Faculty of Computer & Information Technology

Sign Language Recognition

By:

Amira Ahmed Hashem	**2001213**
Amira Hashem Ahmed	**2000739**
Mona Hassan Abdelhafiz	**2000769**
Sama Alaa Hassan	**2001081**
Sara Ahmed Hashem	**2000987**
Rawan Hamada Ali	**2001021**
Zyad Medhat Abdelraheem	**2001083**

Under the Supervision of:

Prof. Dr. Ahmed Abdelrahim

Professor of Computer and Information Technology Egyptian E-Learning University

Eng. Aya Magdy

Assistant Lecturer in the Faculty of Computer and Information Technology at Egyptian E-LearningUniversity

Assiut 2024

Acknowledgment

We express our sincere gratitude and appreciation to all those who contributed to the successful completion of our graduation project in the Faculty of Computer & Information Technology at the Egyptian E-Learning University.

Firstly, we extend our deepest appreciation to **Dr. Ahmed Abdelrahim**, our esteemed project supervisor and distinguished professor at the Egyptian E-Learning University.

His guidance, expertise, and unwavering support were instrumental in shaping our project and helping us overcome challenges along the way. We are truly grateful for his invaluable insights, patience, and dedication to our academic growth.

We also wish to extend our sincere thanks to **Eng. Aya Magdy**, our dedicated Assistant Lecturer at the Egyptian E-Learning University. Her expertise, constructive feedback, and guidance throughout the project have been immensely valuable. Her continuous encouragement and commitment to our development played a significant role in our project's success.

Furthermore, we express our gratitude to the Faculty of Computer & Information Technology at the Egyptian E-Learning University for providing us with an excellent educational environment and the

necessary resources to pursue our graduation project. The faculty's commitment to academic excellence and emphasis on practical learning greatly enhanced our educational journey.

Additionally, we extend our thanks to our classmates and friends for their unwavering support throughout this project. Their encouragement, constructive discussions, and collaborative spirit enriched our learning experience and inspired us to reach our goals.

Lastly, we acknowledge our families for their constant support, understanding, and encouragement throughout our academic journey. Their love, patience, and belief in our abilities have been the driving force behind our accomplishments.

We are sincerely grateful to all the individuals and institutions mentioned above for their contributions to our graduation project. Their guidance, support, and confidence in our capabilities have been invaluable in shaping our academic and professional growth.

Thank you all sincerely.

Table of Contents:

List of Figures.

Chapter 1

INTRODUCTION

Sign language has long been a crucial means of communication for millions of individuals worldwide who have speech and hearing impairments. This unique form of communication relies on a complex system of gestures, body movements, and facial expressions to convey meaning effectively.

However, it is not always accessible or understood by the public, which can lead to communication barriers and social isolation for those who depend on it. In an increasingly interconnected world, bridging this communication gap is of paramount importance.

As a result, there has been growing interest in developing technologies that can assist in breaking down these barriers, allowing for more seamless interaction between the hearing and non-hearing communities.

One such promising approach is the development of a real-time digital translator that can convert sign language to text, enabling more efficient communication between individuals with speech disabilities and those unfamiliar with sign language where deaf and hearing-impaired people face many challenges and problems in their daily lives.

They suffer from difficulty communicating with people who have not mastered sign language.

It is difficult for them to exchange thoughts, feelings, and daily needs. They may feel isolated and cut off from society. This social isolation can negatively affect mental health. And emotional to them.

Therefore, the idea of our project is to create a mobile application that is a silent conversation to learn sign language for individuals who suffer from deafness or hearing loss, to bridge the communication gap between the deaf community and the hearing world by enabling simultaneous translation of sign language gestures using all algorithms and computer vision techniques.

Deep learning network (CNN) techniques are used to analyses images and extract important information from them. Thus, we can improve communication between deaf and normal people and enhance understanding, participation, and expression of their needs.

The development of a real-time sign language translation system using hand gesture recognition and Convolutional Neural Networks has the potential to revolutionize communication for people with speech disabilities.

By focusing on improving the accuracy and usability of our proposed solution, we hope to create a valuable tool that can break down communication barriers and promote greater social inclusion for those who rely on sign language.

This project represents a significant step forward in leveraging the power of deep learning and computer vision technologies to make a meaningful difference in the lives of people with speech disabilities and their communities. Another critical aspect of this project is to ensure that the resulting system is user-friendly and accessible to a wide range of users, including those with minimal technical expertise.

To achieve this, we will focus on developing an intuitive user interface that simplifies the process of capturing sign language gestures and displaying the corresponding translation in real-time.

We will also consider the integration of our system with popular communication platforms and devices, allowing users to seamlessly incorporate the technology into their daily lives we recognize the importance of validating our proposed solution in real-world scenarios.

To this end, we will conduct thorough testing and evaluation of the system's performance under various conditions, such as different lighting environments, user demographics, and sign language dialects.

We will also seek feedback from the speech-disabled community and sign language interpreters to ensure that our system addresses their needs effectively and accurately.

Chapter 2

BACKGROUND

2.1 Machine Learning

Machine learning is a branch of artificial intelligence (AI) and computer science that revolves around leveraging data and algorithms to mimic the way humans learn, progressively enhancing its accuracy. Our machine learning tutorial is tailored for both students and professionals. It introduces machine learning, covering a broad spectrum of techniques such as supervised and unsupervised learning. You'll delve into concepts like regression, classification models, and clustering.

Machine learning operations are broadly categorized into two main types:

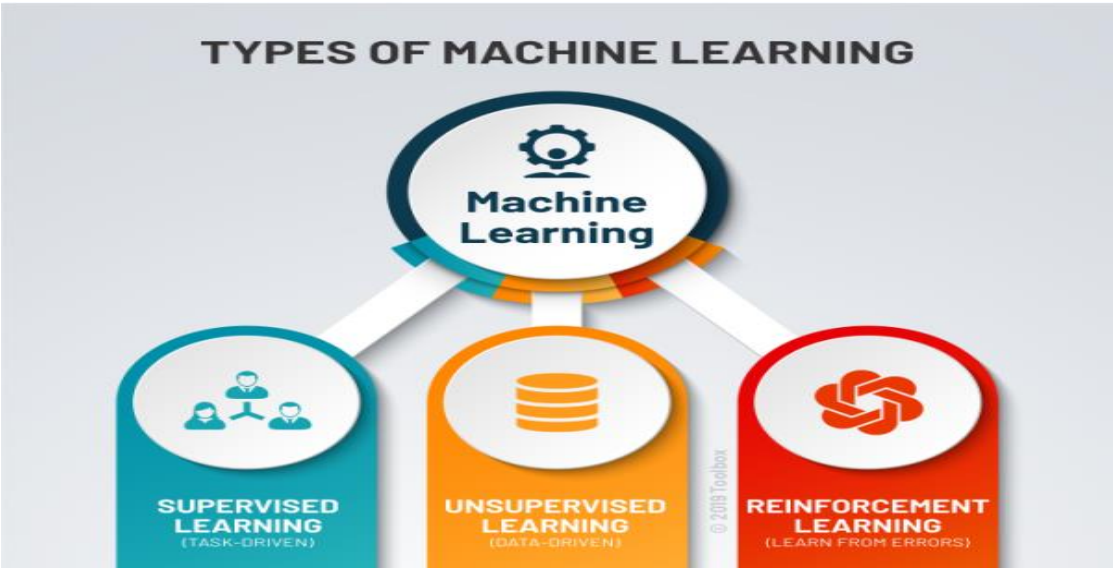


Figure 1Types of machine learning

2.1.1Supervised learning:

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labeled data. Even though the data needs to be labeled accurately for this method to work, supervised learning is extremely powerful when used in the right circumstances.

In supervised learning, the ML algorithm is given a small training dataset to work with. This training dataset is a smaller part of the bigger dataset and serves to give the algorithm a basic idea of the problem, solution, and data points to be dealt with. The training dataset is also very similar to the final dataset in its characteristics and provides the algorithm with the labeled parameters required for the problem.

The algorithm then finds relationships between the parameters given, essentially establishing a cause-and-effect relationship between the variables in the dataset.

2.1.2Unsupervised machine learning:

Unsupervised machine learning holds the advantage of being able to work with unlabeled data. This means that human labor is not

required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program.

In supervised learning, the labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work off of, resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner, with no input required from human beings.

The creation of these hidden structures is what makes unsupervised learning algorithms versatile. Instead of a defined and set problem statement, unsupervised learning algorithms can adapt to the data by dynamically changing hidden structures. This offers more post-deployment development than supervised learning algorithms.

2.1.3Reinforcement Learning

Reinforcement Learning directly takes inspiration from how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged, or ‘reinforced’, and non-favorable outputs are discouraged or ‘punished’.

Based on the psychological concept of conditioning, reinforcement learning works by putting the algorithm in a work environment with an interpreter and a reward system. In every iteration of the algorithm, the output result is given to the interpreter, which decides whether the outcome is favorable or not.

In case of the program finding the correct solution, the interpreter reinforces the solution by providing a reward to the algorithm. If the outcome is not favorable, the algorithm is forced to reiterate until it finds a better result. In most cases, the reward system is directly tied to the effectiveness of the result.

In typical reinforcement learning use-cases, such as finding the shortest route between two points on a map, the solution is not an absolute value. Instead, it takes on a score of effectiveness, expressed in a percentage value. The higher this percentage value is, the more reward is given to the algorithm. Thus, the program is trained to give the best possible solution for the best possible reward.

2.2 Deep Learning

Deep learning [19] is a subset of machine learning that involves the use of artificial neural networks to solve complex problems. Neural networks are computational models that are inspired by the structure and function of the human brain.

The term "deep learning" refers to the fact that these algorithms are typically composed of multiple layers of artificial neurons that are trained using large datasets.

Each layer of neurons is designed to transform the input data in some way, with the goal of learning increasingly abstract and complex representations of the data as it moves through the layers.

Deep learning algorithms are designed to learn representations of data that allow them to automatically extract features and patterns from the data, making them particularly well-suited for tasks such as image recognition, speech recognition, natural language processing, and more. Deep learning has revolutionized the field of artificial intelligence by enabling breakthroughs in areas such as computer vision, natural language processing, and speech recognition.

It has also been used to solve complex problems in science, engineering, and medicine, such as drug discovery, climate modelling, and protein folding. The key advantage of deep learning is its ability to automatically learn from data without the need for manual feature engineering.

This makes it a powerful tool for tasks such as object detection, image classification, and speech recognition, where the complexity of the data makes it difficult to specify a set of rules or features by hand. There are many different types of deep learning architectures, each of which is designed to solve a particular type of problem.

Convolutional neural networks (CNNs) are commonly used for image recognition tasks, while recurrent neural networks (RNNs)mare used for sequential data such as speech or text

2.3 Mobile Application (Flutter & Firebase):

Flutter is a popular open-source UI software development kit created by Google. It's designed to help developers build natively compiled applications for mobile, web, and desktop from a single codebase.

One of the standout features of Flutter is its fast development cycle. Developers can see changes in real-time thanks to its hot reload feature, which instantly updates the app as they write code. This makes the development process more efficient and iterative.

Flutter uses the Dart programming language, which was also developed by Google. Dart is known for its simplicity and ease of learning, making it accessible to developers with various levels of experience.

Another advantage of Flutter is its expressive and flexible UI framework. It provides a wide range of customizable widgets that allow developers to create beautiful and fluid user interfaces. These widgets can adapt to different screen sizes and resolutions, providing a consistent experience across platforms.

Flutter's performance is also impressive, as it compiles code directly to native machine code, eliminating the need for a bridge between

the code and the platform. This results in smooth animations, fast rendering, and overall high performance.

Furthermore, Flutter has a growing ecosystem of packages and plugins that extend its functionality. This allows developers to easily integrate features like maps, authentication, and push notifications into their apps without having to build everything from scratch.

Overall, Flutter has gained popularity among developers for its speed, flexibility, and ability to create high-quality apps for multiple platforms with a single codebase. It continues to evolve, with regular updates and improvements from both Google and the open-source community.

Cross-Platform Development: Flutter allows developers to build apps for multiple platforms, including iOS, Android, web, and desktop, using a single codebase. This enables companies to reach a wider audience and streamline their development process by reducing the need to maintain separate codebases for each platform.

Material Design and Cupertino Widgets: Flutter provides both Material Design widgets (for Android apps) and Cupertino widgets (for iOS apps), allowing developers to create apps that adhere to the design guidelines of each platform. This ensures that apps look and feel native, providing users with a familiar experience.

State Management: Flutter offers various state management solutions to help developers manage the state of their applications effectively. Options include built-in state management solutions like setState, provider, and Riverpod, as well as external libraries such as Redux, MobX, and Bloc.

Internationalization and Localization: Flutter supports internationalization and localization out of the box, making it easy for developers to create apps that support multiple languages and regions. This is essential for reaching global audiences and ensuring that users can interact with the app in their preferred language.

Integration with Firebase: Firebase is Google's mobile and web application development platform, and Flutter provides seamless integration with Firebase services such as authentication, cloud storage, real-time database, cloud messaging, and more. This allows developers to leverage powerful backend services without writing server-side code.

Performance Monitoring and Debugging: Flutter comes with built-in tools for performance monitoring and debugging, such as the Flutter DevTools suite. These tools help developers identify performance bottlenecks, optimize their code, and debug issues more efficiently, ultimately improving the quality of their apps.

Community and Support: Flutter has a large and active community of developers who contribute to its ecosystem by creating plugins, packages, tutorials, and other resources. This community support makes it easier for developers to learn Flutter, troubleshoot problems, and stay up to date with the latest developments in the framework.

These are just a few additional aspects of Flutter that contribute to its popularity and effectiveness as a cross-platform development framework. As Flutter continues to evolve, we can expect to see even more features and improvements that further enhance its capabilities.

Firebase is a comprehensive platform developed by Google for building mobile and web applications. It offers a wide range of services that help developers with various aspects of app development, including backend infrastructure, database management, authentication, analytics, and more.

Here are some key components and features of Firebase:

1. **Realtime Database**: Firebase provides a NoSQL cloud database that allows developers to store and sync data between users in real time. It's particularly well-suited for applications that require real-time updates, such as chat apps, collaborative tools, and multiplayer games.

2. **Firestore**: Firestore is Firebase's newer, more scalable NoSQL database that offers features like automatic scaling, offline support, and more complex querying capabilities. It's designed to support larger-scale applications and provides a more flexible data model compared to the Realtime Database.

3. **Authentication**: Firebase Authentication makes it easy to add user authentication to your app using email/password, social logins (Google, Facebook, Twitter, etc.), phone number authentication, and more. It handles user management, authentication, and identity verification, allowing developers to focus on building the core features of their app.

4. **Cloud Functions**: Cloud Functions for Firebase allows developers to run backend code in response to events triggered by Firebase features and HTTPS requests. This serverless compute platform lets developers extend the functionality of their app without managing servers, enabling tasks like sending notifications, processing data, and integrating with third-party services.

5. **Cloud Storage**: Firebase Storage provides secure cloud storage for user-generated content such as images, videos, and files. It offers powerful features like resumable uploads, access control, and integration with Firebase Authentication, making it easy to store and serve media files in your app.

6. **Cloud Messaging**: Firebase Cloud Messaging (FCM) enables developers to send notifications and messages to users across platforms (iOS, Android, and web). It supports various types of messages, including notifications, data messages, and topic-based messaging, allowing developers to engage with users and keep them informed.

7. **Analytics**: Firebase Analytics provides insights into user behavior and app performance, helping developers understand how users interact with their app, identify trends, and make data-driven decisions to optimize the user experience.

8. **Performance Monitoring**: Firebase Performance Monitoring helps developers identify performance issues in their app, such as slow startup times, long-running operations, and network latency. It provides detailed metrics and traces to diagnose performance bottlenecks and improve app responsiveness.

9. **Remote Config**: Firebase Remote Config allows developers to dynamically configure and personalize their app without requiring a new release. It enables A/B testing, feature flags, and targeted rollouts, allowing developers to iterate quickly and optimize the user experience based on real-time feedback.

