/200
24/24 [==============================] - 2s 94ms/step - loss: 1.1152e-04 - categorical_accuracy: 1.0000
Epoch 63/200
24/24 [==============================] - 2s 97ms/step - loss: 1.0400e-04 - categorical_accuracy: 1.0000
Epoch 64/200
24/24 [==============================] - 2s 87ms/step - loss: 9.6947e-05 - categorical_accuracy: 1.0000
Epoch 65/200
24/24 [==============================] - 2s 89ms/step - loss: 9.4331e-05 - categorical_accuracy: 1.0000
Epoch 66/200
24/24 [==============================] - 2s 97ms/step - loss: 8.9686e-05 - categorical_accuracy: 1.0000
Epoch 67/200
24/24 [==============================] - 2s 90ms/step - loss: 8.3982e-05 - categorical_accuracy: 1.0000
Epoch 68/200
24/24 [==============================] - 2s 80ms/step - loss: 8.0532e-05 - categorical_accuracy: 1.0000
Epoch 69/200
24/24 [==============================] - 2s 77ms/step - loss: 7.8805e-05 - categorical_accuracy: 1.0000
Epoch 70/200
24/24 [==============================] - 2s 74ms/step - loss: 7.3628e-05 - categorical_accuracy: 1.0000
Epoch 71/200
24/24 [==============================] - 2s 68ms/step - loss: 7.1003e-05 - categorical_accuracy: 1.0000
Epoch 72/200
24/24 [==============================] - 2s 82ms/step - loss: 6.7394e-05 - categorical_accuracy: 1.0000
Epoch 73/200
24/24 [==============================] - 2s 74ms/step - loss: 6.5128e-05 - categorical_accuracy: 1.0000
Epoch 74/200
24/24 [==============================] - 2s 64ms/step - loss: 6.3224e-05 - categorical_accuracy: 1.0000
Epoch 75/200
24/24 [==============================] - 2s 79ms/step - loss: 6.0764e-05 - categorical_accuracy: 1.0000
Epoch 76/200
24/24 [==============================] - 3s 105ms/step - loss: 5.6727e-05 - categorical_accuracy: 1.0000
Epoch 77/200
24/24 [==============================] - 2s 88ms/step - loss: 5.4248e-05 - categorical_accuracy: 1.0000
Epoch 78/200
24/24 [==============================] - 2s 98ms/step - loss: 5.1654e-05 - categorical_accuracy: 1.0000
```

```
Epoch 79/200
24/24 [==============================] - 2s 73ms/step - loss: 4.9561e-05 - categorical_accuracy: 1.0000
Epoch 80/200
24/24 [==============================] - 2s 78ms/step - loss: 4.7651e-05 - categorical_accuracy: 1.0000
Epoch 81/200
24/24 [==============================] - 2s 87ms/step - loss: 4.7084e-05 - categorical_accuracy: 1.0000
Epoch 82/200
24/24 [==============================] - 2s 71ms/step - loss: 4.4084e-05 - categorical_accuracy: 1.0000
Epoch 83/200
24/24 [==============================] - 2s 78ms/step - loss: 4.2621e-05 - categorical_accuracy: 1.0000
Epoch 84/200
24/24 [==============================] - 2s 71ms/step - loss: 4.0873e-05 - categorical_accuracy: 1.0000
Epoch 85/200
24/24 [==============================] - 2s 75ms/step - loss: 4.0191e-05 - categorical_accuracy: 1.0000
Epoch 86/200
