ROYAL GLOBAL UNIVERSITY
GUWAHATI

"Sign Language Translation using Convolutional Neural Network"

A Progress Report

Submitted by

Harsh Gupta (202025010)

Khriesezo Peseyie (202025016)

Under the guidance of:

Mr. Spandan Kumar Barthakur

Department of Computer Science & Engineering

Royal School of Engineering & Technology

Royal Global University, Guwahati, Assam

(December, 2023)

ROYAL GLOBAL UNIVERSITY
GUWAHATI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING ROYAL SCHOOL OF ENGINEERING & TECHNOLOGY

Certificate of Approval

This is to certify that the project work entitled "Sign Language using Convolutional Neural Network" is a genuine piece of work carried out by the students: Khriesezo Peseyie, Roll No: 202025016 and Harsh Gupta, Roll No: 202025010 of 7th semester, B. Tech, Computer Science and Engineering Department, Royal School of Engineering and Technology. This project is carried out under my guidance and supervision and submitted for the partial fulfillment of the requirement for the degree of Bachelor of Engineering (CSE), under Royal Global University for the session 2022-2023.

Date: **Project Guide: Mr. Spandan Barthakur**

Place: Guwahati **Assistant Professor**

ROYAL GLOBAL UNIVERSITY

GUWAHATI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING ROYAL SCHOOL OF ENGINEERING & TECHNOLOGY

Forwarding Certificate

It is certified that the work contained in the report titled "**Sign Language Conversion using Convolutional Neural Network**" by **Khriesezo Peseyie** bearing Roll No. 202025016 and **Harsh Gupta** bearing Roll No. 202025010 of **Bachelor of Technology, 7th Semester** under the **Computer Science and Engineering**, **Royal School of Engineering and Technology**, under the guidance of **Mr. Spandan Barthakur, Assistant Professor** has been presented in a manner satisfactory to permit its acceptance as a prerequisite to the degree for which has been submitted.

Date: **Name of the HOD/ Coordinator**

Place: Guwahati(Designation)

ROYAL GLOBAL UNIVERSITY
GUWAHATI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING ROYAL SCHOOL OF ENGINEERING & TECHNOLOGY

Declaration by the Students

We, Khriesezo Peseyie and Harsh Gupta, bearing Roll Nos. 202025016 and 202025010, students of B.Tech, Computer Science and Engineering under Royal School of Engineering and Technology, hereby declare that this Final Semester Project Report/Dissertation entitled “Sign language Conversion by Convolutional Neural Network” is a bonafide project work undertaken by us, during the period of September 2023 to December 2023, as partial fulfillment of the requirements of the degree of Bachelor of Technology Computer Science of the Assam Royal Global University, Guwahati.

Further, we declare that this report has not been submitted by us elsewhere for the award of any Degree, Diploma or Certificate and not linked to any other qualification.

Date:

Place: Guwahati

Khriesezo Peseyie	**Harsh Gupta**
Roll no.: 2020250016	**Roll no.: 202025010**

Acknowledgement

This project has thus far been one of our most hard worked accomplishments of our time in education and would have been incomplete without the help of many acquaintances.

We give our gratitude and thanks to our guide; Mr. Spandan Barthakur, Assistant Professor, Department of Computer Science & Engineering, RSET for advising us in our endeavours and helping us complete the project. His patience and forgiving nature has helped in guiding us towards the proper direction of our work. Without his approval of our work, we would not be able to come up with better results in our models.

We would also like to give our thanks to our family, friends and teachers for their thoughtful words of support. Without them, this project would not be possible.

Abstract

The "Sign Language Translation using Convolutional Neural Network" project addresses the communication gap between the Deaf and Hard of Hearing community and the wider population by leveraging advanced machine learning techniques. Sign language, being a complex and expressive form of communication, poses unique challenges for automated translation. This project employs Convolutional Neural Networks (CNNs), a powerful subset of deep learning, to develop a robust system capable of recognizing and translating sign language gestures in real-time.

The key components of the project include the development of a comprehensive gesture recognition model trained on a diverse dataset of sign language gestures. This model is designed to generalize effectively across various signs, expressions, and users. The real-time translation system is implemented to process live video feeds, providing instantaneous conversion of sign language gestures into text or spoken language. The user interface is designed to be intuitive and user-friendly, catering to both sign language users and those unfamiliar with sign language, thereby fostering inclusive communication.

The project aims to contribute to a more connected and inclusive society by breaking down communication barriers. Its scalability and adaptability accommodate a wide range of sign languages and dialects, considering the linguistic diversity within the sign language community. The proposed solution holds the potential to enhance accessibility and promote meaningful interactions, ultimately fostering a world where individuals of all abilities can communicate effortlessly. The Sign Language Translation using Convolutional Neural Network project represents a significant step towards creating a more inclusive and accessible future.

TABLE OF CONTENT

LIST OF FIGURES

Chapter 1: Introduction

1.a Convolutional Neural Network

In a world characterized by diverse communication methods, bridging the gap between individuals with different abilities is crucial for fostering inclusivity. One such group facing communication challenges is the Deaf and Hard of Hearing community, for whom sign language serves as a primary mode of communication. Recognizing the importance of facilitating seamless communication between the Deaf and the hearing population, the project titled "Sign Language Translation using Convolutional Neural Network" aims to leverage advanced machine learning techniques to create an innovative solution.

Sign language is a rich and expressive form of communication, employing gestures, facial expressions, and body movements. Traditional methods of bridging the communication gap, such as interpreters or text-based communication, may not always be readily available or practical. This project seeks to harness the power of Convolutional Neural Networks (CNNs), a type of deep learning algorithm widely used in image recognition tasks, to automatically translate sign language gestures into text or spoken language.

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

1. Convolution Layer:

In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position.

As we continue this process we will create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour.

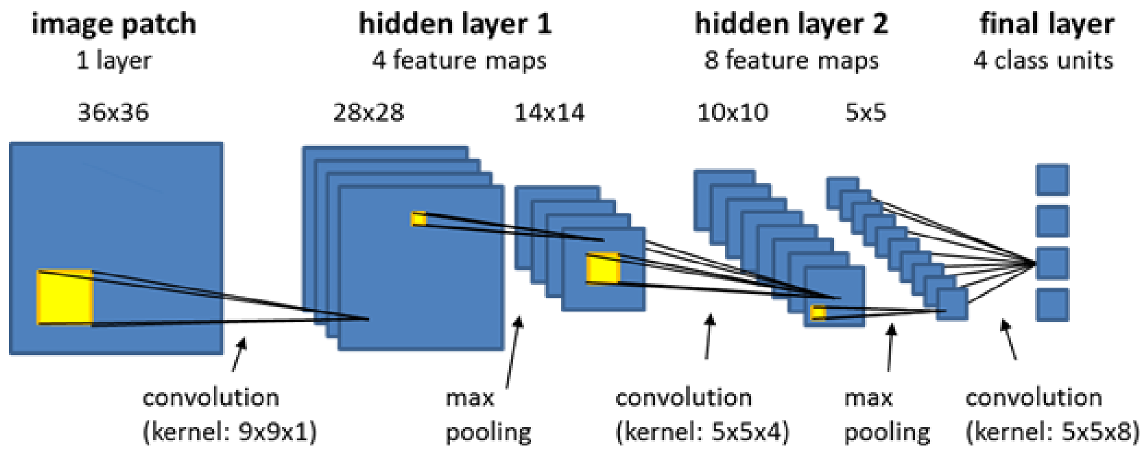


Fig.1.1 Convolutional Layer

2. Pooling Layer:

We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. There are two types of pooling:

a. Max Pooling: In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get an activation matrix half of its originalSize.

b. Average Pooling: In average pooling, we take advantage of of all Values in a window.