10. **Machine Learning**: Firebase ML provides pre-trained models and APIs for adding machine learning capabilities to your app, such as text recognition, image labeling, and natural language processing. It makes it easy to integrate AI-powered features into your app without requiring extensive machine learning expertise.

Overall, Firebase offers a comprehensive suite of tools and services that simplify app development, reduce infrastructure complexity, and enable developers to focus on building great user experiences. Its seamless integration with other Google services and its robust set of features makes it a popular choice among developers for building scalable, feature-rich applications.

Firebase offers robust authentication services that make it easy to implement secure sign-in and login functionality in mobile and web applications. Here's how Firebase can be used for sign-in and login:

1. **Email/Password Authentication**: Firebase Authentication provides built-in support for email and password authentication. Developers can enable this authentication method in their Firebase project, allowing users to create an account using their email address and a password. Firebase handles the authentication flow securely, including password hashing and salting, account verification, and password reset functionality.

2. **Social Sign-In**: Firebase Authentication supports social sign-in methods, such as Google Sign-In, Facebook Login, Twitter Auth, GitHub Auth, and more. Developers can easily integrate these social authentication providers into their app, allowing users to sign in using their existing social media accounts. Firebase handles the OAuth flow and provides secure access tokens for authenticated users.

3. **Phone Number Authentication**: Firebase Authentication also supports phone number authentication, allowing users to sign in using their mobile phone number. This method is particularly useful for apps that target users in regions where email addresses may not be prevalent or for apps that require two-factor authentication (2FA) via SMS.

4. **Custom Authentication**: Firebase Authentication provides support for custom authentication systems, allowing developers to integrate their own authentication backend with Firebase. This can be useful for apps that already have existing authentication systems or require custom authentication logic.

5. **Single Sign-On (SSO):** Firebase Authentication supports single sign-on (SSO) across multiple platforms and devices. Once a user is authenticated on one device, Firebase automatically manages their authentication state and provides seamless access to protected

resources on other devices without requiring the user to sign in again.

6. **User Management**: Firebase Authentication provides a suite of user management features, including account creation, email verification, password reset, user profile management, and account deletion. Developers can easily manage user accounts and access user information through the Firebase console or programmatically using the Firebase Admin SDK.

7. **Security Rules Integration**: Firebase Authentication seamlessly integrates with Firebase Security Rules, allowing developers to enforce fine-grained access control policies based on the authenticated user's identity and attributes. This ensures that only authorized users can access protected resources within the app.

8. **Analytics and Insights**: Firebase Authentication provides analytics and insights into user authentication events, such as sign-in attempts, successful sign-ins, failed sign-ins, and account creations. Developers can track user engagement, monitor authentication trends, and identify potential security issues using Firebase Analytics.

By leveraging Firebase Authentication, developers can implement secure, scalable, and user-friendly sign-in and login functionality in their applications, without the need to manage complex authentication infrastructure or compromise on security. Firebase handles the heavy lifting, allowing developers to focus on building great user experiences.

Chapter 3

LITERATURE REVIEW (Related Work)

Abstract:

A literature review is a survey of scholarly sources on a specific topic. It provides an overview of current knowledge, allowing you to identify relevant theories, methods, and gaps in the existing research that you can later apply to your paper, thesis, or dissertation topic.

One of our steps in our project is to collect a good amount of research papers and read them and understand them so we collected 5 paper each and 35 combined and choose 10 most accuracy once and get the details of them.

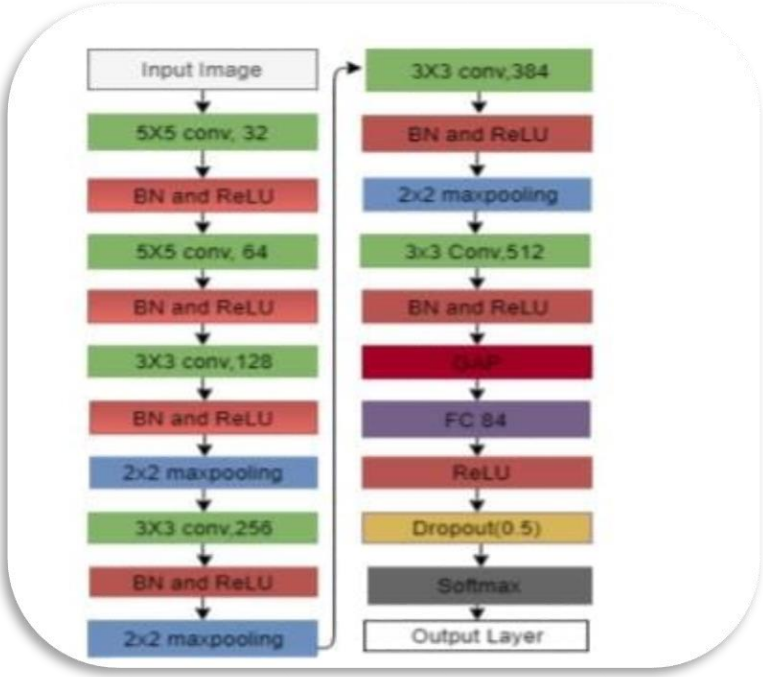
Page number	Name	year	Algorithm	Dataset
1	A New Benchmark on ASL Recognition using CNN	2019	CNN	**Synthetic ASL Alphabet**
2	Sign Language Alphabet Recognition Using Convolution Neural Network	2021	CNN	**Sign Language (ENG Alphabet)**

3	CNN Model for American Sign Language Recognition	2020	CNN	**Sign Language MNIST**
4	Hyper tuned Deep Convolutional Neural Network for Sign Language Recognition	2023	CNN	**Sign Language MNIST**
5	Sign Language to Sentence Interpreter Using Convolutional Neural Network in Real Time	2022	CNN	**ASL(American Sign Language) Alphabet Dataset**
6	Classification of Sign Language Characters by Applying a Deep Convolutional Neural Network	2020	CNN	**Sign Language MNIST**
7	Sign language recognition system for communicating to people with disabilities	2023	CNN	**ASL Hand sign Dataset (Grayscale & Thresholder)**

8	A Translator for Sign Language to Multilingual Text and Speech	2023	CNN	**Beginner to Intermediate NLP Tutorial**
9	Reconstruction of CNN for Sign Language Recognition	2020	CNN	**ASL Fingerspelling Recognition w/ TensorFlow**
10	Arabic Sign Language Recognition and Generating Arabic Speech Using Convolutional Neural Network	2020	CNN	**RGB Arabic Alphabets Sign Language Dataset**

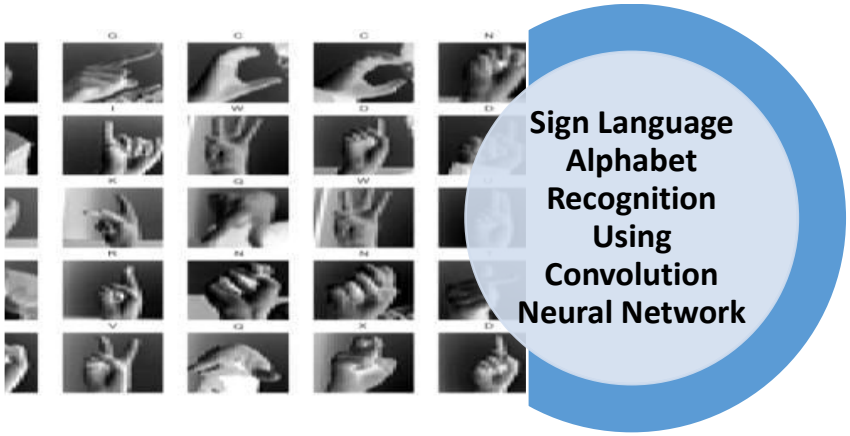
Paper1:

A New Benchmark on Ameri can Sign Language Recognition using Con volutional Neural Network



The listening or hearing impaired (deaf/dumb) people use a set of signs, called sign language instead of speech for communication among them. However, it is very challenging for non-sign language speakers to communicate with this community using signs. It is very necessary to develop an application to recognize gestures or actions of sign languages to make easy communication between the normal and the deaf community. The American Sign Language (ASL) is one of the mostly used sign languages in the World, and considering its importance, there are already existing methods for recognition of ASL with limited accuracy. The objective of this study is to propose a novel model to enhance the accuracy of the existing methods for ASL recognition.

Paper2:



Sign Language plays an indispensable role in the lives of people who have speaking and hearing disabilities. Recognition of American Sign Language using Computer Vision is very challenging due to its increasing complexity and high intraclass variations. In this paper, convolutional neural networks (CNNs) are used to recognize the ASL Alphabets.

This algorithm is useful to recognize it as a deep network, which is expected for the ASL alphabet classification task. Pre-Processing steps of the MNIST dataset are done in the first phase.

After the first phase, different important features of pre-processed hand gesture image are computed. In the final phase, depending on the properties computed or calculated in the initial phases, the accuracy and AUC score of the network model with which it can

recognize the sign language Alphabets were detected. The proposed CNN network has achieved an AUC score of 0.9981 and an accuracy of 0. 9963.Keywords—Convolution neural network (CNN), American Sign Language (ASL), ASL number, Data augmentation, Sign language recognition component.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
=================================================================
Total params: 172,441
Trainable params: 171,993
Non-trainable params: 448
```

Paper3:

CNN Model for American Sign Language Recognition

Layers

```
convl = Sequential() convl.add(Conv2D(32,
kernel_size=(3,3),activation = 'relu
', input_shape= input_shape ))
convl.add(BatchNormalization())

convl.add(Conv2D(32, kernel_size=(3,3),activation = 'relu
' ))
convl.add(BatchNormalization())
convl.add(MaxPooling2D(pool_size = (2,2)))
#convl.add(Dropout(0.2))

convl.add(Conv2D(64, kernel_size=(3,3),activation = 'relu
' )) convl.add(BatchNormalization())
convl.add(MaxPooling2D(pool_size =
(2,2)))
#convl.add(Dropout(0.2))
 convl.add(Conv2D(128, kernel_size=(3,3),activation =
'rel u' )) convl.add(MaxPooling2D(pool_size = (2,2)))
 convl.add(Flatten())
convl.add(Dense(128,activation = 'relu'))
convl.add(Dense(25, activation = 'softmax'))
```

Regularization Function

