

Abstract

Sign language serves as a vital means of communication for individuals with hearing and speech impairments. However, the limited understanding of sign language among the general population creates communication barriers. This study focuses on developing a deep learning-based system for real-time sign language detection to bridge this gap. Convolutional Neural Networks (CNNs) are employed to recognize hand gestures and classify them into corresponding signs. The model is trained on a diverse dataset of hand gestures, incorporating image preprocessing techniques such as background removal, normalization, and augmentation to improve accuracy and generalization. Feature extraction and classification are performed using deep learning architectures, ensuring high precision and robustness.

The sign language detection system using deep learning primarily involves data collection, preprocessing, model training, and real-time prediction. The process begins with gathering a dataset of sign language gestures, which can include static images or video frames. These images undergo preprocessing techniques such as background removal, grayscale conversion, normalization, and data augmentation to improve model robustness. A Convolutional Neural Network (CNN) is used for feature extraction and classification. CNNs automatically learn spatial hierarchies of features from images, making them highly effective for gesture recognition. The model is trained using labeled sign language data, where it learns to associate specific hand gestures with corresponding words or letters. For real-time detection, a webcam or mobile camera captures the user's gestures, which are then processed and classified using the trained deep learning model. The recognized sign is displayed as text or speech output, enabling effective communication. The system can be deployed in mobile apps or assistive devices, significantly improving accessibility for the deaf and hard-of-hearing community. Future improvements may include multi-language support and dynamic gesture recognition.

Index

Sr No.	Topic	Page No.
1.	Introduction	1-5
2.	Literature Survey	6-8
3.	Problem Statement	9-11
4.	Methodology	12-17
5.	Implementation	18-122
6.	Result	23-29
7.	Conclusion	30
8.	Code Snippet	31-32
9.	References	33

1. Introduction

Sign language is a visual language that is primarily used by people who are deaf or hard of hearing. It involves using hand gestures, facial expressions, and body language to convey meaning and communicate. While sign language is an important means of communication for the deaf and hard of hearing community, it can be challenging for those who do not know the language to understand and communicate with them. In recent years, there has been a growing interest in developing sign language detection and recognition systems using computer vision and machine learning techniques. These systems have the potential to improve accessibility and communication for the deaf and hard of hearing community, by automatically recognizing sign language gestures and translating them into text or speech.

A Convolutional Neural Network (CNN) is a deep learning algorithm extensively used for image recognition, object detection, and gesture recognition. CNNs are particularly effective in sign language detection, as they analyze image patterns to recognize hand gestures, facial expressions, and body movements. Unlike traditional machine learning methods, CNNs automatically extract relevant features from images, eliminating the need for manual feature engineering. This makes them highly efficient for detecting sign language gestures, as they can process visual information with high accuracy and scalability.

CNNs are preferred for sign language detection due to their ability to automatically extract features, capture spatial hierarchies, and handle variations in lighting and backgrounds. They analyze hand positions, shapes, and movement patterns, ensuring reliable gesture recognition. Since sign language is inherently visual, CNNs can process images and videos frame by frame to detect gestures accurately.

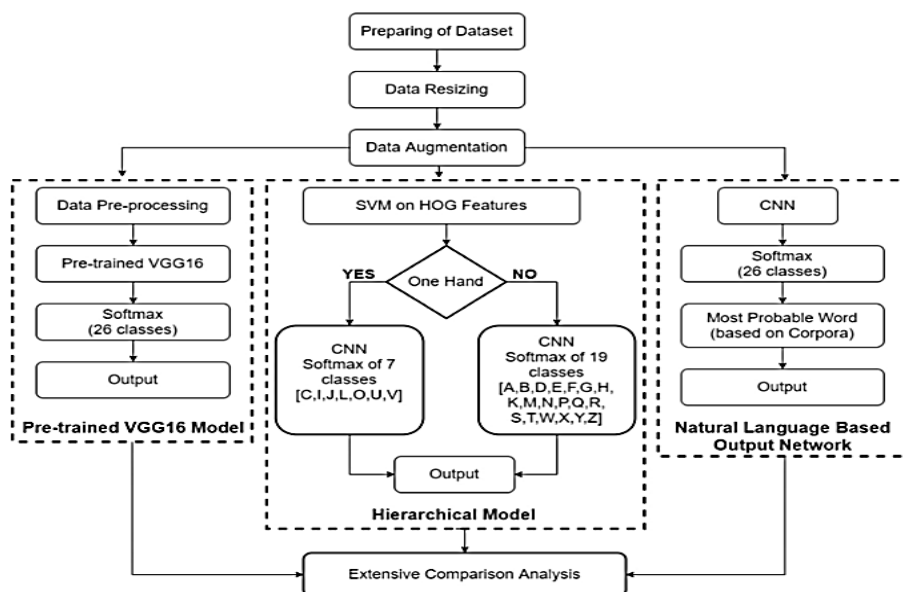
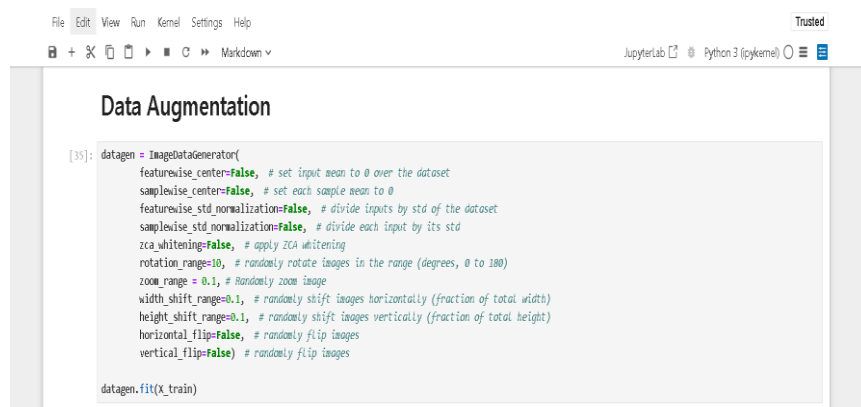


Fig 1.1 Sign Language Detection Technique using CNN Algorithm

As shown in Figure 1.1, the project will be structured into 3 distinct functional blocks, Data Resizing, Data Augmentation and Data Processing. The block diagram is simplified in detail to abstract some of the minutiae:

Data Resizing:- Data resizing is a crucial preprocessing step in sign language detection to ensure all images have a uniform size before feeding them into a deep learning model. Since sign language datasets often contain images of varying resolutions, resizing standardizes the dimensions, improving model efficiency and accuracy. Using Jupyter Notebook, we can resize images using Python libraries such as OpenCV and TensorFlow/Keras. The first step is to import necessary libraries, including OpenCV for image processing, NumPy for numerical operations, and Matplotlib for visualization. Next, images are loaded from the dataset directory, and a sample image is displayed to verify correct loading. The resizing process is then performed using OpenCV's `cv2.resize()` function, setting a fixed resolution, typically 128x128 pixels. This size is commonly used in deep learning models to balance computational efficiency and feature retention.

Data Augmentation:- Data augmentation is a crucial technique in sign language detection, as it increases the diversity of training data by applying transformations such as rotation, flipping, zooming, and brightness adjustments. Since sign language recognition relies on hand gestures, augmented images help models generalize better, reducing overfitting and improving accuracy in real-world scenarios. Using Jupyter Notebook, we can apply data augmentation with TensorFlow/Keras ImageDataGenerator to enhance the dataset before training deep learning models. The first step involves importing necessary libraries, including TensorFlow/Keras, OpenCV, and Matplotlib for image processing and visualization. Next, we define augmentation transformations such as rotation (20 degrees), width/height shifts (20%), shear transformation, zooming (20%), horizontal flipping, and brightness adjustments. These transformations help the model recognize gestures under different conditions, such as varying lighting and hand positions.



```
[35]: datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # apply ZCA whitening
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1, # Randomly zoom image
    width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
    horizontal_flip=False, # randomly flip images
    vertical_flip=False) # randomly flip images

datagen.fit(x_train)
```

Fig 1.2 Data Augmentation

Data Preprocessing:- Data Preprocessing in Sign Language Detection Using Jupyter Notebook

Data preprocessing is an essential step in sign language detection, ensuring that images are correctly formatted and optimized for deep learning models. This process includes loading images, resizing them, normalizing pixel values, augmenting data, and converting datasets into a structured format suitable for training. The first step involves importing necessary libraries, including OpenCV for image processing, NumPy for numerical operations, and Matplotlib for visualization. Next, images are loaded from the dataset directory, and a sample image is displayed to verify correct loading. Since deep learning models require uniform input dimensions, all images are resized to 128x128 pixels, ensuring consistency across the dataset.



```

Jupyter signlanguage22 Last checkpoint 2 months ago
File Edit View Run Kernel Settings Help
Python 3 (ipykernel)

Preprocessing the Data

[07] X_train = train_df.drop(labels = ["label"],axis = 1)
y_train = train_df["label"]

[08] X_test = test_df.drop(labels = ["label"],axis = 1)
y_test = test_df["label"]

[09] X_train = np.array(X_train, dtype="float32")
X_test = np.array(X_test, dtype="float32")
y_train = np.array(y_train, dtype="float32")
y_test = np.array(y_test, dtype="float32")

[10] # Reshape the data to match the input shape of the CNN
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

[11] X_train = X_train / 255.0
X_test = X_test / 255.0

[12] # Convert the labels to one-hot encoded format
num_classes = 25
y_train = tf.keras.utils.to_categorical(y_train, num_classes=num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=num_classes)

[13] print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

(27405, 28, 28, 1)
(27405, 25)
(7172, 28, 28, 1)
(7172, 25)
  
```

Fig 1.3 Pre-Processing Data

Image recognition in sign language is a rapidly evolving field that leverages computer vision and artificial intelligence to interpret hand gestures and convert them into text or speech. This technology relies on deep learning models, particularly convolutional neural networks (CNNs), to detect and classify hand movements, shapes, and positions. By training these models on large datasets of sign language images and videos, researchers enhance accuracy and efficiency. Advanced techniques such as real-time gesture tracking, skeleton-based models, and 3D motion analysis further improve recognition. Challenges include variations in hand shapes, lighting conditions, and occlusions, which require robust algorithms for precise interpretation.