24/24 [==============================] - 2s 80ms/step - loss: 3.8165e-05 - categorical_accuracy: 1.0000
Epoch 87/200
24/24 [==============================] - 2s 86ms/step - loss: 3.6704e-05 - categorical_accuracy: 1.0000
Epoch 88/200
24/24 [==============================] - 2s 89ms/step - loss: 3.5387e-05 - categorical_accuracy: 1.0000
Epoch 89/200
24/24 [==============================] - 2s 69ms/step - loss: 3.4098e-05 - categorical_accuracy: 1.0000
Epoch 90/200
24/24 [==============================] - 2s 75ms/step - loss: 3.3467e-05 - categorical_accuracy: 1.0000
Epoch 91/200
24/24 [==============================] - 2s 75ms/step - loss: 3.3088e-05 - categorical_accuracy: 1.0000
Epoch 92/200
24/24 [==============================] - 2s 65ms/step - loss: 3.1772e-05 - categorical_accuracy: 1.0000
Epoch 93/200
24/24 [==============================] - 2s 73ms/step - loss: 2.9243e-05 - categorical_accuracy: 1.0000
Epoch 94/200
24/24 [==============================] - 3s 105ms/step - loss: 2.8215e-05 - categorical_accuracy: 1.0000
Epoch 95/200
24/24 [==============================] - 2s 93ms/step - loss: 2.7281e-05 - categorical_accuracy: 1.0000
Epoch 96/200
24/24 [==============================] - 2s 79ms/step - loss: 2.6212e-05 - categorical_accuracy: 1.0000
Epoch 97/200
24/24 [==============================] - 2s 73ms/step - loss: 2.5783e-05 - categorical_accuracy: 1.0000
```

```
Epoch 98/200
24/24 [==============================] - 2s 83ms/step - loss: 2.4750e-05 - categorical_accuracy: 1.0000
Epoch 99/200
24/24 [==============================] - 2s 89ms/step - loss: 2.3881e-05 - categorical_accuracy: 1.0000
Epoch 100/200
24/24 [==============================] - 2s 76ms/step - loss: 2.3080e-05 - categorical_accuracy: 1.0000
Epoch 101/200
24/24 [==============================] - 2s 84ms/step - loss: 2.2503e-05 - categorical_accuracy: 1.0000
Epoch 102/200
24/24 [==============================] - 2s 73ms/step - loss: 2.2162e-05 - categorical_accuracy: 1.0000
Epoch 103/200
24/24 [==============================] - 2s 94ms/step - loss: 2.0884e-05 - categorical_accuracy: 1.0000
Epoch 104/200
24/24 [==============================] - 2s 66ms/step - loss: 2.0420e-05 - categorical_accuracy: 1.0000
Epoch 105/200
24/24 [==============================] - 2s 96ms/step - loss: 1.9755e-05 - categorical_accuracy: 1.0000
Epoch 106/200
24/24 [==============================] - 2s 90ms/step - loss: 1.9335e-05 - categorical_accuracy: 1.0000
Epoch 107/200
24/24 [==============================] - 2s 69ms/step - loss: 1.8831e-05 - categorical_accuracy: 1.0000
Epoch 108/200
24/24 [==============================] - 2s 74ms/step - loss: 1.8216e-05 - categorical_accuracy: 1.0000
Epoch 109/200
24/24 [==============================] - 2s 78ms/step - loss: 1.8270e-05 - categorical_accuracy: 1.0000
Epoch 110/200
