INDIAN SIGN LANGUAGE RECOGNITION SYSTEM

Prepared by

Aahona Sinha Roy (12230621054)

Aditya Ghosh (12230621042)

Chandrani Das Sarma (12230621038)

Sattam Bhattacharyya (12230621028)

Under the esteemed guidance of

Mr. Ritesh Prasad

Project Report

Submitted in the partial fulfillment of the requirement for the degree of

B.Tech in Artificial Intelligence and Machine Learning

Department of Artificial Intelligence and Machine Learning

St. Thomas' College of Engineering & Technology

Affiliated to

Maulana Abul Kalam Azad University of Technology, Kolkata

June 2025

St. Thomas' College of Engineering & Technology

Department of Artificial Intelligence and Machine Learning

This is to certify that the work in preparing the project titled **Indian Sign Language Recognition System** is carried out by

Aahona Sinha Roy (12230621054)

Aditya Ghosh (12230621042)

Sattam Bhattacharya (12230621028)

Chandrani Das Sarma (12230621038)

under my guidance during the session (2024-2025) and accepted in the partial fulfillment of the requirement for the degree of Artificial Intelligence and Machine Learning.

Mr. Ritesh Prasad
Assistant Professor
Department of Computer Science Engineering and Technology,
St. Thomas' College of Engineering & Technology

Mr A. K Siromoni
HOD (AIML),
Department of Artificial Intelligence and Machine Learning,
St Thomas' College of Engineering and Technology

St. Thomas' College of Engineering & Technology

Department of Artificial Intelligence and Machine Learning

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project mentor Mr. Ritesh Prasad for his invaluable guidance and support throughout the duration of the project. His expertise, encouragement and unwavering commitment have been instrumental in shaping the direction and success of this endeavour.

We are also grateful to other faculty members of our institute, for their support, assistance and encouragement throughout the project. Their contributions have been deeply appreciated and have significantly enriched the outcome of this project report.

Aahona Sinha Roy University roll number : 12230621054	Aditya Ghosh University roll number : 12230621042
Sattam Bhattacharya University roll number : 12230621028	Chandrani Das Sarma University roll number : 12230621038

St. Thomas' College of Engineering & Technology

Department of Artificial Intelligence and Machine Learning

DECLARATION

We declare that this written submission represents our ideas in our own words and we have adequately cited and referenced the original sources. We also declare that we have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aahona Sinha Roy
University roll number : 12230621054

Aditya Ghosh
University roll number : 12230621042

Sattam Bhattacharya
University roll number : 12230621028

Chandrani Das Sarma
University roll number : 12230621038

Table of Contents

I.PREAMBLE

I.I Vision and Mission of the institute

Vision of the institute

To evolve as an industry-oriented, research-based Institution for creative solutions in various engineering domains, with an ultimate objective of meeting technological challenges faced by the Nation and the Society.

Mission of the institute

1. To enhance the quality of engineering education and delivery through accessible, comprehensive, and research-oriented teaching-learning-assessment processes in the state-of-art environment.

2. To create opportunities for students and faculty members to acquire professional knowledge and develop managerial, entrepreneurial, and social attitudes with highly ethical and moral values.

3. To satisfy the ever-changing needs of the nation with respect to evolution and absorption of sustainable and environment-friendly technologies for effective creation of a knowledge-based society in the global era.

Vision and Mission of the Department

Vision of the Department

To continually improve upon the teaching-learning processes and research with a goal to develop quality technical manpower with sound academic and practical experience, who can respond to challenges and changes happening dynamically in Computer Science and Engineering.

Mission of the Department

1. To inspire the students to work with the latest tools and to make them industry-ready.
2. To impart research-based technical knowledge.
3. To groom the department as a learning centre to inculcate advanced technologies in Computer Science and Engineering with social and environmental awareness

I.II Program Outcome (PO) and Program Specific Outcome (PSO):

Program Outcome (PO):

PO1. **Technical Proficiency:** Graduates will demonstrate advanced skills in mobile application development, specializing in image recognition, database management, and real-time alarm systems.
PO2. **Problem-solving Skills:** Graduates will exhibit strong problem-solving capabilities, addressing the unique challenges faced by visually impaired individuals in medication management through innovative and user-friendly solutions.

PO3. **Ethical Considerations:** Graduates will integrate ethical considerations into the app's design, ensuring privacy, security, and compliance with healthcare regulations.

PO4. User-Centric Design: Graduates will apply principles of user-centric design, creating an intuitive and accessible interface tailored to the specific needs of visually impaired users.

PO5. **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

PO6. Project Management: Graduates will demonstrate effective project management skills, ensuring timely delivery and continuous improvement of the medication management app.

PO7. **Healthcare Domain Knowledge:** Graduates will acquire a foundational understanding of healthcare practices, enabling them to align app functionalities with the needs of both users and healthcare providers.

PO8. Continuous Learning: Graduates will exhibit a commitment to continuous learning, staying updated on emerging technologies and advancements in healthcare to enhance app features.

PO9. **User Training and Support:** Graduates will design and implement comprehensive user training modules and ongoing support systems to facilitate effective app utilization by visually impaired individuals.

PO10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions..

PO11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team to manage projects and in multi-disciplinary environments.

PO12. **Life-long Learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcome (PSO):

PS01: Programming: Apply programming skills to provide efficient and cost-effective solutions for engineering problems.

PS02: Research: Contribute towards the knowledge base through significant research activities in recent trends.

PS03: Industry Readiness: Develop themselves into professionals who are ready to serve the industry and society at large.

I.III PO and PSO Mapping with Justification:

	PO I	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
ISLRS	3	3	3	3	3	-	-	-	-	3	-	3	3	-	3

JUSTIFICATION:

PO1: We're honing our skills in computer vision, machine learning, and natural language processing to develop a robust and accurate ISL interpretation system.

PO2: We're identifying and addressing the unique challenges faced by the deaf community, creating innovative solutions to bridge the communication gap.

PO3: We're prioritizing ethical AI practices, ensuring fairness, privacy, and accessibility in our system.

PO4: We're designing a user-friendly interface that caters to the specific needs of deaf and hearing individuals.

PO5: We have continuously investigated various research papers to find the existing solution to the problem and effectively implemented that on our project in search of a better outcome.

PO10: We're effectively communicating our technical findings and project progress to our mentor to guide us through the project and also we are effectively working as a team to provide solutions for the desired problem statement.

PO12: Our project has given us the ability to work with Deep Neural Networks that is indeed a life long learning in the field of computer science and Technology.

PSO1: (Programming) Apply programming skills for providing efficient and cost effective solutions for engineering problems.

PSO3: (Industry Readiness) Develop themselves into professionals who are ready to serve the industry and society at large

ABSTRACT

The "Indian Sign Language Recognition System" aims to develop an automated system for recognizing Indian Sign Language (ISL) gestures, enabling effective communication between the hearing-impaired community and the general public. Sign language recognition has been a challenging task due to the complexity of hand shapes, movements, and contextual variations in gestures. This project leverages convolutional neural network (CNN) networks, a type of recurrent neural network (RNN) known for its ability to capture long-term dependencies in sequential data, to improve the accuracy and robustness of gesture recognition. The proposed approach offers significant improvements in recognition accuracy over traditional methods by incorporating temporal dependencies and handling variations in user behaviour, background conditions, and real-time gesture sequences. The final system can be applied in various real-world scenarios, including sign language interpreters, assistive technologies, and communication platforms for the hearing-impaired. This work contributes to bridging the communication gap and fostering greater inclusivity for individuals with hearing disabilities in India.

1. INTRODUCTION

1.1 Problem Statement:

Indian Sign Language and Gesture Recognition for Deaf and Hard of Hearing (DHH) Community of India. The Indian Deaf and Hard of Hearing (DHH) community faces significant communication barriers due to limited ISL accessibility and understanding. This hinders social inclusion, educational and employment prospects, and overall quality of life. Developing advanced ISL recognition systems through computer vision and machine learning can bridge this communication gap and empower the DHH community.

1.2 Objective :

The primary objective of this project is to develop a comprehensive and robust Sign Language Recognition System using neural net models like Convolutional Neural Network (CNN) and advanced preprocessing techniques. This will leverage technology to improve communication methods for the DHH community of India.

1.3 Brief Discussion on Problem:

People who are deaf or hard of hearing have difficulties on a daily basis, particularly in communication. According to the World Health Organisation, by 2030, there will be 630 million people in the world, and by 2050, 900 million. India, the second-largest country on Earth, is home to some 63 million deaf or hearing-impaired individuals. Due to the high cost and limited availability of hearing aids and other assistive devices, only a small fraction of individuals with hearing impairments have access to hearing aids and other assistive devices. Deaf and hard-of-hearing persons in India may benefit greatly from technological developments like sign language recognition systems in terms of accessibility and communication. Deaf and hard-of-hearing persons frequently encounter communication barriers since sign language is often the primary means of communication for many members of the deaf community. The current state of the art in ISL recognition employs computer vision and machine learning techniques to recognize sign language gestures from video data automatically [1]. Using computer vision algorithms to monitor the movements of the hands and other body parts in the video and extract features that can be used to recognize specific signs is one of the most common approaches. Then, machine learning algorithms, such as neural networks, can be trained on these features to learn how to identify various signals in the video data. In some instances, these methods have yielded promising results, but they also have some limitations. One difficulty is the high variability and complexity of sign language gestures, which can make it difficult to recognise signs accurately in various contexts and by different signers. The absence of standardised datasets and evaluation metrics makes it difficult to compare and evaluate the performance of different ISL recognition methods. In addition, many existing methods require significant amounts of annotated data to train machine learning models, which can be costly and time-consuming to collect. Recent advances in computer vision and machine learning are advancing ISL recognition and paving the way for more accurate and efficient methods in the future, despite these obstacles.

Predominantly Sign Language has 3 types, mainly :

- Fingerspelling: used to spell words letter by letter
- word level vocabulary: These are particular gestures/actions for a word. Majority of the people use this mode of communication
- Non-manual features: Facial expression posture and etc

While gestures for words have various forms, Fingerspelling is broadly classified into 26 alphabets and 1-9 number signs (which are all static images). These images are shared below in Figure 1 [2].