Output



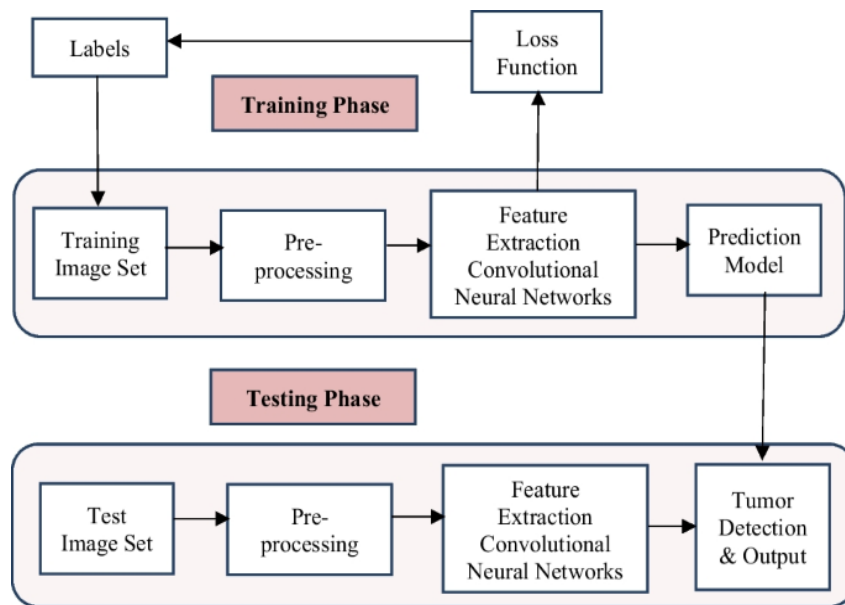


Fig 1.4 Architecture of CNN

A Convolutional Neural Network (CNN) is a type of deep learning model widely used for image classification, object detection, and pattern recognition tasks. The architecture of a CNN is typically organized into several layers, each performing a specific role in feature extraction and learning. It starts with the input layer, which receives the image data in the form of pixel matrices (e.g., 28x28 for grayscale or 224x224x3 for RGB images). Next comes the convolutional layer, where multiple filters (kernels) slide over the input image to detect features such as edges, textures, or shapes. These filters create feature maps that highlight important regions of the image. After convolution, an activation function—usually ReLU (Rectified Linear Unit)—is applied to introduce non-linearity into the model, enabling it to learn complex patterns. This is followed by a pooling layer, commonly using max pooling, which reduces the spatial size of the feature maps and retains the most important information, improving efficiency and reducing overfitting. The process of convolution, activation, and pooling is often repeated in multiple layers. Finally, the output is passed through fully connected layers, where the network makes predictions or classifications based on the extracted features. This layered structure forms the basis of a CNN block diagram.

After convolution, an activation function, most commonly ReLU (Rectified Linear Unit), is applied to the feature maps. This layer introduces non-linearity into the model, allowing it to learn more complex representations. Without activation functions, the network would behave like a linear model regardless of the number of layers. Next, a pooling layer is used to reduce the spatial dimensions of the feature maps while preserving the most important information. The most common type is max pooling, which selects the maximum value from each region of the feature map, thus making the model more robust to small translations in the input.

In addition to the core components of a CNN—input, convolution, activation, pooling, and fully connected layers—there are other important concepts and techniques that enhance the performance and training of the network. One such concept is padding, which involves adding extra pixels (usually zeros) around the input image or feature map. Padding helps maintain the spatial dimensions of the output after convolution, ensuring that important edge features are not lost during processing. It also allows deeper networks without excessive shrinking of feature maps.

Another crucial element is stride, which determines how far the filter moves during the convolution operation. A stride of 1 means the filter shifts one pixel at a time, while higher strides reduce the output size and computational load. Stride and padding are both tunable parameters that influence how much detail the CNN captures.

To train a CNN, a loss function is used to measure how far the predicted output is from the actual label. Backpropagation and optimization algorithms like Stochastic Gradient Descent (SGD) or Adam adjust the filter weights to minimize this loss. Techniques such as dropout and batch normalization are also used to prevent overfitting and speed up training. Overall, CNNs are highly efficient and scalable models, essential for modern visual recognition tasks.

The Convolutional Neural Network (CNN) algorithm offers numerous advantages that make it one of the most powerful and widely used techniques in the field of deep learning, particularly for image and pattern recognition tasks. One of the key advantages of CNNs is their ability to automatically learn relevant features from raw input data. Unlike traditional machine learning algorithms that require manual feature extraction, CNNs eliminate the need for hand-engineered features by using convolutional filters to detect patterns such as edges, textures, and shapes directly from the data. This makes them highly efficient and adaptable to complex visual recognition problems.

Another major advantage is the concept of parameter sharing in convolutional layers. Instead of having a separate weight for every connection, CNNs use the same filter across different regions of the input, significantly reducing the number of parameters and computational requirements. This not only speeds up training but also helps the model generalize better by focusing on the most important features. Additionally, sparse connectivity—where each neuron is connected only to a local region of the input—makes CNNs more efficient than fully connected networks, especially for high-dimensional inputs like images.

2. Literature Survey

Sign language detection is a crucial field in computer vision and natural language processing, aiming to bridge the communication gap between the hearing and speech-impaired communities and the general population. Advances in deep learning, computer vision, and sensor technologies have significantly improved the accuracy and efficiency of sign language detection systems.

[1] The system produced by the authors recognizes ISL gestures from mobile camera videos with none additional sensors to detect hand regions. Single-handed static and dynamic gestures and double-handed static gestures are identified in this system. The system uses CNN algorithm for feature extraction. After pre-processing the image, the centroid of the binary image is estimated. Using the gesture separation algorithm, the gesture is recognized, static as well as dynamic. The advantage of this system is that it is economical and can be implemented with a mobile camera, making it very user-friendly. But the disadvantage is that it is not efficient under cluttered backgrounds and different illumination conditions.

A Deep Learning-based Indian Sign Language Recognition System

[2] This system was successfully trained on all ISL static alphabets with a training accuracy of 99.93% and with testing and validation accuracy of 98.64%. Facial expression and context analysis are the other part not included in this project.

An Efficient Binarized Neural Network for Recognizing Two Hands Indian Sign Language Gestures in Realtime Environment.

[3] Considering the challenges of sign language recognition, on targeted embedded platforms, authors have proposed the novel architecture of a binarized neural network with binary values of weights and activations using bitwise operations. The advantage is using this architecture achieves an overall accuracy of 98.8% which is higher than other existing methods while the disadvantage is This system misclassifies some signs of M, N, E because of their similar kind of shapes.

[4] Deaf Mute Communication Interpreter- A Review 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 Touch-screen. All the above mentioned three sub-divided methods make use of various sensors, accelerometer, a suitable micro-controller, 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. The five subdivided methods are- SLIM module, TESSA, Wi-See Technology, SWI_PELE System and Web-Sign Technology.

[5] An Efficient Framework for Indian Sign Language Recognition Using Wavelet Transform :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.

[6] Hand Gesture Recognition Using PCA in : In this paper authors presented a scheme using a databasedriven hand gesture recognition based upon skin color model approach and thresholding approach along with an effective template matching which can be effectively used for human robotics applications and similar other applications.. Initially, hand region is segmented by applying skin color model in YCbCr color space. In the next stage thresholding is applied to separate foreground and background. Finally, template based matching technique is developed using Principal Component Analysis (PCA) for recognition.

[7] Hand Gesture Recognition System For Dumb People : Authors presented the static hand gesture recognition system using digital image processing. For hand gesture feature vector SIFT algorithm is used. The SIFT features have been computed at the edges which are invariant to scaling, rotation, addition of noise. 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. RGB-to-GRAY segmentation technique was used to minimize the chances of false detection. 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 pc. The key feature in this system is the real time gesture to text conversion. The processing steps include: gesture extraction, gesture matching and conversion to speech.

[8] An Automated System for Indian Sign Language Recognition in : In this paper a method for automatic recognition of signs on the basis of shape based features is presented. For segmentation of hand region from the images, Otsu's thresholding algorithm is used, that chooses an optimal threshold to minimize the within-class variance of thresholded black and white pixels. Features of segmented hand region are calculated using Hu's invariant moments that are fed to Artificial Neural Network for classification. Performance of the system is evaluated on the basis of Accuracy ,Sensitivity and Specificity.

[9] Hand Gesture Recognition for Sign Language Recognition: A Review in Authors presented various method of hand gesture and sign language recognition proposed in the past by various researchers. For deaf and dumb people, Sign language is the only way of communication.

[10] A Review on Feature Extraction for Indian and American Sign Language in : Paper presented the recent research and development of sign language 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. SignPro- An Application Suite for Deaf and Mute .

In [11]: Author presented application that helps the deaf and Mute person to communicate with the rest of the world using sign language. The key feature in this system is the real time gesture to text conversion. The processing steps include: gesture extraction, gesture matching and conversion to speech. Gesture extraction involves use of various image processing techniques such as histogram matching, bounding box computation, skin colour segmentation and region growing. Techniques applicable for Gesture matching include feature point matching and correlation based matching. The other features in the application include voicing out of text and text to gesture conversion. Offline Signature Verification Using Surf Feature Extraction and Neural Networks Approach.

3. Problem Statement



Communication is defined as the act of sharing or exchanging information, ideas or feelings. To establish communication between two people, both of them are required to have knowledge and understanding of a common language. But in the case of deaf and mute people, the means of communication are different. Deaf is the inability to hear and mute is the inability to speak. They communicate using sign language among themselves and with normal people but normal people do not take seriously the importance of sign language. Not everyone possesses the knowledge and understanding of sign language which makes communication difficult between a normal person and a deaf and mute person. To overcome this barrier, one can build a model based on machine learning. A model can be trained to recognize different gestures of sign language and translate them into English. This will help a lot of people in communicating and conversing with deaf and mute people. The existing Indian Sign Language Recognition systems are designed using machine learning algorithms with single and double-handed gestures but they are not real-time. In this propose, we propose a method to create an Indian Sign Language and email system.

Problem Description:-

Despite technological advancements, the lack of effective sign language interpretation tools remains a critical issue. Traditional methods of translation rely on human interpreters, which may not always be available or affordable. Some solutions use sensor-based gloves, but these can be expensive, uncomfortable, and impractical for widespread adoption.