24/24 [==============================] - 2s 69ms/step - loss: 1.8367e-05 - categorical_accuracy: 1.0000
Epoch 111/200
24/24 [==============================] - 2s 77ms/step - loss: 1.6943e-05 - categorical_accuracy: 1.0000
Epoch 112/200
24/24 [==============================] - 2s 73ms/step - loss: 1.6170e-05 - categorical_accuracy: 1.0000
Epoch 113/200
24/24 [==============================] - 2s 77ms/step - loss: 1.5649e-05 - categorical_accuracy: 1.0000
Epoch 114/200
24/24 [==============================] - 2s 66ms/step - loss: 1.5226e-05 - categorical_accuracy: 1.0000
Epoch 115/200
24/24 [==============================] - 2s 74ms/step - loss: 1.4853e-05 - categorical_accuracy: 1.0000
Epoch 116/200
24/24 [==============================] - 2s 83ms/step - loss: 1.4220e-05 - categorical_accuracy: 1.0000
Epoch 117/200
24/24 [==============================] - 2s 92ms/step - loss: 1.4075e-05 - categorical_accuracy: 1.0000
```

```
Epoch 118/200
24/24 [==============================] - 2s 71ms/step - loss: 1.3908e-05 - categorical_accuracy: 1.0000
Epoch 119/200
24/24 [==============================] - 2s 77ms/step - loss: 1.3293e-05 - categorical_accuracy: 1.0000
Epoch 120/200
24/24 [==============================] - 2s 67ms/step - loss: 1.2705e-05 - categorical_accuracy: 1.0000
Epoch 121/200
24/24 [==============================] - 2s 70ms/step - loss: 1.2415e-05 - categorical_accuracy: 1.0000
Epoch 122/200
24/24 [==============================] - 2s 81ms/step - loss: 1.2048e-05 - categorical_accuracy: 1.0000
Epoch 123/200
24/24 [==============================] - 2s 61ms/step - loss: 1.1774e-05 - categorical_accuracy: 1.0000
Epoch 124/200
24/24 [==============================] - 2s 81ms/step - loss: 1.1560e-05 - categorical_accuracy: 1.0000
Epoch 125/200
24/24 [==============================] - 2s 84ms/step - loss: 1.1228e-05 - categorical_accuracy: 1.0000
Epoch 126/200
24/24 [==============================] - 2s 75ms/step - loss: 1.0930e-05 - categorical_accuracy: 1.0000
Epoch 127/200
24/24 [==============================] - 2s 86ms/step - loss: 1.0946e-05 - categorical_accuracy: 1.0000
Epoch 128/200
24/24 [==============================] - 2s 84ms/step - loss: 1.0462e-05 - categorical_accuracy: 1.0000
Epoch 129/200
24/24 [==============================] - 2s 82ms/step - loss: 1.0040e-05 - categorical_accuracy: 1.0000
Epoch 130/200
24/24 [==============================] - 2s 88ms/step - loss: 9.7861e-06 - categorical_accuracy: 1.0000
Epoch 131/200
24/24 [==============================] - 2s 74ms/step - loss: 9.7901e-06 - categorical_accuracy: 1.0000
Epoch 132/200
24/24 [==============================] - 2s 79ms/step - loss: 9.3582e-06 - categorical_accuracy: 1.0000
Epoch 133/200
24/24 [==============================] - 2s 87ms/step - loss: 9.0893e-06 - categorical_accuracy: 1.0000
Epoch 134/200