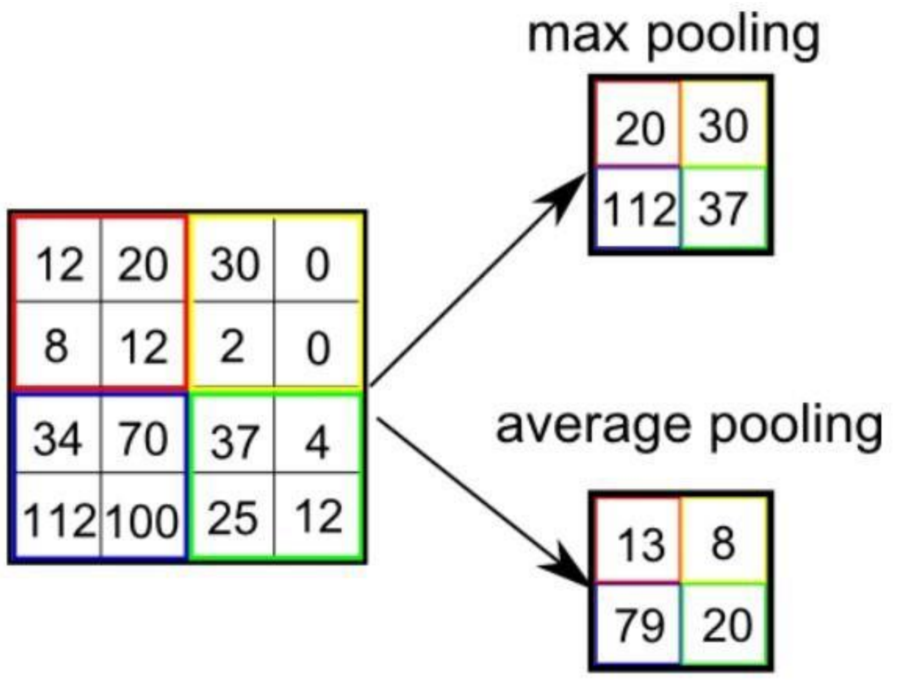


Fig.1.2 Pooling Layer

3. Fully Connected Layer:

In convolution layer, neurons are connected only to a local region, while in a fully connected region, we will connect all the inputs to neurons.

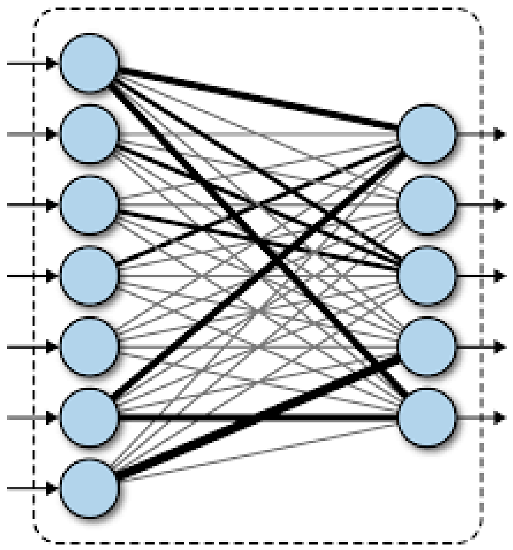


Fig.1.3 Fully Connected Layer

4. Final Output Layer:

After getting values from fully connected layer, we will connect them to the final layer of neurons that will predict the probability of each image to be in different classes.

1.b Problem Statement

The project's significance lies in its potential to transform lives. Deaf and Hard of Hearing individuals often face challenges in accessing information and participating fully in various aspects of life. Our system, through its ability to convert sign language into text and speech, offers a powerful tool for communication and integration. It opens doors to educational opportunities, facilitates effective communication in workplaces, and enhances social interactions. This project represents not only a technical achievement but also a commitment to fostering inclusivity and understanding in our society, emphasizing that everyone, regardless of their hearing abilities, deserves to be an active and engaged part of our diverse world.

1.c Motivation:

For interaction between normal people and D&M people a language barrier is created as sign language structure since it is different from normal text. So, they depend on vision-based communication for interaction.

If there is a common interface that converts the sign language to text, then the gestures can be easily understood by non-D&M people. So, research has been made for a vision-based interface system where D&M people can enjoy communication without really knowing each other's language.

The aim is to develop a user-friendly Human Computer Interface (HCI) where the computer understands the human sign language.

1.d Objectives

- **Sign Language Recognition** - To create a robust machine learning model capable of accurately recognizing a wide range of sign language gestures, ensuring precision and reliability in the conversion process.
- **English Text Generation** - To develop a natural language processing component that can seamlessly convert recognized sign language gestures into clear and coherent English text, preserving the intended meaning and nuances.
- **Speech Synthesis** - To integrate text-to-speech synthesis technology that enables the conversion of the generated English text into natural sounding speech, ensuring effecting communication with the hearing word.

Literature Review

[1] Sunitha K. A et al. , “Deaf Mute Communication Interpreter- A Review”

Communication between the deaf and non-deaf has always been a very cumbersome task. This paper aims to cover the various prevailing methods of deaf-mute communication interpreter system. The two broad classification of the communication methodologies used by the deaf –mute people are Wearable Communication Device and Online Learning System. Under Wearable communication method, there are Glove based system, Keypad method and Handicom Touchscreen. All the above mentioned three sub-divided methods make use of various sensors, accelerometer, a suitable microcontroller, a text to speech conversion module, a keypad and a touch-screen. The need for an external device to interpret the message between a deaf –mute and non-deaf-mute people can be overcome by the second method i.e online learning system. The Online Learning System has different methods under it, five of which are explained in this paper. The five sub-divided methods are SLIM module, TESSA, Wi-See Technology, SWI_PELE System and Web-Sign Technology. The working of the individual components used and the operation of the whole system for the communication purpose has been explained in detail in this paper.

[2] Mathavan Suresh Anand et al., “An Efficient Framework for Indian Sign Language Recognition Using Wavelet Transform”

Hand gesture recognition system is considered as a way for more intuitive and proficient human computer interaction tool. The range of applications includes virtual prototyping, sign language analysis and medical training. In this paper, an efficient Indian Sign Language Recognition System (ISLR) is proposed for deaf and dump people using hand gesture images. The proposed ISLR system is considered as a pattern recognition technique that has two important modules: feature extraction and classification. The joint use of Discrete Wavelet Transform (DWT) based feature extraction and nearest neighbour classifier is used to recognize the sign language. The experimental results show that the proposed hand gesture recognition system achieves maximum 99.23% classification accuracy while using cosine distance classifier.

[3] Pratibha Pandey et al., “Hand Gesture Recognition for Sign Language Recognition: A Review”

For deaf and dumb people, Sign language is the only way of communication. With the help of sign language, these physical impaired people express their emotions and thoughts to other person. Since for the common person, it is difficult to understand these language therefore often these physically challenged has to keep the translator along with them to communicate with the world. Hence sign language recognition has become empirical task. Since sign language consist of various movements and gesture of hand therefore the accuracy of sign language depends on the accurate recognition of hand gesture. This paper present various method of hand gesture and sign language recognition proposed in the past by various researchers.

[4] Nakul Nagpal et al., “Design Issue and Proposed Implementation of Communication Aid for Deaf & Dumb People”

Deaf and dumb people communicate among themselves using sign languages, but they find it difficult to communicate with the outside world. This paper proposes a full proof system to aid communication of deaf and dumb people communicating using Indian sign language (ISL) with normal people where hand gestures will be converted into appropriate text message. The hand gestures corresponding to ISL English alphabets, numbers and words will be captured through a webcam. Main objective is to design an algorithm to convert dynamic gesture to text at real time. Finally after testing is done the system will be implemented on android platform and will be available as an application for smart phone and tablet.

[5] Neelam K. Gilorkar et al., “Real Time Detection And Recognition Of Indian And American Sign Language Using Sift”

This paper emphases a real time vision based system for hand gesture recognition for human computer interaction in many applications. The system can recognize 35 different hand gestures given by Indian and American Sign Language or ISL and ASL at faster rate with virtuous accuracy. The gestures are classified on the basis of keypoints. RGB-to-GRAY segmentation technique was used to minimize the chances of false detection. We proposed a method of improvised Scale Invariant Feature Transform (SIFT) and same was used to extract features. The system is model using MATLAB. To design and efficient user friendly hand gesture recognition system, a GUI model has been implemented. The chief benefit of SIFT algorithm is its high processing speed which produced results in real time. The accuracy of system was computed up to 92-96%.