Sign language recognition is a complex problem due to several challenges:

1. **Variability in Gestures:** Sign languages consist of static signs (hand postures) and dynamic signs (hand movements). Recognizing both types accurately requires advanced motion tracking and deep learning techniques.
2. **Diversity of Sign Languages:** Different regions have distinct sign languages with unique gestures and grammar structures. A universal detection system must be adaptable to multiple languages.
3. **Environmental Factors:** Variations in lighting, background clutter, and camera angles can affect gesture recognition in real-world scenarios.
4. **Hand and Skin Variations:** Differences in hand size, skin color, and orientation can impact the accuracy of vision-based recognition systems.
5. **Real-time Processing:** For practical applications in video conferencing, education, and public services, sign language detection systems must operate efficiently in real time.

Proposed Solution:-

This project aims to develop an AI-powered sign language detection system using computer vision and deep learning techniques. The system will process video frames or images, extract hand movement features, and classify them into corresponding words, letters, or sentences.

Key components of the proposed solution include:

- **Deep Learning Models:** Convolutional Neural Networks (CNNs) for image feature extraction and Recurrent Neural Networks (RNNs) or Transformers for sequential gesture recognition.
- **Dataset Training:** Large-scale sign language datasets with labeled gestures will be used to train the model.
- **Computer Vision Techniques:** Hand tracking, keypoint detection, and background segmentation to improve accuracy.
- **Real-time Recognition:** Optimized algorithms for fast processing on mobile devices, webcams, or embedded systems.
- **User-Friendly Interface:** An interactive application that translates sign language into text or speech, making communication accessible to non-signers.

Expected Impact:-

The implementation of a sign language detection system will bridge the communication gap between the deaf community and the general population. It will improve accessibility in education, workplaces, public services, and social interactions. This technology can be integrated into mobile applications, smart devices, and assistive tools to provide real-time sign language interpretation, enhancing inclusion and independence for deaf individuals.

By leveraging artificial intelligence, this project aims to create an efficient, accurate, and scalable solution for sign language recognition, making communication more inclusive and accessible for everyone.

With the rapid advancement of artificial intelligence, particularly in computer vision, the Convolutional Neural Network (CNN) algorithm has become a foundational tool for image processing and pattern recognition. Despite its widespread use and success in tasks such as image classification, object detection, and facial recognition, there remains a lack of comprehensive understanding and accessibility to advanced information about the inner workings and limitations of CNNs. Many learners and practitioners use CNNs as black-box models without fully understanding the theoretical foundations, architectural variations, optimization challenges, and advanced enhancements such as residual connections, dilated convolutions, and attention mechanisms.

This gap in understanding can lead to inefficient model designs, poor generalization, overfitting, and suboptimal use of computational resources. Moreover, as CNNs are being applied in critical areas such as medical diagnostics, autonomous driving, and security systems, it is essential to ensure transparency, reliability, and explainability in their functioning. Current educational and research materials often focus on basic CNN structures and overlook more advanced concepts like transfer learning, fine-tuning, model interpretability, and recent architectural innovations like ResNet and DenseNet.

Therefore, the problem lies in the need for a structured and detailed study of advanced information about CNN algorithms, addressing both theoretical and practical aspects. This includes understanding how CNNs can be optimized for better performance, how they handle various data types, how to prevent overfitting in deeper models, and how to interpret their decisions. Bridging this knowledge gap will empower developers, researchers, and students to design more efficient, accurate, and explainable CNN-based systems, ultimately pushing the boundaries of AI-driven technologies across various industries.

In addition to the lack of deep theoretical understanding, another major challenge in advancing knowledge about CNN algorithms is the limited exposure to practical implementation strategies and real-world applications. Many academic courses and tutorials provide surface-level guidance, often focusing only on small datasets like MNIST or CIFAR-10. However, real-world problems involve high-resolution images, imbalanced datasets, noisy data, and the need for deployment on resource-constrained devices. This requires an understanding of optimization techniques such as data augmentation, learning rate scheduling, batch normalization, and hardware-efficient model design.

Furthermore, explainability and model interpretability have become crucial, especially in sensitive domains like healthcare, finance, and autonomous systems. Understanding techniques like Grad-CAM, saliency maps, and feature visualization is essential for building trust in CNN-based systems. There is also a growing need to integrate CNNs with other architectures such as RNNs or Transformers for multimodal tasks. Addressing these advanced topics is vital for building robust, scalable, and ethical AI solutions.

4. Methodology

Sign language detection is a complex task that integrates computer vision, machine learning, and deep learning techniques to recognize and interpret hand gestures. Traditional approaches rely on image processing techniques such as hand tracking, background subtraction, and keypoint detection using tools like OpenPose or MediaPipe Hands. Machine learning methods, including Support Vector Machines (SVM), Hidden Markov Models (HMMs), and Dynamic Time Warping (DTW), have been used to classify gestures based on extracted features. However, deep learning has significantly improved accuracy with models like Convolutional Neural Networks (CNNs) for static gesture recognition and Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks for temporal sequence processing. Advanced models, including 3D CNNs and Transformer-based architectures like SignBERT, further enhance the understanding of complex sign language structures. Additionally, sensor-based approaches, such as glove-based systems with accelerometers or motion capture devices like Leap Motion and Kinect, offer alternative detection methods, especially in challenging environments. Hybrid models that combine vision-based and sensor-based techniques provide improved robustness. With continuous advancements in AI and deep learning, sign language detection systems are becoming more accurate, contributing to greater accessibility for the deaf and hard-of-hearing communities.

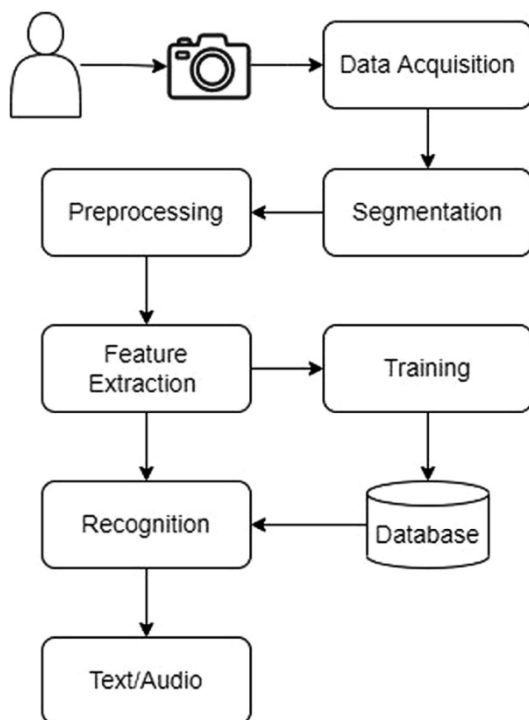


Fig 4.1 Working of CNN

In recent years, deep learning methodologies have significantly advanced sign language detection capabilities. Convolutional Neural Networks (CNNs) are widely used to automatically extract spatial features from individual frames, while Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) capture the temporal evolution of gestures across video sequences. More sophisticated models, such as 3D CNNs, process spatiotemporal data concurrently, and Transformer-based architectures are emerging to model contextual relationships over long sequences.

Sensor-based approaches also contribute to sign language detection by employing wearable devices like data gloves embedded with accelerometers and that

data gloves embedded with accelerometers and gyroscopes, which provide precise motion and orientation data.

Software uses in Sign Language Detection:- Sign language detection relies on various software tools, frameworks, and libraries that facilitate image processing, machine learning, deep learning, and real-time gesture recognition. It uses Jupyter notebook, Anaconda, VS Code and User Interfaces.

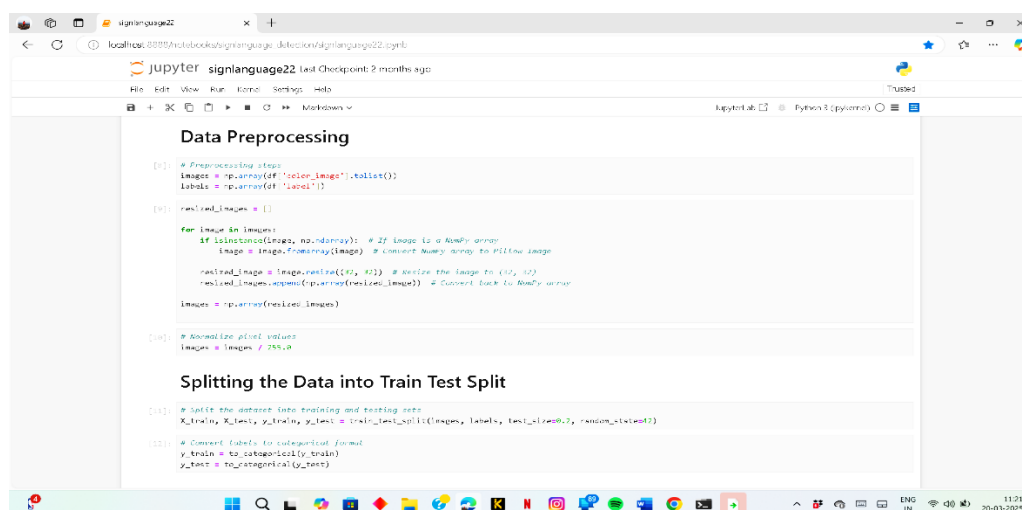


Fig 4.2 Jupyter notebook with command line

Jupyter Notebook in Sign Language Detection

Jupyter Notebook is a powerful tool for developing and experimenting with sign language detection models due to its interactive nature and compatibility with various machine learning and deep learning frameworks. It allows researchers and developers to write code, visualize data, and document their findings in a single environment, making it ideal for building and testing sign language recognition systems.

1. Data Preprocessing in Jupyter Notebook

Jupyter Notebook provides an interactive platform to preprocess sign language datasets. Libraries like OpenCV and MediaPipe are used to capture and process images and videos, including hand tracking and keypoint detection. Pandas and NumPy assist in managing large datasets, while Matplotlib and Seaborn are useful for visualizing data distributions and gesture patterns.

2. Model Development and Training

Jupyter Notebook supports deep learning frameworks like TensorFlow, Keras, and PyTorch, enabling the creation of models such as Convolutional Neural Networks (CNNs) for static gesture recognition and Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) for sequence-based sign detection.

3. Real-Time Sign Language Detection

With Jupyter Notebook, researchers can integrate OpenCV's VideoCapture function to process real-time video streams for detecting and classifying sign language gestures. Additionally, MediaPipe Hands and OpenPose enable real-time hand tracking, improving accuracy in gesture recognition.

Using VS Code to Run Python Libraries for Sign Language Detection

Visual Studio Code (VS Code) is a powerful and lightweight code editor widely used for running Python libraries in sign language detection projects. It provides an efficient environment for coding, debugging, and executing machine learning and deep learning models.