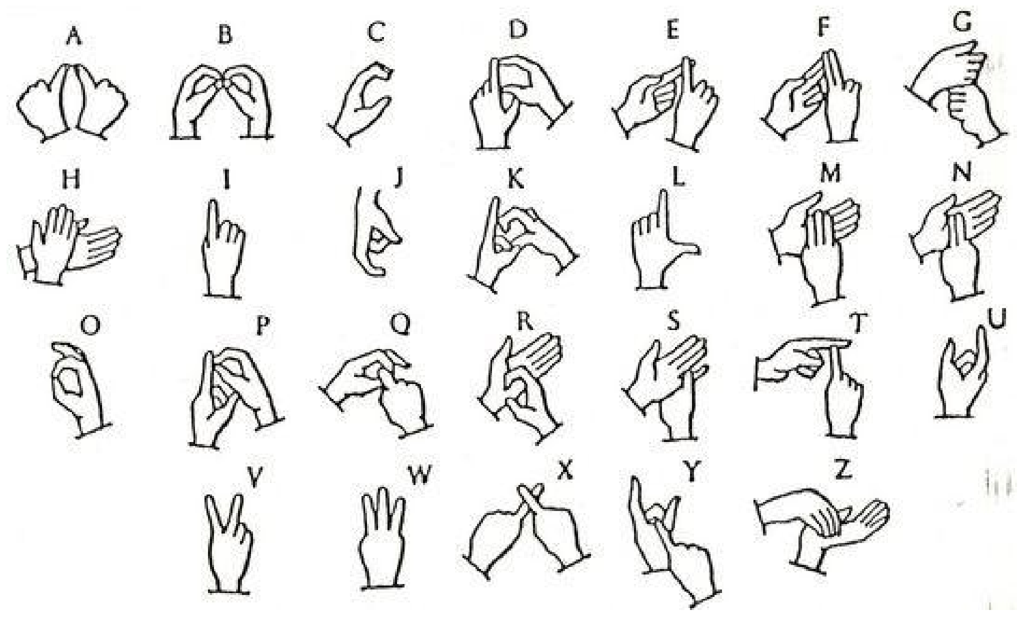


fig 1 : A-Z Indian Sign Gestures

1.4 Tools, Libraries and Frameworks used : We have used various libraries throughout the tenure of our project. They are listed below :

1.4.1. PIP (Package installer Python) : PIP is the official package installer for Python. It allows you to install, upgrade, and manage third-party libraries and packages from the Python Package Index (PyPI).

1.4.2. opencv : OpenCV (Open Source Computer Vision Library) is an open-source library designed for real-time computer vision and image processing tasks. It was originally developed by Intel and is now supported by the open-source community**.** We have used " OpenCV 4.11.0".

1.4.3. numpy : NumPy is a powerful open-source Python library used for numerical computing. It provides support for multidimensional arrays (ndarray) and a large collection of mathematical functions to operate on these arrays.

1.4.4. Tensorflow : TensorFlow is a comprehensive, open-source platform for machine learning developed by Google. It provides a wide range of tools and libraries for building and deploying machine learning models.

1.4.5. Matplotlib Matplotlib is a popular Python library for data visualization, widely used in scientific computing, machine learning, and data analysis. It enables the creation of static, animated, and interactive plots. In this project all the graphical representation and confusion matrix is done using matplotlib.

1.4.6. Keras : Keras is an open-source deep learning library written in Python. It provides a high-level API for building and training neural networks, making it user-friendly and easy to experiment with.

1.4.7. Pandas: Pandas is an open-source Python library widely used for data manipulation and analysis, particularly with tabular data (like spreadsheets or SQL tables). It provides powerful and flexible data structures. We used it to print the data frame of the metrics.

1.4.8. Streamlit : Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science projects. With just a few lines of code, you can turn your Python scripts into interactive web applications without needing any front-end experience. It allows users to visualize data, build interactive widgets like sliders and buttons, and display charts and images in real time. Streamlit is popular for rapid prototyping and sharing models, making it a great tool for data scientists and developers.

1.4.9. Seaborn: Seaborn is a powerful and popular data visualization library in Python, built on top of Matplotlib. It provides a high-level interface for creating informative and aesthetically pleasing statistical graphics, making it particularly well-suited for exploratory data analysis and understanding relationships within datasets.

1.5 Organization and Planning:

1.5.1 Organization :

We have organised the project report into various chapters which includes various aspects of our project. Chapter 1 includes the introduction of the problem and mainly understanding our objective and plans. While chapter 2 follows a literature survey of the research papers and articles that have been crucial in our project journey. chapter 3 in our project includes complex analysis of the problem along with brief emphasis on the keywords and concepts that are needed to tackle our problem statement in hand. Chapter 4 and 5 contains the brief solution and methodologies implemented in our project. Chapter 6 shows the test results and metrics that have been evaluated. Finally we have concluded the report by discussing the future work. Sample codes are also included in the report to provide detailed information on the codes that have been used. All the models are integrated in a single app with a user interface that will provide real time gesture classification.

1.5.2 Planning :

We have organised the project into multiple phases as listed below. Each phase contained detailed study of the key topics and libraries that have been used in the project. The Gantt chart serves as a cornerstone in project planning, offering a visual roadmap that delineates tasks, timelines, and dependencies. Its significance lies in providing a comprehensive and intuitive view of the project schedule, allowing mentors and team members to grasp the entire project lifecycle at a glance. By incorporating task dependencies, resource allocation, and progress tracking, Gantt charts enable effective project management by aiding in the identification of critical paths, resource bottlenecks, and potential risks. This visual tool fosters communication and collaboration among stakeholders, making it an essential component for effective project planning and execution. Its ability to adapt to changes in scope, track progress, and optimise timelines makes the Gantt chart an invaluable asset in ensuring projects are completed successfully and on schedule

The ISL Recognition System using the LSTM project was systematically divided into six phases, as detailed in the Gantt chart below, spanning from August to December:

- Research and Literature Review
- Dataset Preparation
- Model Development
- Testing and Evaluation

- API Integration
- Integration and Documentation

The Gantt chart provided in Figure 2 is a visual reference for the project timeline and task distribution.

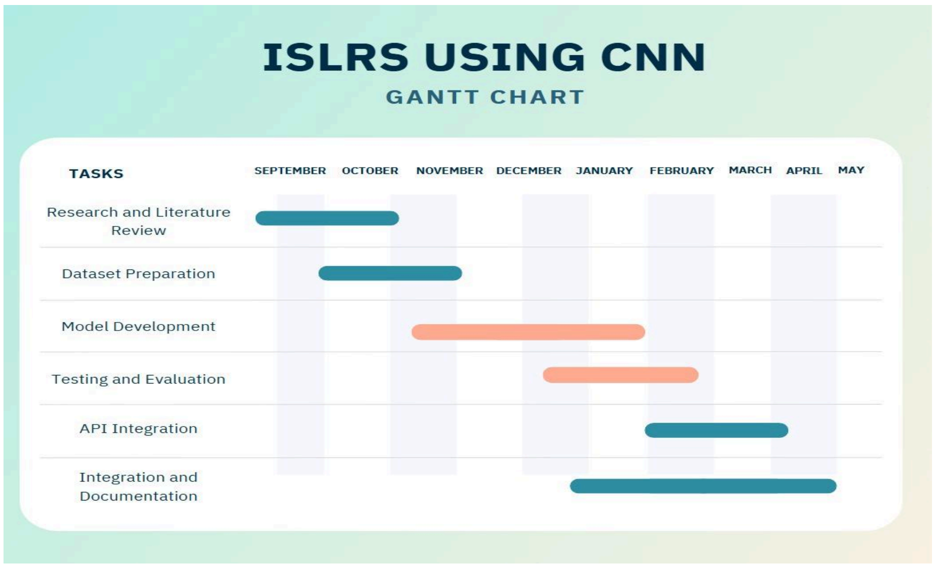


fig 2 : Gnatt chart of the project

Through this Gantt chart (shown in Fig 2) the planning of the Sign Language Recognition System project is shown as it visually represents tasks, dependencies, and timelines. It provides a clear overview of the project's schedule, helps allocate resources efficiently, identifies critical paths, and ensures timely completion of key milestones such as data collection, refinement, model training, and UI design. This visual roadmap enhances project management, facilitates coordination among team members, and enables effective tracking of progress throughout the project lifecycle.

2. Literature Survey

In recent years, significant advancements have been witnessed in sign language recognition (SLR) systems, particularly in Indian sign language (ISL). Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN), have emerged as a powerful tool for modelling the temporal dependencies inherent in sign language gestures. Various research studies have explored the application of LSTM networks for ISL recognition, demonstrating promising results. One notable approach in [2] involves utilizing MediaPipe, a versatile framework developed by Google, to extract relevant features from video frames. These features, which capture hand and body movements, are then fed into an LSTM network for classification. By leveraging the temporal processing capabilities of LSTM, this method effectively recognizes dynamic gestures, leading to improved accuracy. Another promising direction in [3] involves combining Convolutional Neural Networks (CNNs) with LSTM networks. CNNs excel at extracting spatial features from images, while LSTMs are adept at capturing temporal dependencies. By integrating these two architectures, researchers have achieved state-of-the-art results in ISL recognition. The CNN extracts features from each frame, and the LSTM processes the sequence of features to classify the sign. While significant progress has been made, several challenges remain in the field of ISL recognition. One key challenge is the limited availability of large-scale, high-quality datasets. To address this issue, researchers have explored techniques like data augmentation and transfer learning to improve model performance. Additionally, real-time sign language recognition remains a challenging task, requiring efficient algorithms and hardware acceleration. Future research directions in this field include exploring more advanced deep learning architectures, such as Transformer networks, to capture complex dependencies within sign language gestures. Additionally, incorporating attention mechanisms can help focus on the most relevant parts of the input sequence, leading to improved accuracy and robustness. As technology continues to advance, we can expect further breakthroughs in ISL recognition, ultimately empowering the deaf and hard-of-hearing community.