[6] Neelam K. Gilorkar et al., “A Review on Feature Extraction for Indian and American Sign Language”

In the field of human computer interaction (HCI) Sign Language has been the emphasis of significant research in real time. Such systems are advance and they are meant to substitute interpreters. Recently several techniques have been advanced in this area with the improvement of image processing and artificial intelligence techniques. Indian Sign Language(ISL) are double handed and hence it is more intricate compare to single handed American Sign Language(ASL). Many researchers use only single ASL or ISL sign for creating their database. In this paper recent research and development of sign language are reviewed based on manual communication and body language. Sign language recognition system typically elaborate three steps pre-processing, feature extraction and classification. Classification methods used for recognition are Neural Network(NN), Support Vector Machine(SVM), Hidden Markov Models(HMM), Scale Invariant Feature Transform (SIFT),etc.

[7] Priyanka Sharma, “Offline Signature Verification Using Surf Feature Extraction and Neural Networks Approach”

In this paper we will evaluate the use of SURF features in handwritten signature verification. For each known writer we will take a sample of three genuine signatures and extract their SURF descriptors. In this paper, off-line signature recognition & verification using neural network is proposed, where the signature is captured and presented to the user in an image format. Signatures are verified based on parameters extracted from the signature using various image processing techniques. The Off-line Signature Recognition and Verification is implemented using Matlab.

Chapter 3: Libraries Used

3.a CVZone

CVZone typically refers to a Python library known for computer vision and image processing tasks. However, it's important to note that the landscape of libraries and tools in the field of computer vision is dynamic, and there may have been changes or new developments since then.

CVZone might be associated with various functionalities related to computer vision, including image processing, object detection, facial recognition, and more. The library is likely designed to provide a set of tools and utilities to simplify the implementation of computer vision tasks, making it easier for developers to work with visual data.

Without the ability to browse the internet for real-time updates, I recommend checking the latest documentation or community resources associated with CVZone for the most accurate and current information. Developers often use such libraries to streamline the development process and leverage pre-built functions for common computer vision tasks. Here, we discuss the features and importance of CVZone:

1. **Purpose:**

   CVZone is a Python library designed for computer vision and image processing tasks.

2. **Functionality:**

   It provides a set of tools and utilities to simplify the implementation of various computer vision tasks.

3. **Tasks Covered:**

   CVZone likely includes functions for tasks such as image processing, object detection, facial recognition, and other common computer vision applications.

4. Ease of Use:

The library is aimed at simplifying the development process by offering pre-built functions and utilities, making it easier for developers to work with visual data.

5. Python-Based:

CVZone is implemented in Python, a widely-used language for machine learning and computer vision applications.

6. Community and Support:

Depending on the library's popularity and developer community, users may find support, documentation, and community-contributed resources.

7. Integration:

CVZone is likely designed to integrate seamlessly with other popular Python libraries for machine learning and image processing, such as OpenCV or NumPy.

8. Versatility:

ay cover a range of computer vision tasks, catering to the diverse needs of developers working on projects involving visual data.

9. Updates:

Like any library, CVZone may receive updates over time to incorporate new features, improvements, and compatibility with the latest technologies.

10. Documentation:

A well-documented library ensures that users can easily understand and implement its functionalities. Checking the documentation is essential for effectively using CVZone.

3.b TensorFlow

TensorFlow is an open-source machine learning library developed by the Google Brain team. It is designed to facilitate the development and deployment of machine learning models, particularly neural networks, across a variety of tasks. Here is an explanation of TensorFlow and its key features:

1. **Computational Graph:**

TensorFlow represents computations as directed graphs, known as computational graphs. Nodes in the graph represent operations, and edges represent the flow of data (tensors) between these operations.

2. **Tensors:**

Tensors are the fundamental data structures in TensorFlow. These are multi-dimensional arrays that can hold numerical data. Tensors flow through the computational graph, carrying data between operations.

3. **Ease of Use:**

TensorFlow provides high-level APIs (such as Keras) that make it easy to define, train, and deploy machine learning models. This abstraction simplifies the process for developers, allowing them to focus on model architecture and training rather than low-level implementation details.

4. **Flexibility:**

TensorFlow offers flexibility by allowing users to define and customize their models at various levels of abstraction. From high-level APIs for quick prototyping to low-level operations for fine-tuning, developers have the freedom to choose the level of abstraction that suits their needs.

5. **Neural Network Support:**

TensorFlow is widely used for building and training deep learning models, particularly neural networks. It provides a comprehensive set of tools for building various neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more.

6. **TensorFlow Ecosystem:**

TensorFlow has a rich ecosystem of tools and extensions. TensorBoard, for example, is a visualization tool that helps users understand and debug their models. TensorFlow Lite is designed for deploying models on mobile and edge devices, and TensorFlow.js enables running models in web browsers.

7. **Community and Documentation:**

TensorFlow has a large and active community of developers. The library is well-documented, making it accessible for both beginners and experienced machine learning practitioners. The community actively contributes to the development of TensorFlow, ensuring regular updates and improvements.

8. **Scalability:**

TensorFlow is designed to scale seamlessly, allowing users to train models on multiple GPUs or distributed computing environments. This scalability is crucial for handling large datasets and complex models.

9. **Open Source:**

TensorFlow is an open-source project, encouraging collaboration and contributions from the community. This open nature has contributed to its widespread adoption in both academia and industry.

In summary, TensorFlow is a powerful and versatile machine learning library that provides a comprehensive set of tools for developing and deploying machine learning models. Its flexibility, scalability, and active community make it a popular choice for researchers and practitioners in the field of machine learning and artificial intelligence.

TensorFlow requires a recent version of pip, so upgrade your pip installation to be sure you're running the latest version.

For GPU users:

pip install tensorflow[and-cuda]

For CPU users:

pip install tensorflow

Verify the installation:

For CPU:

python3 -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))"

For GPU:

python3 -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))"

If a tensor is returned, you've installed TensorFlow successfully.

3.c Keras

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier. Some important points of the Keras library include:

1. **User-Friendly API:**

   Known for its simplicity and ease of use, Keras provides a user-friendly interface that abstracts many complexities, making it accessible to beginners while remaining powerful for experienced researchers.

2. **Backend Agnostic:**

   Keras is designed to be backend-agnostic, allowing users to seamlessly switch between popular backend engines such as TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK).

3. **Modularity:**

   The library is modular, enabling users to construct neural network models layer by layer. This modular approach allows for easy model customization and experimentation.

4. **Model Types:**

   Keras supports the creation of various types of models, including sequential models for simple linear stacks of layers and functional API for more complex model architectures.

5. **Wide Adoption:**

Originally developed as a user-friendly interface atop Theano, Keras gained immense popularity and became the official high-level API for TensorFlow, solidifying its place as a widely used tool in the deep learning community.

6. **Extensibility:**

Users can extend Keras by developing custom layers, loss functions, and other components. This extensibility enables the integration of specialized functionality into neural network models.

7. **Pre-processing Capabilities:**

Keras includes utilities for data preprocessing, such as image and text preprocessing. These tools facilitate the preparation of data for training and testing.

8. **Integration with TensorFlow:**

Keras seamlessly integrates with TensorFlow, allowing users to leverage TensorFlow's powerful features while benefiting from Keras's high-level abstractions.

9. **Community and Documentation:**

With an active community, Keras has extensive documentation, tutorials, and a wealth of online resources. This community support is valuable for both beginners and experienced practitioners.

10. **Standardization:**

Keras has become a de facto standard for building neural networks due to its simplicity and versatility. Its API design principles have influenced the development of high-level APIs in other deep learning frameworks.

11. **Portability:**

Keras models can be easily saved, serialized, and shared. This portability is crucial for deploying models across different environments.