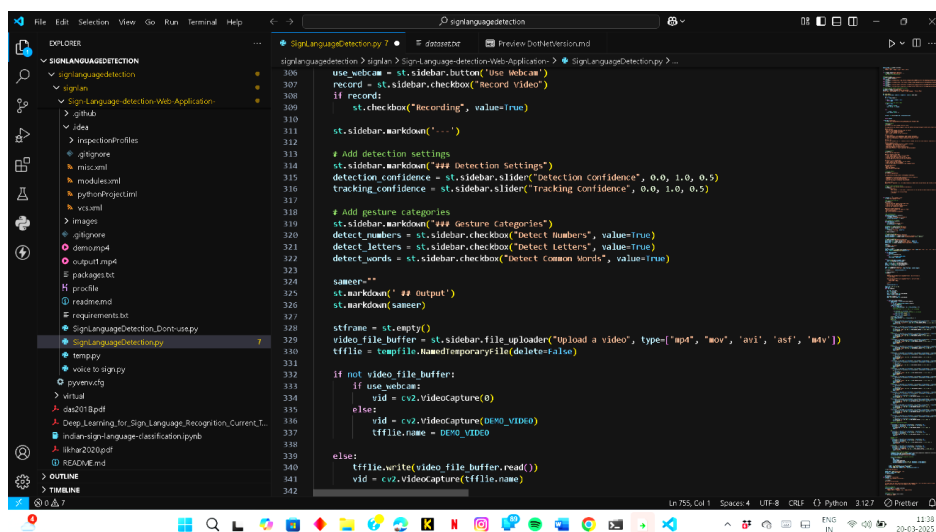


Fig 4.3 VS Code with UI Interface

1. Setting Up VS Code for Python Development

VS Code supports Python through the Python extension, which enables features like IntelliSense, debugging, and Jupyter Notebook integration. Users can install required libraries such as OpenCV, MediaPipe, TensorFlow, Keras, and PyTorch using the integrated terminal with commands like `pip install opencv-python mediapipe tensorflow keras torch torchvision`. The virtual environment feature helps manage dependencies and avoid conflicts between libraries.

2. Running Python Libraries for Image and Video Processing

For sign language detection, VS Code supports real-time image and video processing with OpenCV, allowing developers to capture frames from a webcam using `cv2.VideoCapture()`. MediaPipe Hands can be integrated to detect keypoints on hands and track finger movements, essential for recognizing sign language gestures.

3. Implementing Machine Learning and Deep Learning Models

VS Code enables seamless execution of machine learning models using Scikit-learn for gesture classification and deep learning frameworks like TensorFlow, Keras, and PyTorch for training Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Model training, validation, and evaluation can be done efficiently, with debugging support provided by VS Code's interactive terminal and variable explorer.

Hardware Used in Sign Language Detection:-

Sign language detection requires specialized hardware to capture, process, and interpret gestures accurately. These devices include cameras, sensors, processors, and wearable technology that enhance gesture recognition and real-time processing.

1. Cameras for Image and Video Capture

- **Webcams:** High-resolution webcams, such as the Logitech C920 or Razer Kiyo, capture real-time video of hand gestures for sign language recognition.
- **Depth Cameras:** Devices like the Microsoft Kinect and Intel RealSense provide depth information, improving gesture tracking in three dimensions.
- **Infrared Cameras:** Used for low-light environments, helping detect hand shapes and movements in different lighting conditions.

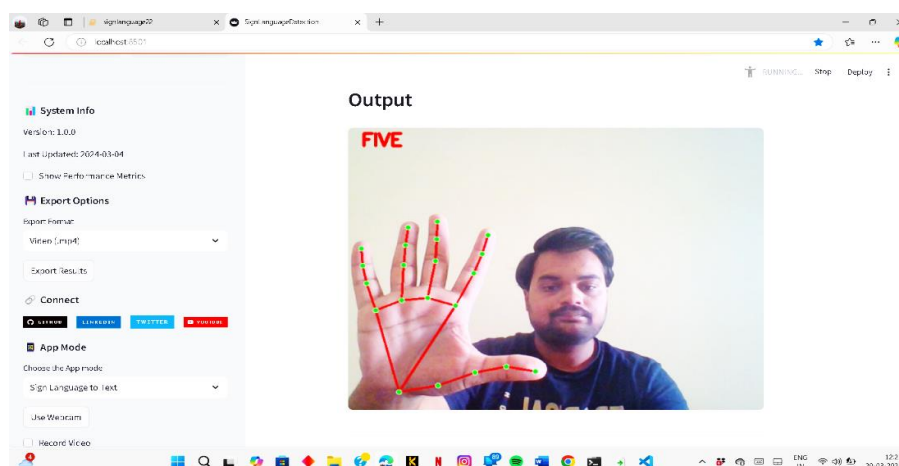


Fig 4.4 Sign Detection using User Interface

Datasets Used in Sign Language Detection

Sign language detection relies on high-quality datasets to train machine learning and deep learning models for accurate gesture recognition. These datasets contain images, videos, and keypoint annotations of various sign language gestures, including fingerspelling and continuous signing.

1. American Sign Language (ASL) Datasets

American Sign Language (ASL) datasets are essential for training machine learning and deep learning models in sign language detection. These datasets contain images, videos, and annotations of ASL gestures, including fingerspelling, isolated signs, and continuous signing. Below are some widely used ASL datasets:

1. ASL Fingerspelling Dataset

- **Description:** This dataset contains images of ASL fingerspelling hand gestures representing individual letters of the alphabet.
- **Usage:** It is widely used for recognizing static hand gestures, particularly for applications in text conversion and real-time communication.
- **Format:** Labeled images of hand gestures, often captured from multiple angles.

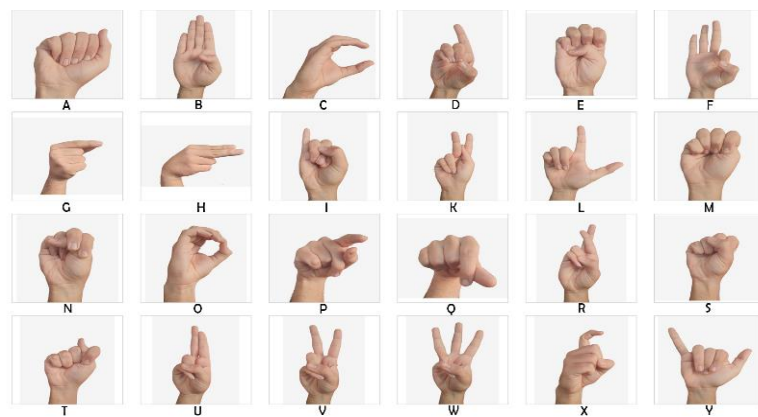


Fig 4.5 American Sign Language Dataset

Indian Sign Language (ISL) detection dataset:- These are essential for training AI models to recognize gestures, fingerspelling, and full sentences. Several publicly available datasets cater to different aspects of ISL recognition. The (ISL) dataset contains images and videos covering commonly used ISL words and phrases, making it useful for basic gesture recognition. The Indian Sign Language Recognition Dataset (ISLRD) consists of hand gesture images representing ISL words and alphabets, primarily used for static hand gesture recognition. Another significant dataset, the Multi-View Indian Sign Language Dataset (MV-ISL), captures ISL gestures from multiple angles, enhancing recognition accuracy and enabling multi-angle sign language detection.

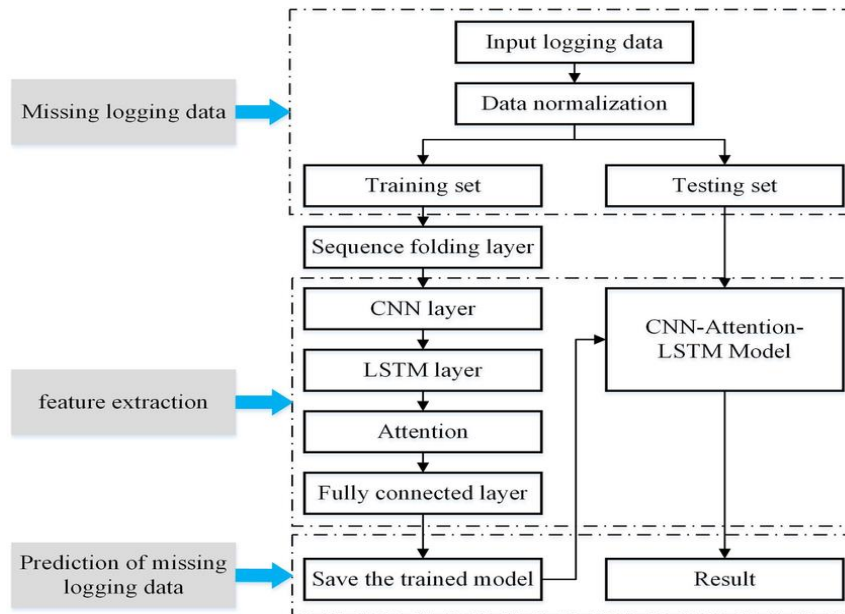


Fig 4.7 Methodology Behind CNN Algorithm

The methodology of a Convolutional Neural Network (CNN) involves a sequence of well-defined steps that allow the network to learn features from image data and perform tasks such as classification or object detection. The process begins with the input layer, where the raw image is fed into the network as a multidimensional array representing pixel intensities. This image is then passed to the first convolutional layer, where multiple filters (kernels) slide over the image to extract low-level features such as edges and textures. Each filter produces a feature map highlighting specific patterns within the image.

After convolution, an activation function like ReLU (Rectified Linear Unit) is applied to introduce non-linearity, allowing the network to learn more complex features. This is followed by a pooling layer, typically max pooling, which reduces the dimensionality of the feature maps while retaining the most important information. This process of convolution, activation, and pooling may be repeated multiple times to form a deep hierarchy of features.

Once high-level features are extracted, the output is flattened into a one-dimensional vector and passed through one or more fully connected layers (dense layers), where the actual classification or prediction is made. The final layer often uses a softmax activation function (for multi-class classification) to produce probability distributions over possible output classes.

During training, a loss function (e.g., cross-entropy loss) is used to evaluate the difference between predicted and actual labels. Through backpropagation, the network adjusts its weights using optimization algorithms like Stochastic Gradient Descent (SGD) or Adam to minimize this loss. Additional techniques such as dropout, batch normalization, and data augmentation are often employed to improve generalization and prevent overfitting. This structured methodology allows CNNs to effectively learn and generalize from visual data, making them highly powerful for various real-world applications.

