

A
Project Report
on
Sign Language Recognition using CNN

By

Rishika Garg	1901330100224
Sonia Tyagi	1901330100277
Aanchal Rattan	1901330100001
Satyam Rai	2001330109024

Under the Supervision of

Ms. Ishu Varshney
Assistant Professor
(Computer Science and Engineering)

Submitted to the department of Computer Science and Engineering

For the partial fulfillment of the requirements

for award of Bachelor of Technology

in

Computer Science and Engineering



Noida Institute of Engineering & Technology Gr. Noida
Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India
May, 2022-2023

LIST OF CONTENT

	PAGE NO.
CERTIFICATE	6
ACKNOWLEDGEMENT	7
ABSTRACT	8-9
CHAPTER 1	Introduction
	1.1 American Sign Language (ASL)
	1.2 Sign Language Recognition (SLR)
CHAPTER 2	Literature Review
CHAPTER 3	Methodologies
	3.1 Data Acquisition
	3.2 Pre-Processing the Image
	3.3 Feature Extraction
	3.4 Classification
	3.5 Gesture Recognition
	3.6 Predicting the Gestures
	3.7 Steps to Build SLR using Convolutional Neural Network
CHAPTER 4	Tools and Dependencies
	4.1 Keras
	4.2 OpenCV

CHAPTER 5	Loading and Preprocessing Data	38-40
CHAPTER 6	Training the Model	41-46
	6.1 Convolutional	
CHAPTER 7	Creating A window with OpenCV	47-49
CHAPTER 8	Inference	50-53
CHAPTER 9	Conclusion	54-55
REFERENCE		56-60

LIST OF FIGURES

FIGURES	DESCRIPTION	Page no.
1.1.1	American Sign Language	10
1.1.2	Sign Language Recognition	11
1.2.1	Data Glove (example of SLR)	13
1.2.2	TensorFlow Object Detection API	13
2.1	English Alphabet for Sign Language	14
2.2	Leap Motion Controller	16
2.3	Various Sign language Recognition method	17
3.1	Proposed Block Diagram for Model	20
3.1.1	Data Acquisition Through Camera Sensor	21
3.2.1	Preprocessing the image (RGB to gray scale)	23
3.3.1	Feature Extraction of Captured image	24
3.4.1	Classification of American Sign Language	25
3.5.1	Gesture Recognition	27
3.6.1	Predicting Gestures	28
3.7.1	Importing necessary libraries	29
3.7.2	Loading MNIST	30
3.7.3	Preprocessing	30
3.7.4	Designing CNN Architecture	31
3.7.5	Compiling the model	31
3.7.6	Evaluating the model	32

3.7.7	Making predictions	32
4.1.1	Keras	34
4.1.2	Keras in Sign language Recognition	35
4.2.1	OpenCV	36
4.2.2	Sign Language Recognition using OpenCV	37
5.1	Frequency of each label	38
5.2	Plot of pixelated Data 1	39
5.3	Plot of pixelated Data 2	40
6.1	Output Screen of 50 Epochs	41
6.2	Model Summary	42
6.3	Various Layers in our Model	44
6.1.1	Architecture of CNN gesture recognition model	45
6.1.2	Various Layers of CNN	46
7.1	Creating OpenCV Input Window	47
7.2	Functions to Preprocess Input image	48
7.3	OpenCV window with object detection rectangle	49
8.1	Model Accuracy	50
8.2	Command to run Prediction Script	51
8.3	Example of Sign Language detection (o)	52
8.4	Example of Sign Language detection (w)	53
8.5	Accuracy of our model	53

Certificate

This is to certify that the Project report entitled “**Sign language recognition using CNN**” is a record of the work done by the following students:

Student name	Roll No.
Rishika Garg	1901330100224
Sonia Tyagi	1901330100277
Aanchal Rattan	1901330100001
Satyam Rai	2001330109024

I provided supervision and guidance for this project throughout the academic year **2022-23**. The present report is being submitted to the **Noida Institute of Engineering & Technology, Greater Noida**, as a partial fulfillment of the requirements for the **B.Tech. degree in Computer Science and Engineering** from **Dr A P J Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India**.

I wish them all the best for all the endeavors.

Signature of Guide
Ishu Varshney
(Assistant Professor,
Computer Science and
Engineering)

ACKNOWLEDGEMENT

We would like to extend our heartfelt appreciation to **Ms. Ishu Varshney, Assistant Professor at the Department of Computer Science and Engineering, Noida Institute of Engineering & Technology, Greater Noida, Gautam Budha Nagar, Uttar Pradesh, India**, for her invaluable guidance, assistance, and valuable suggestions.

We sincerely acknowledge the contributions of **Dr. Kumud Saxena, Head of the Department of Computer Science and Engineering, Noida Institute of Engineering & Technology, Greater Noida, Gautam Buddha Nagar, Uttar Pradesh, India**, for her inspiring guidance, unwavering support, and supervision throughout the duration of our project.

Date:

Students Name:

Rishika Garg	1901330100224
Sonia Tyagi	1901330100277
Aanchal Rattan	1901330100001
Satyam Rai	2001330109024

ABSTRACT

Communication plays a vital role in sharing information, ideas, and feelings between individuals. However, communication barriers arise when individuals have different languages or face challenges such as deafness and muteness. Deafness refers to the inability to hear, while muteness refers to the inability to speak. Deaf and dumb individuals communicate through sign language, a visual and gestural system. Unfortunately, not everyone can understand of sign language, hindering effective communication between deaf and dumb individuals and the general population.

To address this communication gap, machine learning-based models can be developed. These models can be trained to recognize and interpret different sign language gestures, facilitating real-time translation into spoken or written language. This advancement would greatly assist in bridging the communication gap between normal individuals and the deaf and dumb community.

This report focuses on the development of a real-time Sign Language Recognition system using transfer learning and machine learning techniques. The proposed methodology involves creating an American Sign Language dataset using a webcam, followed by training a TensorFlow model using transfer learning. Transfer learning allows leveraging pre-trained models to expedite the training process and achieve higher accuracy. The system aims to provide accurate recognition of American Sign Language gestures in real-time.

The evaluation of the system demonstrates promising results, achieving an accuracy level of 94%. This indicates the system's ability to effectively recognize and interpret sign language gestures, facilitating communication between deaf and dumb individuals and the general population. The real-time nature of the system enhances its usability and practicality in various scenarios, such as educational settings, social interactions, and public services.

By utilizing machine learning and transfer learning techniques, this research contributes to the field of Sign Language Recognition, addressing the challenges faced by the deaf and dumb community. The proposed system has the potential to enhance inclusivity, promote effective communication, and empower individuals with hearing and speech impairments.

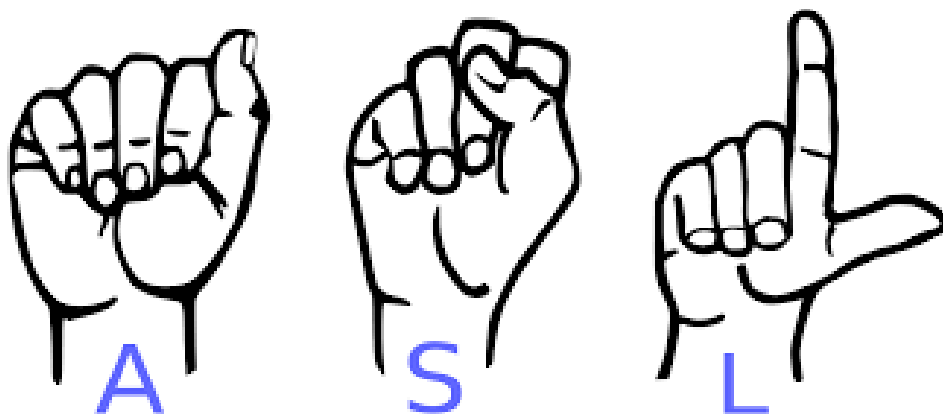
Keywords: American Sign Language, Computer Vision, Machine Learning, Sign Language Recognition (SLR).

CHAPTER 1

INTRODUCTION

1.1 American Sign Language

American Sign Language (ASL) is a visual language used primarily by the deaf and hard-of-hearing community in the United States. It is a complete, distinct language with its own grammar, syntax, and vocabulary. ASL relies on hand movements, facial expressions, and body language to convey meaning, making it a rich and expressive form of communication.



1.1.1 American Sign Language

ASL has a long history that dates back to the early 19th century when the first school for the deaf in America was established in Hartford, Connecticut. Over time, ASL has evolved and developed its own linguistic structure, separate from spoken English. It is important to note that ASL is not a universal language and differs from other sign languages used in different countries.



1.1.2 Sign Language Recognition

The grammar of ASL is different from English grammar. Instead of relying on word order, ASL employs a topic-comment structure where the topic is established first, followed by comments or additional information. This allows for flexibility in expressing ideas and emphasizes the use of facial expressions and body movements to convey nuances of meaning.

ASL vocabulary consists of signs, which are formed by combining hand shapes, movements, and locations. Signs can represent words, phrases, or even concepts. ASL also incorporates fingerspelling, which uses manual alphabet letters to spell out words or convey proper nouns. Additionally, ASL incorporates non-manual markers, such as facial expressions and body postures, which add grammatical information and convey emotions.

ASL plays a crucial role in the lives of the deaf community as it enables effective communication, social interaction, and cultural expression. It is used in various settings, including schools, workplaces, and social gatherings. ASL also has a vibrant Deaf culture associated with it, which encompasses shared experiences, traditions, and values. Deaf individuals who use ASL often have a strong sense of identity and pride in their language and culture.

In recent years, there has been a growing recognition of the importance of ASL and the rights of the deaf community. Many educational institutions now offer ASL classes, allowing individuals to learn and communicate in this unique language. ASL interpreters are also in high demand, bridging the communication gap between the deaf and hearing worlds.

In conclusion, American Sign Language is a visual language that has its own grammar, syntax, and vocabulary. It has a rich history and is an essential tool for communication within the deaf community. ASL's unique structure and expressive nature make it a

fascinating and vibrant language. Its widespread use and recognition contribute to the empowerment and inclusion of the deaf community in American society.

1.2 Sign Language Recognition (SLR)

Sign language recognition (SLR) plays a crucial role in enabling effective communication for individuals with hearing and speech disabilities. Communication, as a fundamental aspect of human interaction, involves the transmission and comprehension of information between individuals. For successful communication, it is essential that the message conveyed by the speaker is received and understood by the listener. Various forms of communication exist, including formal and informal [4], oral and written, non-verbal, feedback, and visual communication, all contributing to effective interaction.

Among these forms, non-verbal communication, which encompasses gestures, facial expressions, and body language, holds particular significance for individuals who are deaf or dumb. Deafness impedes hearing, while dumbness impairs speech, making it challenging for these individuals to establish communication with others. Sign languages have emerged to bridge this communication gap, allowing individuals to convey messages without relying on spoken words. However, a significant barrier remains limited knowledge of sign languages among the general population. Consequently, individuals with hearing and speech impairments face difficulties in communicating with those who have normal hearing, and vice versa.

To address this issue, technology-driven solutions have been developed to facilitate the translation of sign language gestures into commonly spoken languages, such as English. Previous research in this field has explored the use of data gloves, motion capturing systems, sensors [6], and vision based SLR systems. Machine learning algorithms, coupled with tools like MATLAB [1], have been employed to develop Indian Sign Language Recognition systems, achieving impressive accuracy levels ranging from 93% to 96%. However, real-time SLR systems remain a pressing need in the domain.



1.2.1 Data Glove (example of SLR system)

This report aims to contribute to the advancement of SLR systems by developing a real-time solution using the TensorFlow object detection API. The system will be trained using a dataset created through webcam capture. The subsequent sections of this report delve into the related work on SLR systems, data acquisition and generation techniques, the methodology employed in developing the system, experimental evaluations, and concluding remarks. The objective is to enhance communication accessibility for individuals with hearing and speech disabilities, fostering inclusivity and enabling seamless interaction between diverse individuals.