```
history1=convl.fit_generator(batches,steps_per_epoch=2196
4//128,epochs=30,validation_data=val_batches,validation_s
teps=5491//128, use_multiprocessing=True)
```

This paper proposes a model based on convolutional neural network for hand gesture recognition and classification. The dataset uses 26 different hand gestures, which map to English alphabets A–Z. Standard dataset called Hand Gesture Recognition available in Kaggle website has been considered in this paper. The dataset contains 27,455 images (size 28 * 28) of hand gestures made by different people. Deep learning technique is used based on CNN which automatically learns, and extracts features for classifying

each gesture. The paper does comparative study with four recent works. The proposed model reports 99% test accuracy.

Paper4:

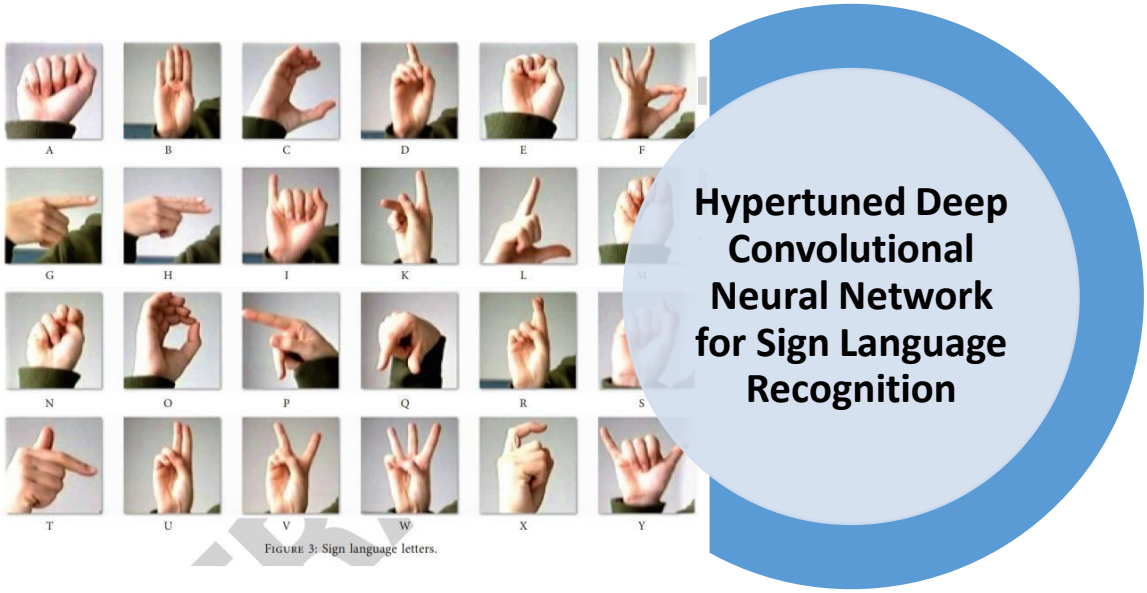
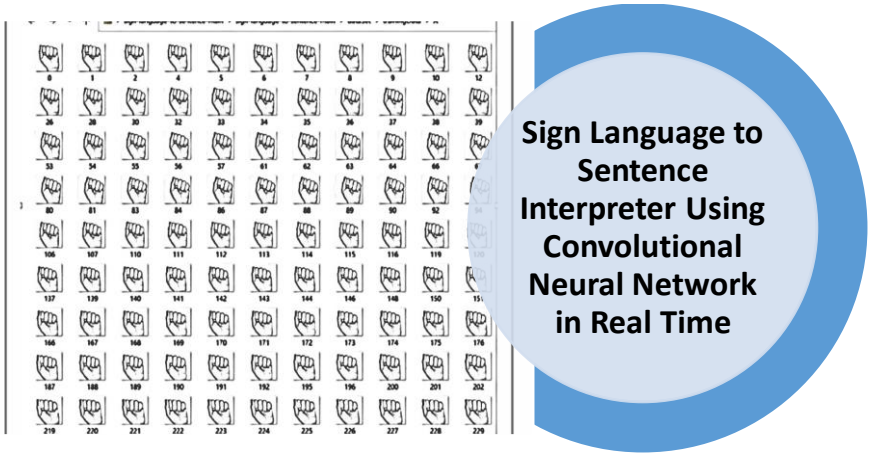


FIGURE 3: Sign language letters.

Sign language plays a pivotal role in the lives of impaired people having speaking and hearing disabilities. They can convey messages using hand gesture movements. American Sign Language (ASL) recognition is challenging due to the increasing intra-class similarity and high complexity. This paper used a deep convolutional neural network for ASL alphabet recognition to overcome ASL recognition challenges. This paper presents an ASL recognition approach using a deep convolutional neural network. The performance of the DeepCNN model improves with the amount of given data; for this purpose, we applied the data augmentation technique to expand the

size of training data from existing data artificially. According to the experiments, the proposed DeepCNN model provides consistent results for the ASL dataset. Experiments prove that DeepCNN gives a better accuracy gain of 19.84%, 8.37%, 16.31%, 17.17%, 5.86%, and 3.26% as compared to various state-of-the-art approaches.

Paper5:



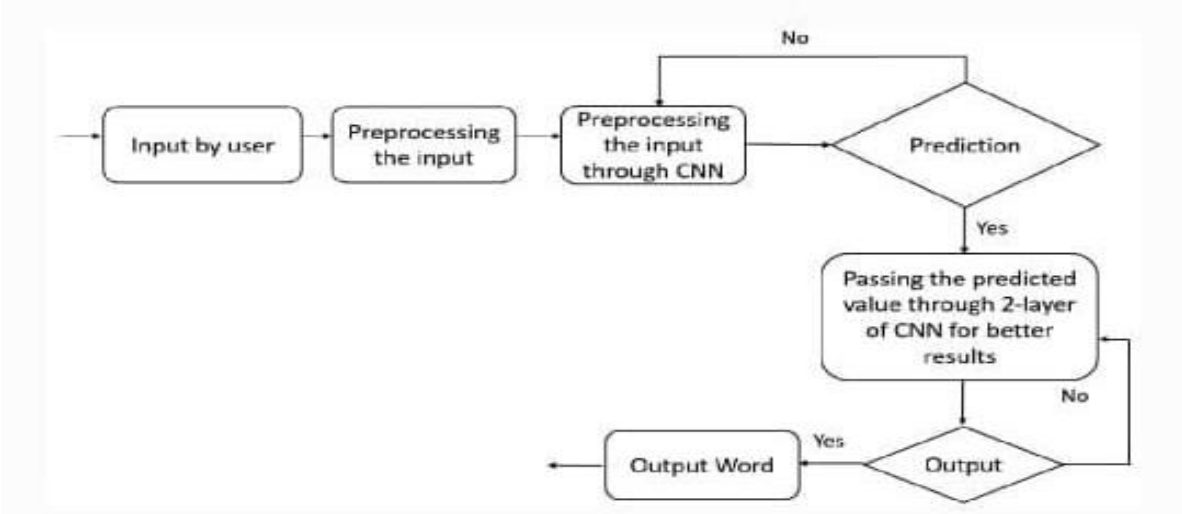
Sign languages are used by several people across the globe as their primary language. Sign language is a non-verbal language that people use to communicate with each other by using facial/body expressions, postures, and a set of gestures. It is based on visual orientation and is natural. It is primarily used as a means of communication with hearing impaired or deaf individuals. Artificial intelligence (AI) has been utilized to break down communication

barriers between the hearing and deaf communities, seeking to create automatic sign language generation and recognition systems as well as to “connect with the machines”.

This article discusses about a desktop application that interprets finger spelling based on American Sign Language in real time using neural networks.

In this method, the hand gesture is photographed using a camera, put through a filter that eliminates RGB background noise and utilizes a Gaussian filter to capture the edges of the hands, and then put through a classifier that determines the type of hand motion it belongs to.

The character is then shown, and a sentence is then formed by adding both. The employed method has a 98.7% accuracy rate for the alphabet’s 26 letters.



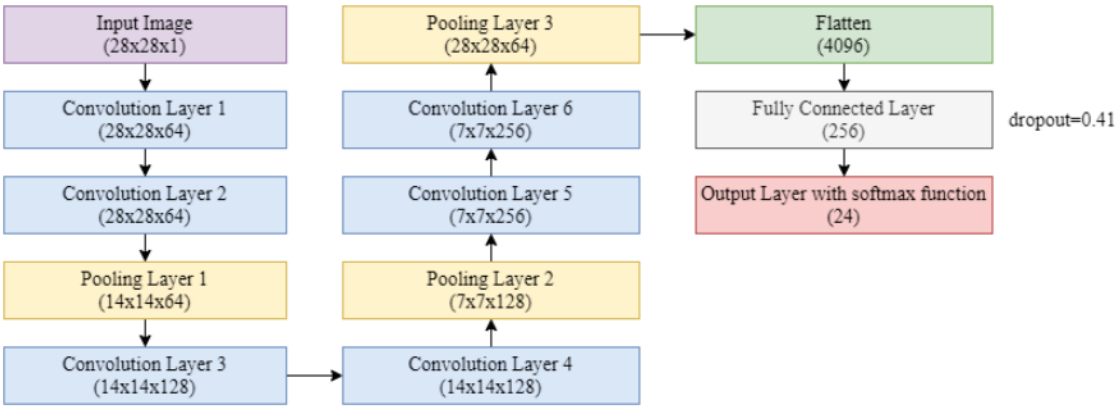
Paper 6 :

Classification of Sign Language Characters by Applying a Deep Convolutional Neural Network

Having a massive community of almost 466 million deaf-mute people all over the world, sign language recognition has always fascinated researchers to develop sophisticated models that can successfully recognize sign languages.

Because of not being a universal language, sign language differs in terms of languages and communities. Previously, various research have been conducted on different sign languages. In this study, we considered the Sign Language MINST dataset. Previously, different classifiers like support vector machine, random forest, multilayer perceptron, etc. have been introduced for sign language recognition. Recently, shallow CNN and Capsule Networks have obtained better results. Therefore, in this research, we proposed a deep convolutional neural network model to achieve the successful identification of the sign linguistics alphabets.

After implementing the model, we produced an overall accuracy of 97.62% and comparison with previous research revealed that our proposed model outperformed all previously introduced models.



Paper 7 :

this research aims to recognize hand gestures or ASL, which the system will change into text that can be read in real-time, making communication with people with special needs easier. Hand gesture recognition is also in Human-Computer Interaction (HCI) because it interacts with the user directly.

Human-Computer Interaction (HCI) is the study, planning, or design of interaction between users and computers.

One of the functional interactions for a hand gesture recognition system is displaying text composed of alphabets read by the system. In this research, we will utilize Computer Vision and Pattern Recognition technology to create a desktop application that can detect hand movements in real-time using a webcam/live camera.

Afterward, we will use American Sign Language (ASL) datasets and the Convolutional Neural Networks (CNN) classification system. This research focuses on the accuracy of recognizing letters of the alphabet and provides the results in a text in real-time.

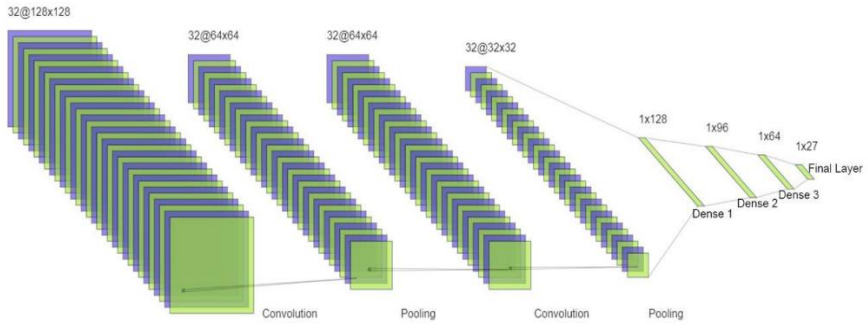
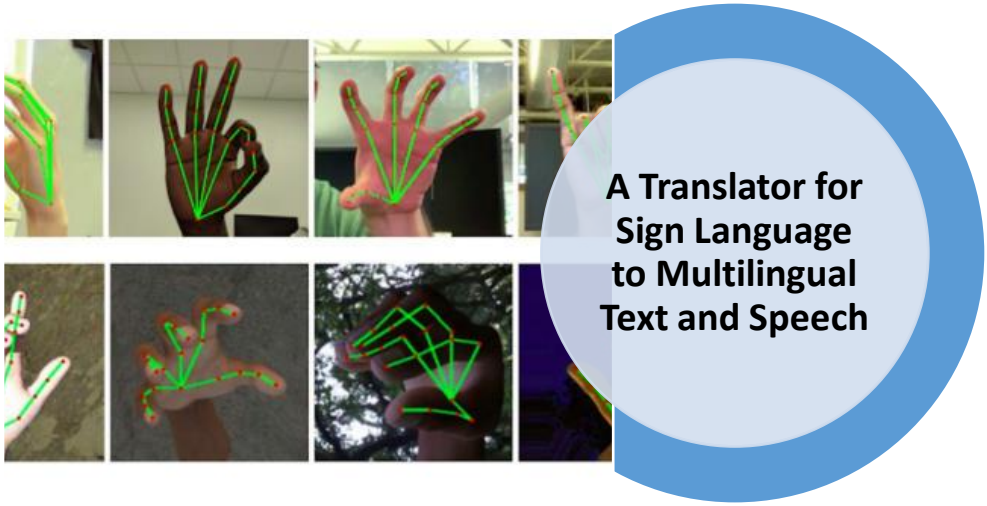


Fig. 2. Model Architecture

Paper 8:



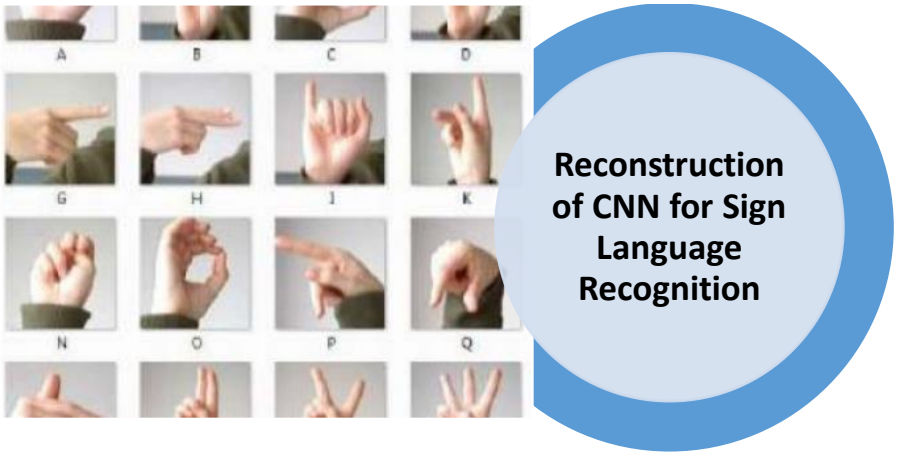
The system's design and implementation are based on the use of Media Pipe, a cross-platform framework for building multimodal applied machine learning pipelines. The system uses deep learning models for sign language recognition and translation and employs speech synthesis techniques to generate spoken language output.

The project involved data collection, preprocessing, and model training using deep learning architectures such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The performance of the system was evaluated using various evaluation metrics, including accuracy, precision, and recall. The results obtained demonstrate that the system is capable of accurately recognizing and translating sign language gestures into

written and spoken language in multiple languages with high accuracy

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 30, 64)            442112

lstm_4 (LSTM)                (None, 30, 128)           98816

lstm_5 (LSTM)                (None, 64)                49408

dense_3 (Dense)              (None, 64)                4160

dense_4 (Dense)              (None, 32)                2080

dense_5 (Dense)              (None, 3)                 99

=================================================================
Total params: 596,675
Trainable params: 596,675
Non-trainable params: 0
_________________________________________________________________
```

Paper 9:



This paper presents a Sign Language translation model using Convolutional Neural Networks (CNN).

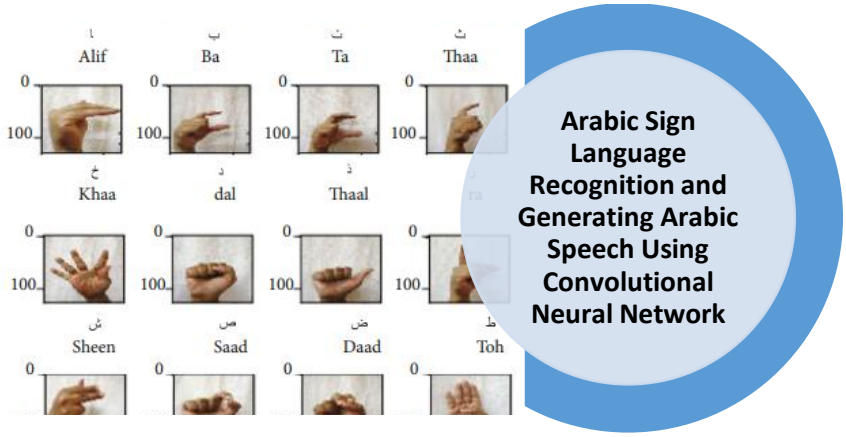
Sign language is a language which allows mute and hearing-impaired people to communicate. It is a visually oriented, nonverbal communication which facilitates communication through body/facial postures, expressions and a collection of gestures. To contribute to the wellbeing of the affected population, we are motivated to implement a vision-based system to avert their day-to-day challenges. Our proposed model constitutes object detection and classification phases. The first module is made up of single shot multi-box detector (SSD) used for hand detection.

The second module constitutes convolutional neural network plus a fully connected network utilized to constructively translate the detected signs into text. The proposed model is implemented using American sign language fingerspelling dataset. The proposed system outperformed other published results in the comparative analysis, hence recommended for further exploitation in sign language recognition problems.

TABLE I MODEL PROPERTIES

Layer type	*Output shape*	*Parameters*
conv2d_1 (Conv2D)	(None, 26, 26,16)	160
Max_pooling2d_1	(None, 13, 13,16)	0
conv2d_2 (Conv2D)	(None, 11, 11,32)	4640
max_pooling2d_2	(None, 5, 5, 32)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	18496
max_pooling2d_3	(None, 1, 1, 64)	0
flatten_1 (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 768)	49920
dense_2 (Dense)	(None, 128)	98432
dense_3 (Dense)	(None, 24)	3096

Paper 10:

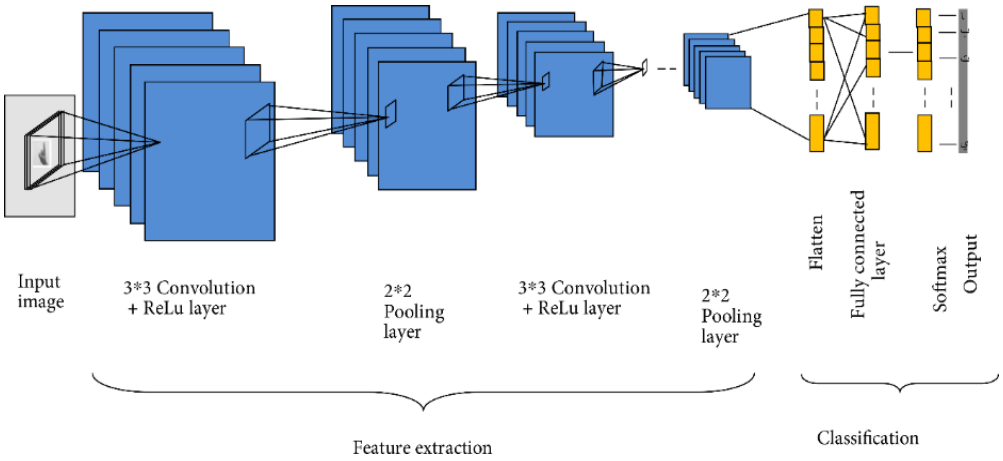


the recent progress in the computer vision field has geared us towards the further exploration of hand signs/gestures' recognition with the aid of deep neural networks. Arabic sign language has witnessed unprecedented research activities to

recognize hand signs and gestures using the deep learning model. A vision-based system by applying CNN for the recognition of Arabic hand sign-based letters and translating them into Arabic speech is proposed in this paper.

The proposed system will automatically detect hand sign letters and speak out the result with the Arabic language with a deep learning model. This system gives 90% accuracy to recognize the Arabic hand sign-based letters which assures it as a highly dependable system.

The accuracy can be further improved by using more advanced hand gestures recognizing devices such as Leap Motion or Xbox Kinect. After recognizing the Arabic hand sign-based letters, the outcome will be fed to the text into the speech engine which produces the audio of the Arabic language as an output.



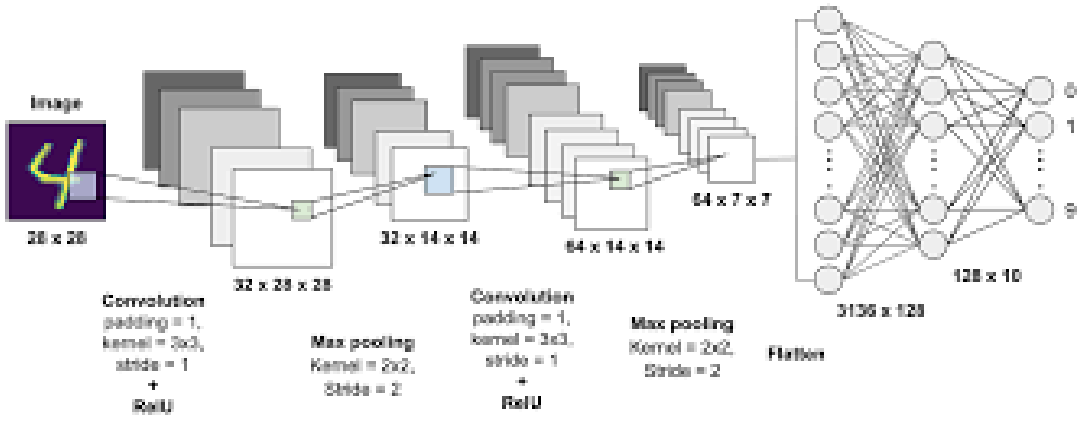
Chapter 4

SYSTEM METHODOLOGY

4.1 CNN model:

We will use the CNN model, which is a type of neural network that is used in computer vision, and it is one of the very important fields in deep learning. It processes and recognizes data (images and videos). It is designed to deal with this type of two- and three-dimensional data and extract information from it. This is done through several layers that make up the model, which enables it to better recognize images and classify them correctly. It can deal with a huge amount of data. The more numerous and diverse data is available, the better its work.