3. Concepts and Problem Analysis

3.1 Concept of Hand Sign Recognition

Sign language detection apps are innovative software applications designed to bridge communication gaps between deaf and hearing individuals. Leveraging cutting-edge technology such as computer vision and machine learning, these apps have the remarkable ability to recognize and interpret sign language gestures in real-time.

In everyday scenarios, these apps prove invaluable. Imagine a smartphone screen displaying a person using a sign language detection app to effortlessly order food at a restaurant, breaking barriers and making interactions more accessible. Through the seamless translation of sign language into text or spoken language, these apps empower deaf individuals to engage with the broader community. The picture below shows an example of hand gesture detection model.

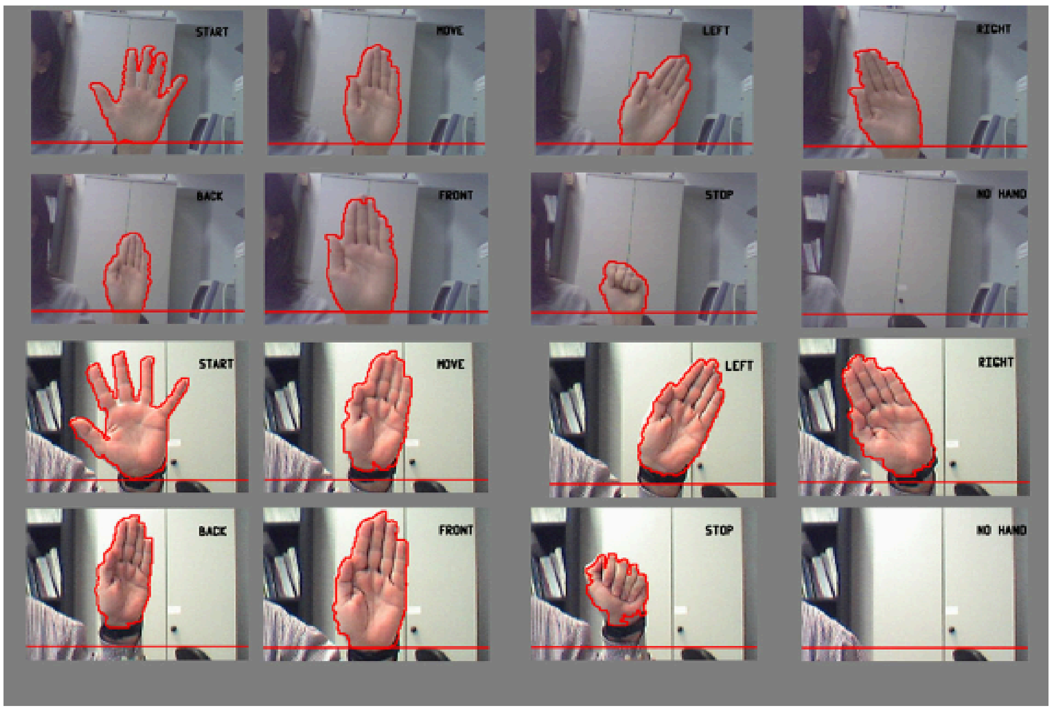


fig 3 : Detection of Hand gestures

Furthermore, a graphic representation of the sign language alphabet showcases the app's prowess in recognizing and translating gestures into meaningful communication. This technology not only facilitates communication but also fosters understanding and acceptance, ultimately creating a more inclusive society. As these apps continue to evolve and improve, their impact on the lives of deaf and hard-of-hearing individuals is bound to expand. With their potential to break down barriers and create connections, sign language detection apps represent a significant step towards a more accessible and inclusive world.

3.2 Convolution Neural Network :

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

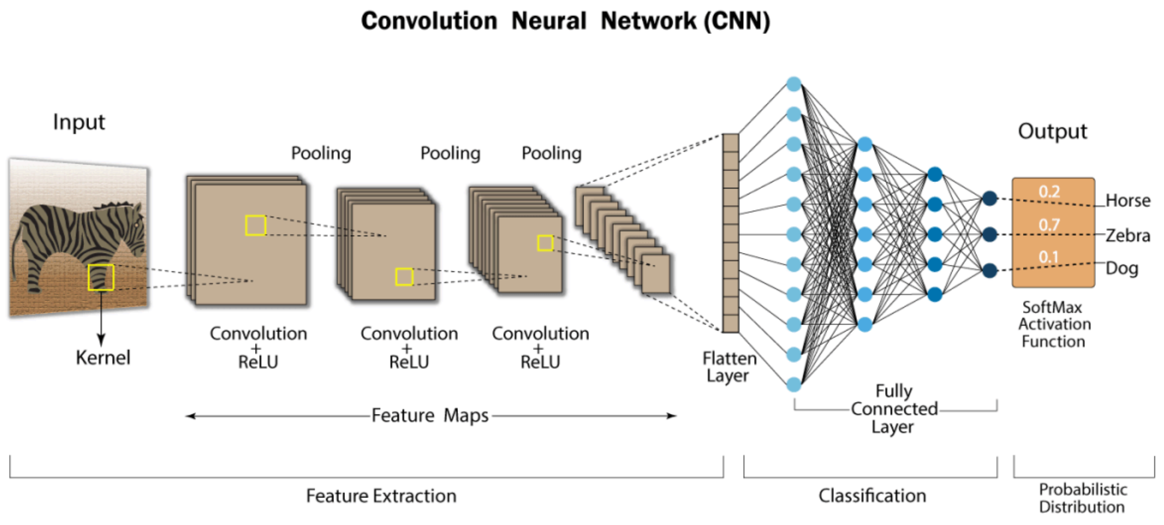


fig 4 : Convolutional Neural Network

1. Convolution Layer : In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color.

2. Pooling Layer : We use a pooling layer to decrease the size of the activation matrix and ultimately reduce the learnable parameters. There are two type of pooling :

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

b) Average Pooling : In average pooling we take the average of all values in a window. The picture given below shows a basic idea of the working of max pooling and average pooling.

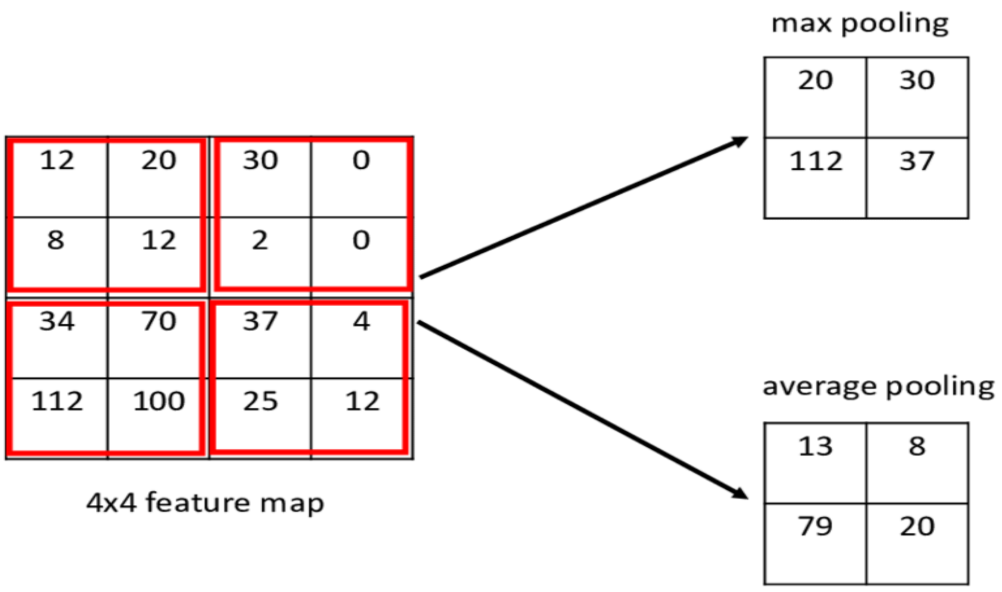


fig 5 : Max pooling and average pooling

3. Fully Connected Layer : In convolution layer neurons are connected only to a local region, while in a fully connected region, we'll connect all the inputs to neurons. The picture below shows a fully connected layer of a Convolution Neural Network.

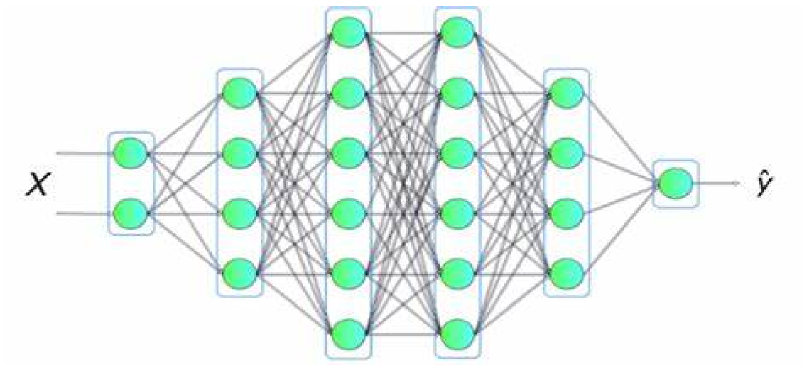


fig 6 : Fully connected layer

3.3 Architecture of the CNN

1. Convolutional Layer 1: The first convolutional layer consists of 32 filters with a 3×3 kernel size, applied to the 128×128 input image. This layer is responsible for extracting fundamental features such as edges, corners, and simple textures. The ReLU activation function is used to introduce non-linearity, ensuring the model can capture complex patterns beyond a simple linear transformation. The output feature maps are 126×126, slightly reduced due to the convolution operation while retaining spatial information.

2. Max Pooling Layer 1: Max pooling is a downsampling technique that helps reduce computational complexity while preserving essential features. In this layer, a 2×2 pooling filter with a stride of 2 is applied, reducing the spatial dimensions from 126×126 to 63×63. This process selects the most dominant features from local regions, making subsequent layers more efficient at recognizing meaningful patterns.

3. Convolutional Layer 2: A second convolutional layer follows, applying 32 filters with a 3×3 kernel size to the 63×63 input from the previous pooling layer. By building on the foundational features detected earlier, this layer focuses on extracting higher-level textures and shapes. The output feature map is 60×60, reflecting a slight reduction in dimensions due to convolution operations. Again, ReLU activation is used to introduce non-linearity, ensuring the network can distinguish complex image features.