1.2.2 TensorFlow Object Detection API

By leveraging state-of-the-art technology and robust machine learning techniques, this research seeks to facilitate effective communication between individuals with hearing and speech impairments and those with normal hearing. Through the development of a real-time SLR system, it is anticipated that the barriers posed by limited sign language knowledge will be mitigated, promoting understanding and inclusivity in society. Future work in this field will explore further improvements and advancements in SLR technology, aiming to enhance accuracy, speed, and user-friendliness to empower individuals with hearing and speech disabilities.

CHAPTER 2

LITRATURE REVIEW

Sign languages are a structured set of hand gestures with specific meanings, used by individuals with hearing impairments to communicate in their day-to-day life [13]. These visual languages rely on the movements of hands, face, and body as means of communication. It is noteworthy that there are more than 300 distinct sign languages worldwide [7]. Despite the existence of numerous sign languages, the percentage of individuals who are knowledgeable in any of them remains low. Consequently, this poses a significant challenge for individuals with hearing impairments to communicate freely with others. Sign Language Recognition (SLR) offers a solution by enabling communication in sign language without requiring knowledge of it. SLR systems recognize gestures and translate them into commonly spoken languages, such as English.

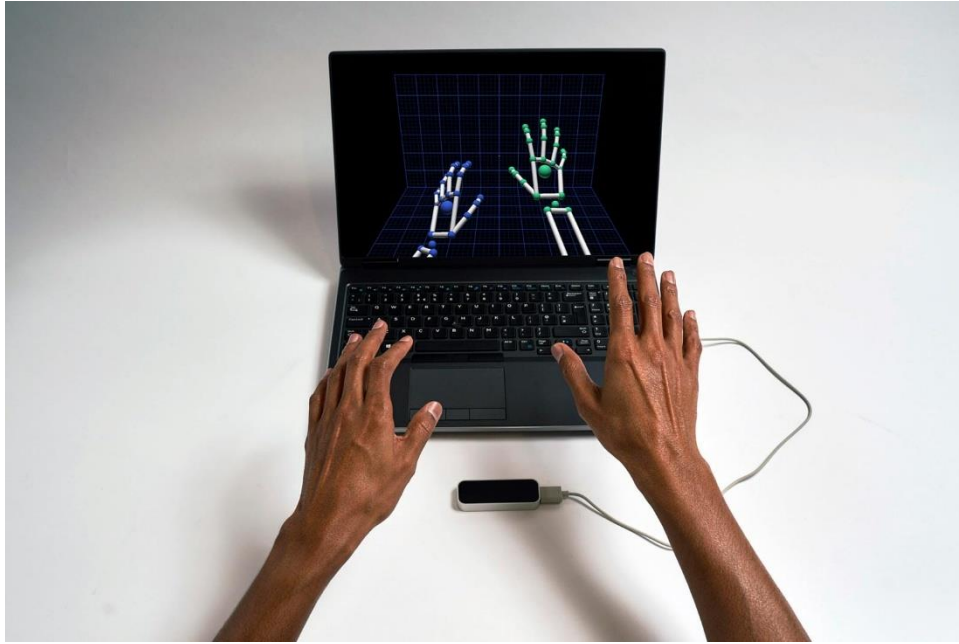


2.1 English Alphabets for Sign Language

SLR is an extensive research field that has seen significant progress, but there are still several areas that require attention. Machine learning techniques have revolutionized electronic systems by enabling decision-making based on data and past experiences. The classification algorithms used in SLR typically rely on two datasets: the training dataset and the testing dataset. The training set provides the necessary experience to the classifier [2], while the testing set evaluates the performance of the model. Numerous researchers have developed effective methods for data acquisition and classification [13][5], employing different approaches.

Previous research in Sign Language Recognition (SLR) can be divided into two primary approaches based on the method of data acquisition: direct measurement methods and vision-based approaches. Direct measurement methods involve utilizing motion data gloves, motion capturing systems, or sensors to extract precise motion data from fingers, hands, and other body parts. This leads to the development of robust SLR methodologies that rely on accurate measurements. In contrast, vision-based SLR approaches focus on extracting spatial and temporal information from RGB images to classify gestures. Typically, these methods involve tracking and isolating hand regions before performing gesture recognition. Hand detection techniques often involve methods like semantic segmentation and skin color detection, taking advantage of the distinct color characteristics of human skin. However, to avoid misidentification of other body parts as hands, modern hand detection methods incorporate face detection and subtraction, as well as background subtraction, to specifically identify the moving parts in each scene. To ensure precise and reliable hand tracking, especially when obstacles are present, researchers have utilized filtering techniques such as Kalman and particle filters. These techniques help improve the accuracy and robustness of hand tracking in SLR systems.

To acquire data in sign language recognition (SLR) systems, various devices and approaches are utilized. The primary device employed for input processing in SLR is a camera [3]. Additionally, other devices such as the Microsoft Kinect can be used, which provides both color and depth video streams. The depth data from the Kinect is particularly useful for background segmentation. Apart from these devices, alternative methods for data acquisition include accelerometers and sensory gloves. Another system commonly utilized for data acquisition is the Leap Motion Controller (LMC) [16][19], developed by the technology company "Leap Motion," now known as "Ultraleap," based in San Francisco. The LMC operates at an approximate rate of 200 frames per second and is capable of detecting and tracking hands, fingers, and finger-like objects.



2.2 Leap motion Controller

Due to the scarcity of available sign language datasets, many researchers opt to record their own training datasets from signers [6]. This is done as finding existing sign language datasets poses a challenge.

In summary, data acquisition in SLR systems involves the use of various devices, as well as alternative methods like accelerometers and sensory gloves. Researchers often resort to recording their own training datasets due to the limited availability of sign language datasets.

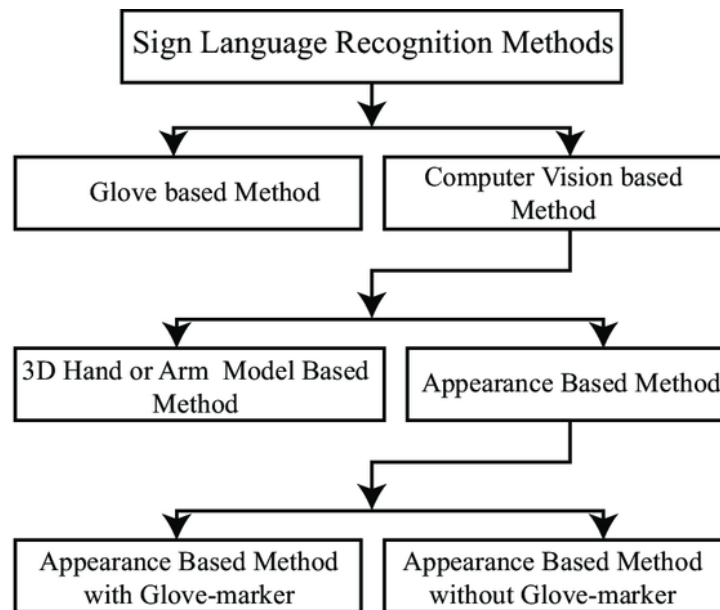
Several different processing methods have been utilized in the development of sign language recognition (SLR) systems. These methods have been explored in previous studies [14][17][18] and encompass a range of techniques.

One widely employed approach is the Hidden Markov Model (HMM), which has proven to be effective in SLR [12]. In addition to HMM, researchers have also explored neural network-based models [15][23][29][27][20], such as Artificial Neural Networks (ANN), Naïve Bayes Classifier (NBC), Multilayer Perceptron (MLP) [16],

unsupervised neural network models like the Self-Organizing Map (SOM) [34] and Self-Organizing Feature Map (SOFM), as well as the Simple Recurrent Network (SRN) [30]. Other approaches include the utilization of Support Vector Machines (SVM) [22] and 3D convolutional residual networks [28].

Furthermore, researchers have developed their own methodologies for SLR, such as the wavelet-based method [21] and the Eigen Value Euclidean Distance approach [30].

These diverse processing methods illustrate the range of techniques employed in SLR research. Each method offers unique advantages and has been explored to improve the accuracy and effectiveness of SLR systems. By building upon these prior approaches, this research aims to contribute to the development of an advanced SLR system.



2.3 Various Sign language Recognition methods

Using various processing methods and application systems has yielded different accuracy results in sign language recognition. The Light-HMM model achieved an accuracy of 83.6%, while the MSHMM model achieved 86.7%. The SVM model attained 97.5% accuracy, the Eigen Value model achieved 97% accuracy, and the Wavelet Family model achieved a perfect 100% accuracy. It is important to note that accuracy is influenced by multiple factors, including the dataset size, image clarity in the dataset acquired through different methods, and the devices utilized. While

different models have demonstrated high accuracy, the choice of processing model alone is not the sole determinant of accuracy.

Two categories of sign language recognition (SLR) systems exist: isolated SLR and continuous SLR. Isolated SLR focuses on recognizing individual gestures, where each image is assigned, a label representing an alphabet, digit, or specific gesture. On the other hand, continuous SLR goes beyond isolated gesture classification. It enables the recognition and translation of complete sentences instead of single gestures [33][25]. Despite the progress made in SLR research, there are still numerous deficiencies that necessitate further investigation and study in the field.

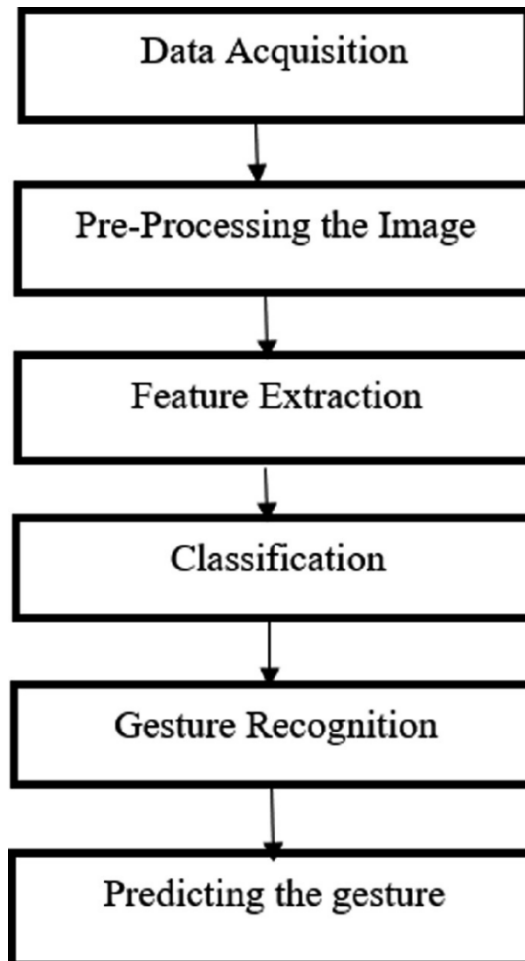
Several issues and challenges require attention in the field of sign language recognition (SLR) [33][2][1][6]. These include:

1. Isolated SLR methods necessitate labor-intensive labeling for each word.
2. Continuous SLR methods rely on isolated SLR systems as building blocks, utilizing temporal segmentation as pre-processing. However, this process is complex and inevitably leads to errors that propagate into subsequent steps. Additionally, sentence synthesis during post-processing poses its own challenges.
3. The devices required for data acquisition in SLR systems are often expensive, necessitating the development of affordable alternatives for commercialization.
4. While web cameras can serve as alternatives to higher specification cameras, the captured images may suffer from blurriness, compromising the overall quality.
5. Data acquisition through sensors introduces various issues such as noise, poor human manipulation, inadequate ground connection, and others.
6. Inaccuracies in vision-based methodologies occur due to the overlapping of hands and fingers, which can pose challenges for accurate recognition.
7. Limited availability of large datasets poses a challenge for training and evaluating SLR systems.
8. Misconceptions exist regarding sign language, such as the misconception that sign language is the same worldwide. Sign languages are based on the spoken language of the corresponding region, leading to linguistic variations.

Addressing these issues and challenges will contribute to the advancement of SLR technology, improving its accuracy, usability, and applicability. By developing more efficient labeling methods, overcoming errors in continuous SLR systems, exploring cost-effective data acquisition options, refining vision-based techniques, and dispelling misconceptions, progress can be made in creating robust and reliable SLR systems. Additionally, efforts to curate and make available large and diverse sign language datasets will further enhance the development and evaluation of SLR systems, enabling more inclusive and effective communication for individuals with hearing and speech disabilities.