24/24 [==============================] - 2s 77ms/step - loss: 8.8297e-06 - categorical_accuracy: 1.0000
Epoch 135/200
24/24 [==============================] - 2s 84ms/step - loss: 8.6916e-06 - categorical_accuracy: 1.0000
Epoch 136/200
24/24 [==============================] - 2s 83ms/step - loss: 8.4156e-06 - categorical_accuracy: 1.0000
```

```
Epoch 137/200
24/24 [==============================] - 2s 72ms/step - loss: 8.2374e-06 - categorical_accuracy: 1.0000
Epoch 138/200
24/24 [==============================] - 2s 75ms/step - loss: 8.0135e-06 - categorical_accuracy: 1.0000
Epoch 139/200
24/24 [==============================] - 2s 90ms/step - loss: 7.8365e-06 - categorical_accuracy: 1.0000
Epoch 140/200
24/24 [==============================] - 2s 72ms/step - loss: 7.6182e-06 - categorical_accuracy: 1.0000
Epoch 141/200
24/24 [==============================] - 2s 76ms/step - loss: 7.5140e-06 - categorical_accuracy: 1.0000
Epoch 142/200
24/24 [==============================] - 2s 83ms/step - loss: 7.3393e-06 - categorical_accuracy: 1.0000
Epoch 143/200
24/24 [==============================] - 2s 73ms/step - loss: 7.1172e-06 - categorical_accuracy: 1.0000
Epoch 144/200
24/24 [==============================] - 2s 72ms/step - loss: 7.0172e-06 - categorical_accuracy: 1.0000
Epoch 145/200
24/24 [==============================] - 2s 79ms/step - loss: 6.8582e-06 - categorical_accuracy: 1.0000
Epoch 146/200
24/24 [==============================] - 2s 75ms/step - loss: 6.6404e-06 - categorical_accuracy: 1.0000
Epoch 147/200
24/24 [==============================] - 2s 79ms/step - loss: 6.4718e-06 - categorical_accuracy: 1.0000
Epoch 148/200
24/24 [==============================] - 2s 79ms/step - loss: 6.3608e-06 - categorical_accuracy: 1.0000
Epoch 149/200
24/24 [==============================] - 2s 72ms/step - loss: 6.2943e-06 - categorical_accuracy: 1.0000
Epoch 150/200
24/24 [==============================] - 2s 81ms/step - loss: 6.0675e-06 - categorical_accuracy: 1.0000
Epoch 151/200
24/24 [==============================] - 2s 87ms/step - loss: 5.8837e-06 - categorical_accuracy: 1.0000
Epoch 152/200
24/24 [==============================] - 2s 70ms/step - loss: 5.7267e-06 - categorical_accuracy: 1.0000
Epoch 153/200
24/24 [==============================] - 2s 79ms/step - loss: 5.7012e-06 - categorical_accuracy: 1.0000
Epoch 154/200
24/24 [==============================] - 2s 84ms/step - loss: 5.4884e-06 - categorical_accuracy: 1.0000
Epoch 155/200
24/24 [==============================] - 2s 74ms/step - loss: 5.3896e-06 - categorical_accuracy: 1.0000
Epoch 156/200
24/24 [==============================] - 2s 77ms/step - loss: 5.2687e-06 - categorical_accuracy: 1.0000
```

```
Epoch 157/200
24/24 [==============================] - 2s 84ms/step - loss: 5.1440e-06 - categorical_accuracy: 1.0000
Epoch 158/200