4. Max Pooling Layer 2: The second pooling layer further down samples the feature map using a 2×2 pooling filter with a stride of 2, reducing the feature map size from 60×60 to 30×30. This step ensures that only the most significant patterns are retained, preventing unnecessary computational overhead while enhancing feature selection. Pooling plays a crucial role in making the network more robust to variations in input images.

5. Fully Connected (Dense) Layers: Once the convolutional and pooling layers have transformed the raw image into structured feature maps, the data is flattened into a 1D vector of 28,800 values (30×30×32). The first dense layer consists of 128 neurons, processing the extracted features for classification purposes. A Dropout rate of 0.5 is applied, randomly deactivating neurons to prevent overfitting. The second dense layer has 96 neurons, continuing feature refinement for optimal classification before the final output layer.

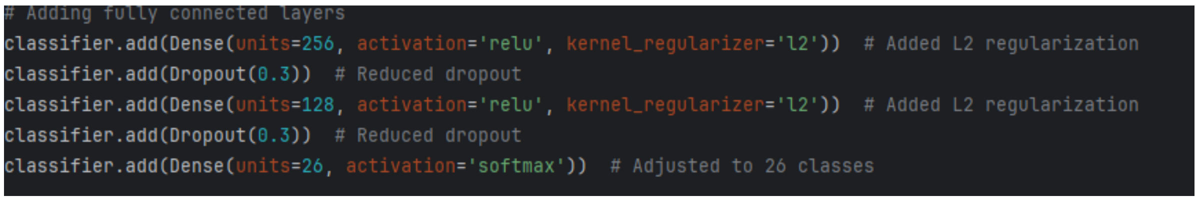
6. Output Layer: The final dense layer consists of 29 neurons, each corresponding to a unique class in the dataset. A softmax activation function is applied to convert raw output values into probability distributions, ensuring the model selects the class with the highest probability. The softmax function is essential for multi-class classification, allowing the CNN to predict the most probable category for a given input image.

7. Regularization Techniques: To improve generalization and prevent overfitting, several regularization techniques are employed. Batch normalization stabilizes training by normalizing activations, improving convergence rates. Dropout (used in dense layers) helps prevent reliance on specific neurons by randomly deactivating connections. L2 regularization penalizes large weight values, ensuring the network learns efficiently without unnecessary complexity. Together, these techniques enhance model robustness and performance across diverse datasets.

3.4 Problem Analysis

3.2.1 Mismatch in Number of Classes

Problem: The number of classes in the dataset does not match the output layer of your model.



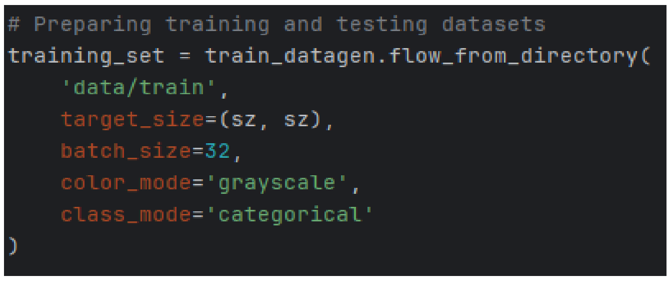
```
# Adding fully connected layers
classifier.add(Dense(units=256, activation='relu', kernel_regularizer='l2'))  # Added L2 regularization
classifier.add(Dropout(0.3))  # Reduced dropout
classifier.add(Dense(units=128, activation='relu', kernel_regularizer='l2'))  # Added L2 regularization
classifier.add(Dropout(0.3))  # Reduced dropout
classifier.add(Dense(units=26, activation='softmax'))  # Adjusted to 26 classes
```

fig 7 : Adding fully connected layers

Solution: Ensure that the number of classes in the dataset matches the output layer. If the dataset has a different number of classes, adjust the output layer accordingly.We check the number of classes detected by printing the class indices.

3.2.2 Data Directory Structure

Problem: The directory structure for the training and testing datasets may not be set up correctly.



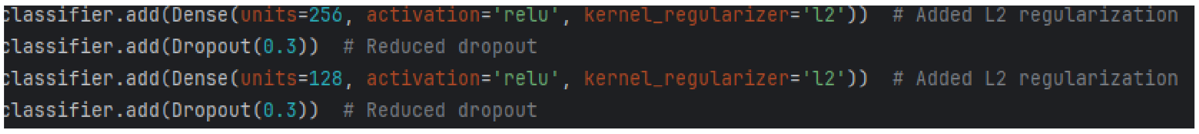
```
# Preparing training and testing datasets
training_set = train_datagen.flow_from_directory(
    'data/train',
    target_size=(sz, sz),
    batch_size=32,
    color_mode='grayscale',
    class_mode='categorical'
)
```

fig 8 : Preparing training and testing datasets.

Solution: We ensure that the data/train and data/test directories contain subdirectories for each class.

3.2.3. Model Overfitting

Problem: The model may overfit if the training accuracy is significantly higher than the validation accuracy.



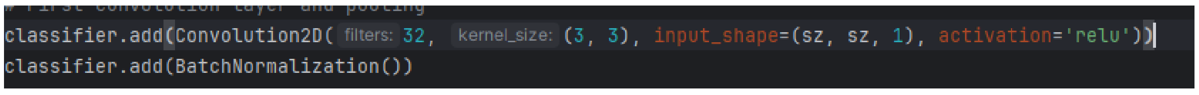
```
classifier.add(Dense(units=256, activation='relu', kernel_regularizer='l2'))  # Added L2 regularization
classifier.add(Dropout(0.3))  # Reduced dropout
classifier.add(Dense(units=128, activation='relu', kernel_regularizer='l2'))  # Added L2 regularization
classifier.add(Dropout(0.3))  # Reduced dropout
```

fig 9 : Adding dropout layers

Solution: We increased the dropout rate and added more regularization. We also considered using early stopping to prevent overfitting.

3.2.4. Incompatible Input Shape

Problem: The input shape of the images did not match the expected shape in the model.



```
# First convolution layer and pooling
classifier.add(Convolution2D( filters: 32,  kernel_size: (3, 3), input_shape=(sz, sz, 1), activation='relu'))
classifier.add(BatchNormalization())
```

fig 10 : Feeding images into the model.

Solution: We ensured that the images being fed into the model are of the correct shape. When we are using grayscale images, the shape should be (128, 128, 1). When we are using RGB images, change the input shape to (128, 128, 3).

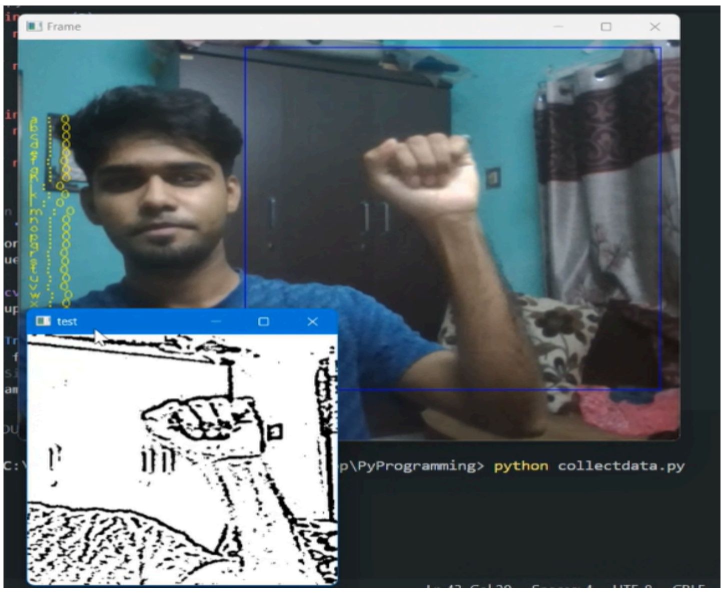
4. Design and Methodology

The system is a vision based approach. All the signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

4.1 Dataset Generation

For the project we tried to find already made datasets but we couldn't find datasets in the form of raw images that matched our requirements. All we could find were the datasets in the form of RGB values. Hence we decided to create our own data set. Steps we followed to create our data set are as follows.

We used the Open computer vision(OpenCV) library in order to produce our dataset.Firstly we captured around 800 images of each of the symbols in ASL for training purposes and around 200 images per symbol for testing purposes. First we capture each frame shown by the webcam of our machine. In each frame we define a region of interest (ROI) which is denoted by a blue bounded square as shown in the image below.



```
p\PyProgramming> python collectdata.py
```

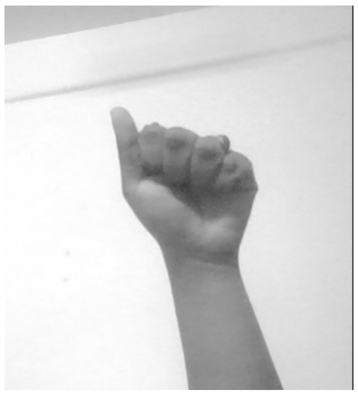


fig 11 : Data collection and conversion of RGB image into grayscale and gaussian blur filter

From this whole image we extract our ROI which is RGB and convert it into gray scale Image as shown on the right picture.

After converting the image to grayscale, the function applies a Gaussian blur filter using ‘GaussianBlur’, which helps to reduce noise and smooth the image, which is particularly important for improving the results of the thresholding operations that follow. The kernel size for the blur is set to (5, 5), and the standard deviation is set to 2, which controls the extent of the blurring effect. We apply our gaussian blur filter to our image as it helps us extract various features of our image.

4.2 Classification of Gestures

The approach which we used for this project is : Our approach uses two layers of algorithm to predict the final symbol of the user.

Algorithm Layer :

1. Apply gaussian blur filter and threshold to the frame taken with opencv to get the processed image after feature extraction.