Keras's combination of simplicity, modularity, and powerful features has contributed to its widespread adoption in the deep learning community. Whether for rapid prototyping or complex model development, Keras provides a versatile and user-friendly platform for building and experimenting with neural networks.

You can install Keras from PyPI via:

```
pip install —upgrade keras
```

3.d OpenCV (CV2)

OpenCV(Open Source Computer Vision) or CV2 is an open source library of programming functions used for real-time computer-vision. It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE.

OpenCV was initially developed by Intel in 1999 and later supported by Willow Garage and Itseez.

Here are some key aspects and features of OpenCV:

1. **Image Processing:**

   OpenCV includes a wide range of functions for basic and advanced image processing tasks. This includes operations like resizing, cropping, filtering, thresholding, and more.

2. **Computer Vision:**

   OpenCV is widely used in computer vision applications. It provides functionalities for feature detection, object recognition, image stitching, camera calibration, and geometric transformations.

3. **Machine Learning:**

OpenCV integrates machine learning algorithms and tools. It supports popular machine learning frameworks like TensorFlow and PyTorch. OpenCV's machine learning capabilities include object detection, face recognition, and image classification.

4. **Camera Calibration:**

OpenCV provides tools for camera calibration, which is essential in computer vision applications. Calibration helps correct lens distortion and provides accurate measurements in real-world units.

5. **Object Detection:**

OpenCV includes pre-trained models for object detection using deep learning techniques. These models can detect and recognize objects in images or video streams.

6. **Video Analysis:**

OpenCV supports video processing, allowing developers to analyze and manipulate video streams. It includes functionalities for video capture, playback, and processing.

7. **Cross-Platform Support:**

OpenCV is cross-platform and can be used on various operating systems, including Windows, Linux, and macOS. It also has interfaces for different programming languages such as C++, Python, and Java.

8. **Community and Documentation:**

OpenCV has a large and active community of developers. The library is well-documented, making it easier for users to understand and implement various functionalities.

9. **Open Source:**

OpenCV is released under the BSD license, which makes it free to use, modify, and distribute. This open-source nature has contributed to its widespread adoption and continuous development.

OpenCV is employed in a diverse range of applications, including robotics, augmented reality, medical image analysis, autonomous vehicles, and more. Its versatility and comprehensive set of tools make it a popular choice for researchers, developers, and engineers working in computer vision and related fields.

- Below Python packages are to be downloaded and installed to their default locations.
    - Python 3.x (3.4+) or Python 2.7.x from here.
    - Numpy package (for example, using pip install numpy command).
    - Matplotlib (pip install matplotlib) (Matplotlib is optional, but recommended since we use it a lot in our tutorials).

- Install all packages into their default locations. Python will be installed to C:/Python27/ in case of Python 2.7.

- After installation, open Python IDLE. Enter import numpy and make sure Numpy is working fine.

- Download latest OpenCV release from GitHub or SourceForge site and double-click to extract it.

- Go to opencv/build/python/2.7 folder.

- Copy cv2.pyd to C:/Python27/lib/site-packages.
- Copy the opencv_world.dll file to C:/Python27/lib/site-packages
- Open Python IDLE and type following codes in Python terminal.

```
import cv2 as cv
print( cv.__version__ )
```

3.e Media Pipe

MediaPipe is an open-source framework developed by Google that provides a comprehensive set of tools for building applications with real-time perception capabilities, particularly in the fields of computer vision and machine learning. MediaPipe simplifies the development of applications that involve the processing of multimedia data, such as video and audio streams.

MediaPipe allows developers to build complex pipelines for processing multimedia data using modular components called graphs. These graphs are composed of reusable building blocks, making it easier to create and experiment with different configurations.

Key features of MediaPipe include:

1. **Cross-Platform Support:**

   MediaPipe is designed to be cross-platform, allowing developers to create applications for different operating systems, including Android, iOS, Linux, and Windows.

2. **Modular Architecture:**

   MediaPipe follows a modular architecture, which means that developers can use individual components (called "calculators") for specific tasks without having to use the entire framework. This modularity makes it flexible and adaptable to a variety of use cases.

3. Pre-Trained Models:

MediaPipe comes with pre-trained models for various perception tasks. These models are trained on large datasets and can be easily integrated into applications, saving developers time and resources.

4. Efficient Processing:

MediaPipe is designed for real-time processing, and it aims to provide efficient solutions for demanding applications. It utilizes techniques like optimized algorithms and hardware acceleration to achieve real-time performance.

5. Media Processing Pipeline:

MediaPipe introduces the concept of a media processing pipeline, where data flows through a series of processing stages. Each stage performs a specific task, and the pipeline allows developers to chain together different components to create complex applications.

6. Community and Documentation:

MediaPipe has an active community of developers, and it is supported by extensive documentation. This makes it easier for new users to get started and for experienced developers to find resources and support.

7. Customization:

MediaPipe provides developers with the ability to customize and extend the functionality according to their specific needs. This flexibility is useful for creating applications with unique requirements.

8. **Integration with TensorFlow Lite:**

MediaPipe integrates with TensorFlow Lite, allowing developers to use machine learning models created with TensorFlow within the framework. This enables the development of applications that leverage the power of machine learning.

MediaPipe is often used in applications such as augmented reality, gesture recognition, body tracking, and other real-time computer vision scenarios. Its flexibility and efficiency make it a valuable tool for developers working on projects that involve visual perception and analysis.

To install MediaPipe on Windows, you can use the following steps:

- **Install Required Dependencies:**
    - Open a command prompt or PowerShell window as an administrator.
    - Run the following command to install required dependencies:

      pip install mediapipe

- **Verify Installation:**
    - After the installation is complete, you can verify that MediaPipe is installed correctly by creating a simple Python script.
    - Create a new Python file, for example, test_mediapipe.py.
    - Open the file in a text editor and add the following code:

```
import mediapipe as mp
# Create a MediaPipe Hands object
hands = mp.solutions.hands
# Create a MediaPipe Drawing object
mp_drawing = mp.solutions.drawing_utils
# Initialize the Hands model
with hands.Hands() as hands_model:
    # Your code here
pass
```

- **Run the Script:**

  Open a command prompt or PowerShell window and navigate to the directory where you saved test_mediapipe.py.

```
python test_mediapipe.py
```

If everything is installed correctly, you should see the script run without errors. This script initializes the MediaPipe Hands model, and you can modify it to perform various hand tracking tasks.

3.f Tkinter

Tkinter is a standard Python library for creating graphical user interfaces (GUIs). It provides a set of tools and widgets (graphical elements) that enable developers to build desktop applications with a graphical user interface. Tkinter is based on the Tk GUI toolkit, which originated as a part of the Tcl (Tool Command Language) scripting language.

Key features and concepts of Tkinter include:

1. **Widgets:** Tkinter provides a variety of widgets that can be used to create different GUI elements, such as buttons, labels, text boxes, canvas, frames, and more. These widgets serve as the building blocks for constructing GUIs.

2. **Geometry Management:** Tkinter includes geometry managers, such as pack, grid, and place, which help organize and arrange widgets within the application window. These managers simplify the layout of the GUI components.

3. **Event Handling:** Tkinter supports event-driven programming. Events, such as button clicks or keypresses, can be bound to specific functions or methods, allowing developers to respond to user interactions.