5. Implementation

5.1 Dataset

We have used multiple datasets and trained multiple models to achieve good accuracy.

5.1.1 ASL Alphabet

The data is a collection of images of the alphabet from the American Sign Language, separated into 29 folders that represent the various classes. The training dataset consists of 87000 images which are 200x200 pixels. There are 29 classes of which 26 are English alphabets A-Z and the rest 3 classes are SPACE, DELETE, and, NOTHING. These 3 classes are very important and helpful in real-time applications.

5.1.2 Sign Language Gesture Images Dataset

The dataset consists of 37 different hand sign gestures which include A-Z alphabet gestures, 0-9 number gestures, and also a gesture for space which means how the deaf (hard hearing) and dumb people represent space between two letters or two words while communicating. Each gesture has 1500 images which are 50x50 pixels, so altogether there are 37 gesture which mean there 55,500 images for all gestures.

5.2 Data Pre-processing

An image is nothing more than a 2-dimensional array of numbers or pixels which are ranging from 0 to 255. Typically, 0 means black, and 255 means white. Image is defined by mathematical function $f(x,y)$ where 'x' represents horizontal and 'y' represents vertical in a coordinate plane. The value of $f(x,y)$ at any point is giving the pixel value at that point of an image. Image Pre-processing is the use of algorithms to perform operations on images. It is important to Pre-process the images before sending the images for model training. For example, all the images should have the same size of 200x200 pixels. If not, the model cannot be trained.



Fig 5.2.1 Sample image without preprocessing

Output



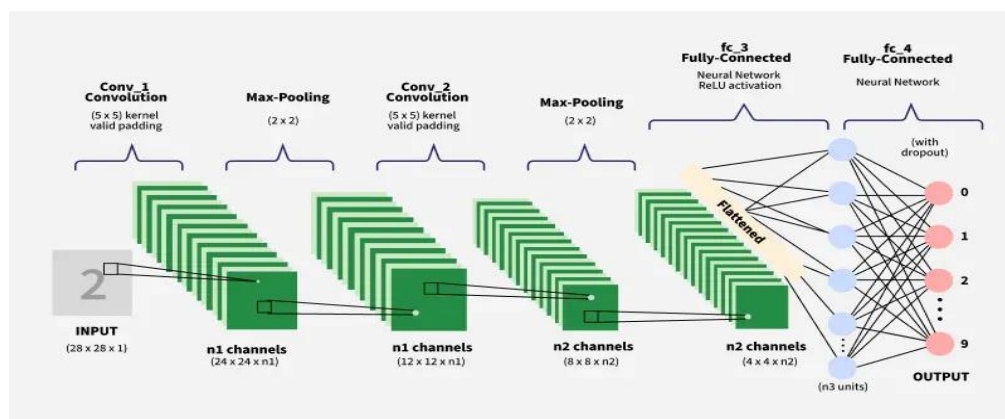
Fig 5.2.3 Pre-Processed image

5.3 Convolution Neural Networks (CNN)

Computer Vision is a field of Artificial Intelligence that focuses on problems related to images and videos. CNN combined with Computer vision is capable of performing complex problems

How CNNs Work?

1. **Input Image:** The CNN receives an input image, which is typically preprocessed to ensure uniformity in size and format.
2. **Convolutional Layers:** Filters are applied to the input image to extract features like edges, textures, and shapes.
3. **Pooling Layers:** The feature maps generated by the convolutional layers are downsampled to reduce dimensionality.
4. **Fully Connected Layers:** The downsampled feature maps are passed through fully connected layers to produce the final output, such as a classification label.
5. **Output:** The CNN outputs a prediction, such as the class of the image.



5.3.1 Convolutional Neural Network

5.3.2 Pooling

After the convolution operation, the pooling layer will be applied. The pooling layer is used to reduce the size of the image. There are two types of pooling:

1. Max Pooling
2. Average Pooling

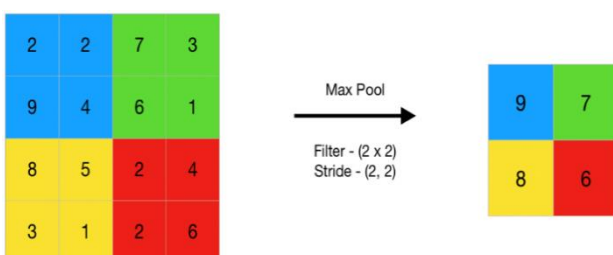


Fig 5.3.3 Max Pooling

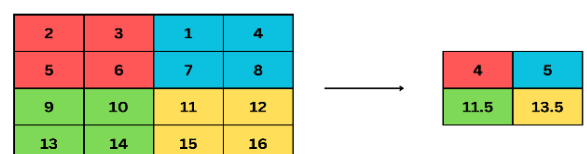


Fig 5.3.4 Average Pooling

Max Pooling:-

Max pooling is a crucial operation used in Convolutional Neural Networks (CNNs), a class of deep learning models particularly effective in analyzing visual imagery. CNNs mimic the visual cortex of animals and are widely used in image recognition, classification, object detection, and various other computer vision tasks. Among the key components of a CNN are the convolutional layers, activation functions, pooling layers, and fully connected layers. Max pooling specifically falls under the pooling layer category and plays a pivotal role in reducing the dimensionality of feature maps while retaining the most salient information. The operation works by sliding a fixed-size window (commonly 2x2 or 3x3) over an input feature map and selecting the maximum value within that window. This process effectively down-samples the feature map, reducing its spatial dimensions (width and height) but not its depth, which is essential for reducing computational cost, memory usage, and overfitting while maintaining critical features.

Average Pooling:-

Average pooling is a widely used downsampling technique in Convolutional Neural Networks (CNNs) that reduces the spatial dimensions (width and height) of feature maps while retaining important information. Unlike max pooling, which selects the highest value within a defined window (such as 2x2 or 3x3), average pooling computes the mean of all values within the window. This results in a smoother and more generalized representation of the input features. The primary goal of average pooling is to reduce the size of feature maps, thereby decreasing computational requirements and helping to prevent overfitting by eliminating less important details.

Average pooling maintains more background and contextual information compared to max pooling, which tends to highlight only the most prominent features. This makes average pooling useful in scenarios where overall information matters more than sharp feature detection—for example, in texture analysis or low-noise environments. Like max pooling, average pooling is a non-learnable operation and introduces translational invariance, allowing the model to be more robust to slight shifts in the input. Although max pooling is more commonly used in modern CNN architectures due to its effectiveness in highlighting strong features, average pooling still has its place, particularly in tasks requiring softer feature representation or during the final stages of some networks, like in global average pooling.

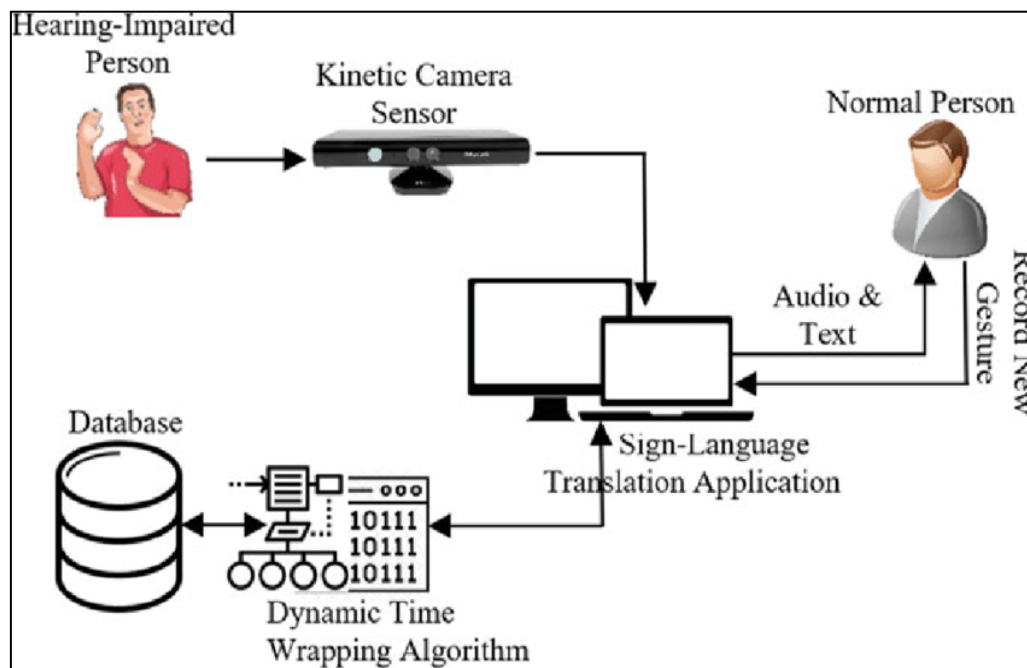


Fig 5.4 Implementation of Sign Language Using CNN Algorithm

Sign language recognition aims to translate gestures into text or speech to bridge communication between hearing-impaired individuals and others. A popular and effective method for this task involves using Convolutional Neural Networks (CNNs), which are powerful for image classification and recognition tasks.

The implementation begins by collecting a dataset of images or video frames representing different sign language gestures. Popular datasets include the American Sign Language (ASL) alphabet dataset. Each image is labeled according to the gesture it represents. Preprocessing steps such as resizing, grayscale conversion, and normalization are applied to ensure consistency and improve model performance.

A CNN model is then designed, typically with multiple convolutional layers followed by pooling layers to extract and reduce spatial features. These layers are followed by fully connected (dense) layers that perform classification based on the extracted features. The final output layer uses a softmax activation function to predict the probability of each gesture class.

The model is trained using a training dataset and optimized using loss functions like categorical cross-entropy and optimizers such as Adam. Validation data is used to monitor overfitting and fine-tune hyperparameters.

After training, the model can be tested in real-time applications. A camera captures the hand gesture, which is processed and fed into the trained CNN model. The model predicts the corresponding sign, which is then displayed or spoken using text-to-speech technology.

Working of Sign Language Detection

Sign language detection is a computer vision and deep learning-based approach to interpret hand gestures used in sign languages and convert them into readable or spoken text. It plays a crucial role in bridging the communication gap between hearing-impaired individuals and others. The working of a sign language detection system involves several key stages:

1. Data Acquisition

The process begins by collecting a dataset containing images or video sequences of hand gestures representing different signs. These could include alphabet signs, numbers, or complete words and phrases. Datasets can be captured using a standard camera, webcam, or depth sensor (like Kinect or Leap Motion).