2. This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word.

3. Space between the words are considered using the blank symbol.

4.3 CNN Model :

We construct the CNN architecture, where a sequential model is initialized, and several convolution layers are added. The first layer uses 32 filters with a kernel size of 3x3 and applies the ReLU activation function. Each convolution layer is followed by batch normalization to stabilize and accelerate the training, and max pooling layers are added to reduce the spatial dimensions of the feature maps. The architecture includes 3 convolutional layers with increasing filter size (32, 64 and 128), which allows the model to learn increasingly complex features from the input images. After flattening the output from the convolutional layers, fully connected i.e. the dense layers are added with L2 regularization applied to help the model to prevent overfitting. The final layer uses the softmax function to output probabilities for 29 classes, indicating that the model is designed for a multi-class classification task.

1. **1st Convolution Layer :** The input picture has a resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.

2. 1st Pooling Layer : The pictures are downsampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is downsampled to 63x63 pixels.

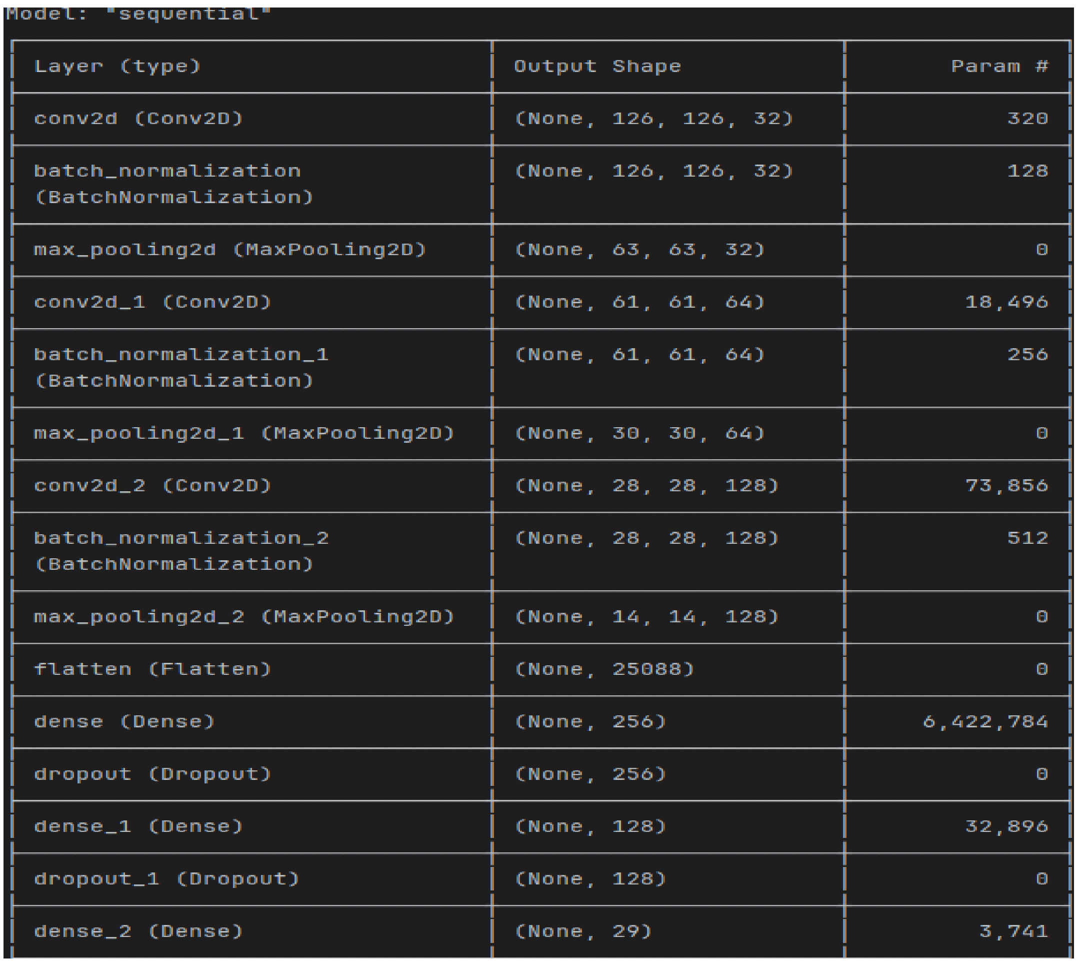
3. 2nd Convolution Layer : Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer.It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each).This will result in a 60 x 60 pixel image.

4. 2nd Pooling Layer : The resulting images are downsampled again using max pool of 2x2 and is reduced to 30 x 30 resolution of images.

5. 1st Densely Connected Layer : Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an array of 30x30x32 =28800 values. The input to this layer is an array of 28800 values. The output of this layer is fed to the 2nd Densely Connected Layer.We are using a dropout layer of value 0.5 to avoid overfitting.

6. 2nd Densely Connected Layer : Now the output from the 1st Densely Connected Layer are used as an input to a fully connected layer with 96 neurons.

7. Final layer: The output of the 2nd Densely Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying.



```
Model: "sequential"
┌──────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                     │ Output Shape           │       Param # │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d (Conv2D)                  │ (None, 126, 126, 32)   │           320 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization              │ (None, 126, 126, 32)   │           128 │
│ (BatchNormalization)             │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 63, 63, 32)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)                │ (None, 61, 61, 64)     │        18,496 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1            │ (None, 61, 61, 64)     │           256 │
│ (BatchNormalization)             │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 30, 30, 64)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                │ (None, 28, 28, 128)    │        73,856 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2            │ (None, 28, 28, 128)    │           512 │
│ (BatchNormalization)             │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 14, 14, 128)    │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)                │ (None, 25088)          │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 256)            │     6,422,784 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)                │ (None, 256)            │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 128)            │        32,896 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)              │ (None, 128)            │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                  │ (None, 29)             │         3,741 │
```

fig 13 : Table Summary for the CNN architecture

4.4 Activation Function :

We have used ReLu (Rectified Linear Unit) in each of the layers(convolutional as well as fully connected neurons). ReLu calculates max(x,0) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features.It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time.

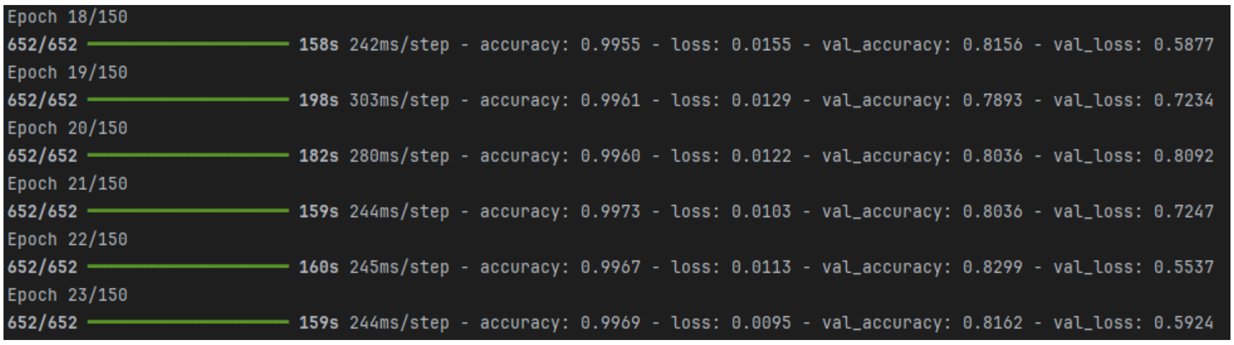
Pooling Layer : We apply Max pooling to the input image with a pool size of (2, 2) with ReLU activation function.This reduces the amount of parameters thus lessening the computation cost and reduces overfitting.

Dropout Layers: The problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples.This layer "drops out" a random set of activations in that layer by setting them to zero.The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out in [4].

Optimizer : We have used Adam optimizer for updating the model in response to the output of the loss function. Adam combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm(ADA GRAD) and root mean square propagation(RMSProp)

4.5 Training and Testing :

We convert our input images(RGB) into grayscale and apply gaussian blur to remove unnecessary noise.We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128. We feed the input images after preprocessing to our model for training and testing after applying all the operations mentioned above. The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using the softmax function. At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which are not the same as labeled value and is zero exactly when it is equal to the labeled value. Therefore we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy. As we have found out the cross entropy function, we have optimized it using Gradient Descent. In fact, the best gradient descent optimizer is called Adam Optimizer.



```
Epoch 18/150
652/652 ━━━━━━━━━━━━━━━━━━━━ 158s 242ms/step - accuracy: 0.9955 - loss: 0.0155 - val_accuracy: 0.8156 - val_loss: 0.5877
Epoch 19/150
652/652 ━━━━━━━━━━━━━━━━━━━━ 198s 303ms/step - accuracy: 0.9961 - loss: 0.0129 - val_accuracy: 0.7893 - val_loss: 0.7234
Epoch 20/150
652/652 ━━━━━━━━━━━━━━━━━━━━ 182s 280ms/step - accuracy: 0.9960 - loss: 0.0122 - val_accuracy: 0.8036 - val_loss: 0.8092
Epoch 21/150
652/652 ━━━━━━━━━━━━━━━━━━━━ 159s 244ms/step - accuracy: 0.9973 - loss: 0.0103 - val_accuracy: 0.8036 - val_loss: 0.7247
Epoch 22/150
652/652 ━━━━━━━━━━━━━━━━━━━━ 160s 245ms/step - accuracy: 0.9967 - loss: 0.0113 - val_accuracy: 0.8299 - val_loss: 0.5537
Epoch 23/150
652/652 ━━━━━━━━━━━━━━━━━━━━ 159s 244ms/step - accuracy: 0.9969 - loss: 0.0095 - val_accuracy: 0.8162 - val_loss: 0.5924
```

fig 14: Training data with Epochs.

The model's accuracy increases and loss value decreases with every epoch as shown in the figure above.

The model is then trained, which takes the training and validation datasets, specifies the number of epochs, and includes the early stopping callback. After training, the model architecture and the model weights are saved in a specified format. Finally, we include a function to plot the training and validation accuracy and loss over the epochs, providing visual feedback on the model's performance during training.