CHAPTER 3

METHODOLOGIES



3.1 Proposed block diagram for the model

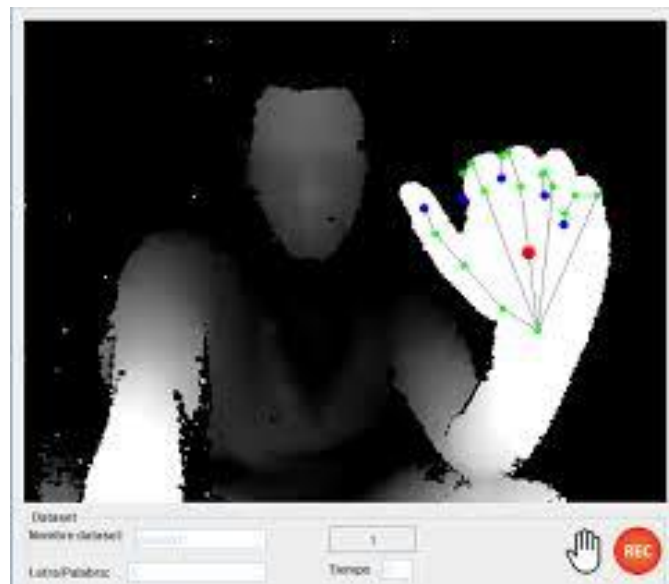
3.1 Data Acquisition

Data acquisition plays a crucial role in sign language recognition systems, enabling accurate and effective communication between individuals who use sign language and those who do not. Sign language recognition aims to bridge the gap between spoken

and sign languages by automatically interpreting and translating sign language gestures into textual or spoken language. To achieve this, a large amount of sign language data needs to be acquired.

Data acquisition in sign language recognition involves capturing and recording sign language gestures using various techniques. One common approach is to use sensor-based devices, such as gloves or motion-capture systems, which can track and record the movements and positions of the hands and other relevant body parts during signing. These devices capture data in the form of spatial coordinates, angles, or joint rotations.

Another method of data acquisition involves using video cameras to record sign language performances. These recordings capture the visual information of the signer's gestures and facial expressions, which are important cues for sign language interpretation. The videos are then processed to extract relevant features for recognition algorithms.



3.1.1 Data Acquisition through Camera Sensor

To ensure accurate and diverse data, it is essential to collect data from a diverse group of sign language users, considering variations in signing styles, regional dialects, and individual nuances. This diversity enhances the robustness and generalization capabilities of sign language recognition systems.

Once the data is acquired, it is annotated with corresponding linguistic information, such as glosses or translations, to create a labeled dataset for training machine learning models. This labeled dataset is used to develop and refine recognition algorithms, enabling the system to accurately recognize and interpret sign language gestures in real-time.

In summary, data acquisition in sign language recognition involves capturing sign language gestures using sensor-based devices or video recordings. The acquired data, annotated with linguistic information, serves as the foundation for training machine learning models to accurately recognize and interpret sign language, facilitating effective communication between sign language users and non-sign language users.

3.2 Pre- Processing the Image

Preprocessing plays a crucial role in sign language recognition systems, as it helps enhance the accuracy and efficiency of the overall process. Image preprocessing techniques are applied to raw sign language images to improve their quality, reduce noise, and extract meaningful features. This stage is typically performed before the actual recognition or classification of signs takes place.

One important step in preprocessing is image normalization, which aims to standardize the image size, orientation, and color. This ensures that all images are consistent and comparable, regardless of variations in lighting conditions or camera angles. Normalization techniques such as resizing, rotation correction, and color adjustment are commonly employed to achieve this.

Another key aspect of preprocessing is noise reduction. Sign language images may be affected by various types of noise, including sensor noise, motion blur, or compression artifacts. Techniques such as filtering, denoising, and deblurring can be employed to remove unwanted noise and enhance the clarity of the sign gestures.

Furthermore, feature extraction is a critical preprocessing step. It involves identifying and extracting relevant features from the preprocessed image, which will be used for subsequent recognition or classification. Feature extraction techniques can include edge detection, contour analysis, and texture analysis. These methods help capture the distinctive visual elements of sign gestures, enabling accurate identification and discrimination.



3.2.1 Preprocessing the image (RGB to gray scale)

Overall, preprocessing in sign language recognition involves a series of techniques to standardize, clean, and extract meaningful features from raw sign language images. This preprocessing stage sets the foundation for subsequent recognition or classification algorithms, ultimately contributing to the accurate interpretation of sign language gestures.

3.3 Feature Extraction

Feature extraction plays a crucial role in sign language recognition, enabling accurate interpretation of manual gestures and movements used in sign languages. Sign language recognition systems rely on extracting meaningful and discriminative features from video or image data to capture the distinct characteristics of different signs. These features serve as input for machine learning algorithms to classify and interpret signs.

One commonly used technique for feature extraction in sign language recognition is hand tracking. This involves detecting and tracking the hand region within the video or image frames. Various computer vision algorithms, such as color-based segmentation, template matching, or deep learning-based approaches, can be employed to accomplish this task. Hand tracking allows for precise localization of the hand, which is essential for extracting relevant features.

Another important aspect of feature extraction in sign language recognition is capturing motion information. Dynamic features, such as hand trajectories, velocity, and acceleration, provide valuable insights into the temporal dynamics of sign gestures. Techniques like optical flow analysis or motion history image generation can be employed to extract motion-related features. These features enable the recognition

system to differentiate between different signs based on their specific movement patterns.

Furthermore, hand shape and orientation are crucial features in sign language recognition. Shape-based features involve extracting information about the configuration and pose of the hand. This can be achieved through techniques like shape context, contour-based descriptors, or hand posture estimation algorithms. By capturing the unique shape and orientation characteristics of signs, these features contribute significantly to accurate recognition.



3.3.1 Feature Extraction of Captured image

In summary, feature extraction in sign language recognition involves techniques for hand tracking, motion analysis, and shape-based feature extraction. By combining these features, sign language recognition systems can effectively interpret and understand the manual gestures and movements used in sign languages, enabling improved communication and interaction between individuals who use sign language and the broader community.

3.4 Classification

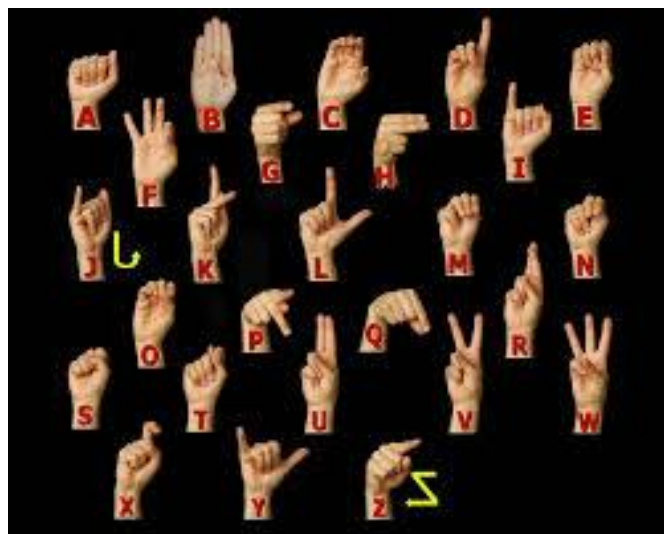
Classification in sign language recognition plays a crucial role in accurately interpreting and understanding sign language gestures. Sign language is a visual-

gestural communication system used by deaf and hard-of-hearing individuals. It relies on various hand shapes, movements, and facial expressions to convey meaning.

In sign language recognition, classification involves the identification and categorization of different sign language gestures. The process typically starts with capturing video data of the signer's hands and face using cameras or sensors. The captured data is then preprocessed to extract relevant features, such as hand shape, hand movement, and facial expressions.

Once the features are extracted, a classification algorithm is employed to assign a specific label or meaning to the sign gesture. Machine learning techniques, such as deep learning and support vector machines, are commonly used for this purpose. These algorithms are trained on large datasets of sign language gestures, enabling them to learn patterns and associations between input features and corresponding signs.

The success of classification in sign language recognition depends on several factors, including the quality and quantity of training data, the choice of features, and the performance of the classification algorithm. Researchers continually work to improve the accuracy and efficiency of classification models by exploring novel techniques, developing robust feature extraction methods, and incorporating real-time processing capabilities.



3.4.1 Classification of ASL

Accurate classification in sign language recognition has numerous applications, including assistive communication devices for deaf individuals, automatic sign language translation systems, and educational tools for sign language learners. It holds the potential to bridge communication barriers and enhance inclusivity for the deaf community, promoting equal access to information and opportunities.

3.5 Gesture Recognition

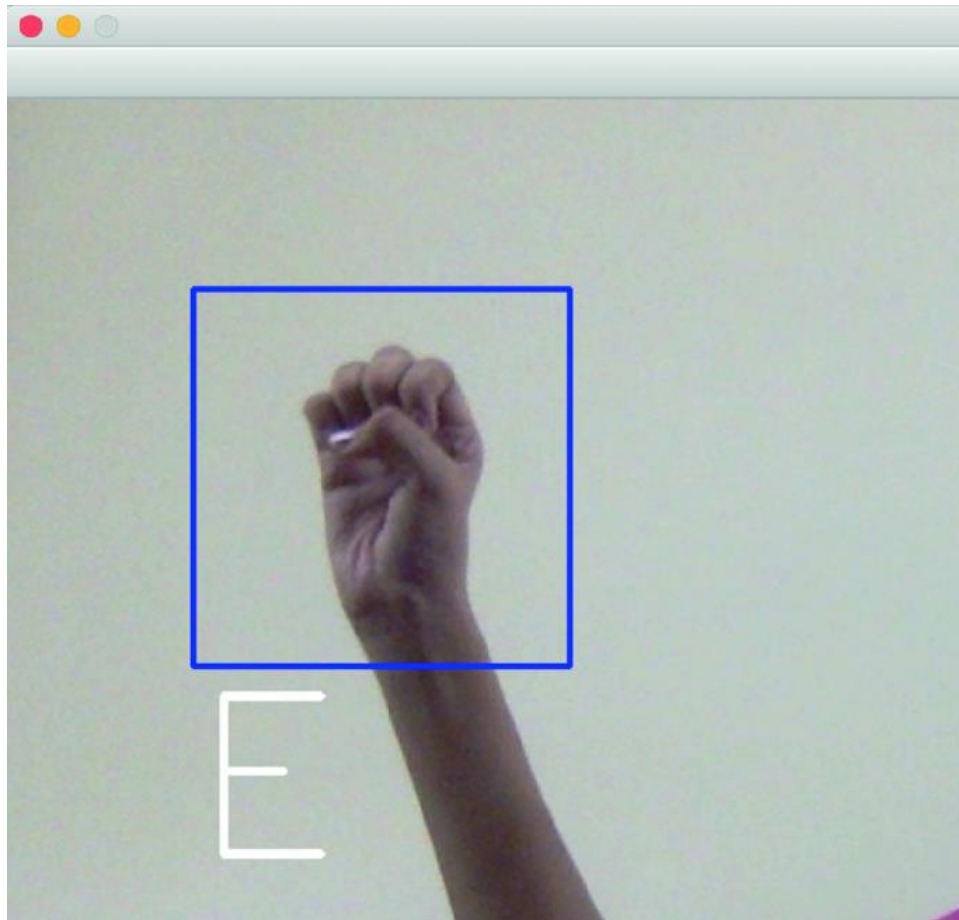
Gesture recognition plays a crucial role in sign language recognition, facilitating effective communication between individuals who are deaf or hard of hearing and the hearing community. Sign language is a visual and gestural form of communication that uses hand shapes, facial expressions, and body movements to convey meaning.

In sign language recognition systems, gesture recognition algorithms are employed to analyze and interpret the gestures performed by the signer. These algorithms capture and process data from various sensors, such as cameras or gloves, to recognize and understand the signs being made.

The recognition process involves several steps. First, the system captures video or sensor data of the signer's gestures. Then, it extracts relevant features from the data, such as hand shapes, movement trajectories, and facial expressions. These features are then fed into machine learning models or pattern recognition algorithms, which have been trained on sign language datasets, to classify and recognize the corresponding signs.