24/24 [==============================] - 2s 75ms/step - loss: 5.0140e-06 - categorical_accuracy: 1.0000
Epoch 159/200
24/24 [==============================] - 2s 83ms/step - loss: 4.9201e-06 - categorical_accuracy: 1.0000
Epoch 160/200
24/24 [==============================] - 2s 88ms/step - loss: 4.8782e-06 - categorical_accuracy: 1.0000
Epoch 161/200
24/24 [==============================] - 2s 71ms/step - loss: 4.7578e-06 - categorical_accuracy: 1.0000
Epoch 162/200
24/24 [==============================] - 2s 80ms/step - loss: 4.6939e-06 - categorical_accuracy: 1.0000
Epoch 163/200
24/24 [==============================] - 2s 88ms/step - loss: 4.5171e-06 - categorical_accuracy: 1.0000
Epoch 164/200
24/24 [==============================] - 2s 71ms/step - loss: 4.4262e-06 - categorical_accuracy: 1.0000
Epoch 165/200
24/24 [==============================] - 2s 72ms/step - loss: 4.3284e-06 - categorical_accuracy: 1.0000
Epoch 166/200
24/24 [==============================] - 2s 87ms/step - loss: 4.2293e-06 - categorical_accuracy: 1.0000
Epoch 167/200
24/24 [==============================] - 2s 73ms/step - loss: 4.1355e-06 - categorical_accuracy: 1.0000
Epoch 168/200
24/24 [==============================] - 2s 82ms/step - loss: 4.0553e-06 - categorical_accuracy: 1.0000
Epoch 169/200
24/24 [==============================] - 2s 76ms/step - loss: 3.9838e-06 - categorical_accuracy: 1.0000
Epoch 170/200
24/24 [==============================] - 2s 76ms/step - loss: 3.9837e-06 - categorical_accuracy: 1.0000
Epoch 171/200
24/24 [==============================] - 2s 77ms/step - loss: 3.8064e-06 - categorical_accuracy: 1.0000
Epoch 172/200
24/24 [==============================] - 2s 74ms/step - loss: 3.7696e-06 - categorical_accuracy: 1.0000
Epoch 173/200
24/24 [==============================] - 3s 103ms/step - loss: 3.6195e-06 - categorical_accuracy: 1.0000
Epoch 174/200
24/24 [==============================] - 3s 104ms/step - loss: 3.5693e-06 - categorical_accuracy: 1.0000
Epoch 175/200
24/24 [==============================] - 2s 99ms/step - loss: 3.4745e-06 - categorical_accuracy: 1.0000
```

```
Epoch 176/200
24/24 [==============================] - 2s 94ms/step - loss: 3.4086e-06 - categorical_accuracy: 1.0000
Epoch 177/200
24/24 [==============================] - 2s 96ms/step - loss: 3.3441e-06 - categorical_accuracy: 1.0000
Epoch 178/200
24/24 [==============================] - 2s 89ms/step - loss: 3.2514e-06 - categorical_accuracy: 1.0000
Epoch 179/200
24/24 [==============================] - 2s 66ms/step - loss: 3.1990e-06 - categorical_accuracy: 1.0000
Epoch 180/200
24/24 [==============================] - 2s 73ms/step - loss: 3.1640e-06 - categorical_accuracy: 1.0000
Epoch 181/200
24/24 [==============================] - 2s 76ms/step - loss: 3.1613e-06 - categorical_accuracy: 1.0000