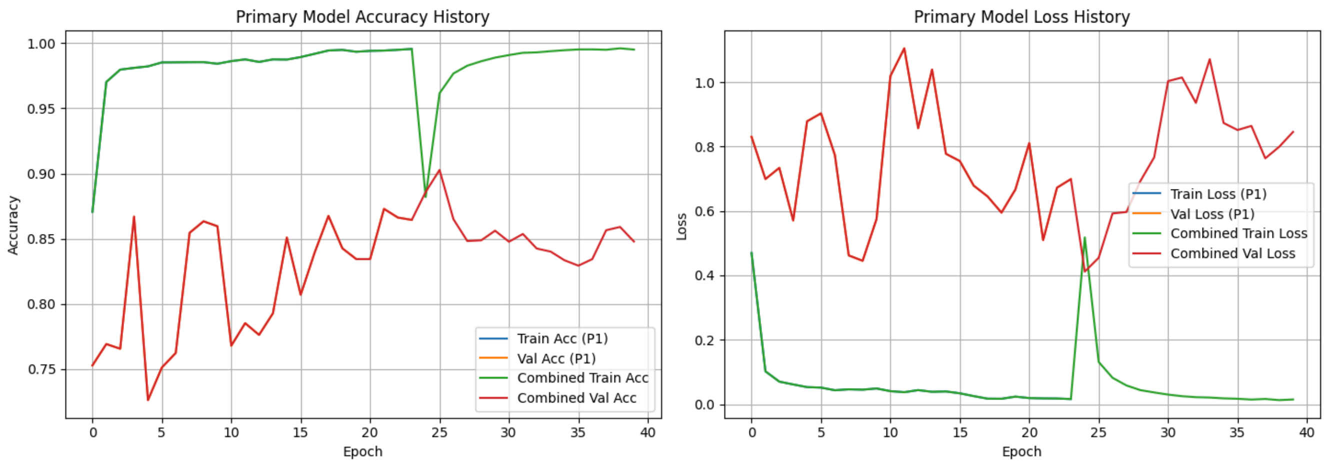


fig 15: Graphs representing Model Accuracy and Model Loss

5. Result Analysis :

We have achieved a training accuracy of 95.8% in our model using only layer 1 of our algorithm , and using the combination of layer 1 and layer 2 we achieve an accuracy of 99.61%, which is a better accuracy then most of the current research papers on american sign language. Here are some comparative analysis of similar kinds on works.

1. CNN-Based Recognition for Flemish Sign Language

Method: Utilized Convolutional Neural Networks (CNNs) with Kinect sensor for hand detection [5].

Error Rate: 2.5%, indicating a high level of accuracy.

Limitation: Dependency on Kinect limits accessibility for users without specialized hardware.

2. Hidden Markov Model-Based Recognition

Method: Built a recognition model using Hidden Markov Model (HMM) classifier with a vocabulary of 30 words [6].

Error Rate: 10.90%, slightly higher than CNN-based approaches.

Limitation: While HMM models perform well for sequential gesture recognition, accuracy is lower compared to CNN-based systems.

3. Static Gesture Recognition for Japanese Sign Language

Method: Recognized 41 static gestures in Japanese Sign Language using depth sensors and CNNs [7]

Accuracy: 86%, demonstrating robust recognition.

Improved Approach: By incorporating depth sensor maps, recognition accuracy for observed signers reached 99.99%, while accuracy for new signers was 83.58% and 85.49% [8].

Limitation: Depth sensor technology improves accuracy but may require specialized equipment.

One thing should be noted that our model doesn't use any background subtraction algorithm while some of the models present above do that. So once we try to implement background subtraction in our project the accuracies may vary. On the other hand most of the above projects use kinect devices but our main aim was to create a project which can be used with readily available resources. A sensor like kinect not only isn't readily available but also is expensive for most of the audience to buy and our model uses a normal webcam of the laptop hence it is a great plus point.

5.1 Evaluation Metrics :

The evaluation of the sign language recognition system incorporates a multifaceted approach, leveraging several key metrics to gauge its performance comprehensively. These metrics include accuracy, precision, recall, F1-score, and a detailed examination via the confusion matrix. Each metric plays a distinctive role in assessing the system's efficacy, offering insights into different aspects of its performance.

Accuracy serves as a foundational metric, measuring the overall correctness of the system's predictions against the ground truth. It provides a simple yet effective indicator of how often the system correctly identifies the intended sign or letter.

Precision, on the other hand, delves deeper into the nature of the system's errors, focusing on the proportion of positive identifications that were indeed correct. This metric is particularly relevant in scenarios where false positives carry significant implications, necessitating a high degree of certainty in the system's predictions.

Recall complements precision by examining the system's ability to detect all relevant instances. It quantifies the proportion of actual positive instances that were correctly identified by the system, emphasising the system's sensitivity to the presence of the target signs or letters.

The F1-score emerges as a harmonious blend of precision and recall, offering a balanced perspective on the system's performance. It is calculated as the harmonic mean of precision and recall, providing a unified measure that accounts for both types of errors—false positives and false negatives. This metric is especially useful in situations where both types of errors are equally undesirable, or when the dataset exhibits an uneven class distribution.

Lastly, the confusion matrix offers a granular breakdown of the system's performance, detailing the counts of true positives, false positives, true negatives, and false negatives. This matrix provides a visual representation of the system's error distribution, allowing for a more nuanced understanding of its strengths and weaknesses

	Metric	**Score**
0	Precision	0. 890925
1	F1	0.872272
2	Recall	0.880086
3	Accuracy	0.890925

fig 16 : Evaluation Metrics of our Model on 75 epochs

5.2. Confusion Matrix :

A confusion matrix is a performance measurement tool for classification models like Convolutional Neural Networks (CNNs). It displays a summary of prediction results on a classification problem by showing the number of correct and incorrect predictions made by the model, compared to the actual outcomes.

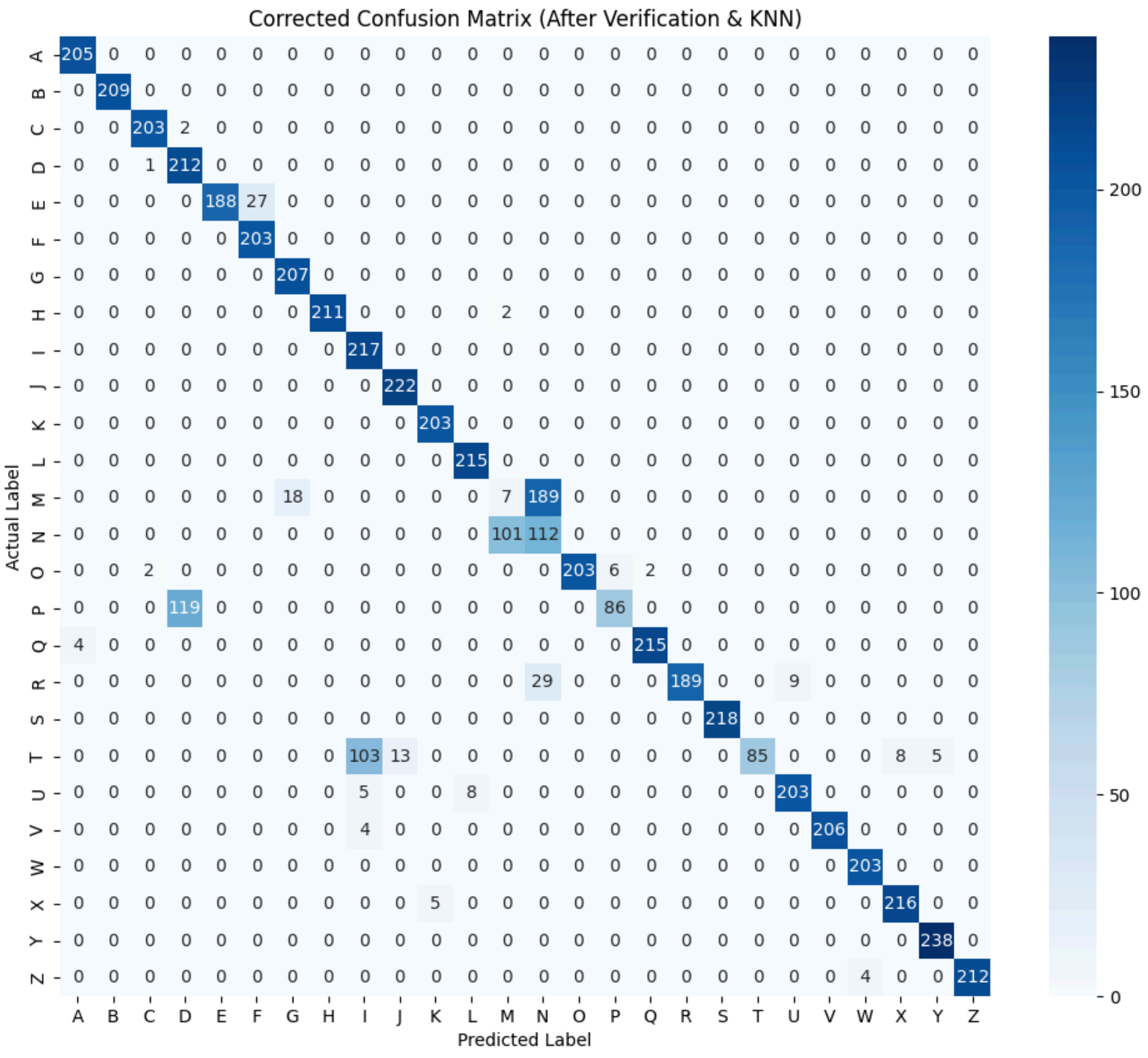
In the context of an Indian Sign Language Recognition System, the confusion matrix helps evaluate how well the CNN model identifies each sign (e.g., alphabets or dynamic gestures). The matrix includes:

- True Positives (TP) – correct predictions for a sign.
- False Positives (FP) – incorrect predictions where another sign is misclassified as the target sign.
- False Negatives (FN) – actual signs that were misclassified as something else.
- True Negatives (TN) – correctly predicted rejections of non-target signs.

By analyzing the confusion matrix, we can compute metrics like **accuracy, precision, recall, and F1-score**, which give deeper insights into the model's strengths and weaknesses—such as which signs are often confused due to visual similarity.