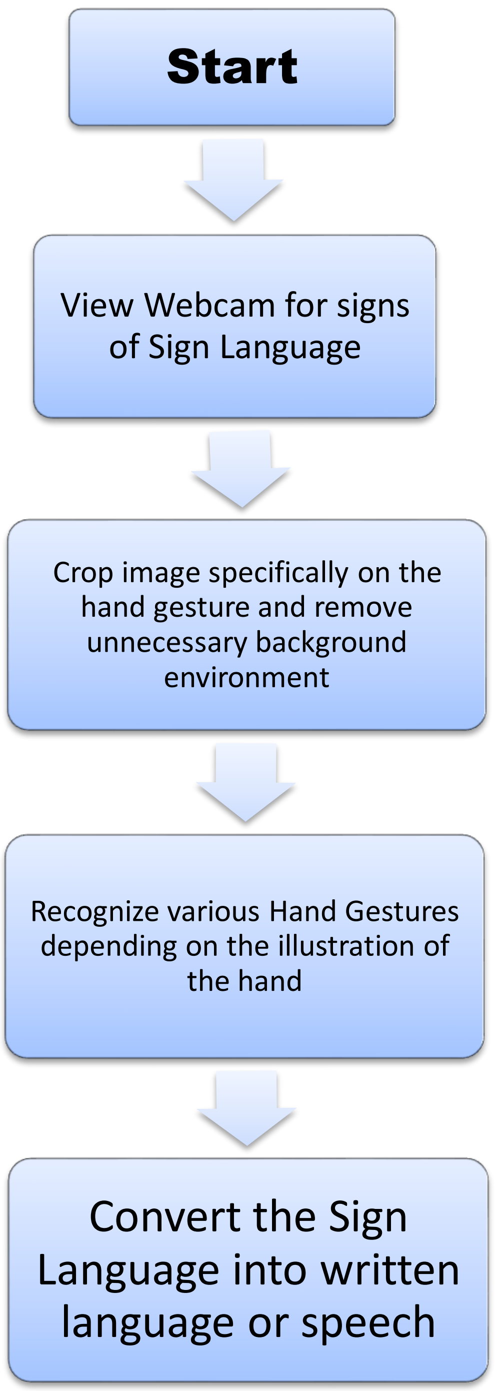
4. **Modal and Modeless Dialogs:** Tkinter supports both modal and modeless dialog boxes. Modal dialogs require user interaction before returning control to the application, while modeless dialogs allow the user to interact with the main window while the dialog is open.

5. **Cross-Platform:** Tkinter is included with Python and is available on most platforms, including Windows, macOS, and Linux. This makes it a convenient choice for cross-platform desktop application development.

6. **Simple Syntax:** Tkinter's syntax is straightforward and easy to learn, making it accessible for beginners. The library follows an object-oriented approach, and creating a basic GUI involves creating and configuring widget objects.

Chapter 4: Methodology

4.a Flowchart



Start
View Webcam for signs of Sign Language
Crop image specifically on the hand gesture and remove unnecessary background environment
Recognize various Hand Gestures depending on the illustration of the hand
Convert the Sign Language into written language or speech

Fig. 4.1 Flowchart of the Project

4.b Algorithm

The algorithm of the project is as follows:

1. **Import Libraries:**
    - Import necessary libraries, including glob for file path handling, matplotlib for plotting, and Keras/TensorFlow for building and training the CNN model.

2. **Load InceptionV3 Model:**
    - Initialize the InceptionV3 model with pre-trained weights from ImageNet.
    - Freeze the weights of the pre-trained layers to prevent them from being updated during training.

3. **Modify the Model Architecture:**
    - Flatten the output of the InceptionV3 model.
    - Add a Dense layer with softmax activation for predicting classes.
    - Create the final model by specifying the input and output layers.

4. **Compile the Model:**
    - Compile the model using categorical cross-entropy as the loss function, Adam optimizer, and accuracy as the evaluation metric.

5. **Data Augmentation:**
    - Use ImageDataGenerator to augment the training data by applying transformations like rescaling, shearing, zooming, and horizontal flipping.

6. **Load Training and Test Data:**
    - Load the training and test datasets using flow_from_directory, specifying the target size, batch size, and categorical class mode.

7. **Model Training:**
    - Train the model using the fit_generator function, providing the training set, validation set, number of epochs, steps per epoch, and validation steps.

8. **Plot Training History:**
    - Plot and visualize the training and validation loss over epochs.
    - Save the loss plot as 'LossVal_loss.png.'

9. **Plot Accuracy:**
    - Plot and visualize the training and validation accuracy over epochs.
    - Save the accuracy plot as 'AccVal_acc.png.'

10. **Save the Model:**
    - Save the trained model in the 'sign.h5' file.

END

4.c Functions & Result

The following functions and code play an important role in the working of the Program:

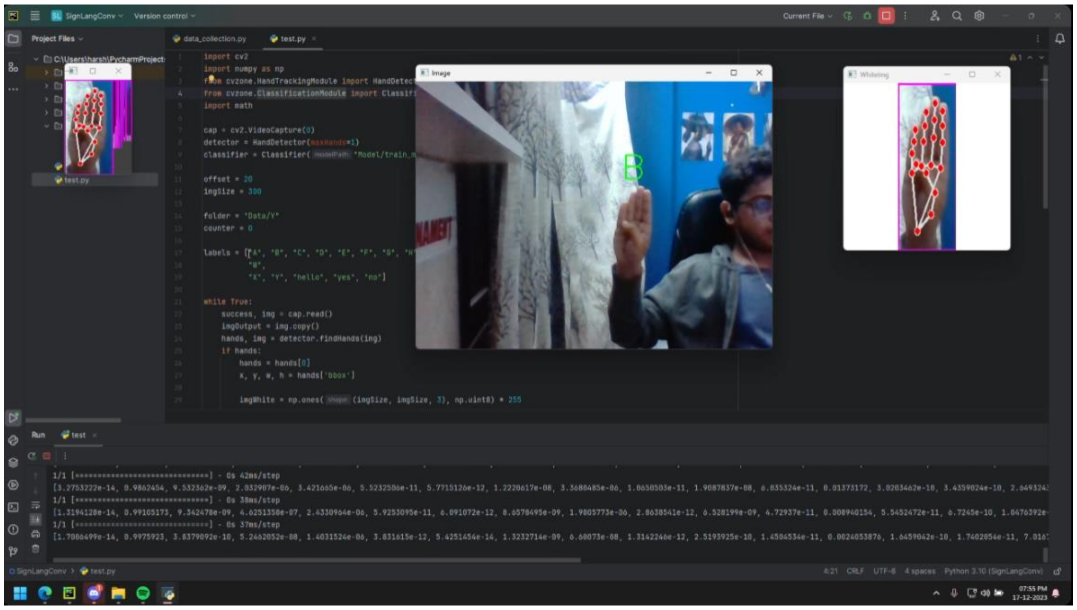
Dataset Generation and Pre-Processing –

We created a python file data_collection.py where the process of data collection and pre processing will take place.

First, we import the necessary libraries, including OpenCV for computer vision tasks, NumPy for numerical operations and the HandDetector module from cvzone library for hand tracking.

This python code aims to generate a dataset for hand images, and the code uses a webcam **cv2.VideoCapture(0)** to capture video frames.

cap = cv2.VideoCapture(0)



```
import cv2
import numpy as np
from cvzone.HandTrackingModule import HandDetect
from cvzone.ClassificationModule import Classifi
import math

cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
classifier = Classifier( modelPath: "Model/train_m

offset = 20
imgSize = 300

folder = "Data/Y"
counter = 0

labels = ["A", "B", "C", "D", "E", "F", "G", "H"
          "W",
          "X", "Y", "hello", "yes", "no"]

while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
    if hands:
        hands = hands[0]
        x, y, w, h = hands['bbox']

        imgWhite = np.ones( shape: (imgSize, imgSize, 3), np.uint8) * 255
```

```
1/1 [==============================] - 0s 42ms/step
[3.2753222e-14, 0.9862454, 9.532362e-09, 2.832907e-06, 3.421665e-06, 5.5232506e-11, 5.7715126e-12, 1.2220617e-08, 3.3680485e-06, 1.0650503e-11, 1.9087837e-08, 6.035324e-11, 0.01373172, 3.8203462e-10, 3.4359024e-10, 2.649324
1/1 [==============================] - 0s 38ms/step
[1.3194128e-14, 0.99105173, 9.342478e-09, 4.6251358e-07, 2.4330964e-06, 5.9253095e-11, 6.091072e-12, 8.6578495e-09, 1.9805773e-06, 2.8638541e-12, 6.528199e-09, 4.72937e-11, 0.008940154, 5.5452472e-11, 6.7245e-10, 1.0476392e-
1/1 [==============================] - 0s 37ms/step
[1.7006499e-14, 0.9975923, 3.8379092e-10, 5.2462052e-08, 1.4031524e-06, 3.831615e-12, 5.4251456e-14, 1.3232714e-09, 6.60073e-08, 1.3142246e-12, 2.5193925e-10, 1.4504534e-11, 0.0024053876, 1.6459042e-10, 1.7402054e-11, 7.016
```