Epoch 182/200
24/24 [==============================] - 2s 73ms/step - loss: 3.0167e-06 - categorical_accuracy: 1.0000
Epoch 183/200
24/24 [==============================] - 2s 79ms/step - loss: 2.9580e-06 - categorical_accuracy: 1.0000
Epoch 184/200
24/24 [==============================] - 2s 74ms/step - loss: 2.8857e-06 - categorical_accuracy: 1.0000
Epoch 185/200
24/24 [==============================] - 3s 104ms/step - loss: 2.8389e-06 - categorical_accuracy: 1.0000
Epoch 186/200
24/24 [==============================] - 3s 104ms/step - loss: 2.8074e-06 - categorical_accuracy: 1.0000
Epoch 187/200
24/24 [==============================] - 2s 104ms/step - loss: 2.7176e-06 - categorical_accuracy: 1.0000
Epoch 188/200
24/24 [==============================] - 2s 85ms/step - loss: 2.6705e-06 - categorical_accuracy: 1.0000
Epoch 189/200
24/24 [==============================] - 2s 76ms/step - loss: 2.6332e-06 - categorical_accuracy: 1.0000
Epoch 190/200
24/24 [==============================] - 2s 73ms/step - loss: 2.5653e-06 - categorical_accuracy: 1.0000
Epoch 191/200
24/24 [==============================] - 2s 73ms/step - loss: 2.5516e-06 - categorical_accuracy: 1.0000
Epoch 192/200
24/24 [==============================] - 2s 73ms/step - loss: 2.4948e-06 - categorical_accuracy: 1.0000
Epoch 193/200
24/24 [==============================] - 2s 72ms/step - loss: 2.4437e-06 - categorical_accuracy: 1.0000
Epoch 194/200
24/24 [==============================] - 2s 72ms/step - loss: 2.4014e-06 - categorical_accuracy: 1.0000
Epoch 195/200
24/24 [==============================] - 2s 89ms/step - loss: 2.3496e-06 - categorical_accuracy: 1.0000
```

```
Epoch 196/200
24/24 [==============================] - 2s 72ms/step - loss: 2.2778e-06 - categorical_accuracy: 1.0000
Epoch 197/200
24/24 [==============================] - 2s 87ms/step - loss: 2.2620e-06 - categorical_accuracy: 1.0000
Epoch 198/200
24/24 [==============================] - 2s 81ms/step - loss: 2.2185e-06 - categorical_accuracy: 1.0000
Epoch 199/200
24/24 [==============================] - 2s 86ms/step - loss: 2.1483e-06 - categorical_accuracy: 1.0000
Epoch 200/200
24/24 [==============================] - 2s 74ms/step - loss: 2.1337e-06 - categorical_accuracy: 1.0000
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            32768

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 26)                858

=================================================================
Total params: 188,090
Trainable params: 188,090
Non-trainable params: 0
```

Fig7.2: 200 Epochs of training model

CHAPTER 8

RESULTS

Here are some pictures of the results obtained from training the model and testing with live video capturing using OpenCV model.

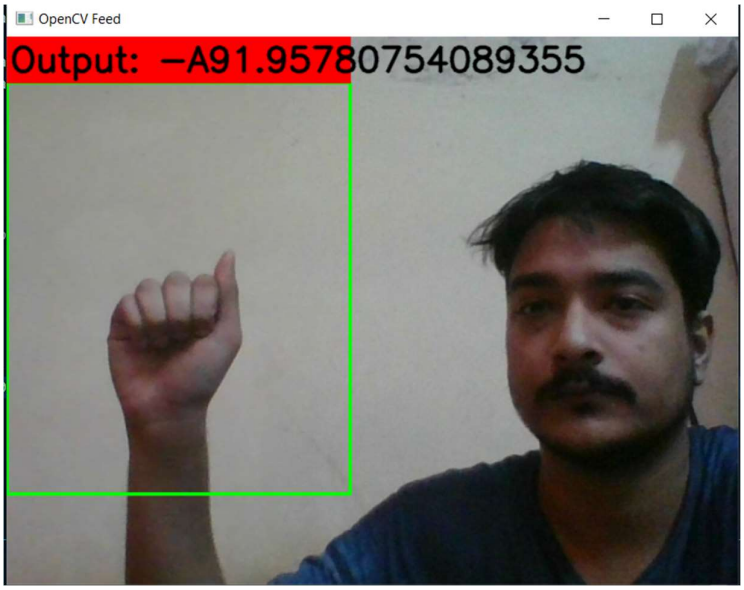

Fig8.1: Predicted A with accuracy of 91.96 %