Neighboring confusions such as 'N' ↔ 'M' and 'Q' ↔ 'R' likely arise from similar hand shapes or overlapping features in data representations; applying contrast boosting and edge detection sharpens the class distinctions. Frequent 'E' vs. 'F' confusions might stem from similar stroke patterns or low feature variability in training data, which is addressed by increasing data augmentation with angle shifts and deformations. Lastly, a high 'U' ↔ 'V' misclassification rate may indicate shared structural similarities in training images, suggesting that adaptive thresholding or finer-scale feature extraction should improve separation.

To enhance prediction accuracy, we've implemented several key strategies: a Lightweight CNN Secondary Verification Model now refines ambiguous predictions, and Confidence Thresholding re-checks any predictions made with low confidence to prevent errors. We also introduced K-Nearest Neighbor (KNN) Post-Prediction Filtering to reduce confusion among similar classes, and Optimized Confusion Matrix Correction to replace uncertain predictions based on observed statistical misclassification trends.



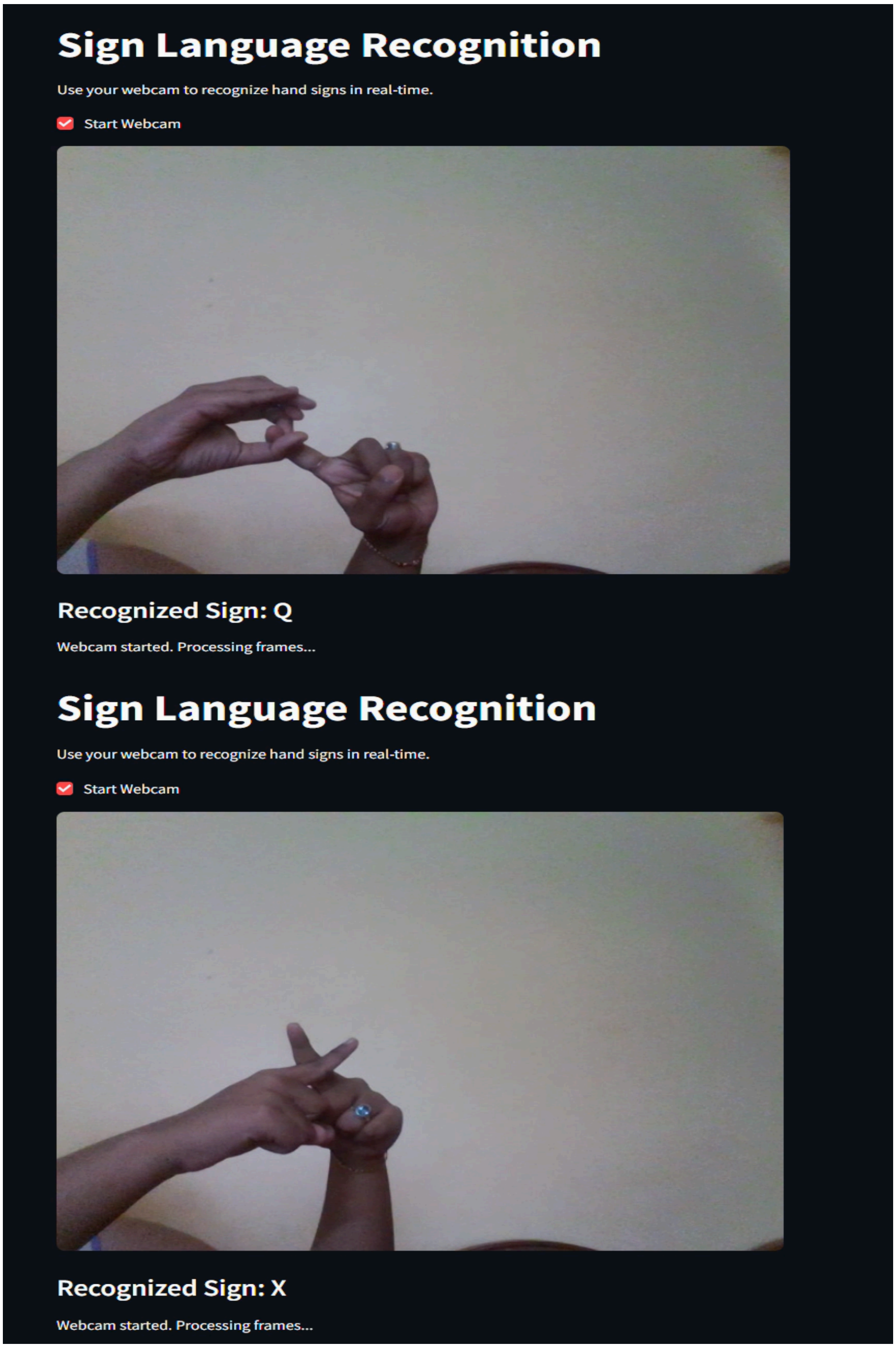
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	205	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	203	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	1	212	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	188	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	203	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	211	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	217	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	222	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	203	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	215	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	18	0	0	0	0	0	7	189	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	101	112	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	2	0	0	0	0	0	0	0	0	0	0	0	203	6	2	0	0	0	0	0	0	0	0	0
P	0	0	0	119	0	0	0	0	0	0	0	0	0	0	0	86	0	0	0	0	0	0	0	0	0	0
Q	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	215	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	189	0	0	9	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	218	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	103	13	0	0	0	0	0	0	0	0	0	85	0	0	0	8	5	0
U	0	0	0	0	0	0	0	0	5	0	0	8	0	0	0	0	0	0	0	0	203	0	0	0	0	0
V	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	206	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	203	0	0	0
X	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	216	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	238	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	212

fig 17 : Confusion Matrix

5.3 GUI Based Output

We built an application using Streamlit to create an interactive web interface for real-time sign language recognition from a webcam feed. The core of the application is encapsulated, which loads a deep learning model from JSON architecture and corresponding weights. The main functionality includes capturing frames from the webcam when the user activates the webcam via a checkbox. Each video frame is flipped to create a mirror effect, and a rectangular region of interest (ROI) is drawn to guide the user where to place their hand. The ROI is cropped, preprocessed using grayscale conversion, Gaussian blurring, and adaptive thresholding to enhance the features relevant for sign prediction. The preprocessed ROI is then fed into the model to obtain a prediction for the hand sign. The live video stream and recognized sign label are displayed continuously in the Streamlit application, updating in near real-time as the webcam captures frames. Users can stop the feed by unchecking the same checkbox.

This immediate visual update helps users understand how well the model interprets their signs, making it suitable for demonstration or learning purposes. The accuracy of the predictions heavily relies on the quality and training of the underlying neural network model and its ability to generalize well from the preprocessed input frames. The preprocessing steps contribute significantly to the model's performance by reducing noise and isolating the hand's shape. However, factors like lighting conditions, background complexity, and hand orientations may affect recognition accuracy. Despite these limitations, the app provides a user-friendly, web-based platform that requires no complex setup beyond a webcam and a browser. With improvements in model robustness and interface usability, this framework could form the foundation for accessible and practical sign language recognition tools.



Sign Language Recognition

Use your webcam to recognize hand signs in real-time.

☑ Start Webcam

Recognized Sign: Q

Webcam started. Processing frames...

Sign Language Recognition

Use your webcam to recognize hand signs in real-time.

☑ Start Webcam

Recognized Sign: X

Webcam started. Processing frames...

fig 18 : Gui based output (Gesture Classification)

6. CONCLUSION AND FUTURE WORK:

The implementation of an Indian Sign Language (ISL) recognition system using Convolutional Neural Networks (CNN) presents a transformative step toward improving accessibility and inclusivity for the deaf and hard-of-hearing communities. CNNs, with their ability to automatically learn spatial hierarchies from visual data, provide a robust framework for accurately classifying hand gestures in ISL. By leveraging deep learning techniques, the system can extract intricate features such as finger positioning, movement dynamics, and hand shapes, ensuring high recognition accuracy across diverse lighting conditions and backgrounds. However, challenges such as variability in hand orientations, dataset limitations, and real-time processing constraints remain key areas for improvement. Future advancements can integrate multimodal approaches, including temporal gesture tracking, 3D hand modeling, and reinforcement learning, to enhance system efficiency and scalability. The ongoing development of ISL recognition systems contributes to fostering social inclusion, enabling smoother interactions between sign language users and non-signers, and reinforcing the importance of technology-driven assistive solutions in bridging communication gaps. As research progresses, refining datasets, optimizing algorithms, and expanding deployment in real-world applications will drive meaningful advancements, ultimately empowering individuals relying on sign language as their primary mode of expression.

Future work on Indian Sign Language recognition using CNN architecture can focus on several key improvements and expansions. Enhancing dataset diversity by incorporating various hand shapes, orientations, and lighting conditions can improve the robustness of the model. Further optimization of CNN architectures, such as using deeper networks or hybrid models with attention mechanisms, could enhance recognition accuracy. Additionally, integrating temporal models like Long Short-Term Memory (LSTM) or Transformer-based architectures can facilitate the recognition of continuous sign language sequences rather than static gestures. Real-time implementation with edge computing or mobile applications would enable accessibility and practicality for users. Collaborations with linguists and sign language experts can help refine the model for cultural and regional variations. Lastly, expanding the system to support multilingual sign recognition and interactive learning platforms could further contribute to inclusivity and accessibility in communication.