Gesture recognition in sign language recognition faces unique challenges compared to other gesture recognition applications. Sign language involves intricate and dynamic movements, making it necessary to capture and analyze fine-grained details accurately. Furthermore, the recognition system must be robust to variations in lighting conditions, different signing styles, and individual differences in hand shapes and movements.

Advancements in computer vision and machine learning techniques have significantly improved gesture recognition in sign language recognition systems. Deep learning approaches, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated promising results in accurately recognizing signs. Additionally, the availability of large sign language datasets has contributed to the development and evaluation of more robust recognition models.



3.5.1 Gesture Recognition

Gesture recognition in sign language recognition has the potential to enhance communication and inclusivity for individuals with hearing impairments. Continued research and development in this field will pave the way for more sophisticated and accurate sign language recognition systems, ultimately fostering better accessibility and understanding between deaf or hard of hearing individuals and the wider community.

3.6 Predicting the gestures

Predicting gestures in sign language recognition is a challenging task that has garnered significant attention in recent years. Sign language is a visual-spatial language used by

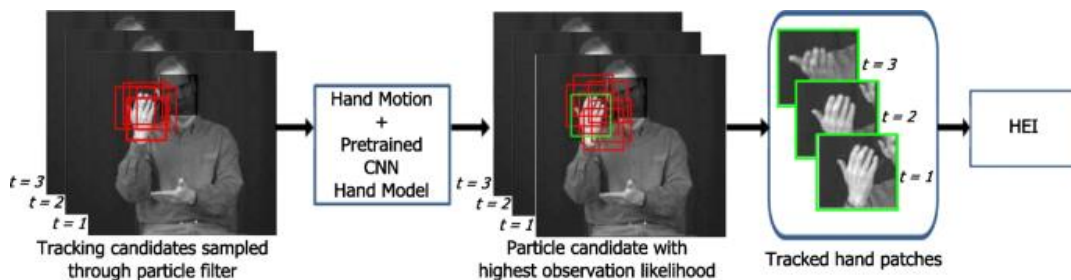
individuals with hearing impairments to communicate. The ability to accurately interpret and predict gestures is crucial for facilitating effective communication between deaf and hearing individuals.

Several approaches have been developed to address this problem. One common method involves using depth sensors or cameras to capture hand movements and extract relevant features. These features can include hand shape, hand position, and motion trajectories. Machine learning algorithms, such as support vector machines, random forests, or deep neural networks, are then employed to classify and predict the corresponding gestures.

To improve accuracy, researchers have also explored incorporating temporal information by analyzing the dynamics of gestures over time. This can be achieved through recurrent neural networks (RNNs) or convolutional neural networks (CNNs) with temporal convolutions. These models can capture the sequential nature of sign language and make predictions based on both the current frame and previous frames.

Furthermore, recent advancements in deep learning have led to the emergence of transformer-based architectures for sign language recognition. Transformers, originally designed for natural language processing tasks, have shown promising results in capturing long-range dependencies and modeling sign language sequences effectively.

However, despite significant progress, there are still challenges to overcome in sign language recognition. Variability in sign language across different individuals, dialects, and contexts poses a considerable obstacle. Additionally, real-time prediction and the ability to handle occlusions or background noise remain active areas of research.



3.6.1 Predicting the gestures

Overall, the development of accurate and robust gesture prediction models in sign language recognition holds immense potential for improving communication and inclusivity for individuals with hearing impairments. Continued research and innovation in this field will contribute to bridging the gap between deaf and hearing communities.

3.7 Steps to build sign language recognition model using CNN

To build a sign language recognition (SLR) model using Convolutional Neural Networks (CNN) and the MNIST dataset, we followed these steps:

1. Import the necessary libraries: Start by importing the required libraries, such as TensorFlow and Keras, which provide tools for building and training neural networks.

```
In [1]: 1 import keras
        2 import numpy as np
        3 import pandas as pd
        4 import cv2
        5 from matplotlib import pyplot as plt
        6 from keras.models import Sequential
        7 from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
        8 from keras.datasets import mnist
        9 import matplotlib.pyplot as plt
       10 from keras.utils import np_utils
       11 from keras.optimizers import SGD

Using TensorFlow backend.
```

3.7.1 Importing necessary libraries

2. Load and preprocess the MNIST dataset: The MNIST dataset consists of images of handwritten digits, but we will repurpose it for sign language recognition. Load the dataset and preprocess it by reshaping the images and normalizing the pixel values.

```
In [2]: 1 train = pd.read_csv('C:/Users/37896/Sign-Language-Recognition-master/sign_mnist_train/train.csv')
        2 train.head(5)
```

```
Out[2]:
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	3	107	118	127	134	139	143	146	150	153	...	207	207	207	207	206	206	206	204	204
1	6	155	157	156	156	156	157	156	158	158	...	69	149	128	87	94	163	175	103	103
2	2	187	188	188	187	187	186	187	188	187	...	202	201	200	199	198	199	198	195	195
3	2	211	211	212	212	211	210	211	210	210	...	235	234	233	231	230	226	225	222	222
4	13	164	167	170	172	176	179	180	184	185	...	92	105	105	108	133	163	157	163	163

5 rows x 785 columns

```
In [3]: 1 test = pd.read_csv('C:/Users/37896/Sign-Language-Recognition-master/sign_mnist_test/test.csv')
        2 test.head(5)
```

```
Out[3]:
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	6	149	149	150	150	150	151	151	150	151	...	138	148	127	89	82	96	106	112	112
1	5	126	128	131	132	133	134	135	135	136	...	47	104	194	183	186	184	184	184	184
2	10	85	88	92	96	105	123	135	143	147	...	68	166	242	227	230	227	226	225	225
3	0	203	205	207	206	207	209	210	209	210	...	154	248	247	248	253	236	230	240	240

3.7.2 Loading the MNIST dataset

```
In [4]: 1 y_train = train['label'].values
        2 y_test = test['label'].values
        3
        4 X_train = train.drop(['label'],axis=1)
        5 X_test = test.drop(['label'], axis=1)
        6
        7 X_train = np.array(X_train.iloc[:,:])
        8 X_train = np.array([np.reshape(i, (28,28)) for i in X_train])
        9
       10 X_test = np.array(X_test.iloc[:,:])
       11 X_test = np.array([np.reshape(i, (28,28)) for i in X_test])
       12
       13 num_classes = 26
       14 y_train = np.array(y_train).reshape(-1)
       15 y_test = np.array(y_test).reshape(-1)
       16
       17 y_train = np.eye(num_classes)[y_train]
       18 y_test = np.eye(num_classes)[y_test]
```

```
In [5]: 1
        2 X_train = X_train.reshape((27455, 28, 28, 1))
        3 X_test = X_test.reshape((7172, 28, 28, 1))
```

3.7.3 Preprocessing

3. Prepare the training and testing data: Split the dataset into training and testing sets. This division allows you to train the model on a portion of the data and evaluate its performance on the remaining portion. In our case data was already spitted into training and testing datasets.

4. Design the CNN architecture: Define the architecture of the CNN model. Typically, it consists of convolutional layers, pooling layers, and fully

connected layers. Experiment with different layer configurations to achieve better accuracy.

```
In [9]: 1 classifier = Sequential()
2 classifier.add(Conv2D(filters=8, kernel_size=(3,3),strides=(1,1),padding='same',input_shape=(28,28,1),activation='relu', da
3 classifier.add(MaxPooling2D(pool_size=(2,2)))
4 classifier.add(Conv2D(filters=16, kernel_size=(3,3),strides=(1,1),padding='same',activation='relu'))
5 classifier.add(Dropout(0.5))
6 classifier.add(MaxPooling2D(pool_size=(4,4)))
7 classifier.add(Dense(128, activation='relu'))
8 classifier.add(Flatten())
9 classifier.add(Dense(26, activation='softmax'))
10 classifier.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
11
```

3.7.4 Designing the CNN architecture

5. Compile the model: Specify the loss function, optimizer, and evaluation metric for the model. The loss function measures the model's performance, the optimizer adjusts the weights during training, and the evaluation metric provides a measure of accuracy.

```
In [7]: 1 classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
2 classifier.fit(X_train, y_train, epochs=50, batch_size=100)
3

Epoch 1/50
275/275 [=====] - 14s 42ms/step - loss: 5.3196 - accuracy: 0.3963
Epoch 2/50
275/275 [=====] - 11s 42ms/step - loss: 0.8084 - accuracy: 0.7275
Epoch 3/50
275/275 [=====] - 11s 40ms/step - loss: 0.5548 - accuracy: 0.8101
Epoch 4/50
275/275 [=====] - 11s 41ms/step - loss: 0.4411 - accuracy: 0.8461
Epoch 5/50
275/275 [=====] - 11s 40ms/step - loss: 0.3472 - accuracy: 0.8792
Epoch 6/50
275/275 [=====] - 11s 40ms/step - loss: 0.3174 - accuracy: 0.8904
Epoch 7/50
275/275 [=====] - 11s 40ms/step - loss: 0.2674 - accuracy: 0.9059
Epoch 8/50
275/275 [=====] - 11s 39ms/step - loss: 0.2416 - accuracy: 0.9162
Epoch 9/50
275/275 [=====] - 11s 39ms/step - loss: 0.2135 - accuracy: 0.9250
Epoch 10/50
275/275 [=====] - 11s 39ms/step - loss: 0.2027 - accuracy: 0.9291
Epoch 11/50
275/275 [=====] - 11s 40ms/step - loss: 0.1900 - accuracy: 0.9348
Epoch 12/50
```

3.7.5 Compiling the Model

6. Train the model: Fit the model to the training data by specifying the number of epochs (iterations) and the batch size. During training, the model will learn to recognize sign language gestures.

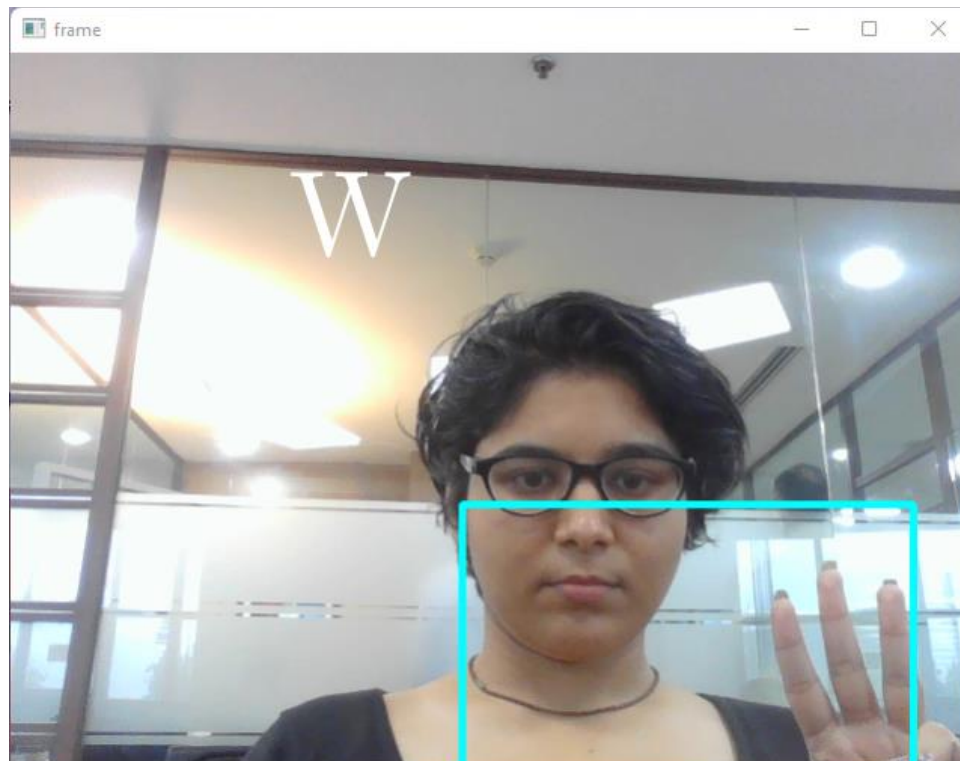
7. Evaluate the model: After training, evaluate the model's performance on the testing data. This step helps assess how well the model generalizes to unseen examples.

```
In [18]: 1 accuracy = classifier.evaluate(x=X_test,y=y_test,batch_size=32)
          2 print("Accuracy: ",accuracy[1])

7172/7172 [=====] - 0s 68us/step
Accuracy: 0.9408812046848857
```

3.7.6 Evaluating the Model

8. Make predictions: Use the trained model to make predictions on new images of sign language gestures. This will allow you to test the model's real-world performance.