2. Preprocessing

Before feeding the data into a model, it must be preprocessed to enhance accuracy. Preprocessing may include:

- Resizing images to a standard size.
- Grayscale conversion or channel normalization.
- Noise removal using filters.
- Hand segmentation, where the hand region is isolated from the background using techniques like background subtraction or skin color detection.

3. Feature Extraction

Convolutional Neural Networks (CNNs) are typically used to extract spatial features from images. Convolutional layers automatically learn patterns such as edges, shapes, and textures of the hand gestures. These features are crucial for distinguishing between different signs.

4. Model Training

The extracted features are passed through fully connected layers, and the model is trained using labeled data. The system learns to map the gesture image to a corresponding class label (e.g., “A” for a certain hand shape). A softmax layer at the end outputs probabilities for each possible class.

5. Gesture Recognition

Once trained, the model can be used for real-time recognition. A live camera feed captures a user’s hand gesture. The image is preprocessed and passed through the trained model, which then predicts the sign being shown. The output can be displayed as text or spoken using text-to-speech tools.

6. Post-processing and Feedback

To enhance user experience, smoothing techniques (e.g., averaging predictions across frames) and user feedback loops are added. This helps in reducing false predictions and improving accuracy over time.

6. Result

The sample dataset is made by collecting images for deep learning using webcam and OpenCV. Label images for sign language detection using Labellmg. The Labellmg software is used for graphically labeling the images that is further used when recognizing the images. Gestures should be labeled with a right label so that we get the gestures recognized correctly later with the rightlabel. The Labellmg software is used for graphically labeling the images that is further used whenrecognizing the images. We have to keep in mind that labeling has to be done correctly i.e, thegesture should be labeled with a right label so that we get the gestures recognized correctly later with the right label.

Once the images are labeled and saved an XML file is created for that image. This XML file contains the information about where the model should be looking in the imageduring the training process. This model is trained for 5 different gestures hence 5 different labels were used for labeling them. For each gesture, 15 images were used and clicked from differentangles. Code used to automatically take pictures and save them to a specific folder. Labeling isdone by drawing a frame around the gesture being performed.. XML file associated with a taggedimage indicating where the model should look for the gesture when training the ML model. The implementation begins with data collection, where a dataset of hand gestures is gathered using a webcam or pre-existing sign language datasets, such as those for American Sign Language (ASL). To enhance accuracy, Mediapipe Hands is used for real-time landmark detection, extracting 21 key points from the hand rather than relying on raw pixel data. Next, in the data preprocessing stage, image frames are resized and converted into grayscale or RGB formats, followed by feature extraction. The model training phase involves a deep learning model, often a CNN or an LSTM-based neural network, trained using TensorFlow/Keras to classify hand gestures into predefined categories.

Output

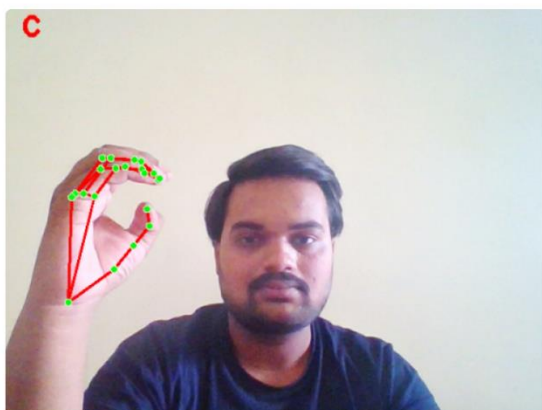


Fig 6.1 Dataset for C

Output



Fig 6.2 Dataset for THANK YOU

Output

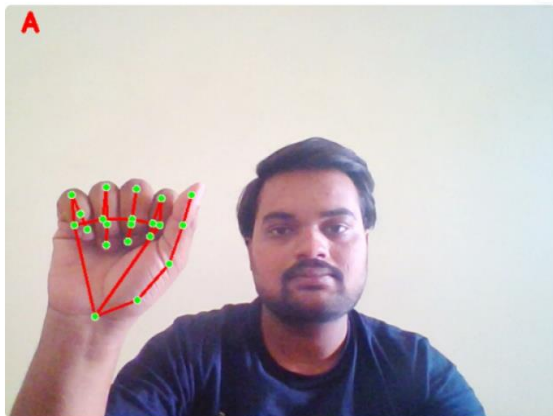


Fig 6.3 Dataset for A

Output



Fig 6.4 Dataset for SEVEN

Output

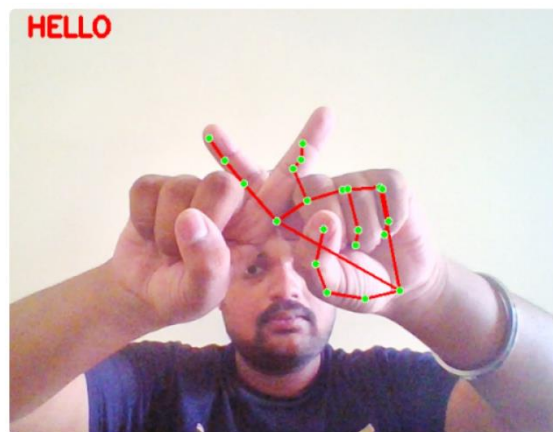


Fig 6.5 Dataset for HELLO

Dataset train and accuracy:-

Sign language detection is an essential task in computer vision and deep learning, requiring a well-structured dataset and an effective model to achieve high accuracy. Several datasets are widely used for training models in sign language detection, including RWTH-BOSTON-104, which provides American Sign Language (ASL) annotations, and WLASL, a large-scale dataset with over 2,000 ASL signs. Other datasets, such as MS-ASL, LSA64, and Chalearn LAP, offer diverse sign language samples that help improve model generalization. Choosing the right dataset is crucial because variations in hand gestures, signer styles, and lighting conditions can impact accuracy.

Training a model for sign language detection involves multiple steps, starting with data preprocessing. Video-based datasets are converted into frames, and techniques like data augmentation (rotation, flipping, and normalization) are applied to improve robustness. Feature extraction methods such as OpenPose, MediaPipe, or deep learning-based convolutional neural networks (CNNs) help identify key hand movements. For static image-based sign recognition, CNNs are effective, whereas sequential models like recurrent neural networks.

The accuracy of sign language detection is typically evaluated using metrics like top-1 and top-5 accuracy, which measure the correctness of predictions, and word error rate (WER), which is useful for continuous recognition tasks. Basic CNN models achieve around 95–98% accuracy..

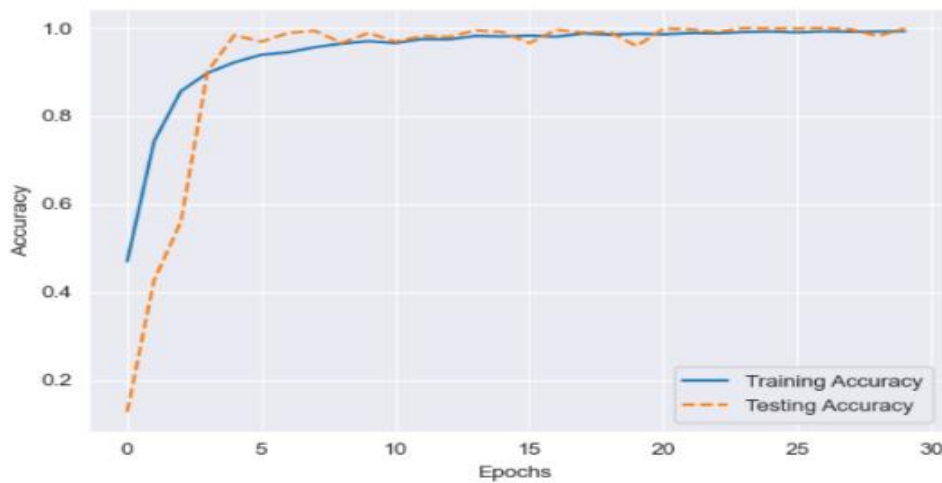


Fig 6.6 Training Accuracy Graph

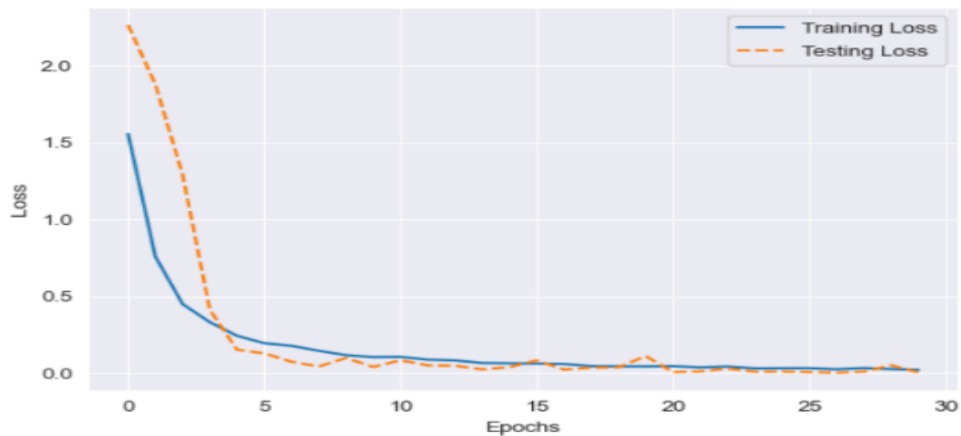


Fig 6.7 Training Loss Function Graph

Results obtained are mentioned below :-

- Accuracy Attained - 0.99111008644104
- Loss - 0.06643851101398468
- F1 Score - 0.98
- Precision - 0.99
- Recall - 0.97

Confusion Matrix:-

A confusion matrix is a crucial evaluation tool for sign language detection models, providing a clear breakdown of classification performance. It helps analyze how well a model distinguishes between different sign language gestures and identifies misclassifications. The confusion matrix is structured as a table where each row represents actual sign labels, and each column represents predicted labels. The key components of a confusion matrix include True Positives (TP), where the model correctly identifies a sign; True Negatives (TN), where the model correctly rejects a non-sign or another sign; False Positives (FP), where the model incorrectly classifies a different sign as the target sign; and False Negatives (FN), where the model fails to detect the correct sign and misclassifies it.