7. REFERENCES

1. Velmula, K. R., Linginani, I., Reddy, K. B., Meghana, P., & Aruna, A. (2021). Indian sign language recognition using convolutional neural networks. In C. Kiran Mai, B. V. Kiranmayee, M. N. Favorskaya, S. Chandra Satapathy, & K. S. Raju (Eds.), Proceedings of International Conference on Advances in Computer Engineering and Communication Systems (Vol. 20, pp. 387–395). Springer, doi: 10.1007/978-981-15-9293-5_35
2. Shamitha S, Badrinath K, "Sign Language Recognition Utilizing LSTM And Mediapipe For Dynamic Gestures Of ISL", IJFMR comm. Magazine, vol-5, September-October 2023.
3. Goyal K, "Indian sign language recognition using media pipe holistic", arXiv preprint arXiv:2304.10256. Apr 20, 2023.
4. F. Obaid, A. Babadi, and A. Yoosofan, "Hand Gesture Recognition in Video Sequences Using Deep Convolutional and Recurrent Neural Networks," Appl. Computer System, vol. 25, no. 1, pp. 57–61, May 2020, doi: 10.2478
5. N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," 2017 Nicograph International (NicoInt) , Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9
6. Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen "Real-time sign language fingerspelling recognition using convolutional neural networks from depth map" 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)
7. Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4), 572–577 (2011)
8. Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925.Springer, Cham

ANNEXURE I

Data Collection:

The code for data collection includes the extensive use of the CV2 library which is responsible for taking inputs of the sign through a webcam interface. The following code explains 3 main parts :

- Directory setup for training and testing data
- Setting up webcam using CV2 library and collecting data real time
- GRAYSCALE conversion of an RGB image
- Binary mask has been put up on the images
- Finally, saving the processed image to the new directory

Sample code is given below:

```
import cv2

import numpy as np

import os

import string


# Create the directory structure

if not os.path.exists("data"):

    os.makedirs("data")

if not os.path.exists("data/train"):

    os.makedirs("data/train")

if not os.path.exists("data/test"):

    os.makedirs("data/test")

for i in range(3):

    if not os.path.exists("data2/train/" + str(i)):

        os.makedirs("data2/train/" + str(i))

    if not os.path.exists("data2/test/" + str(i)):

        os.makedirs("data2/test/" + str(i))


for i in string.ascii_uppercase:

    if not os.path.exists("data/train/" + i):
        os.makedirs("data/train/" + i)
    if not os.path.exists("data/test/" + i):
        os.makedirs("data/test/" + i)

# Train or test
mode = 'train'
directory = 'data/' + mode + '/'
minValue = 70

cap = cv2.VideoCapture(0)
interrupt = -1

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {
        'zero': len(os.listdir(directory + "/0")),
        'one': len(os.listdir(directory + "/1")),
        'two': len(os.listdir(directory + "/2")),....
………..
cap.release()
cv2.destroyAllWindows()
"""
d = "old-data/test/0"
newd = "data/test/0"
for walk in os.walk(d):
    for file in walk[2]:
        roi = cv2.imread(d+"/"+file)
        roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        _, mask = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
        cv2.imwrite(newd+"/"+file, mask)
```

Pre-processing:

The purpose of this code snippet is to copy grayscale images from data/train/<label> into a new directory structure data2/train/<label> (and optionally data2/test/<label>) while optionally applying a custom image processing function. Finally we are parsing through the whole directory and reading the grayscale images.

Sample code is given below:

```
import numpy as np
import cv2
import os
from image_processing import func

# Create output directories
if not os.path.exists("data2"):
  os.makedirs("data2")
if not os.path.exists("data2/train"):
  os.makedirs("data2/train")
if not os.path.exists("data2/test"):
  os.makedirs("data2/test")
path = "data/train"
path1 = "data2"
label = 0
var = 0
c1 = 0
c2 = 0
# Check if the train directory exists
if not os.path.exists(path):
  print(f"Directory '{path}' does not exist.")
else:
  print(f"Contents of '{path}': {os.listdir(path)}")
```

```
# Walk through the train directory
for (dirpath, dirnames, filenames) in os.walk(path):
    print(f"Current directory: {dirpath}")
    print(f"Subdirectories: {dirnames}")
    print(f"Files: {filenames}")

    for dirname in dirnames:
        print(f"Processing directory: {dirname}")
        for (direcpath, direcnames, files) in os.walk(os.path.join(path, dirname)):
            if not os.path.exists(os.path.join(path1, "train", dirname)):
                os.makedirs(os.path.join(path1, "train", dirname))
            if not os.path.exists(os.path.join(path1, "test", dirname)):
                os.makedirs(os.path.join(path1, "test", dirname))

            num = 100000000000000000  # Large number to save all to train
            i = 0

            for file in files:
                var += 1
                actual_path = os.path.join(path, dirname, file)
                actual_path1 = os.path.join(path1, "train", dirname, file)
                actual_path2 = os.path.join(path1, "test", dirname, file)

                img = cv2.imread(actual_path, 0)
                if img is None:
                    print(f"Failed to read image: {actual_path}")
                    continue
                # bw_image = func(actual_path)
                bw_image = img  # Temporarily use the original image for testing

                if i < num:
                    c1 += 1
                    cv2.imwrite(actual_path1, bw_image)
            else:
                c2 += 1
                cv2.imwrite(actual_path2, bw_image)
            i += 1
        label += 1
print(f"Total images processed: {var}")
print(f"Images saved to train: {c1}")
print(f"Images saved to test: {c2}")
```

Data Augmentation

This code sets up a data augmentation strategy using ImageDataGenerator from Keras to enhance the training dataset by applying random transformations such as:

- Normalization (rescale=1./255)
- Image distortions (shear, zoom, rotation)
- Shifts (horizontal and vertical)
- Brightness adjustments
- Pixel filling for newly created areas after transformation

As previously explained data augmentation has played a pivotal role in our project as it has prevented overfitting of the data and helped in increasing diverse samples. Also it helped he training model to variations in data as the data taken through the input window would be real time so very small factors like lighting issue, background etc., might play a huge role in the gesture classification of our model.

The sample code is given below :

```
# Data augmentation for training and testing
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.1,  # Reduced shear range
    zoom_range=0.1,  # Reduced zoom range
    horizontal_flip=True,
    rotation_range=10,  # Reduced rotation
    width_shift_range=0.1,  # Reduced width shift
    height_shift_range=0.1,  # Reduced height shift
    brightness_range=[0.9, 1.1],  # Adjusted brightness range
    fill_mode='nearest'  # Fill mode for new pixels
)
```

Training CNN Model

CNN Architecture :

This is a Convolutional Neural Network (CNN) built using Keras' Sequential API for image classification into 29 categories (likely for sign language recognition including 26 alphabets + 3 extra classes).

The code snippet is given below :

```
# Initializing the CNN
classifier = Sequential()


# First convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 1), activation='relu'))
classifier.add(BatchNormalization())
classifier.add(MaxPooling2D(pool_size=(2, 2)))


# Second convolution layer and pooling
classifier.add(Convolution2D(64, (3, 3), activation='relu'))
classifier.add(BatchNormalization())
classifier.add(MaxPooling2D(pool_size=(2, 2)))


# Third convolution layer and pooling
classifier.add(Convolution2D(128, (3, 3), activation='relu'))
classifier.add(BatchNormalization())
classifier.add(MaxPooling2D(pool_size=(2, 2)))


# Flattening the layers
classifier.add(Flatten())


# Adding fully connected layers
```

```
classifier.add(Dense(units=256, activation='relu', kernel_regularizer='l2'))   # Added L2 regularization

classifier.add(Dropout(0.3))  # Reduced dropout

classifier.add(Dense(units=128, activation='relu', kernel_regularizer='l2'))   # Added L2 regularization

classifier.add(Dropout(0.3))  # Reduced dropout

classifier.add(Dense(units=29, activation='softmax'))  # Adjusted to 29 classes
```

Training the model:

In this code we train the model using the provided dataset. Then we define the number of steps per epoch (usually, one step in one batch of training samples). We specify 200 training cycles, meaning the model will go through the entire dataset 200 times. Then we use ‘test_set’ for validation, allowing the model to check accuracy on the unseen data. The ‘validation_steps’ determines how many validation steps to perform which is usually based on dataset size.

The code snippet given below:

```
# Train the model and store the history

history = classifier.fit(

    training_set,

    steps_per_epoch=len(training_set),

    epochs=200,

    validation_data=test_set,

    validation_steps=len(test_set)

)
```

Saving the model into json format

This code saves the architecture of a trained Keras model (referred to here as classifier) in JSON format, so it can be reloaded later without retraining. So it helps the model to be pipelined into the app without any hesitation.

The code snippet is given below :

```
# Save the model architecture to JSON

model_json = classifier.to_json()

os.makedirs("model", exist_ok=True)
```

```
with open("model/model-bw.json", "w") as json_file:
    json_file.write(model_json)
print('Model architecture saved to model-bw.json')
```

Testing and GUI creation

This part of the code extensively focuses on the testing phase as well as model pipelining into our app using the Streamlit package in python. we will see the code in various parts.

Dependencies used are mentioned below in the code :

```
import streamlit as st
import cv2
import operator
import numpy as np
from keras.models import model_from_json
from string import ascii_uppercase
from PIL import Image
import time
```

The work of the code is to showcase our classification task of indian signs in a realtime environment using a GUI based platform made with the help of Streamlit package. Here we are loading the model saved in json format which will be further used in our classification task.

```
class SignRecognitionApp:
    def _init_(self):
        self.directory = "model/"

        # Load main model
        with open(self.directory + "model-bw.json") as json_file:
            model_json = json_file.read()
        self.model = model_from_json(model_json)
        self.model.load_weights(self.directory + "model-bw.weights.h5")
        self.current_symbol = "None"

    def predict(self, test_image):
        test_image = cv2.resize(test_image, (128, 128))
        test_image = test_image.reshape(1, 128, 128, 1) / 255.0  # normalize

        result = self.model.predict(test_image)

        prediction = {}
        prediction['blank'] = result[0][0]
        idx = 1
        for ch in ascii_uppercase:
            prediction[ch] = result[0][idx]
            idx += 1

        # Sort primary prediction
        prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)
        predicted_symbol = prediction[0][0]

        self.current_symbol = predicted_symbol
        return self.current_symbol
```

Confusion Matrix

This code computes and visualizes a confusion matrix to asses classification accuracy. We build a numpy matrix, tracking the misclassifications by comparing true vs. predicted labels. We use Seaborn's heatmap to highlight correct and incorrect predictions. The matrix helps analyze misclassified gestures, guiding model improvements through feature refinement or preprocessing.

The code snippet given below:

```
# Confusion matrix plotting function using matplotlib
def compute_confusion_matrix(true_labels, pred_labels, class_labels):
    num_classes = len(class_labels)
    cm = np.zeros((num_classes, num_classes), dtype=int)
    for t, p in zip(true_labels, pred_labels):
        cm[t, p] += 1
    return cm
def plot_confusion_matrix(cm, class_labels):
    plt.figure(figsize=(12, 10))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```