3.7.7 Making the Predictions

9. Fine-tune and optimize: Experiment with different hyperparameters, layer configurations, and techniques like data augmentation to improve the model's accuracy and robustness.

10. Deploy the model: Once you are satisfied with the model's performance, you can deploy it for sign language recognition in various applications, such as real-time gesture interpretation or communication aids for individuals with hearing impairments.

Basically, our dataset consists of many images of 24 (except J and Z) American Sign Language alphabets. Each image has a size of 28×28 pixels which means a total of 784 pixels per image.

Alternate approach for this project, the first step is to capture images using a webcam and OpenCV. Once the images are collected, they will be labeled for Sign Language Detection using LabelImg. Next, the TensorFlow Object Detection pipeline configuration should set up to ensure proper functioning. With the configuration in place, Transfer Learning is applied to train a deep learning model. This step involves leveraging pre-trained models and fine-tuning them on the labeled sign language images. As a result of the training process, a robust model will be developed which will be able to detect sign language in real time using the OpenCV library.

CHAPTER 4

TOOLS AND DEPENDENCIES

4.1 Keras

Keras is a popular deep learning framework widely used for various machine learning tasks, including sign language recognition. Sign language is a visual form of communication used by deaf individuals to express themselves. Recognizing and interpreting sign language gestures can enable computers to understand and interact with individuals who use sign language as their primary means of communication.



4.1.1 Keras

In sign language recognition, Keras provides a powerful and user-friendly interface to build and train deep neural networks. Keras offers a high-level API that allows researchers and developers to easily construct complex models for sign language recognition tasks. It simplifies the process of designing neural networks by providing pre-defined layers and functions that can be easily integrated into the model architecture.

One of the key challenges in sign language recognition is capturing the temporal dynamics of the gestures. Sign language involves intricate hand movements and facial expressions that convey meaning. Keras provides recurrent neural network (RNN) layers such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), which are well-suited for modeling sequential data. These layers enable the network to learn long-term dependencies and capture the temporal structure of sign language gestures.

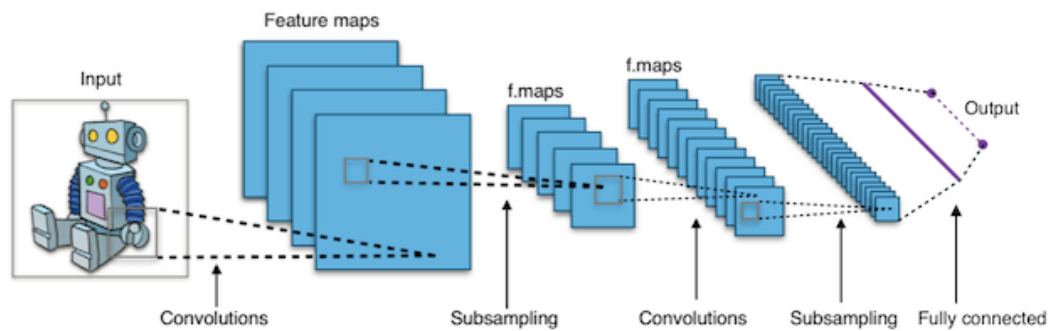
To train a sign language recognition model in Keras, a large dataset of sign language videos or images is required. These datasets typically include samples of various sign language gestures performed by different individuals. Keras provides utilities for data preprocessing, such as image augmentation and sequence padding, which can enhance the model's ability to generalize to unseen sign language gestures.

Once the data is prepared, the model architecture can be defined using Keras' functional API or sequential API. The model architecture can include convolutional layers to extract spatial features from the sign language images or video frames. These features are then fed into the recurrent layers to capture the temporal dynamics. Additional dense layers can be used for classification or regression tasks, depending on the specific application.

During training, Keras offers a range of optimization algorithms, such as stochastic gradient descent (SGD) and Adam, which can be used to update the model's parameters and minimize the loss function. The model's performance can be monitored using various evaluation metrics, including accuracy, precision, and recall.

After training, the model can be deployed for real-time sign language recognition. Keras provides methods for model serialization and deployment, allowing the model to be integrated into applications or embedded systems. This enables the recognition of sign language gestures in real-world scenarios, facilitating communication and interaction between deaf individuals and technology.

In conclusion, Keras is a versatile and powerful deep learning framework that can be effectively used for sign language recognition. Its user-friendly interface, support for recurrent neural networks, and extensive optimization algorithms make it an excellent choice for developing accurate and efficient sign language recognition systems. By leveraging the capabilities of Keras, researchers and developers can contribute to bridging the communication gap between deaf individuals and the wider society.



4.1.2 Keras

4.2 OpenCV

Sign language recognition using OpenCV is a fascinating application of computer vision technology that aims to bridge the communication gap between hearing-impaired individuals and the rest of the world. OpenCV, an open-source computer vision library, provides a powerful framework for processing and analyzing visual data, making it an ideal choice for developing sign language recognition systems.



4.2.1 OpenCV

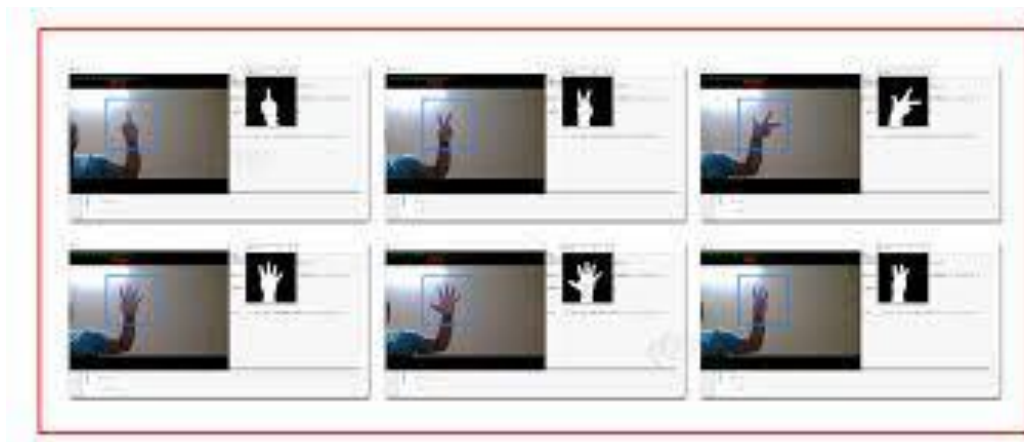
At its core, sign language recognition involves capturing video input of sign language gestures and translating them into meaningful text or spoken words. OpenCV offers a rich set of functionalities that enable developers to extract and interpret relevant features from video frames, allowing for accurate recognition of sign language gestures.

The first step in sign language recognition using OpenCV involves capturing video input from a camera or a pre-recorded dataset. OpenCV provides easy-to-use interfaces for accessing video streams and reading video files. Once the video input is obtained, it can be processed frame by frame.

In sign language recognition, hand tracking is a crucial step. OpenCV provides various methods for detecting and tracking hands in video frames. Techniques like background subtraction, motion detection, and skin color detection can be used to isolate the hand region. Once the hand is detected, its position, shape, and movement can be analyzed using OpenCV's contour detection and tracking algorithms.

After hand tracking, the next step is hand gesture recognition. OpenCV allows developers to extract meaningful features from the hand region, such as finger positions, hand shape, and motion trajectories. Machine learning algorithms, such as support vector machines or convolutional neural networks, can be trained on labeled sign language gesture data to recognize specific signs. OpenCV seamlessly integrates with popular machine learning frameworks like TensorFlow and PyTorch, enabling efficient training and deployment of sign language recognition models.

Once the sign language gestures are recognized, the corresponding text or spoken words can be generated as output. This output can be displayed on a screen, spoken through a text-to-speech engine, or even translated into different languages to facilitate communication between sign language users and non-signers.



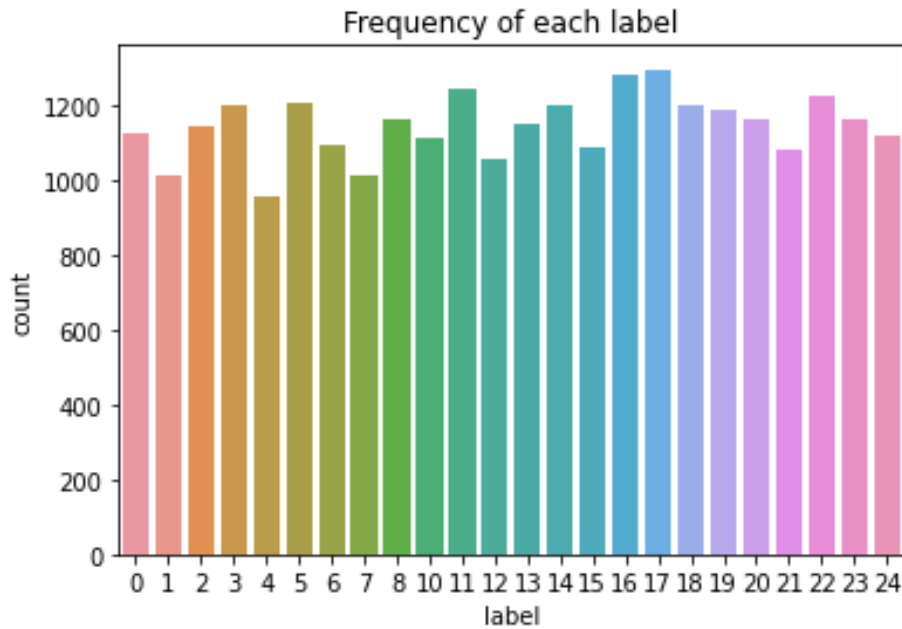
4.2.2 Sign Language recognition using OpenCV

Sign language recognition using OpenCV has the potential to enhance accessibility and inclusivity for the deaf and hard-of-hearing community. It enables real-time interpretation of sign language, allowing individuals to communicate more effectively with the wider society. With ongoing advancements in computer vision and machine learning, coupled with the versatility of OpenCV, the accuracy and reliability of sign language recognition systems continue to improve, paving the way for a more inclusive world.

CHAPTER 5

LOADING AND PREPROCESSING THE DATA

The dataset used in this study was obtained from Kaggle and comprises two primary files: the training and testing csv files. The training dataset contains a collection of over 7,000 samples, each showcasing different gestures from the American Sign Language. The format of the dataset closely follows the structure of the classic MNIST dataset. For each alphabetic letter A-Z, excluding J (9) and Z (25) due to gesture motions, there is a corresponding label (ranging from 0 to 25).



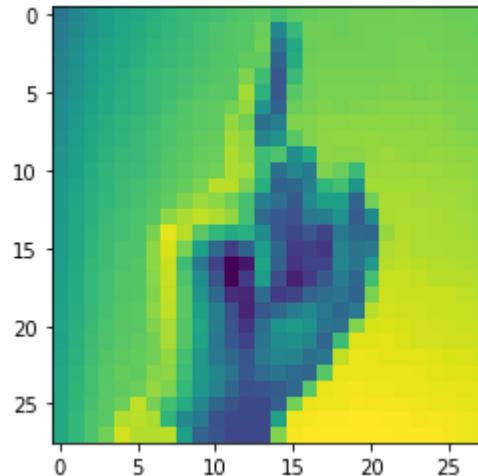
5.1 Frequency of each label

The training dataset consisted of 27,455 cases, while the test dataset had 7,172 cases. Although they were smaller in size compared to the standard MNIST dataset, they

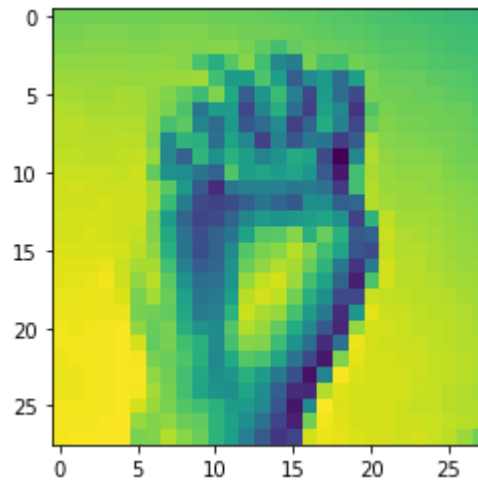
shared similarities in terms of structure. Each dataset had a header row indicating the label and individual pixel values ranging from 1 to 784, representing a grayscale image of size 28x28 with intensity values between 0 and 255.