Fig. 4.2 Webcam Capture Window

A handDetector object is instantiated to identify and track hands and track hands in the video. The maxHands=1 parameter ensures that only one hand is detected at a time.Several variables are defined to control the cropping and resizing of the captured hand images i.e. offset=20 and imgSize=300

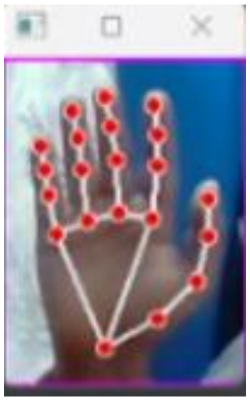


Fig. 4.3 Uncropped Image Window

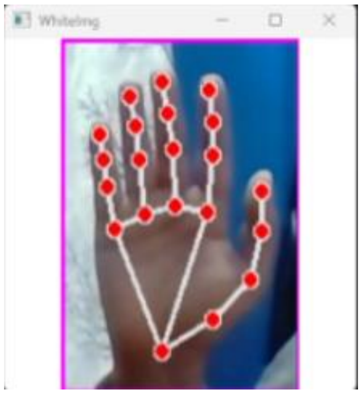
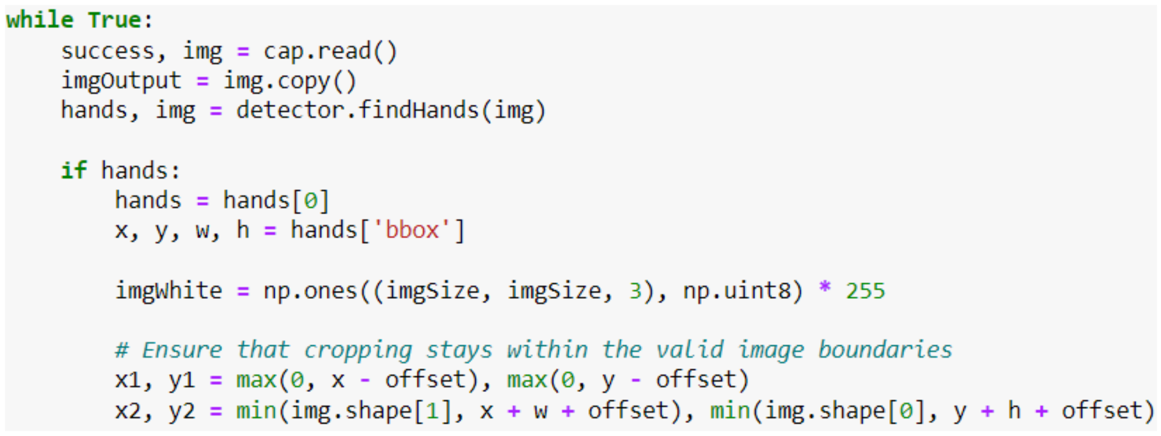


Fig. 4.4 Cropped Image Window

The variable 'folder' specifies the directory where the images will be saved and 'counter' variable keeps track of the number of saved images.

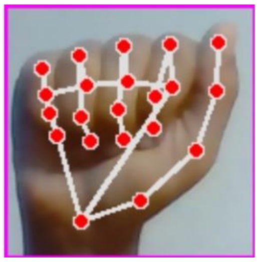
The code then enters a loop to continuously capture video frames and process them for hand detection. Within the loop, the code checks if hands are detected. If hands are present, it proceeds to crop, resize, and save the images.

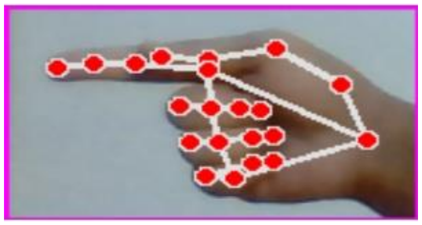


```
while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)

    if hands:
        hands = hands[0]
        x, y, w, h = hands['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255

        # Ensure that cropping stays within the valid image boundaries
        x1, y1 = max(0, x - offset), max(0, y - offset)
        x2, y2 = min(img.shape[1], x + w + offset), min(img.shape[0], y + h + offset)
```

Fig. 4.5 Dataset Capturing Code

Then it calculates the aspect ratio of the detected hand and resizes the image accordingly to fit a square canvas. Finally, the code displays the original image, the cropped hand image, and the resized hand image. A picture of the hand is taken by pressing the ‘s’ button on the keyboard, the code increments the counter, saves the resized hand image, and prints the current count.





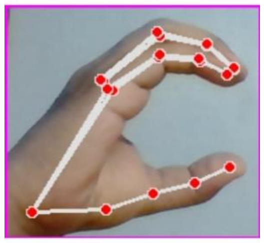


Fig. 4.6 Dataset ‘M’ **Fig. 4.7** Dataset ‘G’ **Fig. 4.8** Dataset ‘C’

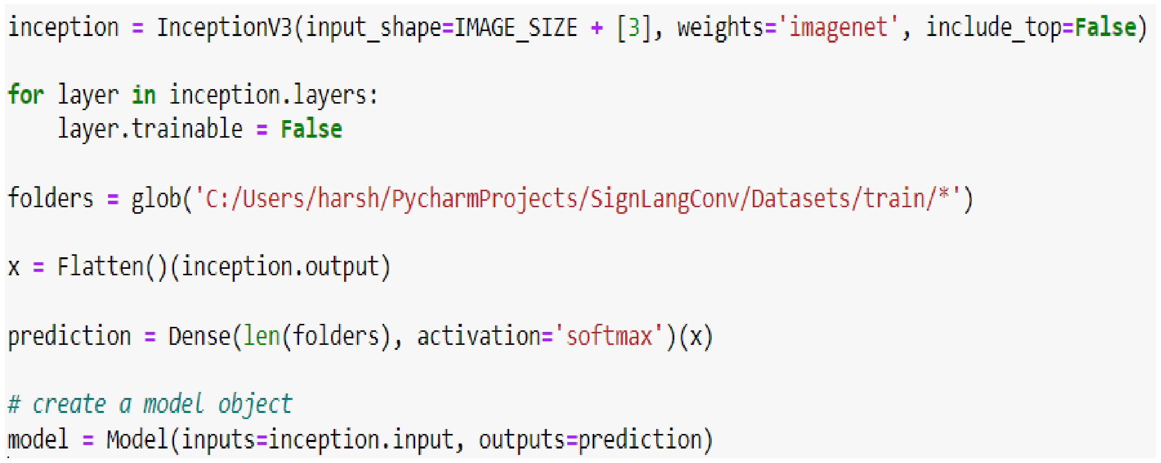
Training –

We created a python file train.py where the process of training our dataset will take place.

First we import the necessary libraries, such as glob for file path handling, matplotlib for plotting, and components from Keras and TensorFlow for building and training the model.

Then we define the path for the training and validation datasets.

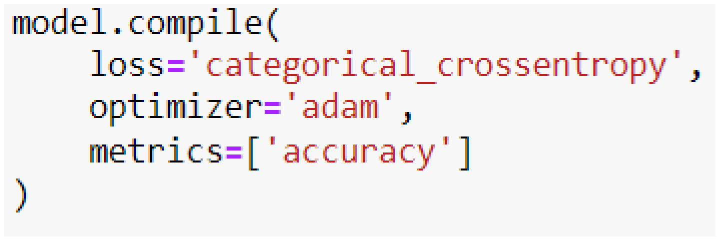
Next, the script loads the InceptionV3 model with pre-trained weights from ImageNet. All layers of the model are set to non-trainable to retain the pre-existing knowledge. The script then adds custom output layers to the InceptionV3 model to suit the specific requirements of the sign language classification task. This includes flattening the output and adding a dense layer with softmax activation for classification. The prediction layer estimates how likely the image will fall under one of the classes. So, the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using SoftMaxfunction.



```
inception = InceptionV3(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

for layer in inception.layers:
    layer.trainable = False

folders = glob('C:/Users/harsh/PycharmProjects/SignLangConv/Datasets/train/*')

x = Flatten()(inception.output)

prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=inception.input, outputs=prediction)
```

Fig. 4.9 Inception model code

The model is compiled with categorical crossentropy loss, the Adam optimizer, and accuracy as the evaluation metric.