Key components of a typical CNN model include:



- Convolutional Layers: These layers apply convolution operations to the input data, using filters or kernels to extract specific features.

- Pooling Layers: Pooling layers down sample the spatial dimensions of the input data, reducing the computational complexity while retaining important features.

- Activation Functions: Non-linear activation functions, like ReLU (Rectified Linear Unit), introduce non-linearity to the model, enabling it to learn complex patterns.

- Fully Connected Layers: These layers connect every neuron from one layer to every neuron in the next layer, enabling the model to make final predictions based on the learned features.

- Flattening: Before feeding data into fully connected layers, the output from previous layers is often flattened into a one-dimensional array.

4.2Steps of Creating a CNN model

Having outlined the steps our model is expected to take, we will delve into a comprehensive exploration of each of these steps throughout the implementation phase.

There are several steps to divide the data in order to build the model.

4.2.1 Data Collection:

During the implementation phase, one crucial aspect is the

systematic collection of data

------	English	Arabic	Spanish	India
Alphapts				
Words				
Sentences				

Figure 2 Data collection

4.2.2 Data Splitting:

This process involves dividing the available dataset into distinct subsets for training, validation, and testing. The training set is used to train the model, the validation set helps tune hyperparameters and avoid overfitting, while the testing set evaluates the model's performance on unseen data. Data splitting is essential for assessing a model's performance objectively and preventing it from overfitting to the training data, allowing for a more reliable evaluation of its capabilities.

ASL Alphabet Dataset:

- X-train= 87000 sample, Y-train= 87000 sample
- Split=%20:80

ASL word Dataset:

- X-train= sample, Y-train= sample
- Split=%20:80

4.2.3 Data Preprocessing:

Data preprocessing is a fundamental step in the implementation of machine learning models, involving the cleaning, transformation, and organizing of raw data to make it suitable for analysis. This

crucial phase aims to enhance the quality of the dataset, address missing or irrelevant information, and standardize the data format.

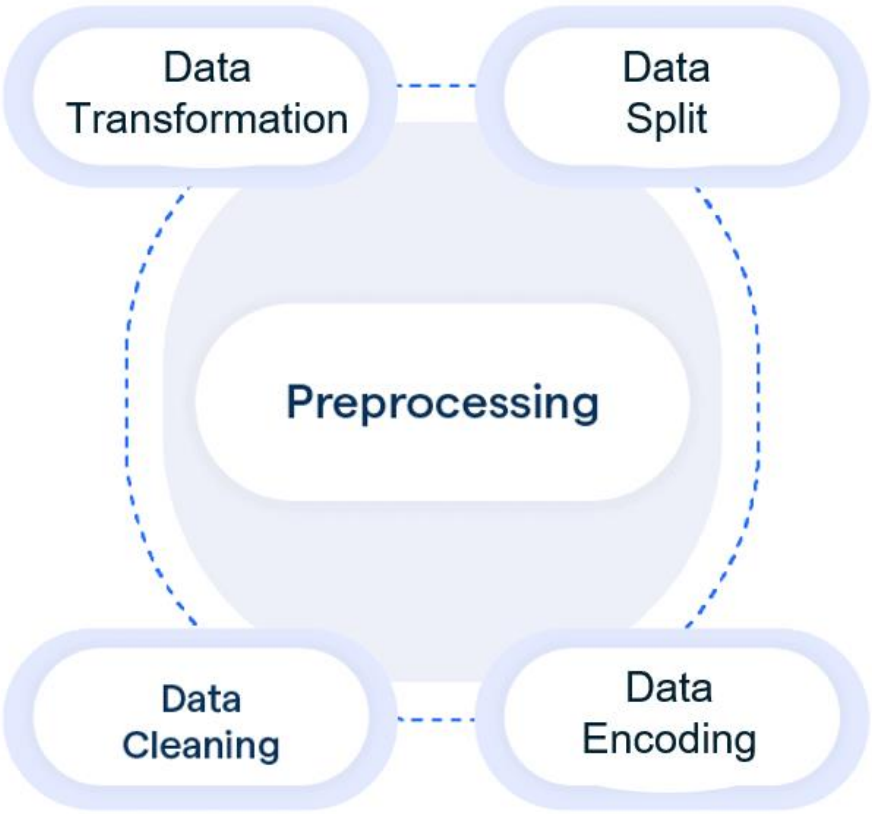


Figure 3Preprocessing

The key aspects of data preprocessing include:

- Data Cleaning: Identifying and handling missing data, outliers, or inaccuracies in the dataset to ensure its integrity.

- Data Transformation: Converting and scaling features to a consistent format or scale, often through techniques like normalization or standardization.

- Data Encoding: Converting categorical variables into a numerical format that can be processed by machine learning algorithms.

- Data Splitting: Dividing the dataset into training, validation, and testing sets for model development, tuning, and evaluation.

Effective data preprocessing lays the foundation for robust and accurate machine learning models by ensuring that the input data is well-organized and suitable for training and testing.

4.2.4CNN Model:

Here, the CNN model is built so that it consists of number of layers that are appropriate for the model to analyze data (images and videos), and it consists of number of hyperparameters that are set and modified until we achieve the highest accuracy of the model.

4.2.5 Model Training:

Model training is a crucial phase in machine learning where a model learns patterns and representations from a given dataset. In the context of a Convolutional Neural Network (CNN) for image-related tasks, the training process involves the following steps:

- **Input Data:**

The training set consists of a collection of input data (such as images) and their corresponding labels. Each input has specific features or characteristics that the model aims to learn.

- **Forward Pass:**

During training, each input is fed into the CNN through a forward pass. This involves applying a series of convolutional operations, activation functions, pooling layers, and fully connected layers to extract hierarchical features from the input.

- **Loss Calculation:**

The model's predictions are compared to the actual labels using a loss function. The loss function quantifies the difference between the predicted output and the true labels, serving as a measure of how well the model is performing.

- **Backward Pass (Backpropagation):**

The gradient of the loss with respect to the model's parameters is calculated through backpropagation. This involves computing the derivative of the loss function with respect to each parameter, indicating how much each parameter contributed to the error.

- **Optimization:**

Optimization algorithms, such as gradient descent, are then employed to update the model's internal parameters (weights and biases). The goal is to minimize the loss function by adjusting these parameters in the direction that reduces the error.

- **Epochs:**

The entire training dataset is usually processed through the model multiple times in what is called an epoch. During each epoch, the model's parameters are updated based on the entire training dataset.

- **Iterations:**

The training process consists of multiple iterations (epochs). The model learns to generalize from the training data, improving its ability to make accurate predictions on unseen examples.

- **Convergence:**

Training continues until the model converges, meaning that the parameters reach a state where further adjustments do not significantly improve performance on the training set.

4.2.6 Model Evaluation:

In it, the trained model is applied to the test data, and Accuracy Measurement. The model's predictions are compared to the actual values in the test set. Accuracy is calculated as the ratio of the

number of correct predictions to the total number of predictions. and can used Other Metrics Besides accuracy, other metrics like recall, precision, and F1-score can be used for binary classification. A confusion matrix might be employed to analyze the model's performance across different classes.

4.2.7 Improve performance:

The hyperparameters are adjusted if the accuracy of the model is low and there is an Overfitting or Bias until we reach the highest accuracy. such as Data Augmentation:

Increase the diversity of the training dataset by applying random transformations to existing data, especially in image processing.

and Hyperparameter Tuning:

Fine-tune hyperparameters like learning rate, batch size, and regularization strength. Utilize techniques such as grid search or random search to explore the hyperparameter space. and other processing such as Model Architecture:

Adjust the complexity of the model architecture. Experiment with adding or removing layers, changing the number of nodes, or utilizing pre-trained models for transfer learning.

4.2.8 Final Model Test:

By completing these steps, you ensure that the model is tested on fresh, unseen data, verifying its ability to recognize sign language gestures and confirming that it functions correctly in real-world scenarios. If the model performs well, it can be considered ready for deployment and use in practical applications. If there are areas for improvement, further adjustments may be made to enhance its performance.

4.2.9 Integrate the model into an application:

Once you have verified the performance and accuracy of your model, the next step is to integrate it into a mobile application. Here's a simplified explanation of this process:

- **Optimize for Mobile Resources:**

Optimize the app to efficiently use mobile resources, considering factors like memory consumption and energy efficiency.

- **Testing:**

Rigorously test the integrated model within the mobile application across various scenarios and devices to ensure robustness and reliability.

- **Deployment:**

Prepare the mobile application for deployment to app stores or distribution platforms.

4.3 Libraries

TensorFlow:

It offers a complete environment for creating and implementing deep learning models inside machine learning. TensorFlow is extensively used in many different applications, including natural language processing, image and audio recognition.

Keras:

is a widely adopted high-level neural networks API that serves as the official high-level API for Tensor Flow It makes training neural networks easy to implement.

Sklearn:

a open- source Machine learning library It provides simple and efficient tools for data analysis and modelling used in Machine learning a pre-processing.

Seaborn:

it is a data visualization library based on Matplotlib It creates it creates statistical graphics it simplifies the complexity of visualizations.

Matplotlib:

Is a comprehensive data visualization library for Python that enables the creation of a wide range of static, animated, and interactive plots.

Os:

provides a way of interaction with the operating system it perform files or dictionaries operations such as making file using osmium () or rename using os.rename() it also useful to construct and deconstruct files and dictionaries paths.

Cv2:

allows us to read and write images and videos in various formats. It supports a wide range of image and video file types and provides functions for capturing video from cameras also uses for image processing tasks like resizing, cropping, rotation and other processes.

Open cv:

a computer vision library it used for image and video processing and object detection and recognition.

NumPy:

it used for numerical computing Its efficient array operations, universal functions, and linear algebra capabilities make it an essential tool for a wide range of scientific and engineering applications, including data analysis, machine learning, and simulations.

Pandas:

open-source data analysis library it supports reading and writing data from/to various file formats, including CSV, Excel, SQL databases, JSON, and more. This makes it easy to import and export data between Pandas and different data sources Pandas also used in data cleaning and pre-processing.

It provides efficient and flexible tools for working with structured data.

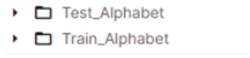
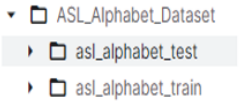
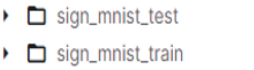
4.4 The Dataset (Done or not)

....	English	Arabic	Spanish	India
Alphapts				
Words				
Sentences				

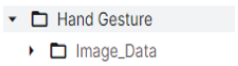
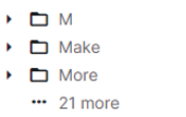
Figure 4The Dataset (Done or not)

4.5 Collection Dataset From Different Language

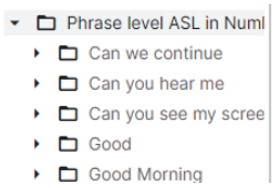
English Alphabet:

	Name	Abstract	size	photo	Source
1	Synthetic ASL Alphabet	A-Z	7 GB		Kaggle
2	ASL(American Sign Language) Alphabet Dataset	A-Z	5 GB		Kaggle
3	Sign Language MNIST	9-0 A-Z	66 MB		Kaggle
4	ASL Alphabet	A-Z , 3 classes for SPACE, DELETE & NOTHING.	1 GB		Kaggle

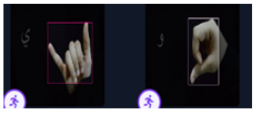
English Words:

	Name	Abstract	size	photo	Source
1	WLASL (World Level American Sign Language) Video	2,000 common different words	5 GB		Kaggle
2	Hand Gesture Recognition	50 words each containing 40 videos	3 GB		Kaggle
3	ASL_and_same words	Image 203000 51 class (4000)	2GB		Kaggle

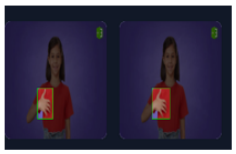
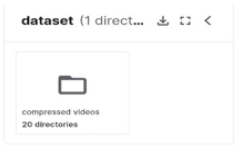
English Sentence:

	Name	Abstract	size	photo	Source
1	Phrase level ASL converted into 1D-array	30 folders has 30 different folders within that each contain 30 NumPy arrays.	212 MB		Kaggle

Arabic Alphabet:

	Name	Abstract	size	photo	Source
1	Arabic Sign Language ArSL dataset	ن ال أ الي ال Image ي Alphabet	139 MB		Kaggle
2	Arabic SL Computer Vision Project	Image	200 MB		Roboflow

Arabic Words:

	Name	Abstract	size	photo	Source
1	Arabic-sign-language-words-detection Computer Vision Project	Image Words	400 MB		Roboflow
2	Arabic sign language dataset	Video	2 GB		Kaggle

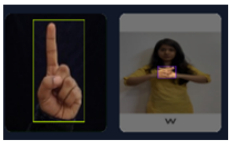
Arabic Sentence:

	Name	Abstract	size	photo	Source
1	Arabic Sign Language	Arabic and English sentences .CSV	445 KB	sn_ar_en_with_letters_tokeniz	Kaggle

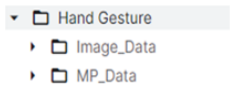
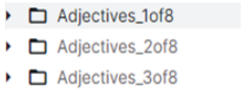
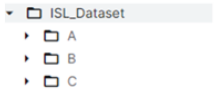
Spanish Sign Language (Alphabet and Words):

	Name	Abstract	size	photo	Source
1	Spanish Sign Language Alphabet (Static)	19 static letters and 8 with movement	5 GB	fondo_blanco A B C D	Kaggle
2	SpanishSignLanguageRecognitionTFG	50 signs / 130 video each Words	4 GB	df_16words df_32words df_50words df_8words	Kaggle

India Alphabet:

	Name	Abstract	size	photo	Source
1	Indian Sign Language Detection Computer Vision Project	0-9 a-z 1748image	7 GB		Roboflow
2	Indian Sign Language Translation Letters n Digits	0-9 23 Eng Letter	5 GB	0 1 2 3	Kaggle
3	indian sign language	A- Y Image	66 MB	A B C D	Kaggle

India words:

	Name	Abstract	size	photo	Source
1	Hand Gesture Recognition	50 words video	3 GB		Kaggle
2	INCLUDE - ISL	4286 file	57 GB		Kaggle
3	Indian Sign Language Dataset	A-Z	3 MB		Kaggle

India Sentence:

	Name	Abstract	size	photo	Source
1	ISL-CSLTR: Indian Sign Language Dataset for Continuous Sign Language Translation and Recognition	700 fully annotated videos, 18863 Sentence level frames, and 1036 word level images .	8GB		Google dataset

4.6 ASL Alphabet Dataset:

The dataset is a collection of images representing the alphabets of American Sign Language. It is divided into 29 folders, each representing a different class. The training dataset consists of 87,000 images, each with dimensions of 200x200 pixels. There are 29 classes in total, including the letters A-Z, as well as three additional classes for SPACE, DELETE, and NOTHING.