For example, in a system detecting five signs—A, B, C, D, and E—a confusion matrix can display how often each sign is correctly classified and where errors occur. If sign A is detected correctly 50 times but misclassified as B twice and as E three times, these errors indicate potential model weaknesses. From the confusion matrix, key performance metrics can be derived, such as accuracy, which measures overall correctness, and precision, which indicates the proportion of correctly predicted signs out of all detected instances. Recall, or sensitivity, measures how effectively the model detects actual signs, while the F1-score provides a balanced measure of precision and recall. Challenges in sign language detection include handshape similarities, motion variations, and environmental factors like lighting and background noise. Similar hand movements may lead to misclassification, while variations in speed or angle can further impact detection accuracy. By analyzing a confusion matrix, researchers can fine-tune models, adjust training data, and improve classification accuracy. This analysis is essential for building robust sign language recognition systems, making them more effective for real-world applications.

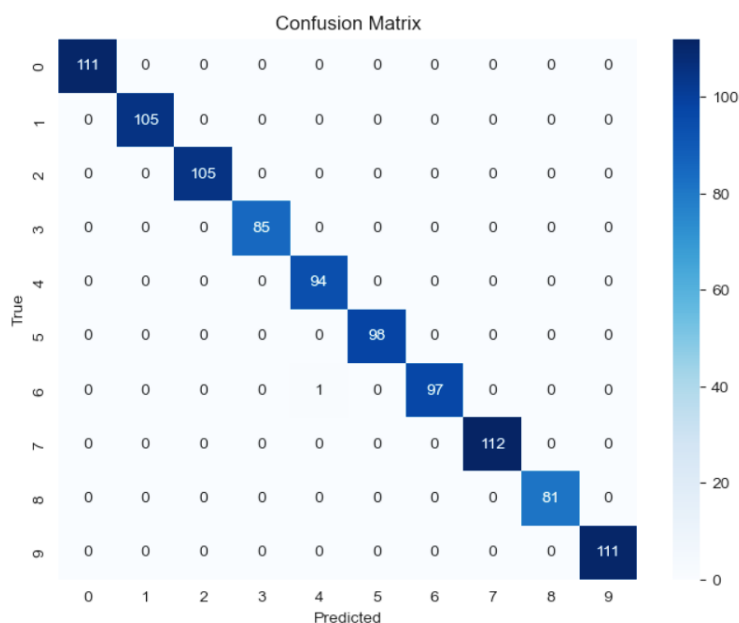


Fig 6.8 Confusion Matrix

Speech to Sign Language Converter:-

A speech-to-sign language converter is an assistive technology that translates spoken language into sign language, enhancing communication between hearing and Deaf or hard-of-hearing individuals. This system typically involves multiple stages, including speech recognition, natural language processing (NLP), sign language translation, and gesture synthesis using avatars or robotic hands.

The process begins with speech recognition, where an automatic speech recognition (ASR) model converts spoken words into text. This is achieved using deep learning techniques like recurrent neural networks (RNNs) and transformer-based models such as Whisper or Wav2Vec. The recognized text is then processed by natural language processing (NLP) techniques to understand the sentence structure and meaning. NLP algorithms identify keywords, context, and grammar to restructure spoken language into sign language syntax, as sign languages often have different grammatical structures from spoken languages.

Once the sentence is structured properly, the next stage is sign language translation, where text is mapped to corresponding signs. This can be achieved using a database of sign language words or machine learning models trained on sign language datasets. Finally, the gesture synthesis stage converts the translated text into sign language gestures. This can be done using animated avatars, virtual human models, or robotic arms capable of performing sign gestures. Motion capture technology and deep learning-based gesture generation further improve the realism of sign movements. Challenges in speech-to-sign language conversion include accurately capturing spoken accents, handling real-time translation, and ensuring sign language accuracy across different regional variations. Additionally, facial expressions and body movements, which are crucial in sign language, must be effectively integrated.

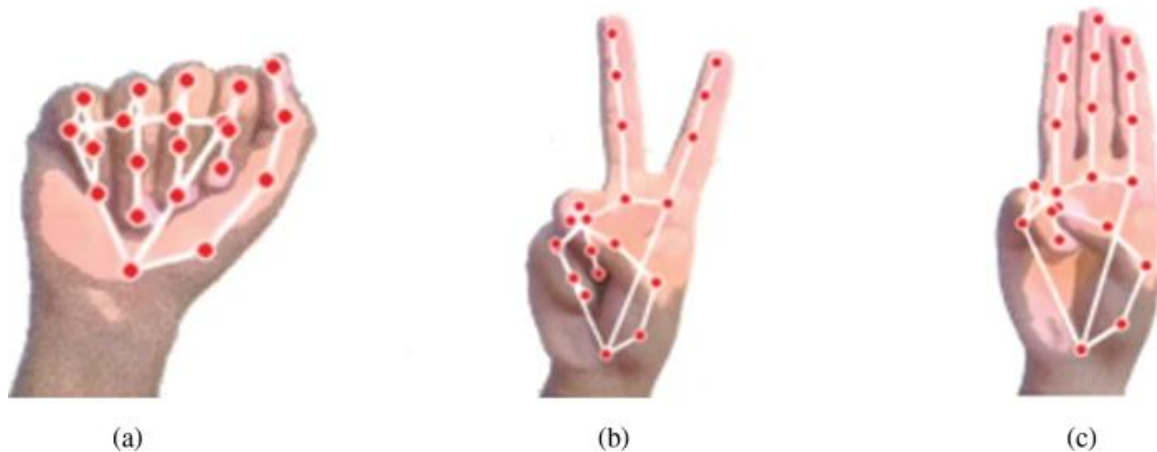


Fig 6.11 sign language detection using

The image above illustrates the process of sign language detection using hand keypoint landmarks. It displays three different hand gestures, each marked with red dots representing key joint positions such as fingertips, knuckles, and the wrist. These dots are connected with white lines to form a skeletal model of the hand. This kind of representation is commonly used in sign language detection systems to identify and classify hand gestures. Tools like MediaPipe Hands or OpenPose are often employed to extract these 21 keypoints from a hand image in real-time using a webcam or camera.

In sign language detection, the first step is to capture the hand gesture through a camera. Then, pose estimation algorithms detect and track the hand landmarks consistently across various poses. These keypoints serve as the basis for creating a vector representation of the gesture, which describes the spatial relationship between different parts of the hand. This vector is then passed into a classification model, which could be a Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) network, or another type of machine learning model. The model processes this input and matches it with a predefined class, which corresponds to a specific sign in the sign language alphabet or numeric system.

Once the gesture is recognized, the system can display the translated output as text or convert it into speech using a text-to-speech engine. This technology enables real-time communication and can be especially useful for assisting individuals with hearing or speech impairments. Applications of such systems include mobile apps, educational tools, virtual interpreters, and other human-computer interaction platforms. The use of keypoint-based detection ensures accuracy and flexibility across different users and lighting conditions, making it a powerful method for sign language recognition.

Expanding further on the concept of sign language detection using hand keypoint landmarks, this approach has revolutionized the way we interact with machines and interpret human gestures. The primary advantage of using hand landmarks over traditional image classification is that it focuses on the geometric structure of the hand, which remains consistent regardless of background, lighting conditions, or even clothing. By relying on skeletal data, the system becomes more robust and efficient, especially in real-time applications.

After detecting the 21 key landmarks on the hand, the spatial coordinates (usually in 2D or 3D space) are extracted and used to form a structured dataset. These coordinates can include the distance between fingers, the angle of joints, and the relative position of the palm and fingertips. This information is critical in differentiating between gestures that may appear visually similar but differ slightly in finger positioning. For instance, the American Sign Language (ASL) signs for “V” and “Peace” may look alike at a glance but have subtle variations in finger angles and spacing that are captured accurately through landmark detection.

Once the hand gesture is represented numerically, the system uses a trained model to classify it. In advanced setups, a sequence of gestures (as in full words or sentences) can be analyzed using Recurrent Neural Networks (RNNs) or LSTM models, which are capable of handling time-series data. These models track the motion and transition between gestures to understand full expressions or phrases in sign language.

Moreover, integrating hand tracking with additional cues like facial expression recognition or body pose estimation can significantly enhance the context and accuracy of sign language interpretation. For example, certain signs may change meaning based on facial expressions, which can be captured using tools like MediaPipe FaceMesh.

Recent advancements also focus on optimizing these systems for mobile platforms. Lightweight models such as MobileNet or TensorFlow Lite enable sign language detection on smartphones, making the technology accessible to a wider audience. This portability opens doors to educational tools, translation apps, and even live interpretation in public spaces.

In summary, sign language detection using hand keypoints is a highly efficient, scalable, and inclusive approach. It bridges communication gaps and empowers individuals with hearing or speech impairments to express themselves in digital environments, promoting greater inclusivity and accessibility in society.

7. Conclusion

In order to sum up, our project represents a major advancement in the field of hand gesture detection technology. Using the vast ISL Translate dataset and a self-trained CNN model, we have obtained impressive accuracy, which has been confirmed by thorough recall and confusion matrix analysis.

This model tries to solve the problem of people who are deaf and mute and who wants to convey what they are saying to others using Sign Language but can't as others haven't learned Sign Language. The model is efficient and highly accurate and hence works without any lag and also if the data is downloaded can be made to work offline. Our technology promises new applications in assistive technologies and interactive interfaces, providing users with smooth and natural interactions thanks to its real-time detection capabilities. Our project opens the door for future innovation and advances gesture recognition technology to unprecedented levels of usefulness and accessibility for a wide range of user demographics.

This project indicate that sign language detection can bridge communication gaps between individuals with hearing impairments and those unfamiliar with sign language. Despite its success, challenges such as variations in lighting conditions, hand positioning, and background noise remain areas for future improvement. Enhancing the dataset with a broader range of gestures, optimizing the model for different sign languages for text-to-speech conversion can further increase the effectiveness of the system. Another crucial feature is the use of deep learning-based classification, particularly Convolutional Neural Networks (CNNs), which analyze image patterns to distinguish between different signs with high accuracy. The model is trained on a large and diverse dataset of hand gestures, improving its ability to recognize variations in hand positioning, lighting conditions, and individual differences among users. The system also integrates gesture-to-text conversion, allowing detected signs to be translated into readable text, which can be displayed on a screen or further processed for text-to-speech conversion, enhancing communication for individuals who rely on sign language. Furthermore, the project supports multilingual sign language recognition, making it adaptable for different sign language standards such as American Sign Language (ASL) and Indian Sign Language (ISL). Additionally, it features a user-friendly interface, designed to provide an intuitive experience for users of all backgrounds. To improve usability, the system incorporates error detection and correction mechanisms, ensuring that misinterpretations are minimized.