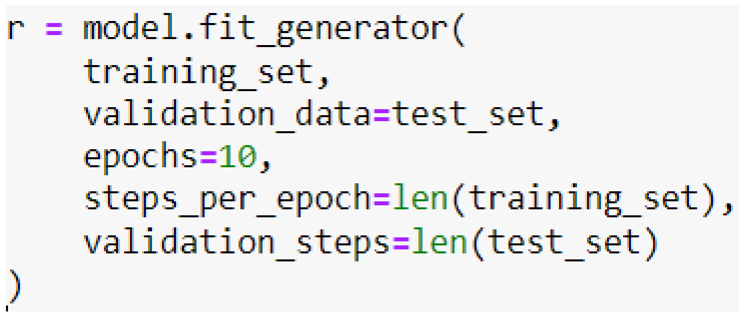


```
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Fig. 4.10 Model Compilation Code

Data augmentation is applied to the training set using the **ImageDataGenerator** from TensorFlow's Keras API. This includes rescaling, shearing, zooming, and horizontal flipping. The training and test datasets are loaded and augmented using the specified **ImageDataGenerator**. Images are resized to match the defined target size, and batches of 32 are used during training and testing.

The model is then trained using the **fit_generator** method with 10 epochs and the number of steps per epoch set to the length of the training set.



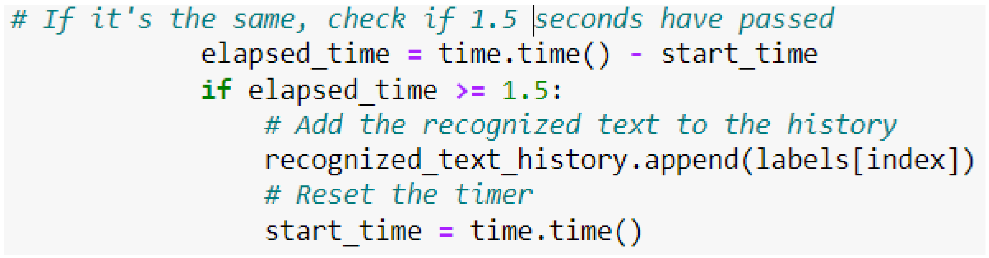
```
r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=10,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)
```

Fig. 4.11 Model Trainer Code

Finally, the script plots and saves the training and validation loss curves as well as the accuracy curves. In the end, the model is saved in the HDF5 format for future use and deployment.

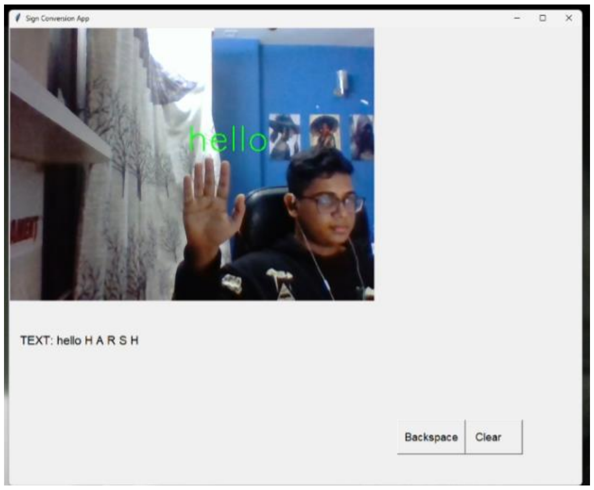
Optimizer - The Adam optimizer also known as Adaptive Moment Estimation, is a popular optimization algorithm used in training deep neural networks, Adam combines the advantages of two other optimization techniques: RMSprop and momentum. Adam maintains two moving averages for each parameter during training: the first moment (mean) and the second moment (uncentered variance). These moments are calculated using exponentially decaying averages of past gradients and squared gradients, respectively. The algorithm also incorporates bias correction to account for the initialization bias in these moving averages.

Output – The script uses a timer to control how often the recognized gesture is added to the history. If the same gesture is recognized for 1.5 seconds, it is added to the history. If a different gesture is recognized, the recognized text is updated, and the timer is reset.



```
# If it's the same, check if 1.5 seconds have passed
            elapsed_time = time.time() - start_time
            if elapsed_time >= 1.5:
                # Add the recognized text to the history
                recognized_text_history.append(labels[index])
                # Reset the timer
                start_time = time.time()
```

Fig. 4.12 Timer for Recognition Code



TEXT: hello H A R S H

Fig. 4.13 Sign Language Conversion Output

Chapter 5: Training and Testing

- We feed the input images after preprocessing to our model for training and testing.
- At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data.
- A graphical image is then generated to showcase the accuracy and loss of the trained model.

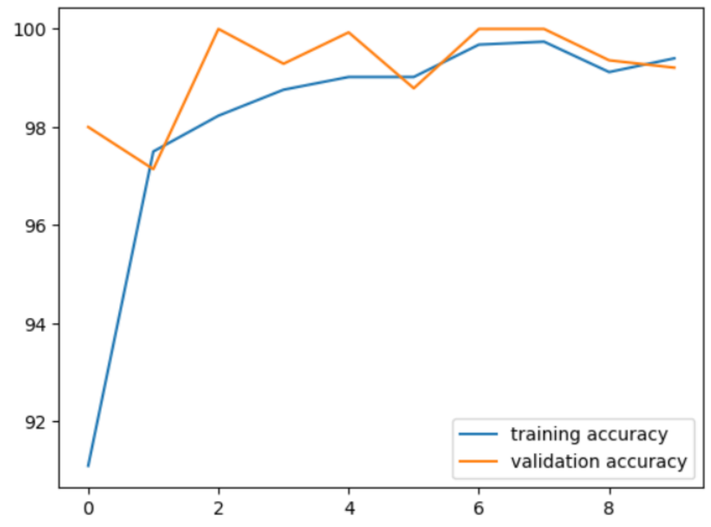


Fig. 5.1 Accuracy Graph

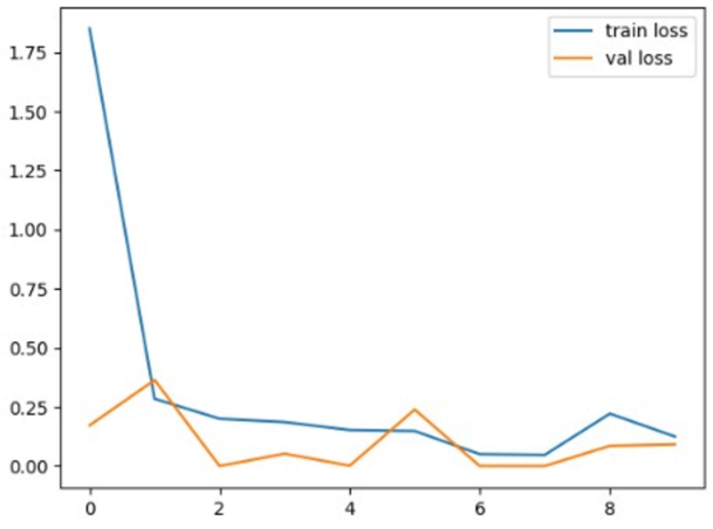
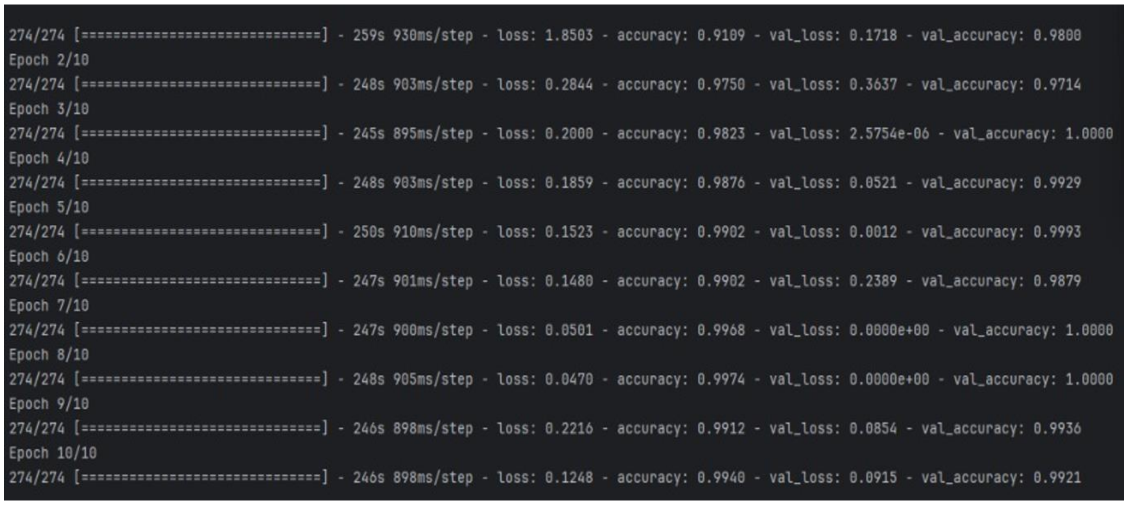


Fig. 5.2 Loss Graph

- Epoch refers to one complete pass through the entire training dataset during the training phase of a neural network. In other words, during one epoch, the algorithm processes the entire dataset once, forward and backward, to update the model's parameters.

- An epoch is a complete iteration over the entire training dataset. In each epoch, the model sees each training example once and updates its parameters accordingly. The number of epochs is a hyperparameter that the data scientist or machine learning engineer can set based on the specific problem and the characteristics of the dataset.