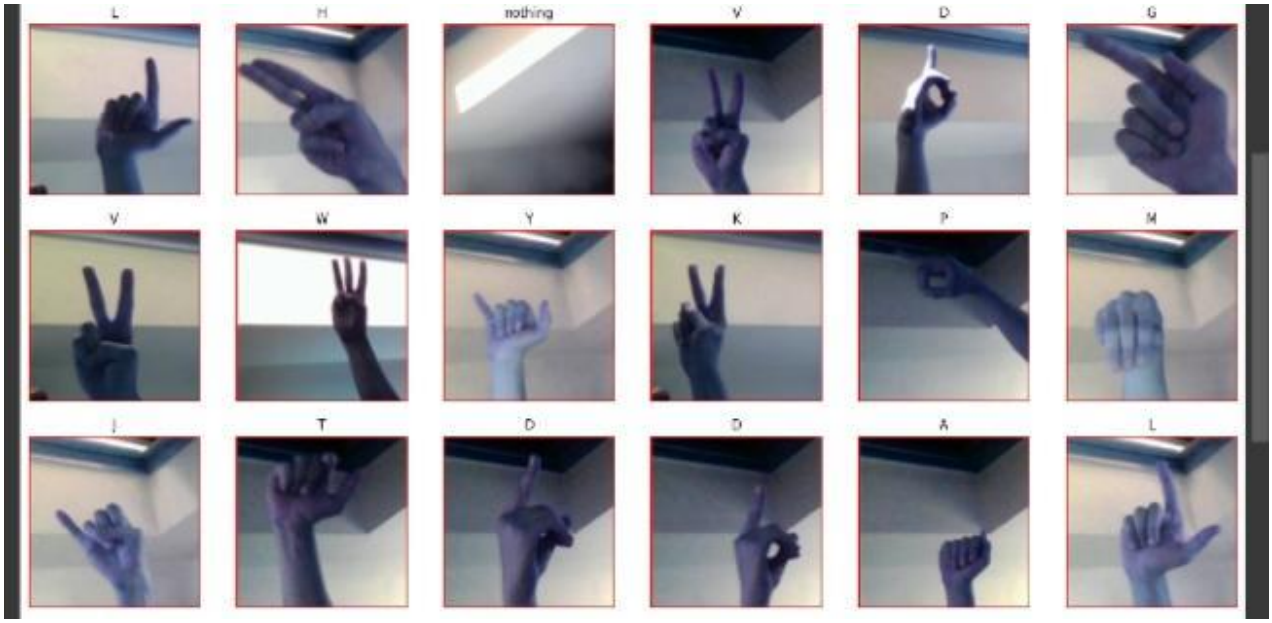


Figure 5 samples from the dataset.

4.6.1 Load Dataset:

We obtained a ready-made dataset from the Kaggle website. It was convenient to download the JSON file from the Kaggle website and upload it to Colab after creating an account on Kaggle. By accessing the dataset in this manner.

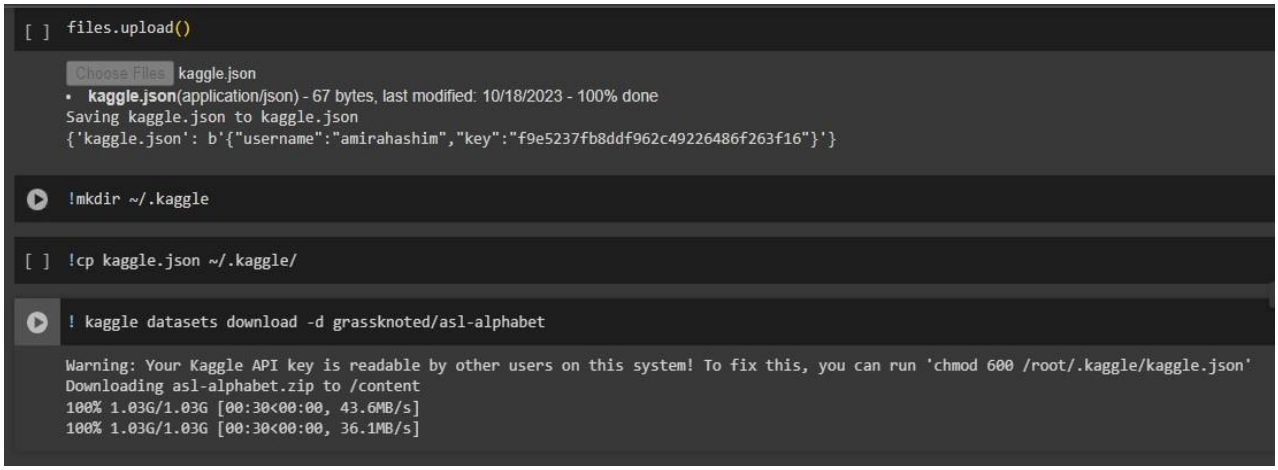


Figure 6 Data collection.

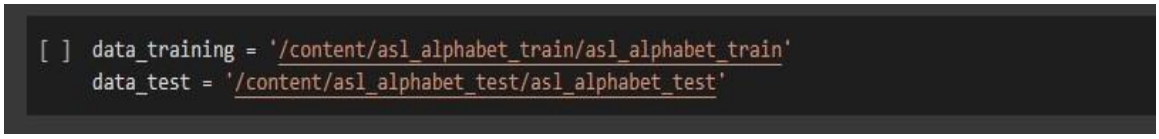


Figure 7 Load dataset.

Load the dataset from Kaggle to Colab, the training data is “data_training " and the test data is “data_test".

```
import os
import glob as gb
for folder in os.listdir(data_training):
    folder_path = os.path.join(data_training, folder)
    if os.path.isdir(folder_path):
        files = gb.glob(os.path.join(folder_path, '*.jpg'))
        print(f'For training data, found {len(files)} in folder {folder}')
```

Figure 8 Counting Images in Each Training Folder and Printing the Results

This code is a loop that scans the training folders in the data_training path. For each folder, the files in it are scanned using glob.glob(), which is used to find all files with the .jpg extension. The number of files in each folder is then printed, indicating the folder name. This can be useful to check how much data is available in each training folder. We found that each folder already contains 3000 images.

```
code = {'A':0, 'B':1, 'C':2, 'D':3, 'E':4, 'F':5, 'G':6, 'H':7, 'I':8, 'J':9, 'K':10, 'L':11, 'M':12, 'N':13, 'O':14,
        'P':15, 'Q':16, 'R':17, 'S':18, 'T':19, 'U':20, 'V':21, 'W':22, 'X':23, 'Y':24, 'Z':25, 'del':26,'nothing':27,
        'space':28}
def getcode(n) :
    for x , y in code.items() :
        if n == y :
            return x
```

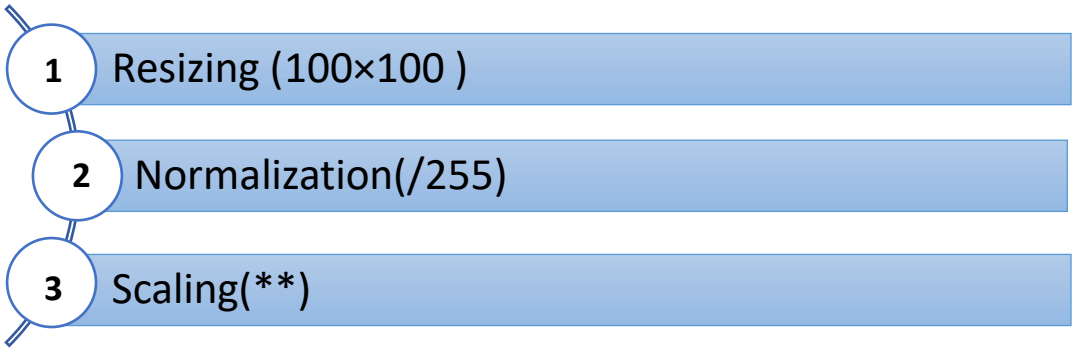
This code is a dictionary that converts letters and symbols into numbers, then provides a function to retrieve the letter corresponding to the specified number.

- code is a dictionary called, in which each letter in the English alphabet is assigned a numeric value, along with the additional codes "del", "nothing", and "space". It is considered a scheme for converting letters into their representative numbers.

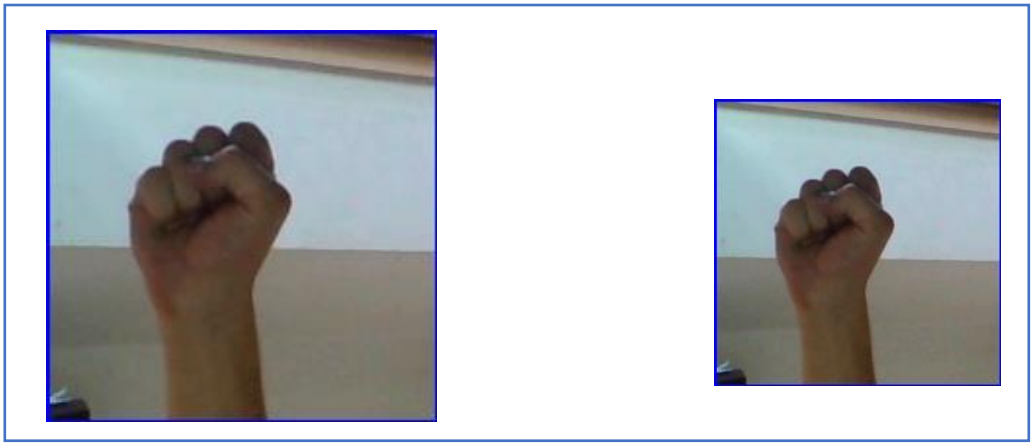
- The getcode() function checks the number passed as the parameter n, searches the code dictionary to find the character corresponding to that number, and returns it. Its purpose is to facilitate the conversion process between the representation of letters as symbols and numbers.

The code provides a means of assigning numbers to letters and symbols and their corresponding recall.

4.6.2 Preprocessing:



At the preprocessing stage, the resizing process, the results of this process produce an image size of 100×100 pixels, from the original image size of 200×200 pixels. This step is taken to reduce the time complexity during model training. The next preprocessing step is Normalization. To improve model performance and ease of training. Finally, Scaling To convert data into a color-compatible range from 0 to 255.



Original size 200×200, after resize 100×100.

```
import cv2
import numpy as np
s = 100
X_train = []
y_train = []
for folder in os.listdir(data_training):
    folder_path = os.path.join(data_training, folder)
    if os.path.isdir(folder_path):
        files = gb.glob(os.path.join(folder_path, '*.jpg'))
        for file in files:
            image = cv2.imread(file)
            info = np.iinfo(image.dtype)  # Get the information of the incoming image type
            image = image.astype(np.float64) / info.max  # normalize the data to 0 - 1
            image = 255 * image  # Now scale by 255
            image = image.astype(np.uint8)
            image_array = cv2.resize(image, (s, s))
            image_array = image_array.astype(np.uint8)
            X_train.append(list(image_array))
            y_train.append(code[folder])
```

Figure 9 Preprocessing Data

This code loads the images and prepares them appropriately for use in training the model. It preprocesses the images, and this code explains what we did in detail.

- It reads images from the training folders, and loads them into two lists, X_train and y_train, where the images are stored in X_train = [] and their corresponding tags are stored in y_train = [].

- The process starts with a ‘for’ loop that goes through all the directories within the data_training path. For each folder, glob.glob() is used to find all image files with ".jpg" extension. Each image is then loaded using OpenCV cv2.imread().

- image = cv2.imread(file): We load the image located in the path specified by the file variable using the cv2.imread() function that belongs to the OpenCV library. The data type of the image is then converted to np.float64 using astype().
- info = np.iinfo(image.dtype): Get the information of the incoming .image type

- image = image.astype(np.float64) / info.max: normalizes the pixel values of the image to a range between 0 and 1 by dividing by the maximum value of the image data type using np.iinfo().
- Then, image = 255 * image: it scales the pixel values back to the standard range between 0 and 255.

- image = image.astype(np.uint8): Then the data type of the image is converted to np.uint8 using astype() again.
- image_array = cv2.resize(image, (s, s)): The image is resized using cv2.resize() to be compatible with the required size (specified by s), which is equal to 100.

- Finally, X_train.append(list(image_array)): the processed image is added to the X_train list containing image data, and y_train.append(code[folder]): the label corresponding to this image is added to the y_train list.

4.6.3 Split Dataset:

The data set was divided into 80% for training and 20% for testing, as the training data contains 87,000, which was divided into these proportions so that we apply training to the data and then test it. and shuffle=True, random_state=100.

```
from sklearn.model_selection import train_test_split
import numpy as np

# Split the data into 80% for training and 20% for validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle=True, random_state=100)

X_train = np.array(X_train)
X_val = np.array(X_val)
y_train = np.array(y_train)
y_val = np.array(y_val)
```

Figure 10 Split dataset.

This code splits the dataset into training and validation sets using the Train_test_split function from the scikit-Learn library. Here is a detailed explanation of the code:

Splitting the Data: The train_test_split function is called to split the dataset into training and validation sets. It takes four main arguments:

- X_train and y_train: The features and labels of the dataset.
- test_size: The proportion of the dataset to include in the validation split, in this case, is 20%.

- Shuffle: Whether to shuffle the dataset before splitting, set to True.
- random_state: Seed used by the random number generator for shuffling the data.

Converting to Numpy Arrays: Convert the resulting lists X_train, X_val, y_train, and y_val to numpy arrays using np.array().

This code divides the dataset into two subsets: one to train the model (X_train and y_train) and one to validate its performance (X_val and y_val). It involves training the model on a piece of data and then evaluating it on unseen data to evaluate its ability to generalize.

4.6.4 Training Data:

At this stage, the design of the CNN model used in the training process is implemented to produce a suitable model for classifying hand sign language images.

After preprocessing the data and ensuring that it is ready to enter the model so that he can conduct training on it and learn the model from it, this stage is based on experience and scientific concepts, as we build a model consisting of a number of deep layers that contain the hyperparameters so that you can extract all The features of the input images are considered one of the most important stages of

the project, and it may take time to modify and adjust the value of the upper parameters until we reach the highest accuracy of the blending, and we will explain all of this in detail.

The implemented CNN model uses hyperparameter values consisting of learning rate, epoch, loss function and optimizer. Table 1 is the specifications of the CNN architecture used in the project.

4.6.5 CNN Model:

Layer type	Layer type Description	Size
Input Layer	Input Image	3×100×100
Conv 1	Convolutional ReLU MaxPooling	32 kernels, 3x3 Window size 2×2
Conv 2	Convolutional ReLU MaxPooling	64 kernels, 5x5 Window size 2×2
Conv 3	Convolutional ReLU MaxPooling	128 kernels, 5x5 Window size 2×2
Conv 4	Convolutional ReLU MaxPooling	256 kernels, 3x3 Window size 2×2512
Flatten	Flatten	512
Fully Connected Layer1	Dense ReLU	512
Fully Connected Layer2	Dense ReLU Dense ReLU	265 182
Fully Connected Layer3		29
Output Layer	Dense SoftMax	

Figure 11 architectural model (table 1).

4.6.6 Hyperparameters:

- Number of convolutional layers: 4-layer cov2D.
- Filters (Kernel Depth).
- Activation Function (relu, SoftMax).
- Pooling (Max).
- Stride Size.
- Kernel Size (Filter Size).
- Epochs' Number (25).
- Optimization Technique (Optimizer)(Adam).
- Batch Size (64)
- Flatten and Fully Connected Layers.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

KerasModel = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(s, s, 3)),
    MaxPooling2D(2, 2),

    Conv2D(64, kernel_size=(5, 5), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, kernel_size=(5, 5), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(256, kernel_size=(3, 3), activation='relu'),

    Flatten(),
    Dense(512, activation='relu'),

    Dense(256, activation='relu'),

    Dense(128, activation='relu'),
    Dense(29, activation='softmax'),
])
```

Figure 12 build the CNN model by Keras, using Conv2D layers, MaxPooling & Denses.

Here we will explain the layers of the model, which contains four layers of CNN (conv2D) and three layers of Fully Connected:

- Conv2D: This layer creates a convolution kernel that is convolved with the input layer to produce a tensor of outputs. Here, you have four convolutional layers with increasing numbers of filters (32, 64, 128, and 256). Each layer uses a ReLU activation function.

- The input layer contains the image size and number of channels, and since the images are RGB, it consists of three channels.

- Kernel Size: The kernel size refers to the dimensions of the filter that is applied to the input data. In the case of images, the kernel size typically specifies a square filter represented by (height, width).
  a kernel size of (3, 3) means that the filter is a 3x3 matrix. During the convolution operation, this 3x3 filter slides over the input image, computing dot products with the underlying image patches.

  The choice of kernel size depends on various factors, including the complexity of the features to be detected and the size of the input images. Smaller kernel sizes capture fine-grained details, while larger kernel sizes capture more global features. Here we used kernel size (3, 3) and (5,5)

- Filters: In a Conv2D layer, the term "filters" refers to the number of unique convolutional kernels applied to the input data. Each filter learns to detect specific patterns or features in the input data. These patterns could represent edges, textures, shapes, or higher-level semantic features depending on the depth of the network.
  Increasing the number of filters allows the network to learn a richer set of features but also increases the computational complexity of the model.

- MaxPooling2D: This layer performs max pooling operation for spatial data. It reduces the spatial dimensions of the output volume. After each convolutional layer, there is a max-pooling

layer to downs ample the feature. The provided configuration uses a 2x2 pooling window.

- Flatten: This layer flattens the input, which is necessary to convert the 2D feature into a 1D vector before passing them to the fully connected layers.

- Fully connected layers come after the flattened layer. In this model, there are three fully connected layers.
  - The first layer has 512 neurons.
  - The second layer has 256 neurons.
  - The third layer has 128 neurons.
- The activation function used here is ReLU, which is used to enhance the learning process.

- The last layer is the output layer, which has 29 neurons. The SoftMax function is used to generate the probability distribution of possible classes for the output. Which is suitable for multi-class classification tasks. The number 29 is the number of classes in classification.