8. Code Snippet

```

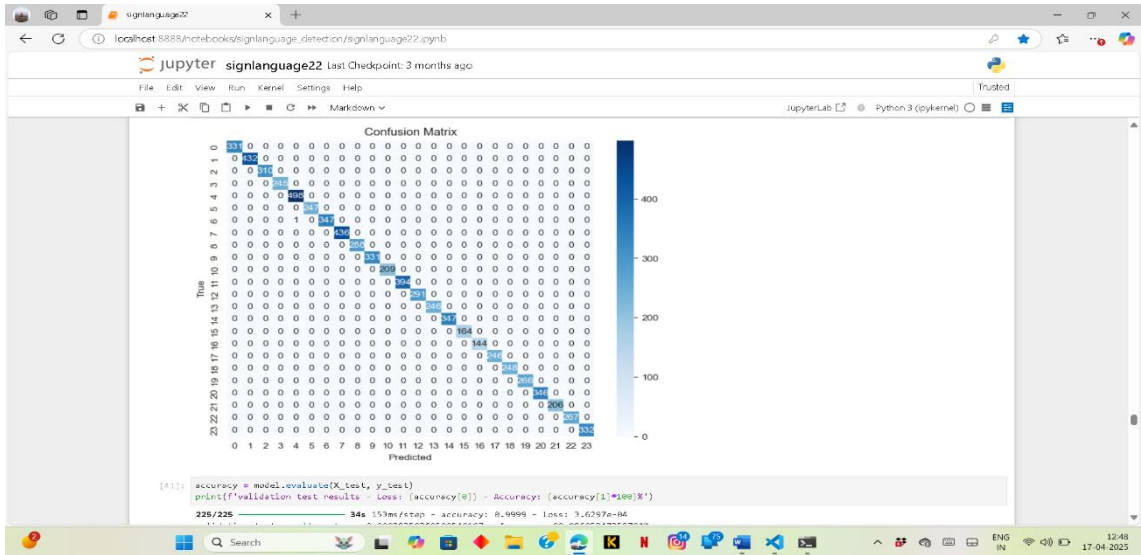
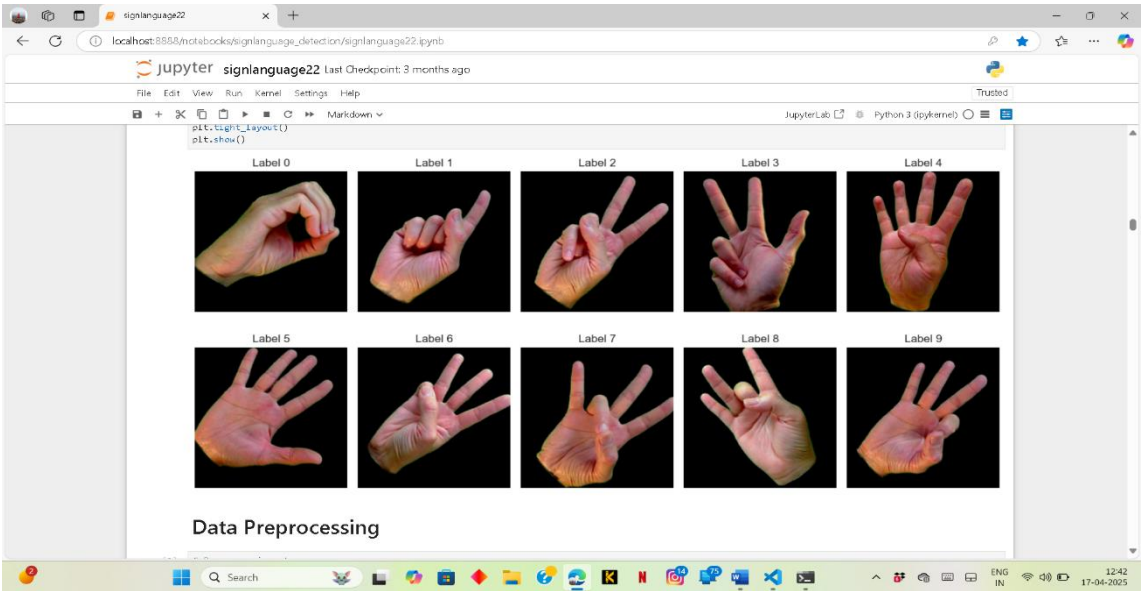
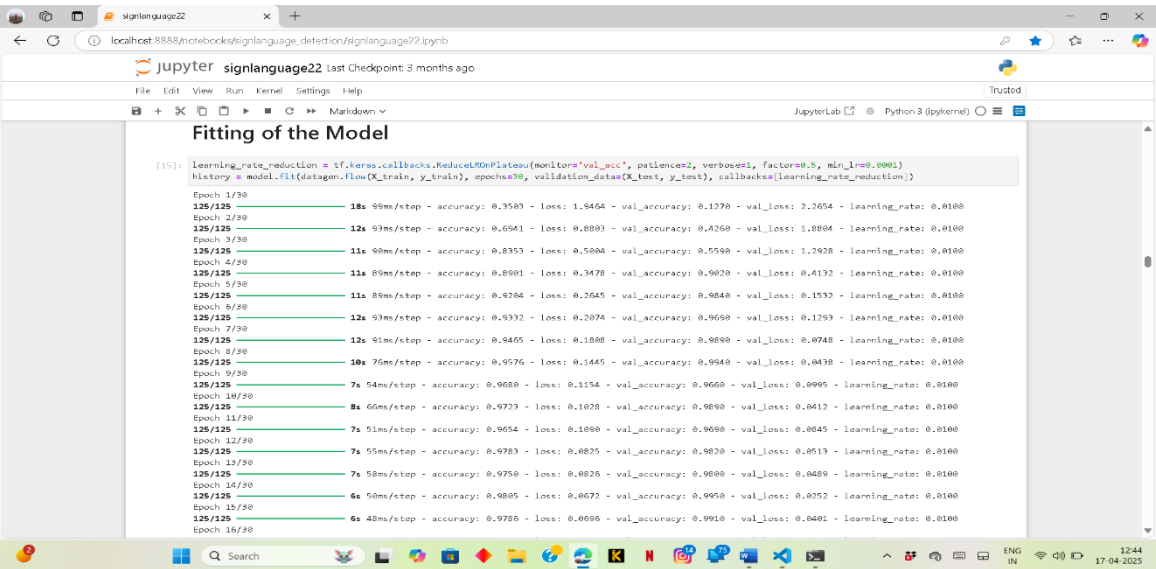
1 import cv2
2 import numpy as np
3 import PIL
4 from PIL import Image
5 import os
6 from PIL import Image
7 import time
8
9 mp_hands = mp.solutions.hands
10 hands = mp_hands.Hands()
11 mp_draw = mp.solutions.drawing_utils
12
13 DEPO_VIDEO = 'demo.mp4'
14 DEPO_IMAGE = 'demo.jpg'
15
16 my_list = []
17
18 st.markdown(
19     """
20     """
21 )
22
23 [True, True, True, True]
24 368 242
25
26 PS C:\Users\sarth\signlanguage_detection\signlanguageDetection>
27 PS C:\Users\sarth\signlanguage_detection\signlanguageDetection>
  
```

```

1 # This file must be used with "source bin/activate" from bash
2 # You cannot run it directly
3
4 deactivate () {
5     # reset old environment variables
6     if [ -n "$OLD_VIRTUAL_PATH" ] ; then
7         PATH="$OLD_VIRTUAL_PATH"
8         export PATH
9         unset OLD_VIRTUAL_PATH
10     fi
11     if [ -n "$OLD_VIRTUAL_PYTHONHOME" ] ; then
12         PYTHONHOME="$OLD_VIRTUAL_PYTHONHOME"
13         export PYTHONHOME
14         unset OLD_VIRTUAL_PYTHONHOME
15     fi
16
17     # call hash to forget past commands. Without forgetting
18     # past commands the $PATH changes we made may not be respected
19     hash -r 2>/dev/null
20
21     if [ -n "$OLD_VIRTUAL_PS1" ] ; then
22         PS1="$OLD_VIRTUAL_PS1"
23         export PS1
24         unset OLD_VIRTUAL_PS1
25     fi
26 }
27
28 [True, True, True, True]
29 368 242
30
31 PS C:\Users\sarth\signlanguage_detection\signlanguageDetection>
32 PS C:\Users\sarth\signlanguage_detection\signlanguageDetection>
  
```

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import plotly.express as px
6 from PIL import Image
7
8 import os
9 import tensorflow as tf
10 from tensorflow import keras
11 from tensorflow.keras import layers
12 from sklearn.model_selection import train_test_split
13 from sklearn.preprocessing import LabelEncoder
14 from tensorflow.keras.models import Sequential
15 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
16 from keras.optimizers import RMSprop
17 from tensorflow.keras.preprocessing.image import ImageDataGenerator
18 from sklearn.metrics import confusion_matrix
19
20 import warnings
21 # Ignore warnings
22 warnings.filterwarnings("ignore")
23
24 Loading the Digit's DataSet
25
26 root_dir = r"C:\Users\sarth\signlanguage_detection\last\American Sign Language Digits Dataset"
  
```



9. 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 , 2019.
- [2] Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan, “ An Efficient Framework for Indian SignLanguage Recognition Using Wavelet Transform” Circuits and Systems, Volume 7, pp 1874- 1883, 2018.
- [3] Mandeep Kaur Ahuja, Amardeep Singh, “Hand Gesture Recognition Using PCA”, International Journal of Computer Science Engineering and Technology (IJCSET), Volume 5, Issue 7, pp. 267-27, July 2020.
- [4] Sagar P.More, Prof. Abdul Sattar, “Hand gesture recognition system for dumb people”.
- [5] International Journal of Science and Research (IJSR)
- [6] Chandandeep Kaur, Nivit Gill, “An Automated System for Indian Sign Language Recognition”, International Journal of Advanced Research in Computer Science and Software Engineering.
- [7] 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 2018 .
- [8] 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.
- [9] 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 (IJCET), Volume 5, Issue 5, pp. 11-18 , May 2017
- [10]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, 2019.
- [11]Ashish Sethi, Hemanth ,Kuldeep Kumar,Bhaskara Rao ,Krishnan R, “Sign Pro-An Application Suite for Deaf and Dumb”, IJCSET , Volume 2, Issue 5, pp-1203-1206, May 2018.