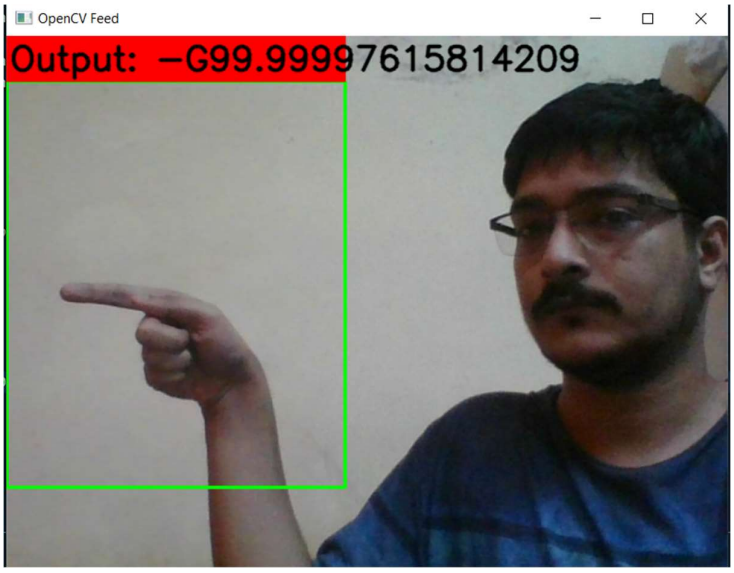

Fig8.2: Predicted G with accuracy of 99.99 %

Fig8.3: Predicted M with accuracy of 100 %

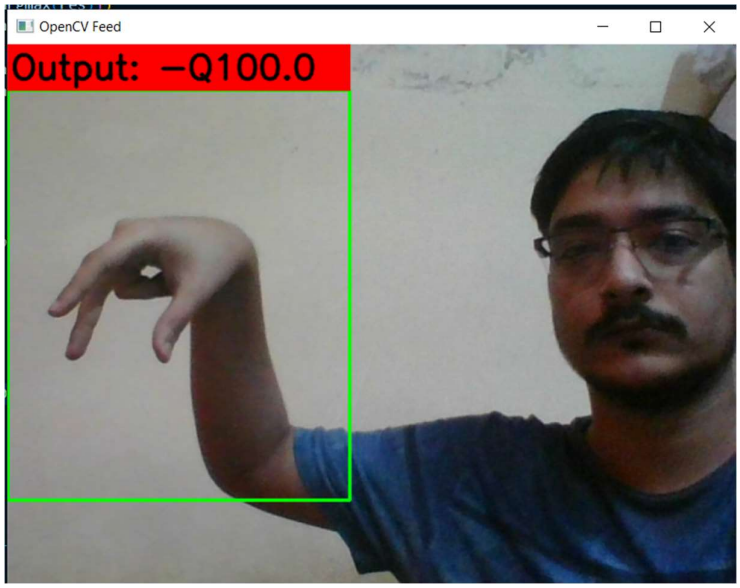

Fig8.4: Predicted Q with accuracy of 100 %

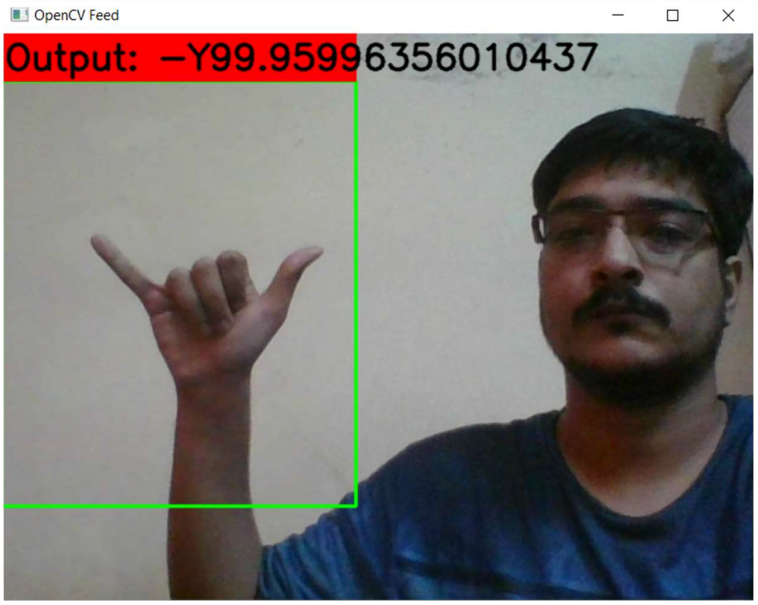

Fig8.5: Predicted Y with accuracy of 99.96 %

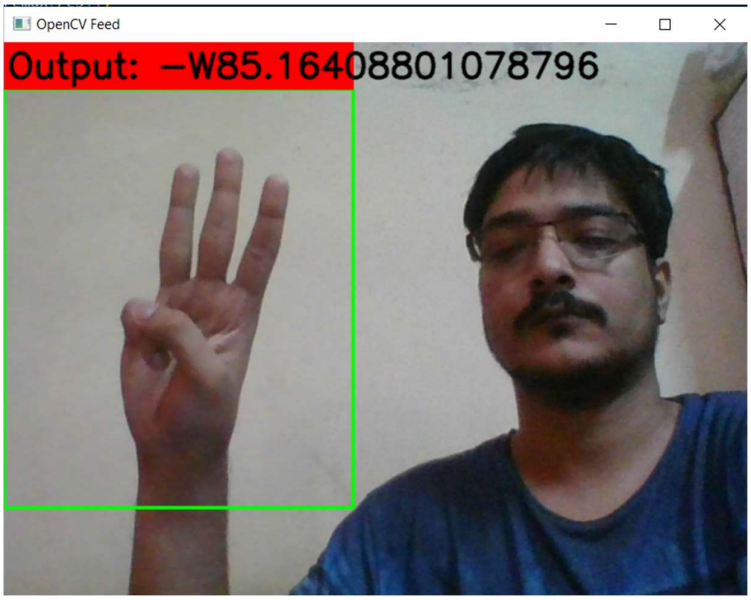

Fig8.6: Predicted W with accuracy of 85.16 %

SIGNS	ACCURACY
A	91.95 %
B	99.47 %

C	82.21 % (Getting results similar to O)
D	99.98 %
E	98.73 %
F	87.45 %
G	99.99 %
H	99.98 %
I	90.49 %
J	82.94 % (Getting results similar to I)
K	90.88 % (Getting results similar to V)
L	80.89 %
M	100 %
N	99.65 % (Getting results similar to S, T)
O	100 %
P	99.18 %
Q	100 %
R	99.99 %
S	89.21 % (Getting results similar to T)

T	99.99 % (Getting results similar to S, N, M)
U	89.78 % (Getting results similar to D)
V	100 %
W	85.16 %
X	95.69 % (Although not coming most of the times)
Y	99.96 %
Z	99.99 %

All the output images along with codes are uploaded in github website:-

https://github.com/Chakraborty98/Sign_Language_Recognition_Using_ML

CHAPTER 9

CONCLUSION

9.1 Regarding Project

Although most of the alphabets have accuracy of above 90%, still the results were not stable and fluctuated very much. We have to work more on the model so that we can get a very stable output.

Next, we can work on voice activation in the project where the signs when showed in front of the camera, the system will play voice for the particular alphabet and letter.

9.2 Regarding Internship

I am truly grateful to Edunet Foundation organization and IBM for accepting my application for internship on Artificial Intelligence and Machine Learning. It was really a very much fruitful internship as I came to know how to make the dataset for the project and even execute the project with higher rate of accuracy. I would like to express my gratitude to the mentors of this project. They really helped me a lot to understand the approach of the project. Overall, I was very much happy to be able to do internship and learn so much from the mentors and the project. In future I would like to again apply for internship under Edunet Foundation.

CHAPTER 10

REFERENCES

1. Ankita Wadhawan, Parteek Kumar, P2020, “Deep learning-based sign language recognition system for static signs”, Springer Nature

2. Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, Benjamin Schrauwen, P2021, “Sign Language Recognition using Convolutional Neural Networks”

3. Md. Moklesur Rahman, Md. Shafiqul Islam, Md. Hafizur Rahman, Roberto Sassi, Massimo W. Rivolta, Md Aktaruzzamank, P2019, “A New Benchmark on American Sign Language Recognition using Convolutional Neural Network”, IEEE Xplore

4. Rachana Patil, Vivek Patil, Abhishek Bahuguna, Mr. Gaurav Datkhile, P2021, “Indian Sign Language Recognition using Convolutional Neural Network”, ICACC