```
[ ] import tensorflow as tf

    opt = tf.keras.optimizers.Adam()
    KerasModel.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 13 compile the model.

model.compile(...): This line compiles the model by specifying the optimizer, loss function, and evaluation metrics.

- opt = tf.keras.optimizers.Adam(): This line defines an Adam optimizer using TensorFlow's tf.keras.optimizers module. Adam is an optimization algorithm commonly used for training neural networks. The optimizer's settings, such as learning rate and momentum, can be customized by passing arguments to the Adam() constructor.

- KerasModel.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy']): This line compiles the Keras model.
- optimizer=opt: Specifies the optimizer to use during training. Here, it's set to the Adam optimizer that we defined earlier.

- loss='sparse_categorical_crossentropy': Defines the loss function to be used during training. 'sparse_categorical_crossentropy' is suitable for classification tasks with integer labels.

- metrics=['accuracy']: Specifies the evaluation metric to monitor during training. In this case, it's accuracy, which measures the proportion of correctly classified images.

- After compiling the model with the optimizer, loss function, and metrics, it's ready to be trained using the specified configurations.

```
Model Details are :
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        896

 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 45, 45, 64)        51264

 max_pooling2d_1 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 18, 18, 128)       204928

 max_pooling2d_2 (MaxPoolin  (None, 9, 9, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 7, 7, 256)         295168

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 512)               6423040

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 29)                3741

=================================================================
Total params: 7143261 (27.25 MB)
Trainable params: 7143261 (27.25 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 14 Summary of the model 1 structure.

4.6.7 Trainig Model:

```
epochs = 25
ThisModel = KerasModel.fit(X_train, y_train,
                           epochs=epochs,
                           batch_size=64,
                           validation_data=(X_val, y_val),
                           verbose=1,
                           shuffle=False)
```

Figure 15 train the model, use 25 epochs.

This code performs the training of the Keras model:

- KerasModel.fit(): This method starts the training process of the model.

- X_train and y_train: These are the training data (input features and corresponding labels) used to train the model.

- epochs=epochs: Specifies the number of epochs for which the model will be trained. In this case, it's set to 25 epochs as defined earlier.

- batch_size=64: Defines the batch size used for training. Training data is divided into batches, and the model's parameters are updated after processing each batch. Here, each batch contains 64 samples.

- validation_data=(X_val, y_val): Specifies the validation data (input features and labels) used to evaluate the model's performance after each epoch. This helps monitor the model's generalization capability and detect overfitting.

- verbose=1: Controls the verbosity mode during training. Setting it to 1 prints progress bars showing the training progress for each epoch.

- shuffle=False: Indicates whether to shuffle the training data before each epoch. In this case, the data is not shuffled. Shuffling the data can help prevent the model from memorizing the order of the training examples and improve generalization.

- After training, the method returns a history object (ThisModel), which contains information about the training process, such as loss and accuracy metrics for each epoch.

```
Epoch 8/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0567 - accuracy: 0.9862 - val_loss: 0.1099 - val_accuracy: 0.9816
Epoch 9/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0510 - accuracy: 0.9872 - val_loss: 0.0208 - val_accuracy: 0.9944
Epoch 10/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0428 - accuracy: 0.9895 - val_loss: 0.0786 - val_accuracy: 0.9838
Epoch 11/25
1088/1088 [==============================] - 27s 24ms/step - loss: 0.0563 - accuracy: 0.9867 - val_loss: 0.0153 - val_accuracy: 0.9955
Epoch 12/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0527 - accuracy: 0.9880 - val_loss: 0.0369 - val_accuracy: 0.9897
Epoch 13/25
1088/1088 [==============================] - 30s 27ms/step - loss: 0.0443 - accuracy: 0.9900 - val_loss: 0.1328 - val_accuracy: 0.9745
Epoch 14/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0266 - accuracy: 0.9942 - val_loss: 0.0095 - val_accuracy: 0.9973
Epoch 15/25
1088/1088 [==============================] - 27s 24ms/step - loss: 0.0552 - accuracy: 0.9887 - val_loss: 0.0735 - val_accuracy: 0.9826
Epoch 16/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0405 - accuracy: 0.9919 - val_loss: 0.0353 - val_accuracy: 0.9920
Epoch 17/25
1088/1088 [==============================] - 32s 29ms/step - loss: 0.0375 - accuracy: 0.9927 - val_loss: 0.0441 - val_accuracy: 0.9925
Epoch 18/25
1088/1088 [==============================] - 27s 25ms/step - loss: 0.0374 - accuracy: 0.9925 - val_loss: 0.0296 - val_accuracy: 0.9940
Epoch 19/25
1088/1088 [==============================] - 28s 26ms/step - loss: 0.0551 - accuracy: 0.9900 - val_loss: 0.0516 - val_accuracy: 0.9930
Epoch 20/25
1088/1088 [==============================] - 31s 29ms/step - loss: 0.0270 - accuracy: 0.9950 - val_loss: 0.0476 - val_accuracy: 0.9896
Epoch 21/25
1088/1088 [==============================] - 28s 26ms/step - loss: 0.0409 - accuracy: 0.9923 - val_loss: 0.0153 - val_accuracy: 0.9967
Epoch 22/25
1088/1088 [==============================] - 29s 26ms/step - loss: 0.0332 - accuracy: 0.9940 - val_loss: 0.0148 - val_accuracy: 0.9970
Epoch 23/25
1088/1088 [==============================] - 32s 30ms/step - loss: 0.0537 - accuracy: 0.9908 - val_loss: 0.1218 - val_accuracy: 0.9759
Epoch 24/25
1088/1088 [==============================] - 33s 30ms/step - loss: 0.0327 - accuracy: 0.9944 - val_loss: 0.0171 - val_accuracy: 0.9960
Epoch 25/25
1088/1088 [==============================] - 30s 27ms/step - loss: 0.0576 - accuracy: 0.9904 - val_loss: 0.0289 - val_accuracy: 0.9945
```

Figure 16 show the final loss & accuracy when use 25 epochs.

4.6.8 Evaluation:

Here, if the result of the evaluation was bad, you start to see where. your problems were exactly, modify the parameters of the Dataset, and do Training more than once to improve the Model.

But it is as shown in the screenshot taken from our model, which is that the model Ours works very well and there are no errors.

```
[ ] # Evaluate the model on the train data
    final_train_loss = ThisModel.history['loss'][-1]
    final_train_accuracy = ThisModel.history['accuracy'][-1]

    print("Final Train Loss:", final_train_loss)
    print("Final Train Accuracy:", final_train_accuracy)

    Final Train Loss: 0.057629186660051346
    Final Train Accuracy: 0.990431010723114
```

Figure 17 Evaluate the model on the training data.

```
# Evaluate the model on the validation data
final_val_loss = ThisModel.history['val_loss'][-1]
final_val_accuracy = ThisModel.history['val_accuracy'][-1]

print("Final Validation Loss:", final_val_loss)
print("Final Validation Accuracy:", final_val_accuracy)

Final Validation Loss: 0.028888490051031113
Final Validation Accuracy: 0.9945402145385742
```

Figure 18 Evaluate the model on the validation data.

After training the model, its training accuracy is 99% and validation accuracy is 99.5%.

```
plt.plot(ThisModel.history['loss'], 'r', label='Training loss')
plt.plot(ThisModel.history['val_loss'], 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.show()
```

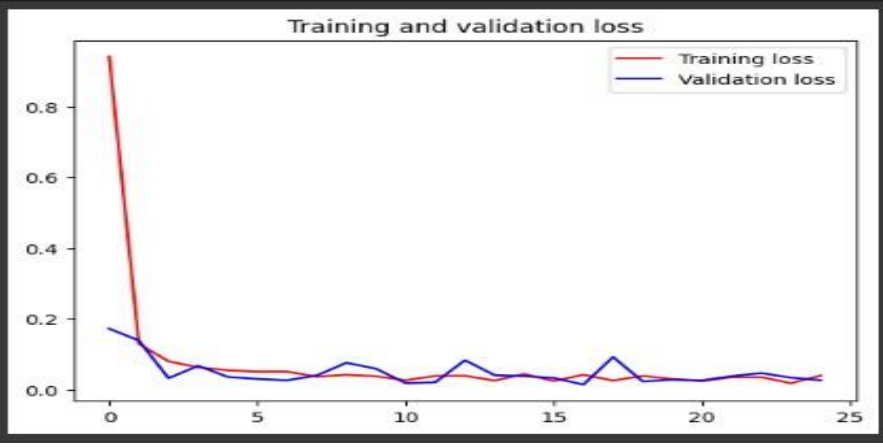


Figure 19 Visualize training result (loss)

```
acc = ThisModel.history['accuracy']
val_acc = ThisModel.history['val_accuracy']
loss = ThisModel.history['loss']
val_loss = ThisModel.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()
plt.show()
```

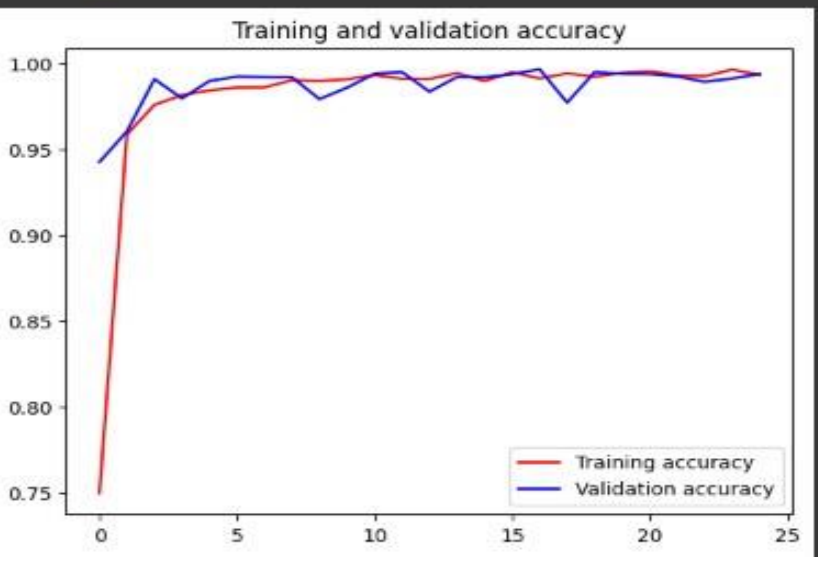


Figure 20 Visualize training result (accuracy)

Here, if the result of the evaluation was bad, you start to see where your problems were exactly, modify the parameters of the Dataset, and do Training more than once to improve the Model.

But it is as shown in the screenshot taken from our model, which is that the model Ours works very well and there are no errors.

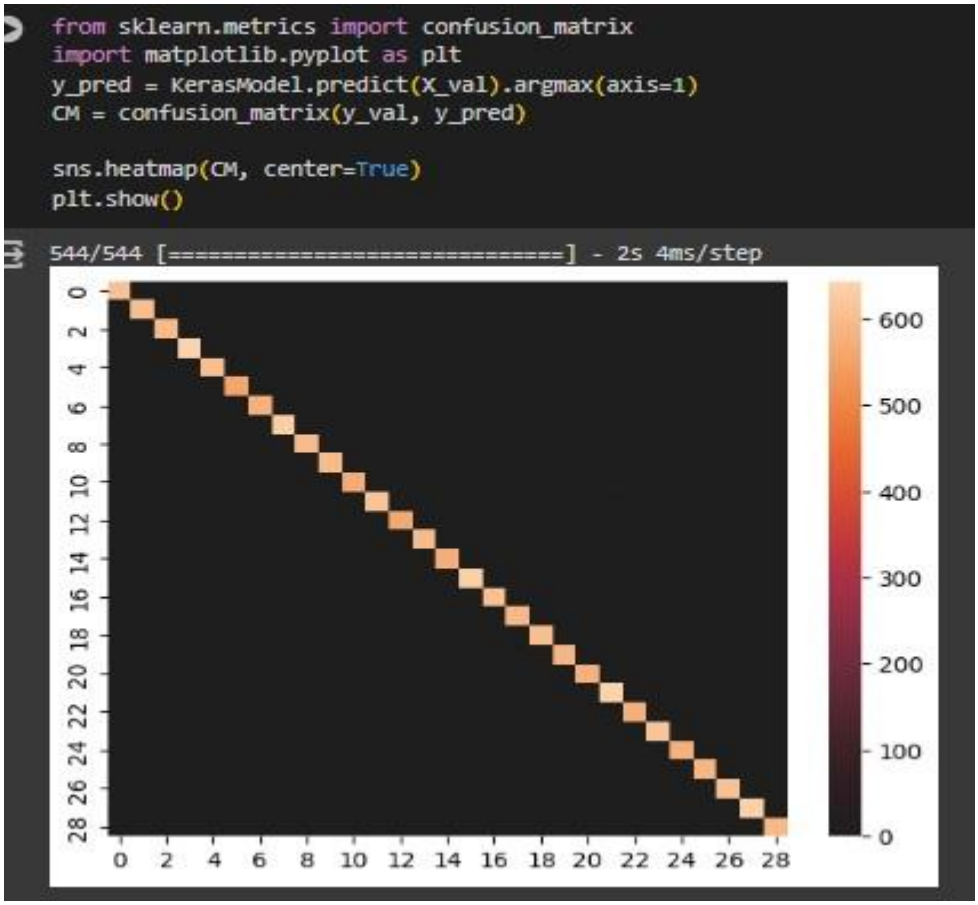


Figure 21Confusion Matrix of The Model Testing

"The provided code evaluates the performance of the Keras model on the validation set by generating a confusion matrix, which depicts

the distribution of true and predicted class labels, and then visualizes the confusion matrix using a heatmap."

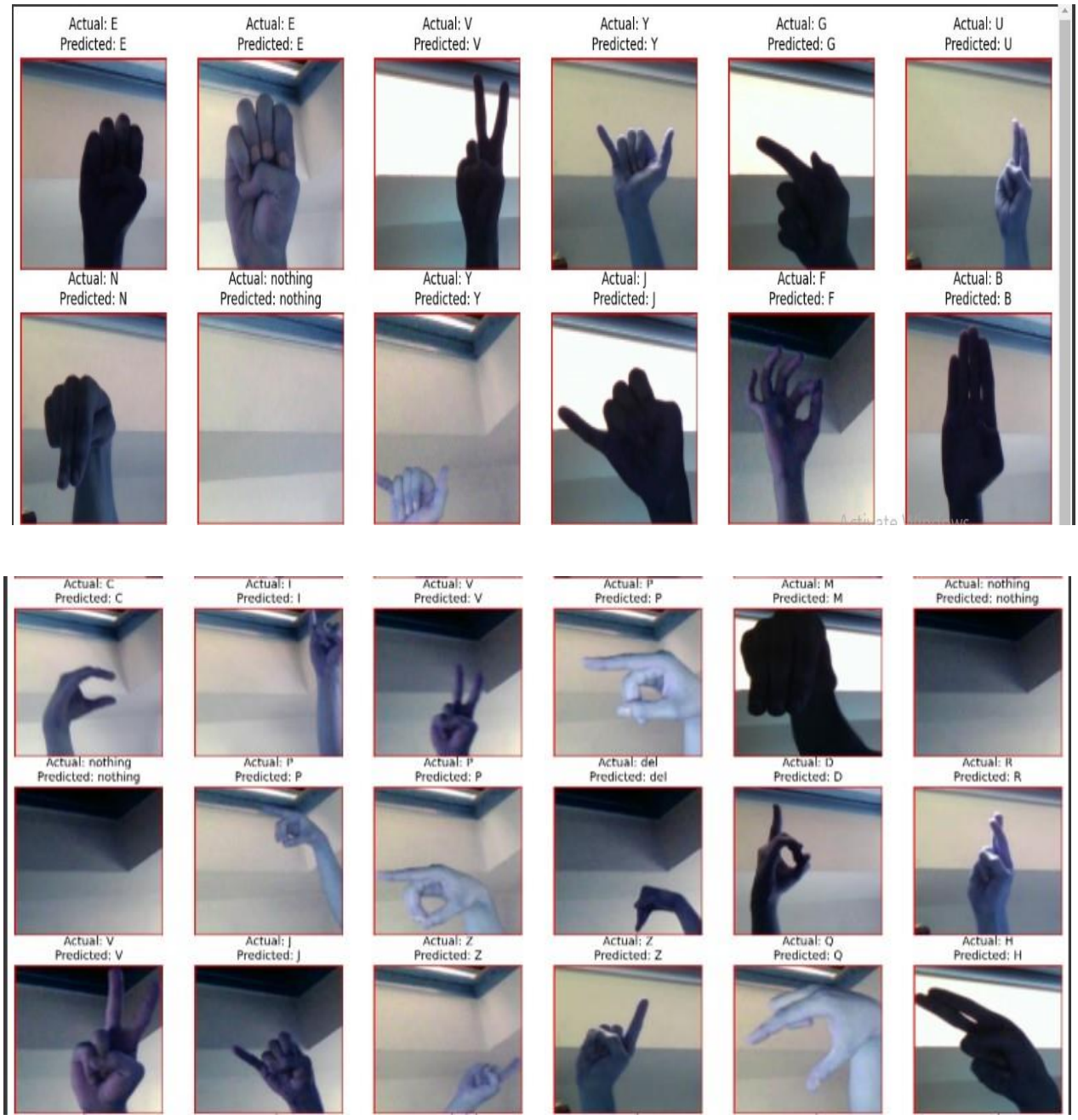


Figure 22 Comparison between the actual value and the expected value.

4.6.9 Prediction:

We made a prediction for a group of images to test the accuracy of the model in recognizing sign language, and it was discovered that the model predicted it correctly.

As we tested the model on a group of images consisting of 51 images, from each classification we took an image to test the model on. These images had not been seen by the model before during the training or verification phase, and the model was able to predict them very correctly. These images illustrate the model's .prediction