Initially, the original hand gesture image data consisted of multiple users performing the gestures against various backgrounds. To expand the dataset, the Sign Language MNIST data was created by significantly augmenting the limited number of color images (1,704) that were not cropped around the hand region of interest. This involved employing an image processing pipeline using ImageMagick, which included steps such as isolating the hands through cropping, converting to grayscale, resizing, and generating over 50 variations to increase the dataset size.

The modification and expansion process employed different filters such as 'Mitchell,' 'Robidoux,' 'Catrom,' 'Spline,' and 'Hermite.' Additionally, random pixilation of around 5%, brightness and contrast adjustments of up to $\pm 15\%$, and rotations of up to 3 degrees were applied. Due to the small dimensions of the original images, these modifications effectively transformed the resolution and class separation in intriguing and controllable ways.



5.2 Plot of pixelated data 1



5.3 Plot of pixelated data 2

To load the data, the dataset provided is in CSV format, which stands for Comma-Separated Values. The dataset consists of four main components: train_X, test_X, train_Y, and test_Y. The train_X and test_X variables hold the pixel values for each image in the dataset, while train_Y and test_Y contain the corresponding labels for those images.

For preprocessing the data, the train_X and test_X variables are structured as arrays containing pixel values. In order to create an image from these values, the data needs to be reshaped. Since the desired image size is 28x28 pixels, the array of pixel values must be divided into groups of 28x28 pixels.

By dividing the array into these groups, each section corresponds to a specific pixel in the image. This process ensures that the pixel values are arranged correctly and that the resulting image maintains the desired dimensions of 28x28 pixels. This preprocessing step is essential to prepare the data for further analysis or model training, as it allows for proper visualization and interpretation of the images contained within the dataset.

CHAPTER 6

TRAINING THE MODEL

The `classifier` object is created as a sequential model, which allows for the sequential addition of layers. The model architecture consists of several layers. The first layer is a 2D convolutional layer (`Conv2D`) with 8 filters, a kernel size of 3x3, and a stride of 1x1. The padding is set to 'same', ensuring that the output size matches the input size. The activation function used is ReLU. The input shape of the layer is defined as (28, 28, 1), indicating the input image dimensions of 28x28 pixels with a single channel (grayscale).

Following the convolutional layer, a max-pooling layer (`MaxPooling2D`) with a pool size of 2x2 is added. This layer reduces the spatial dimensions of the input by selecting the maximum value within each pooling region.

```
Epoch 9/50
27455/27455 [=====] - 2s 87us/step - loss: 0.1016 - acc: 0.9652
Epoch 10/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0914 - acc: 0.9681
Epoch 11/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0865 - acc: 0.9696
Epoch 12/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0849 - acc: 0.9708
Epoch 13/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0761 - acc: 0.9746
Epoch 14/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0677 - acc: 0.9773
Epoch 15/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0706 - acc: 0.9755
Epoch 16/50
27455/27455 [=====] - 2s 88us/step - loss: 0.0586 - acc: 0.9803
Epoch 17/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0664 - acc: 0.9775
Epoch 18/50
27455/27455 [=====] - 2s 87us/step - loss: 0.0519 - acc: 0.9826
```

6.1 Output of 50 Epochs (accuracy ranges from 25-99)

The subsequent layers include another convolutional layer with 16 filters, a dropout layer (`Dropout`) with a rate of 0.5 (used for regularization to prevent overfitting), and another max-pooling layer with a pool size of 4x4.

Afterwards, a dense layer (`Dense`) with 128 units and ReLU activation is added, followed by a flattening layer (`Flatten`) to convert the 2D feature maps into a 1D feature vector.

Finally, the output layer is defined as a dense layer with 26 units (representing the 26 possible classes or categories) and a softmax activation function, which outputs the probability distribution over the classes.

In summary, this code snippet sets up a CNN model for image classification, comprising convolutional, pooling, dropout, and dense layers. The model is designed to take grayscale images with dimensions of 28x28 pixels as input and classify them into one of the 26 possible classes.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 8)	80
max_pooling2d_1 (MaxPooling2)	(None, 14, 14, 8)	0
conv2d_2 (Conv2D)	(None, 14, 14, 16)	1168
dropout_1 (Dropout)	(None, 14, 14, 16)	0
max_pooling2d_2 (MaxPooling2)	(None, 3, 3, 16)	0
dense_1 (Dense)	(None, 3, 3, 128)	2176
flatten_1 (Flatten)	(None, 1152)	0
dense_2 (Dense)	(None, 26)	29978
Total params: 33,402		
Trainable params: 33,402		
Non-trainable params: 0		

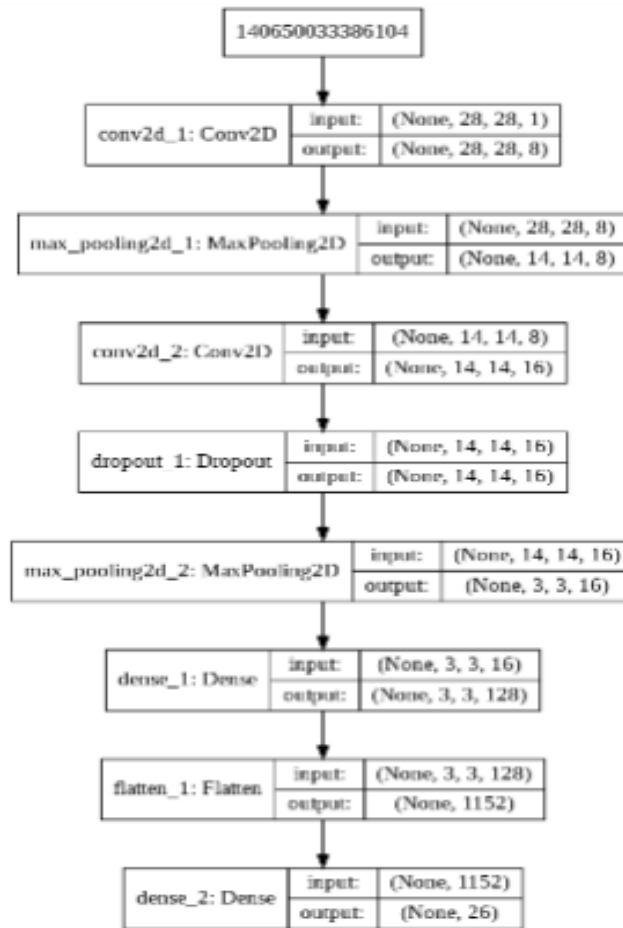
6.2 Model summary

The `compile()` function is used to configure the classifier model. The optimizer is set to 'SGD' (Stochastic Gradient Descent), which is a popular optimization algorithm for training deep learning models. The loss function is specified as 'categorical_crossentropy', which is commonly used for multi-class classification problems. Additionally, the model is set to compute the 'accuracy' metric to evaluate its performance.

The `fit()` function is called to train the classifier model. The training data, `X_train` and `y_train`, are provided as input. The `epochs` parameter is set to 50, indicating the number of times the model will iterate over the entire training dataset during training. The `batch_size` parameter is set to 100, indicating the number of samples that will be used in each update of the model's weights.

By executing these lines of code, the classifier model will be trained using the provided training data for 50 epochs, with the specified batch size. The model will optimize its parameters using the SGD optimizer and minimize the categorical cross-entropy loss. The accuracy metric will be computed to assess the model's performance during training.

Out[16]:



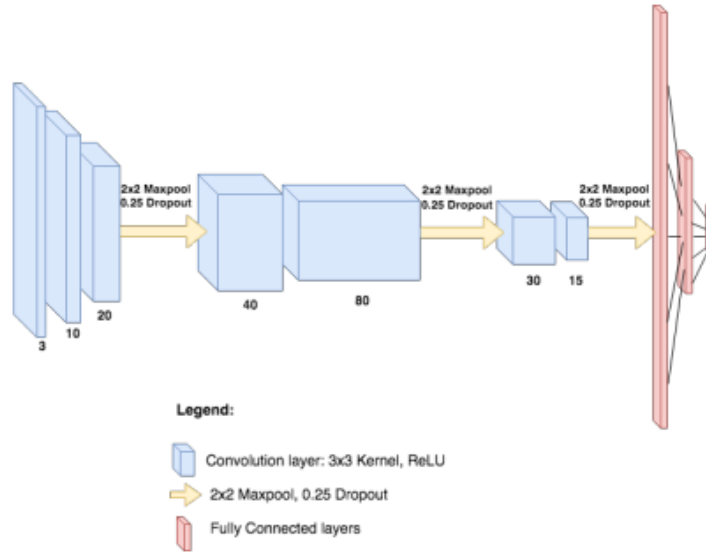
6.3 Various layers in our Model

6.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) have proven to be highly effective in various computer vision tasks, including sign language recognition. CNNs offer a powerful approach for automatically learning discriminative features from input images.

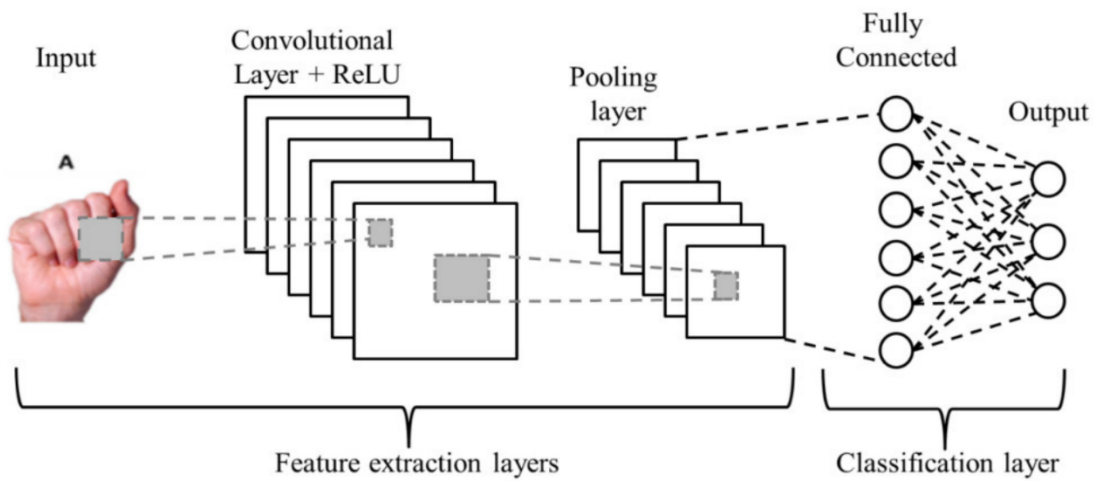
CNNs consist of multiple layers, including convolutional, pooling, and fully connected layers. The convolutional layers apply filters to capture spatial patterns in the input images, while the pooling layers reduce the spatial dimensions, enhancing translation

invariance. The fully connected layers learn high-level representations and make predictions based on the learned features.



6.1.1 Architecture of a CNN gesture recognition model

To apply CNNs to sign language recognition, a labeled dataset of sign language images is required for training. CNN learns to extract relevant features that distinguish different signs. During training, the model adjusts its parameters using optimization techniques like backpropagation to minimize the difference between predicted and actual labels.



6.1.2 Various layers of CNN

Once trained, CNN can recognize and classify sign language gestures in real-time by processing input images through its layers. It can handle variations in hand shape, orientation, and movement, making it suitable for capturing the unique characteristics of sign language.

Overall, CNNs provide a robust and efficient approach for sign language recognition, enabling effective communication between individuals with hearing and speech impairments and those with normal hearing.