```
274/274 [==============================] - 259s 930ms/step - loss: 1.8503 - accuracy: 0.9109 - val_loss: 0.1718 - val_accuracy: 0.9800
Epoch 2/10
274/274 [==============================] - 248s 903ms/step - loss: 0.2844 - accuracy: 0.9750 - val_loss: 0.3637 - val_accuracy: 0.9714
Epoch 3/10
274/274 [==============================] - 245s 895ms/step - loss: 0.2000 - accuracy: 0.9823 - val_loss: 2.5754e-06 - val_accuracy: 1.0000
Epoch 4/10
274/274 [==============================] - 248s 903ms/step - loss: 0.1859 - accuracy: 0.9876 - val_loss: 0.0521 - val_accuracy: 0.9929
Epoch 5/10
274/274 [==============================] - 250s 910ms/step - loss: 0.1523 - accuracy: 0.9902 - val_loss: 0.0012 - val_accuracy: 0.9993
Epoch 6/10
274/274 [==============================] - 247s 901ms/step - loss: 0.1480 - accuracy: 0.9902 - val_loss: 0.2389 - val_accuracy: 0.9879
Epoch 7/10
274/274 [==============================] - 247s 900ms/step - loss: 0.0501 - accuracy: 0.9968 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/10
274/274 [==============================] - 248s 905ms/step - loss: 0.0470 - accuracy: 0.9974 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/10
274/274 [==============================] - 246s 898ms/step - loss: 0.2216 - accuracy: 0.9912 - val_loss: 0.0854 - val_accuracy: 0.9936
Epoch 10/10
274/274 [==============================] - 246s 898ms/step - loss: 0.1248 - accuracy: 0.9940 - val_loss: 0.0915 - val_accuracy: 0.9921
```

Fig. 5.3 Epoch List

Chapter 6: Result

- We have achieved an accuracy of **98.76%** in our model using the fit generator model. Which is better accuracy than most of the current research papers on American Sign Language
- Training of model was done 4 times. The fourth time the results were up to the mark before there high signs of errors and wrong readings.
- Used a new symbol which applied the spacebar function.
- Static data were used for J and Z letters since the model does not support movement based signs.

- Below is the confusion matrix for our results -

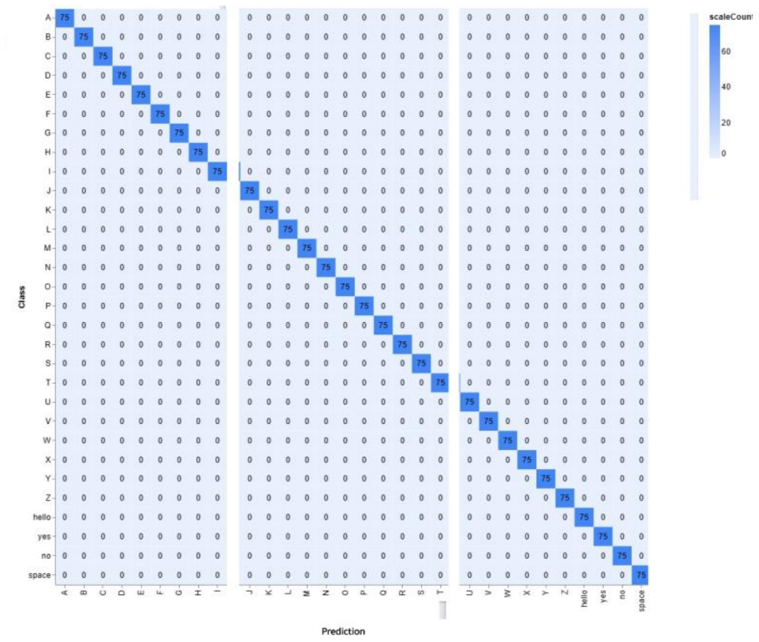


Fig. 6.0 Confusion Matrix

Chapter 7: Future Scope

- To achieve higher accuracy even in case of complex backgrounds and dark places using various background subtraction algorithms.
- To improve the preprocessing to predict gestures in low light conditions with a higher accuracy.
- The project can be build into a web/mobile application to conveniently access the project and use it.
- Currently the project only supports for ASL conversion but with the right data and training the project can be used for various other sign conversion.

Chapter 8: References

[1] Sunitha K. A, Anitha Saraswathi.P, Aarthi.M, Jayapriya. K, Lingam Sunny, “Deaf Mute Communication Interpreter- A Review”, International Journal of Applied Engineering Research ,Volume 11, pp 290-296, 2016.

[2] Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan, “An Efficient Framework for Indian Sign Language Recognition Using Wavelet Transform” Circuits and Systems, Volume 7, pp 1874-1883, 2016.

[3] Pratibha Pandey, Vinay Jain, “Hand Gesture Recognition for Sign Language Recognition: A Review”, International Journal of Science, Engineering and Technology Research (IJSETR), Volume 4, Issue 3, March 2015.

[4] Nakul Nagpal,Dr. Arun Mitra.,Dr. Pankaj Agrawal, “Design Issue and Proposed Implementation of Communication Aid for Deaf & Dumb People”, International Journal on Recent and Innovation Trends in Computing and Communication ,Volume: 3 Issue: 5,pp- 147 – 149.

[5] Neelam K. Gilorkar, Manisha M. Ingle, “Real Time Detection And Recognition Of Indian And American Sign Language Using Sift”, International Journal of Electronics and Communication Engineering & Technology (IJECET), Volume 5, Issue 5, pp. 11-18, May 2014.

[6] Neelam K. Gilorkar, Manisha M. Ingle, “A Review on Feature Extraction for Indian and American Sign Language”, International Journal of Computer Science and Information Technologies, Volume 5 (1) , pp314-318, 2014.

[7] Priyanka Sharma, “Offline Signature Verification Using Surf Feature Extraction and Neural Networks Approach”, International Journal of Computer Science and Information Technologies, Volume 5 (3) , pp 3539-3541, 2014.