```
plt.figure(figsize=(13,8))
for i in range(0,len(test)):
    plt.subplot(4,7,i+1)
    plt.imshow(test[i])
    plt.axis('off')
    plt.title(getcode(y_pred_test[i]))# Predict testing data
y_pred_test = KerasModel.predict(np.array(test)).argmax(axis=1)
```

Figure 23 Predict data.

This code explains how prediction works:

- plt.figure(figsize=(13,8)): Creates a new figure for displaying the images with a specified size of 13x8 inches.
- 'for i in range(0,len(test)):' :Iterates through the images in the test set.
- plt.subplot(4,7,i+1): Specifies the subplot in the created figure to display the current image. It defines a grid of 4 rows and 7 columns, and uses the index i+1 to position the image in the grid.

- plt.imshow(test[i]): Displays the current image from the test set.
- plt.axis('off'): Turns off the axis display for the image.

- plt.title(getcode(y_pred_test[i])): Displays the prediction for the current image as the title of the image. It is assumed that the getcode() function returns the appropriate label for each prediction.

- y_pred_test=KerasModel.predict(np.array(test)).argmax(axis =1): Uses the model to predict the classes for the test set test, then extracts the predicted class for each image using argmax(axis=1).

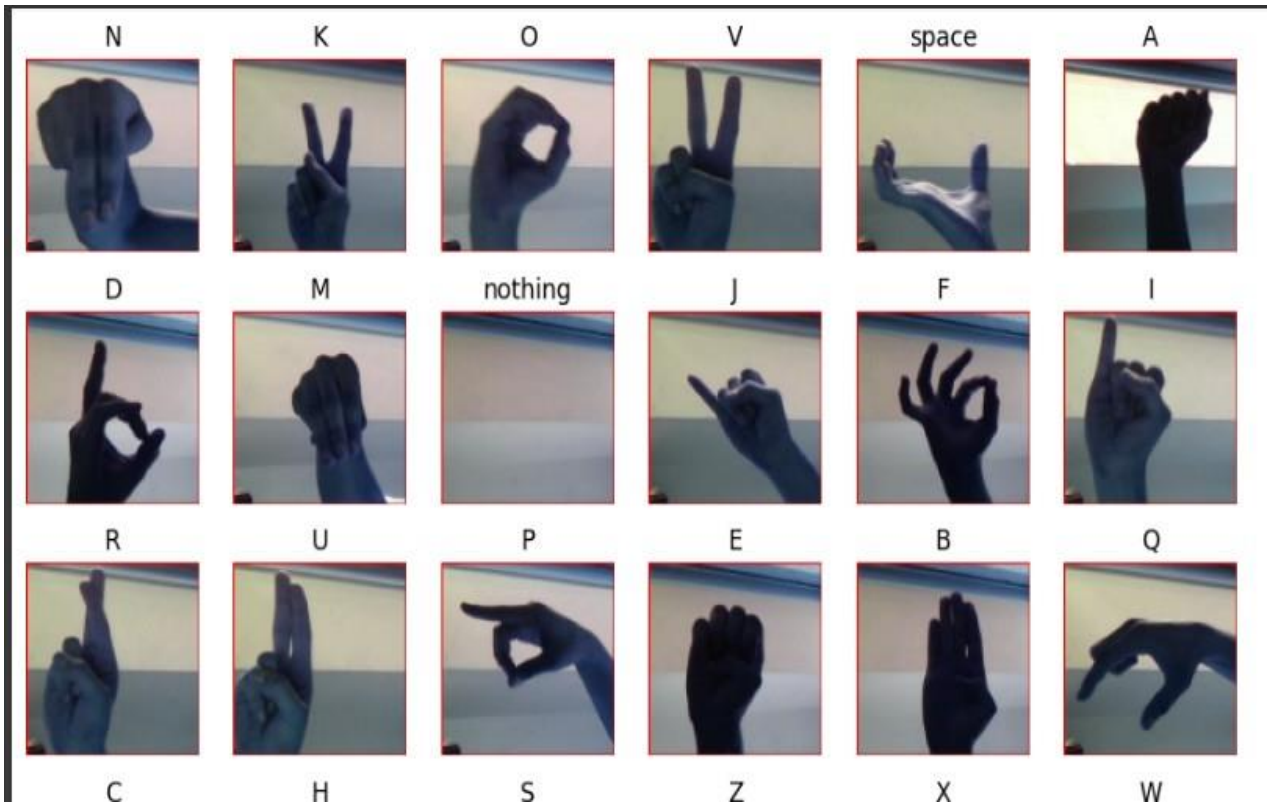


Figure 24 Prediction result.

4.7 ASL_and_same_words

The dataset is a collection of images of Letters (a, b, c, …, z), Words (Help, Like, Brother, Baby, …) and Numbers (1, 2, 3, …, 9) from the American Sign Language, separated 203000 images in 51 folders, folder have 4,000 image, Common images in data sets are 200 x 200 pixels, and there are images 100 x 100 and other images that differ in their dimensions. which represent the various classes.

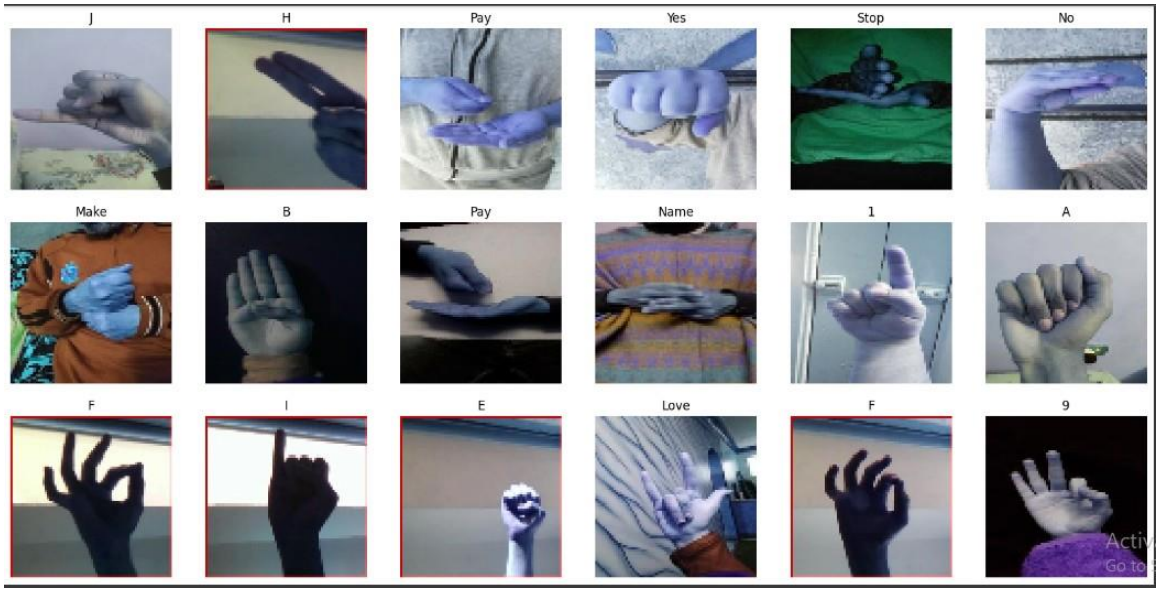


Figure 25 samples from the dataset.

4.7.1 Load Dataset:

We obtained a ready-made dataset from the Kaggle website. It was convenient to download the JSON file from the Kaggle website and

upload it to Colab after creating an account on Kaggle. By accessing the dataset in this manner. And test data from google drive.

Figure 26 Data collection.

Figure 27 Load data.

Load the dataset from Kaggle to colab, the training data is “data training " and the test data from google drive to colab is “data test".

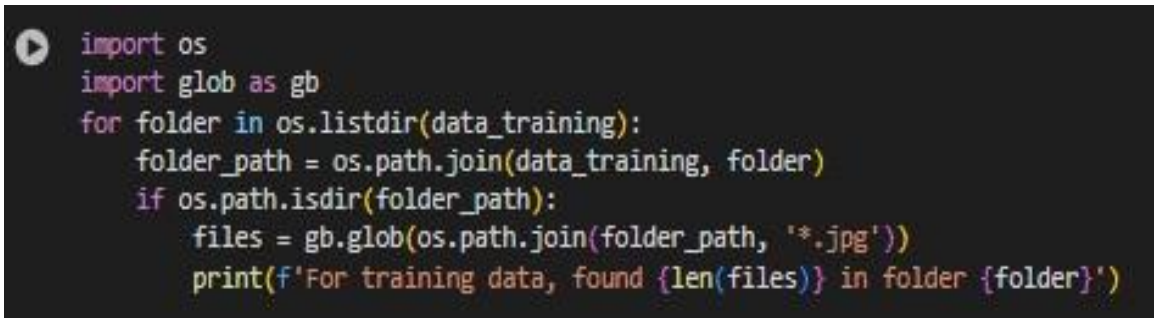


Figure 28 Counting Images in Each Training Folder and Printing the Results

This code is a loop that scans the training folders in the data_training path. For each folder, the files in it are scanned using glob.glob(), which is used to find all files with the .jpg extension. The number of files in each folder is then printed, indicating the folder name. This can be useful to check how much data is available in each training folder. We found that each folder already contains 4000 images.

```
training_dir=data_training
content=sorted(os.listdir(training_dir))
print(content)
len(content)
```

```
['1', '3', '4', '5', '7', '8', '9', 'A', 'B', 'Baby', 'Brother', 'C', 'D', 'Dont_like', 'E', 'F',
```

Figure 29 Preparing directories.

Here is a code snippet that reads the names of subdirectories, which contain images of hand signs, in the specified directory "training Dir".

It sorts the names alphabetically and prints the list of directory names along with the total number of directories, which is equivalent to the number of hand signs in the dataset.

```
code = {
    '1': 0, '3': 1, '4': 2, '5': 3, '7': 4, '8': 5, '9': 6, 'A': 7, 'B': 8, 'Baby': 9, 'Brother': 10, 'C': 11,
    'D': 12, 'Dont_like': 13, 'E': 14, 'F': 15, 'Friend': 16, 'G': 17, 'H': 18, 'Help': 19, 'House': 20, 'I': 21,
    'J': 22, 'K': 23, 'L': 24, 'Like': 25, 'Love': 26, 'M': 27, 'Make': 28, 'More': 29, 'N': 30, 'Name': 31, 'No': 32,
    'O_OR_0': 33, 'P': 34, 'Pay': 35, 'Play': 36, 'Q': 37, 'R': 38, 'S': 39, 'Stop': 40, 'T': 41, 'U': 42, 'V_OR_2': 43,
    'W_OR_6': 44, 'With': 45, 'X': 46, 'Y': 47, 'Yes': 48, 'Z': 49, 'nothing': 50
}

def getcode(n) :
    for x , y in code.items() :
        if n == y :
            return x
```

Figure 30 Mapping Characters to Numeric Codes.

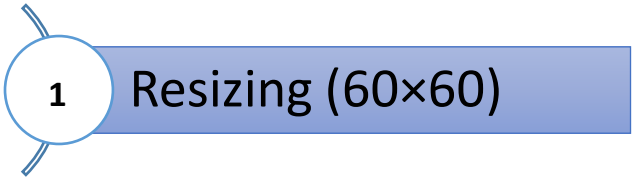
This code is a dictionary that converts letters and symbols into numbers, then provides a function to retrieve the letter corresponding to the specified number.

- code is a dictionary called, in which each letter in the English alphabet is assigned a numeric value, along with the additional codes "del", "nothing", and "space". It is considered a scheme for converting letters into their representative numbers.

- The getcode() function checks the number passed as the parameter n, searches the code dictionary to find the character corresponding to that number, and returns it. Its purpose is to facilitate the conversion process between the representation of letters as symbols and numbers.

The code provides a means of assigning numbers to letters and symbols and their corresponding recall.

4.7.2 Preprocessing:



At the preprocessing stage, the resizing process, the results of this process produce an image size of 60×60 pixels, from the original image that have different dimensions. This step is taken to reduce the time complexity during model training.

Original size 200×200, after resize 100×100.

```
s = 64
X_train = []
y_train = []
import os
import glob as gb
import cv2

for folder in os.listdir(data_training):
    folder_path = os.path.join(data_training, folder)
    if os.path.isdir(folder_path):
        files = gb.glob(os.path.join(folder_path, '*.jpg'))
    for file in files:
        image = cv2.imread(file)
        image_array = cv2.resize(image , (s,s))
        X_train.append(list(image_array))
        y_train.append(code[folder])
```

Figure 31 Preprocessing Data

This code loads the images and prepares them appropriately for use in training the model. It preprocesses the images, and this code explains what we did in detail.

- It reads images from the training folders, and loads them into two lists, X_train and y_train, where the images are stored in X_train = [] and their corresponding tags are stored in y_train = [].

- The process starts with a 'for' loop that goes through all the directories within the data_training path. For each folder, glob.glob() is used to find all image files with ".jpg" extension. Each image is then loaded using OpenCV cv2.imread().

- image = cv2.imread(file): We load the image located in the path specified by the file variable using the cv2.imread() function that belongs to the OpenCV library.

- image_array = cv2.resize(image, (s, s)): The image is resized using cv2.resize() to be compatible with the required size (specified by s), which is equal to 64.

- Finally, X_train.append(list(image_array)): the processed image is added to the X_train list containing image data, and y_train.append(code[folder]): the label corresponding to this image is added to the y_train list.