CHAPTER 7

CREATING A WINDOW WITH OPENCV

To facilitate input from the webcam, a window needs to be created. The captured image is required to be a grayscale image with dimensions of 28x28 pixels. This specific size was chosen because the model was trained using images of the same dimensions. Therefore, in order to ensure compatibility with the trained model, the input image must adhere to the 28x28 grayscale format. By adhering to these specifications, the input image can be properly processed and analyzed by the model for accurate sign language recognition.

```
def main():
    l = []

    while True:
        cam_capture = cv2.VideoCapture(0)
        _, image_frame = cam_capture.read()
        # Select ROI
        im2 = crop_image(image_frame, 300,300,300,300)
        image_grayscale = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)

        image_grayscale_blurred = cv2.GaussianBlur(image_grayscale, (15,15), 0)
        im3 = cv2.resize(image_grayscale_blurred, (28,28), interpolation = cv2.INTER_AREA)
        im4 = np.resize(im3, (28, 28, 1))
        im5 = np.expand_dims(im4, axis=0)
        pred_probab, pred_class = keras_predict(model, im5)
        curr = prediction(pred_class)
        cv2.putText(image_frame, curr, (700, 300), cv2.FONT_HERSHEY_COMPLEX, 4.0, (255, 255, 255), lineType=cv2.LINE_AA)
        # Display cropped image
        cv2.rectangle(image_frame, (300, 300), (600, 600), (255, 255, 00), 3)
        cv2.imshow("frame",image_frame)
        #cv2.imshow("Image4",resized_img)
        cv2.imshow("Image3",image_grayscale_blurred)

        if cv2.waitKey(25) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            break
```

7.1 Creating OpenCV Input Window

The `main()` performs various image processing operations on a webcam-captured image in the context of sign language recognition. Within an infinite loop, the code starts by capturing an image from the webcam using the OpenCV library's `VideoCapture()` function. The captured image frame is stored in the variable `image_frame`.

Next, the code crops the required part of the image using the `crop_image()` function, specifying the dimensions (300, 300, 300, 300). The cropped image (`im2`) is then converted to grayscale using the `cv2.cvtColor()` function with the `COLOR_BGR2GRAY` color conversion parameter.

```
def prediction(pred):
    return(chr(pred+ 65))

def keras_predict(model, image):
    data = np.asarray( image, dtype="int32" )
    pred_probab = model.predict(data)[0]
    pred_class = list(pred_probab).index(max(pred_probab))
    return max(pred_probab), pred_class

def keras_process_image(img):
    image_x = 28
    image_y = 28
    img = cv2.resize(img, (1,28,28), interpolation = cv2.INTER_AREA)
    return img

def crop_image(image, x, y, width, height):
    return image[y:y + height, x:x + width]
```

7.2 Functions to preprocess the input image

To reduce noise and smoothen the image, a Gaussian blur is applied to the grayscale image using the `cv2.GaussianBlur()` function. A kernel size of (15, 15) is used for blurring. The resulting blurred image (`image_grayscale_blurred`) is resized to a dimension of 28x28 pixels using the `cv2.resize()` function.

To comply with the input requirements of certain machine learning models, the image is further modified by resizing the dimensions from (28, 28) to (28, 28, 1) using NumPy's `np.resize()` function. Finally, the image is expanded to include an additional dimension representing the batch size using `np.expand_dims()`, resulting in the final image shape of (1, 28, 28, 1).



7.3 OpenCV window with object detection rectangle

This code snippet provides a basic image preprocessing pipeline that prepares the webcam-captured image for further analysis and classification within a sign language recognition system.

CHAPTER 8

INFERENCE

The results obtained from our model for predicting the alphabets from input images are present in this section. It is important to note that the model outputs integers instead of alphabets due to the labels being represented as integers. For instance, the label 'A' is represented as 1, 'B' as 2, 'C' as 3, and so on. Our trained model achieved an impressive accuracy of 94%, indicating its ability to recognize alphabets with high precision. This accuracy was obtained under favorable conditions, namely plain background and sufficient lighting conditions.

```
1 accuracy = classifier.evaluate(x=X_test,y=y_test,batch_size=32)
2 print("Accuracy: ",accuracy[1])

7172/7172 [=====] - 0s 68us/step
Accuracy:  0.9408812046848857
```

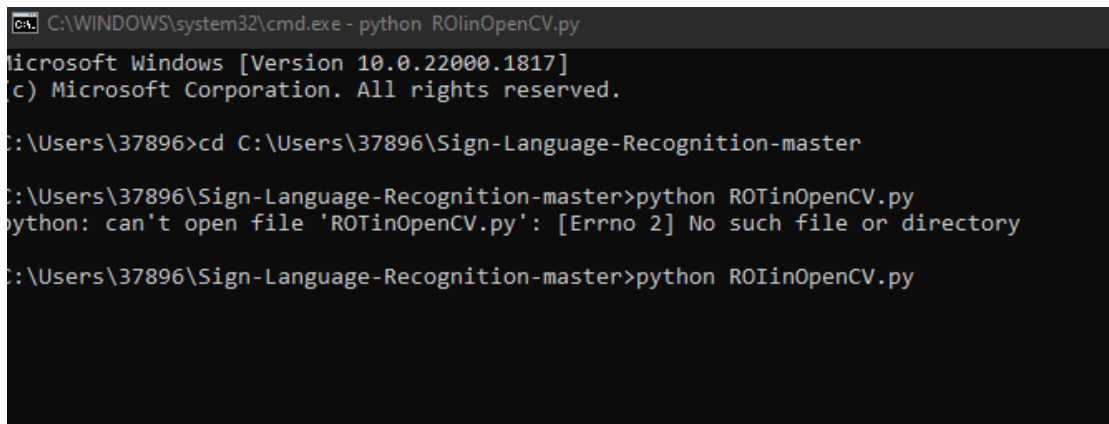
8.1 Model Accuracy

Throughout the evaluation stage, the model consistently exhibited reliable performance by precisely forecasting the corresponding alphabet based on a provided input image. The training procedure entailed comprehensive optimization through the utilization of the Stochastic Gradient Descent (SGD) optimizer and the minimization of the categorical cross-entropy loss function. It is important to note that although our model achieved outstanding results under the specified circumstances, its effectiveness might fluctuate when confronted with intricate backgrounds or insufficient illumination. These particular factors can introduce supplementary hurdles to the accurate identification of alphabets. Nevertheless, when presented with unadorned backgrounds and appropriate lighting conditions, our model effectively demonstrates its ability to discern alphabets without encountering significant difficulties.

To run the final model is saved locally with the name 'CNNmodel.h5', this model is further used to predict the outputs using the script 'ROlinOpenCV.py' in real time.

Final model can be run by following these steps:

1. Open the command line
2. Make sure you are in the directory where all files are stored, if not then change the directory using cd command
3. Run this command



```
C:\WINDOWS\system32\cmd.exe - python ROTinOpenCV.py
Microsoft Windows [Version 10.0.22000.1817]
(c) Microsoft Corporation. All rights reserved.

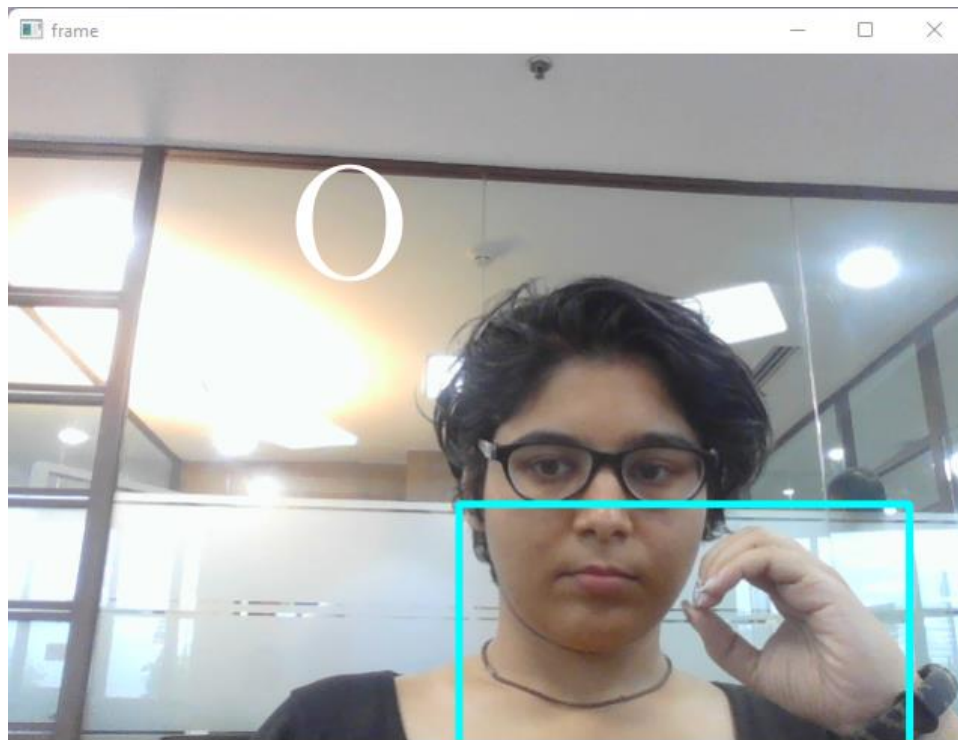
C:\Users\37896>cd C:\Users\37896\Sign-Language-Recognition-master

C:\Users\37896\Sign-Language-Recognition-master>python ROTinOpenCV.py
python: can't open file 'ROTinOpenCV.py': [Errno 2] No such file or directory

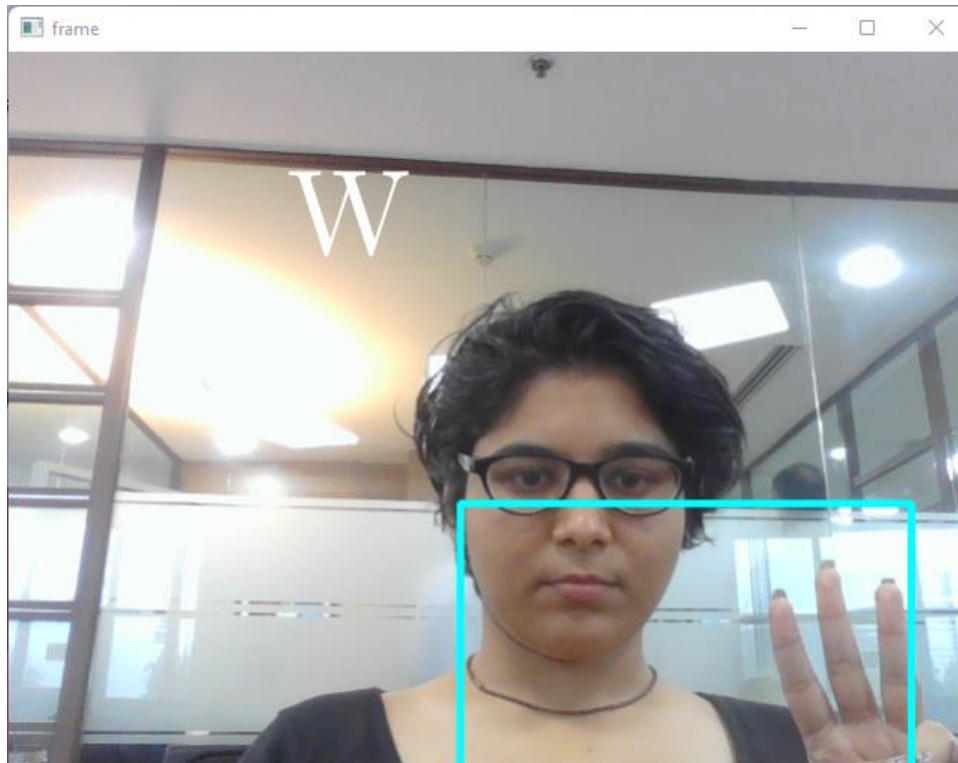
C:\Users\37896\Sign-Language-Recognition-master>python ROIinOpenCV.py
```

8.2 Command to Run the prediction script

4. After running the above command, you will see an OpenCV window, use this to make the model predict the output.
5. Make sure you are in a well-lit room, and your hand gestures are inside the blue borders



8.3 Example of Sign Language detection(O)



8.4 Example of Sign Language Detection(W)

In conclusion, our model exhibits a promising accuracy of 94% in predicting alphabets from input images. Further research and enhancements can be pursued to improve its robustness in diverse environments, ultimately expanding its practical applications in real-world settings.

```
In [18]: 1 accuracy = classifier.evaluate(x=X_test,y=y_test,batch_size=32)
          2 print("Accuracy: ",accuracy[1])

7172/7172 [=====] - 0s 68us/step
Accuracy:  0.9408812046848857
```

8.5 Accuracy of our model

CHAPTER 9

CONCLUSION

Sign languages are visual forms of communication that utilize hand, body, and facial movements. They serve as vital means of expression for individuals with disabilities, enabling them to convey their thoughts and emotions. However, a significant challenge arises from the limited understanding of sign languages among the general population, impeding effective communication. To address this issue, automated systems for Sign Language Recognition (SLR) have been developed, which aim to translate sign language into commonly spoken languages. This study focuses on implementing an SLR system using the TensorFlow object detection API, with training conducted on a dataset containing the American Sign Language alphabet. The proposed system enables real-time detection of sign language gestures, offering a cost-effective solution by utilizing webcam images captured through Python and OpenCV.

The developed SLR system exhibits an average confidence rate of 94%, indicating its accuracy in recognizing sign language gestures. Although this level of performance is commendable, it is important to note that the system's performance is influenced by the size and diversity of the training dataset. In this case, the dataset utilized for training is relatively small and limited. To further enhance the system's capabilities, it is recommended to enlarge and diversify the dataset. By incorporating a more extensive range of gestures, the system would be able to recognize and interpret a broader array of sign language gestures.

In the future, expanding the dataset will enable the SLR system to recognize a wider range of gestures, contributing to its versatility and applicability. Additionally, while this paper focuses on utilizing the TensorFlow model for training and inference, the system can be easily adapted to incorporate alternative models. By exploring different models, the system's performance and accuracy can be further improved, tailoring it to specific requirements and preferences. Furthermore, the system's potential extends beyond the American Sign Language alphabet dataset. By modifying and adapting the dataset, the SLR system can be implemented for other sign languages, opening doors for cross-cultural communication and inclusivity.

In conclusion, this project presents a real-time Sign Language Recognition system developed using the TensorFlow object detection API. The system successfully translates sign language gestures into commonly spoken language, addressing the communication barriers faced by individuals with disabilities. Through webcam image capture using Python and OpenCV, the system provides a cost-effective and accessible solution. While achieving an average confidence rate of 85.45%, there are opportunities for further improvement, such as expanding the dataset and exploring alternative models. By continuing to advance the capabilities of SLR systems, we can foster inclusivity, empower individuals with disabilities, and enable effective communication across different sign languages.

References

- [1] Dutta, K.K., Bellary, S.A.S.: Machine Learning Techniques for Indian Sign Language Recognition. Int. Conf. Curr. Trends Comput. Electr. Electron. Commun. CTCEEC 2017. 333–336 (2018). <https://doi.org/10.1109/CTCEEC.2017.8454988>
- [2] Rosero-Montalvo, P.D., Godoy-Trujillo, P., Flores-Bosmediano, E., Carrascal-Garcia, J., Otero-Potosi, S., Benitez-Pereira, H., Peluffo-Ordonez, D.H.: Sign Language Recognition Based on Intelligent Glove Using Machine Learning Techniques. 2018 IEEE 3rd Ecuador Tech. Chapters Meet. ETCM 2018. (2018). <https://doi.org/10.1109/ETCM.2018.8580268>
- [3] Nikam, A.S., Ambekar, A.G.: Sign language recognition using image-based hand gesture recognition techniques. Proc. 2016 Online Int. Conf. Green Eng. Technol. IC-GET 2016. (2017). <https://doi.org/10.1109/GET.2016.7916786>.
- [4] Kapur, R.: The Types of Communication. MIJ. 6, (2020)
- [5] Zheng, L., Liang, B., Jiang, A.: Recent Advances of Deep Learning for Sign Language Recognition. DICTA 2017 - 2017 Int. Conf. Digit. Image Comput. Tech. Appl. 2017- Decem, 1–7 (2017). <https://doi.org/10.1109/DICTA.2017.8227483>
- [6] Suharjito, Anderson, R., Wiryana, F., Ariesta, M.C., Kusuma, G.P.: Sign Language Recognition Application Systems for Deaf-Mute People: A Review Based on Input Process-Output. Procedia Comput. Sci. 116, 441–448 (2017). <https://doi.org/10.1016/J.PROCS.2017.10.028>
- [7] Bragg, D., Koller, O., Bellard, M., Berke, L., Boudreault, P., Braffort, A., Caselli, N., Huenerfauth, M., Kacorri, H., Verhoef, T., Vogler, C., Morris, M.R.: Sign Language Recognition, Generation, and Translation: An Interdisciplinary Perspective. 21st Int. ACM SIGACCESS Conf. Comput. Access. (2019). <https://doi.org/10.1145/3308561>
- [8] Rautaray, S.S.: A Real Time Hand Tracking System for Interactive Applications. Int. J. Comput. Appl. 18, 975–8887 (2011)
- [9] Zhang, Z., Huang, F.: Hand tracking algorithm based on super-pixels feature. Proc. - 2013 Int. Conf. Inf. Sci. Cloud Comput. Companion, ISCC-C 2013. 629–634 (2014). <https://doi.org/10.1109/ISCC-C.2013.77>.

- [10] Lim, K.M., Tan, A.W.C., Tan, S.C.: A feature covariance matrix with serial particle filter for isolated sign language recognition. *Expert Syst. Appl.* 54, 208–218 (2016). <https://doi.org/10.1016/J.ESWA.2016.01.047>
- [11] Lim, K.M., Tan, A.W.C., Tan, S.C.: Block-based histogram of optical flow for isolated sign language recognition. *J. Vis. Commun. Image Represent.* 40, 538–545 (2016). <https://doi.org/10.1016/J.JVCIR.2016.07.020>.
- [12] Gaus, Y.F.A., Wong, F.: Hidden Markov Model - Based gesture recognition with overlapping hand-head/hand-hand estimated using Kalman Filter. *Proc. - 3rd Int. Conf. Intell. Syst. Model. Simulation, ISMS 2012.* 262–267 (2012). <https://doi.org/10.1109/ISMS.2012.67>.
- [13] Konstantinidis, D., Dimitropoulos, K., Daras, P.: Sign language recognition based on hand and body skeletal data. *3DTV-Conference. 2018-June,* (2018). <https://doi.org/10.1109/3DTV.2018.8478467>
- [14] Cheok, M.J., Omar, Z., Jaward, M.H.: A review of hand gesture and sign language recognition techniques. *Int. J. Mach. Learn. Cybern.* 2017 101. 10, 131–153 (2017). <https://doi.org/10.1007/S13042-017-0705-5>
- [15] Cui, R., Liu, H., Zhang, C.: A Deep Neural Framework for Continuous Sign Language Recognition by Iterative Training. *IEEE Trans. Multimed.* 21, 1880–1891 (2019). <https://doi.org/10.1109/TMM.2018.2889563>
- [16] Mohandes, M., Aliyu, S., Deriche, M.: Arabic sign language recognition using the leap motion controller. *IEEE Int. Symp. Ind. Electron.* 960–965 (2014). <https://doi.org/10.1109/ISIE.2014.6864742>.
- [17] Camgöz, N.C., Koller, O., Hadfield, S., Bowden, R.: Sign language transformers: Joint end-to-end sign language recognition and translation. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* 10020–10030 (2020). <https://doi.org/10.1109/CVPR42600.2020.01004>.
- [18] Wadhawan, A., Kumar, P.: Sign Language Recognition Systems: A Decade Systematic Literature Review. *Arch. Comput. Methods Eng.* 2019 283. 28, 785–813 (2019). <https://doi.org/10.1007/S11831-019-09384-2>.
- [19] Enikeev, D.G., Mustafina, S.A.: Sign language recognition through Leap Motion controller and input prediction algorithm. *J. Phys. Conf. Ser.* 1715, 012008 (2021). <https://doi.org/10.1088/1742-6596/1715/1/012008>.

- [20] Sharma, A., Sharma, N., Saxena, Y., Singh, A., Sadhya, D.: Benchmarking deep neural network approaches for Indian Sign Language recognition. *Neural Comput. Appl.* 2020 3312. 33, 6685–6696 (2020). <https://doi.org/10.1007/S00521-020-05448-8>.
- [21] Kalash, E.A., Garewal, N.S.: Sign Language Recognition System. *Int. J. Comput. Eng. Res.* 6.
- [22] Quocthang, P., Dung, N.D., Thuy, N.T.: A comparison of SimpSVM and RVM for sign language recognition. *ACM Int. Conf. Proceeding Ser.* 98–104 (2017). <https://doi.org/10.1145/3036290.3036322>.
- [23] Bantupalli, K., Xie, Y.: American Sign Language Recognition using Deep Learning and Computer Vision. *Proc. - 2018 IEEE Int. Conf. Big Data, Big Data* 2018. 4896–4899 13 (2019). <https://doi.org/10.1109/BIGDATA.2018.8622141>.
- [24] Kishore, P.V.V., Prasad, M. V.D., Prasad, C.R., Rahul, R.: 4-Camera model for sign language recognition using elliptical fourier descriptors and ANN. *Int. Conf. Signal Process. Commun. Eng. Syst. - Proc. SPACES 2015, Assoc. with IEEE.* 34–38 (2015). <https://doi.org/10.1109/SPACES.2015.7058288>.
- [25] Cui, R., Liu, H., Zhang, C.: Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017.* 2017-Janua, 1610–1618 (2017). <https://doi.org/10.1109/CVPR.2017.175>
- [26] Singha, J., Das, K.: Indian Sign Language Recognition Using Eigen Value Weighted Euclidean Distance Based Classification Technique. *IJACSA) Int. J. Adv. Comput. Sci. Appl.* 4, (2013).
- [27] Kumar, P., Roy, P.P., Dogra, D.P.: Independent Bayesian classifier combination-based sign language recognition using facial expression. *Inf. Sci. (Ny).* 428, 30–48 (2018). <https://doi.org/10.1016/J.INS.2017.10.046>.
- [28] Pu, J., Zhou, W., Li, H.: Iterative alignment network for continuous sign language recognition. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* 2019-June, 4160–4169 (2019). <https://doi.org/10.1109/CVPR.2019.00429>.
- [29] Hore, S., Chatterjee, S., Santhi, V., Dey, N., Ashour, A.S., Balas, V.E., Shi, F.: Indian Sign Language Recognition Using Optimized Neural Networks. *Adv.*

- Intell. Syst. Comput. 455, 553–563 (2017). https://doi.org/10.1007/978-3-319-38771-0_54.
- [30] Gao, W., Fang, G., Zhao, D., Chen, Y.: A Chinese sign language recognition system based on SOFM/SRN/HMM. *Pattern Recognit.* 37, 2389–2402 (2004). <https://doi.org/10.1016/J.PATCOG.2004.04.008>
 - [31] Liang, Z., Liao, S., Hu, B.: 3D Convolutional Neural Networks for Dynamic Sign Language Recognition. *Comput. J.* 61, 1724–1736 (2018). <https://doi.org/10.1093/COMJNL/BXY049>.
 - [32] Pigou, L., Van Herreweghe, M., Dambre, J.: Gesture and Sign Language Recognition with Temporal Residual Networks. *Proc. - 2017 IEEE Int. Conf. Comput. Vis. Work. ICCVW 2017. 2018-Janua*, 3086–3093 (2017). <https://doi.org/10.1109/ICCVW.2017.365>.
 - [33] Huang, J., Zhou, W., Zhang, Q., Li, H., Li, W.: Video-based Sign Language Recognition without Temporal Segmentation
 - [34] Tewari, D., Srivastava, S.K.: A Visual Recognition of Static Hand Gestures in Indian Sign Language based on Kohonen Self-Organizing Map Algorithm. *Int. J. Eng. Adv. Technol.* 165 (2012).