```
X_train = np.array(X_train)
y_train = np.array(y_train)

print(f'X_train shape  is {X_train.shape}')
print(f'y_train shape  is {y_train.shape}')
```

Converting to Numpy Arrays: Convert the resulting lists X_train, X_val, y_train, and y_val to numpy arrays using np.array().

4.7.3 Split Dataset:

The data set was divided into 80% for training and 20% for testing, as the training data contains 200,300, which was divided into these proportions so that we apply training to the data and then test it. and shuffle=True, random_state=100.

```
from sklearn.model_selection import train_test_split

# Split the data into 80% for training and 20% for validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle=True, random_state=100)
```

Figure 32 Split dataset.

This code splits the dataset into training and validation sets using the Train_test_split function from the scikit-Learn library. Here is a detailed explanation of the code:

Splitting the Data: The train_test_split function is called to split the dataset into training and validation sets. It takes four main arguments:

- X_train and y_train: The features and labels of the dataset.
- test_size: The proportion of the dataset to include in the validation split, in this case, is 20%.
- Shuffle: Whether to shuffle the dataset before splitting, set to True.
- random_state: Seed used by the random number generator for shuffling the data.

This code divides the dataset into two subsets: one to train the model (X_train and y_train) and one to validate its performance (X_val and y_val). It involves training the model on a piece of data and then evaluating it on unseen data to evaluate its ability to generalize.

4.7.4 CNN Model:

At this stage, the design of the CNN model used in the training process is implemented to produce a suitable model for classifying hand sign language images.

After preprocessing the data and ensuring that it is ready to enter the model so that he can conduct training on it and learn the model from it, this stage is based on experience and scientific concepts, as we build a model consisting of a number of deep layers that contain the hyperparameters so that you can extract all The features of the input images are considered one of the most important stages of the project, and it may take time to modify and adjust the value of the upper parameters until we reach the highest accuracy of the blending, and we will explain all of this in detail.

The implemented CNN model uses hyperparameter values consisting of learning rate, epoch, loss function and optimizer.

Table 2 is the specifications of the CNN architecture used in the project.

CNN Architecture:

Layer type	Layer type Description	Size
Input Layer	Input Image	3×100×100
Conv 1	Convolutional ReLU MaxPooling	32 kernels, 3x3 Window size 2×2
Conv 2	Convolutional ReLU MaxPooling	64 kernels, 5x5 Window size 2×2
Conv 3	Convolutional ReLU MaxPooling	128 kernels, 5x5 Window size 2×2
Conv 4	Convolutional ReLU MaxPooling	256 kernels, 3x3 Window size 2×2512
Flatten	Flatten	512
Fully Connected Layer1	Dense ReLU	512
Fully Connected Layer2	Dense ReLU	265
Fully Connected Layer3	Dense ReLU	182
Output Layer	Dense SoftMax	51

Figure 33 architectural model (table 2).

4.7.5 Hyperparameters:

Number of convolutional layers: 4-layer cov2D.

Filters (Kernel Depth).

Activation Function (relu, SoftMax).

Pooling (Max).

Stride Size.

Kernel Size (Filter Size).

Epochs' Number (25).

Optimization Technique (Optimizer)(Adam).

Batch Size (64)

Flatten and Fully Connected Layers.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

KerasModel = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(s, s, 3)),
    MaxPooling2D(2, 2),

    Conv2D(64, kernel_size=(5, 5), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, kernel_size=(5, 5), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(256, kernel_size=(3, 3), activation='relu'),

    Flatten(),
    Dense(512, activation='relu'),

    Dense(256, activation='relu'),

    Dense(128, activation='relu'),
    Dense(29, activation='softmax'),
])
```

Figure 34 build the CNN model by Keras, using Conv2D layers, MaxPooling & Denses.

Here we will explain the layers of the model, which contains four layers of CNN (conv2D) and three layers of Fully Connected:

- Conv2D: This layer creates a convolution kernel that is convolved with the input layer to produce a tensor of outputs. Here, you have four convolutional layers with increasing numbers of filters (32, 64, 128, and 256). Each layer uses a ReLU activation function.

- The input layer contains the image size and number of channels, and since the images are RGB, it consists of three channels.

- Kernel Size: The kernel size refers to the dimensions of the filter that is applied to the input data. In the case of images, the kernel size typically specifies a square filter represented by (height, width).
  a kernel size of (3, 3) means that the filter is a 3x3 matrix. During the convolution operation, this 3x3 filter slides over the input image, computing dot products with the underlying image patches.

  The choice of kernel size depends on various factors, including the complexity of the features to be detected and the size of the input images. Smaller kernel sizes capture fine-grained details, while larger kernel sizes capture more global features. Here we used kernel size (3, 3) and (5,5)

- Filters: In a Conv2D layer, the term "filters" refers to the number of unique convolutional kernels applied to the input

data. Each filter learns to detect specific patterns or features in the input data. These patterns could represent edges, textures, shapes, or higher-level semantic features depending on the depth of the network.
Increasing the number of filters allows the network to learn a richer set of features but also increases the computational complexity of the model.

- MaxPooling2D: This layer performs max pooling operation for spatial data. It reduces the spatial dimensions of the output volume. After each convolutional layer, there is a max-pooling layer to downs ample the feature. The provided configuration uses a 2x2 pooling window.

- Flatten: This layer flattens the input, which is necessary to convert the 2D feature into a 1D vector before passing them to the fully connected layers.

- Fully connected layers come after the flattened layer. In this model, there are three fully connected layers.
  - The first layer has 512 neurons.
  - The second layer has 256 neurons.
  - The third layer has 128 neurons.
- The activation function used here is ReLU, which is used to enhance the learning process.

- The last layer is the output layer, which has 29 neurons. The SoftMax function is used to generate the probability distribution of possible classes for the output. Which is

suitable for multi-class classification tasks. The number 51 is the number of classes in classification.

```
[ ] import tensorflow as tf

    opt = tf.keras.optimizers.Adam()
    KerasModel.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 35 compile the model, using adam optimizer, & sparse categorical crossentropy loss.

model.compile(...): This line compiles the model by specifying the optimizer, loss function, and evaluation metrics.

- opt = tf.keras.optimizers.Adam(): This line defines an Adam optimizer using TensorFlow's tf.keras.optimizers module. Adam is an optimization algorithm commonly used for training neural networks. The optimizer's settings, such as learning rate and momentum, can be customized by passing arguments to the Adam() constructor.

- KerasModel.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy']): This line compiles the Keras model.
- optimizer=opt: Specifies the optimizer to use during training. Here, it's set to the Adam optimizer that we defined earlier.

- loss='sparse_categorical_crossentropy': Defines the loss function to be used during training. 'sparse_categorical_crossentropy' is suitable for classification tasks with integer labels.

- metrics=['accuracy']: Specifies the evaluation metric to monitor during training. In this case, it's accuracy, which measures the proportion of correctly classified images.

- After compiling the model with the optimizer, loss function, and metrics, it's ready to be trained using the specified configurations.

```
Model Details are :
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 64)        51264

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 9, 9, 128)         204928

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 2, 2, 256)         295168

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 512)               524800

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 51)                6579

=================================================================
Total params: 1247859 (4.76 MB)
Trainable params: 1247859 (4.76 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 36 Summary of the model structure

4.7.6 Training Model:

```
epochs = 15
ThisModel = KerasModel.fit(X_train, y_train,
                           epochs=epochs,
                           batch_size=64,
                           validation_data=(X_val, y_val),
                           verbose=1,
                           shuffle=False)
```

Figure 37 train the model, use 15 epochs.

This code performs the training of the Keras model:

- KerasModel.fit(): This method starts the training process of the model.
- X_train and y_train: These are the training data (input features and corresponding labels) used to train the model.
- epochs=epochs: Specifies the number of epochs for which the model will be trained. In this case, it's set to 25 epochs as defined earlier.
- batch_size=64: Defines the batch size used for training. Training data is divided into batches, and the model's parameters are updated after processing each batch. Here, each batch contains 64 samples.
- validation_data=(X_val, y_val): Specifies the validation data (input features and labels) used to evaluate the model's performance after each epoch. This helps monitor the model's generalization capability and detect overfitting.

- verbose=1: Controls the verbosity mode during training. Setting it to 1 prints progress bars showing the training progress for each epoch.
- shuffle=False: Indicates whether to shuffle the training data before each epoch. In this case, the data is not shuffled. Shuffling the data can help prevent the model from memorizing the order of the training examples and improve generalization.
- After training, the method returns a history object (ThisModel), which contains information about the training process, such as loss and accuracy metrics for each epoch.

```
Epoch 1/15
2538/2538 [==============================] - 39s 13ms/step - loss: 1.1359 - accuracy: 0.6797 - val_loss: 0.4627 - val_accuracy: 0.8619
Epoch 2/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.3893 - accuracy: 0.8863 - val_loss: 0.3641 - val_accuracy: 0.8921
Epoch 3/15
2538/2538 [==============================] - 29s 12ms/step - loss: 0.2895 - accuracy: 0.9179 - val_loss: 0.3786 - val_accuracy: 0.9019
Epoch 4/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.2488 - accuracy: 0.9323 - val_loss: 0.2932 - val_accuracy: 0.9217
Epoch 5/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.2219 - accuracy: 0.9405 - val_loss: 0.2644 - val_accuracy: 0.9326
Epoch 6/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.2052 - accuracy: 0.9472 - val_loss: 0.2107 - val_accuracy: 0.9462
Epoch 7/15
2538/2538 [==============================] - 29s 12ms/step - loss: 0.1896 - accuracy: 0.9518 - val_loss: 0.2811 - val_accuracy: 0.9326
Epoch 8/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.1885 - accuracy: 0.9540 - val_loss: 0.2241 - val_accuracy: 0.9543
Epoch 9/15
2538/2538 [==============================] - 29s 12ms/step - loss: 0.1911 - accuracy: 0.9552 - val_loss: 0.2373 - val_accuracy: 0.9490
Epoch 10/15
2538/2538 [==============================] - 32s 13ms/step - loss: 0.1885 - accuracy: 0.9562 - val_loss: 0.2155 - val_accuracy: 0.9540
Epoch 11/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.1751 - accuracy: 0.9602 - val_loss: 0.2950 - val_accuracy: 0.9408
Epoch 12/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.1989 - accuracy: 0.9568 - val_loss: 0.1709 - val_accuracy: 0.9646
Epoch 13/15
2538/2538 [==============================] - 29s 12ms/step - loss: 0.1983 - accuracy: 0.9581 - val_loss: 0.2235 - val_accuracy: 0.9556
Epoch 14/15
2538/2538 [==============================] - 32s 13ms/step - loss: 0.2153 - accuracy: 0.9562 - val_loss: 0.3286 - val_accuracy: 0.9365
Epoch 15/15
2538/2538 [==============================] - 30s 12ms/step - loss: 0.1789 - accuracy: 0.9623 - val_loss: 0.2015 - val_accuracy: 0.9605
```

Figure 38 show the final loss & accuracy when use 15 epochs.

```
plt.plot(ThisModel.history['loss'], 'r', label='Training loss')
plt.plot(ThisModel.history['val_loss'], 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.show()
```

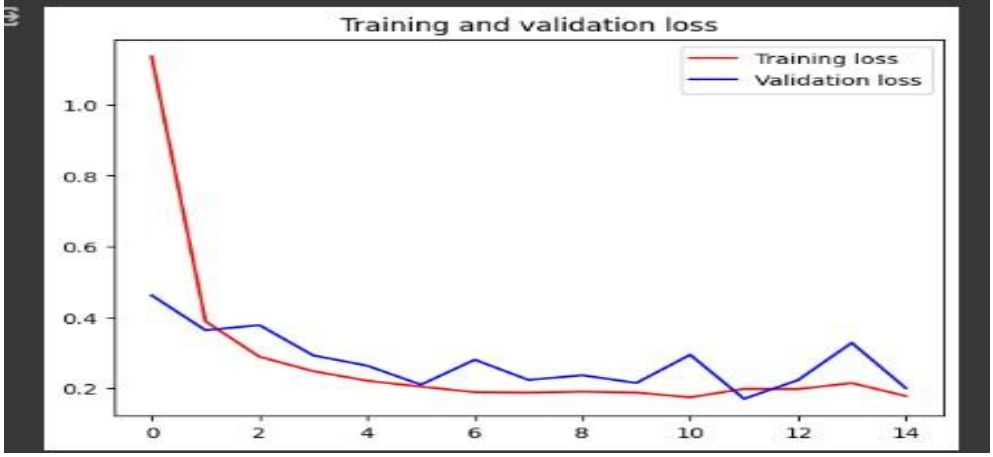


Figure 39 Visualize training result (loss)

```
acc = ThisModel.history['accuracy']
val_acc = ThisModel.history['val_accuracy']
loss = ThisModel.history['loss']
val_loss = ThisModel.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.show()
```

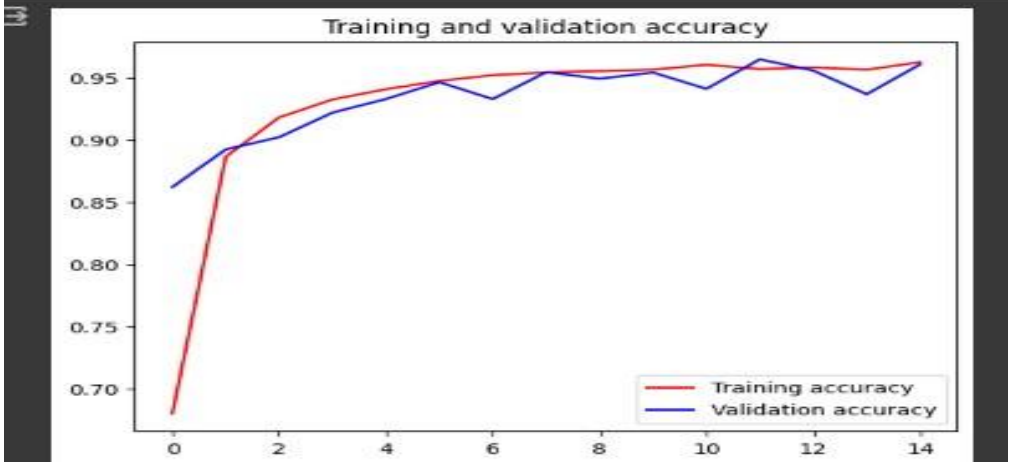


Figure 40 Visualize training result (accuracy)

Here, if the result of the evaluation was bad, you start to see where your problems were exactly, modify the parameters of the Dataset, and do Training more than once to improve the Model.

But it is as shown in the screenshot taken from our model, which is that the model Ours works very well and there are no errors.

4.7.7 Evaluation:

Here, if the result of the evaluation was bad, you start to see where. your problems were exactly, modify the parameters of the Dataset, and do Training more than once to improve the Model.
But it is as shown in the screenshot taken from our model, which is that the model Ours works very well and there are no errors.

```
# Evaluate the model on the train data
final_train_loss = ThisModel.history['loss'][-1]
final_train_accuracy = ThisModel.history['accuracy'][-1]

print("Final Train Loss:", final_train_loss)
print("Final Train Accuracy:", final_train_accuracy)
```

```
Final Train Loss: 0.17890425026416779
Final Train Accuracy: 0.9622783064842224
```

Figure 41 Evaluate the model on the training data.

```
# Evaluate the model on the validation data
final_val_loss, final_val_accuracy = KerasModel.evaluate(X_val, y_val)

print("Final Validation Loss:", final_val_loss)
print("Final Validation Accuracy:", final_val_accuracy)
```

```
1269/1269 [==============================] - 5s 4ms/step - loss: 0.2015 - accuracy: 0.9605
Final Validation Loss: 0.20148353278636932
Final Validation Accuracy: 0.9604679942131042
```

Figure 42 Evaluate the model on the validation data.

After training the model, its training accuracy is 96.23% and validation accuracy is 96%.

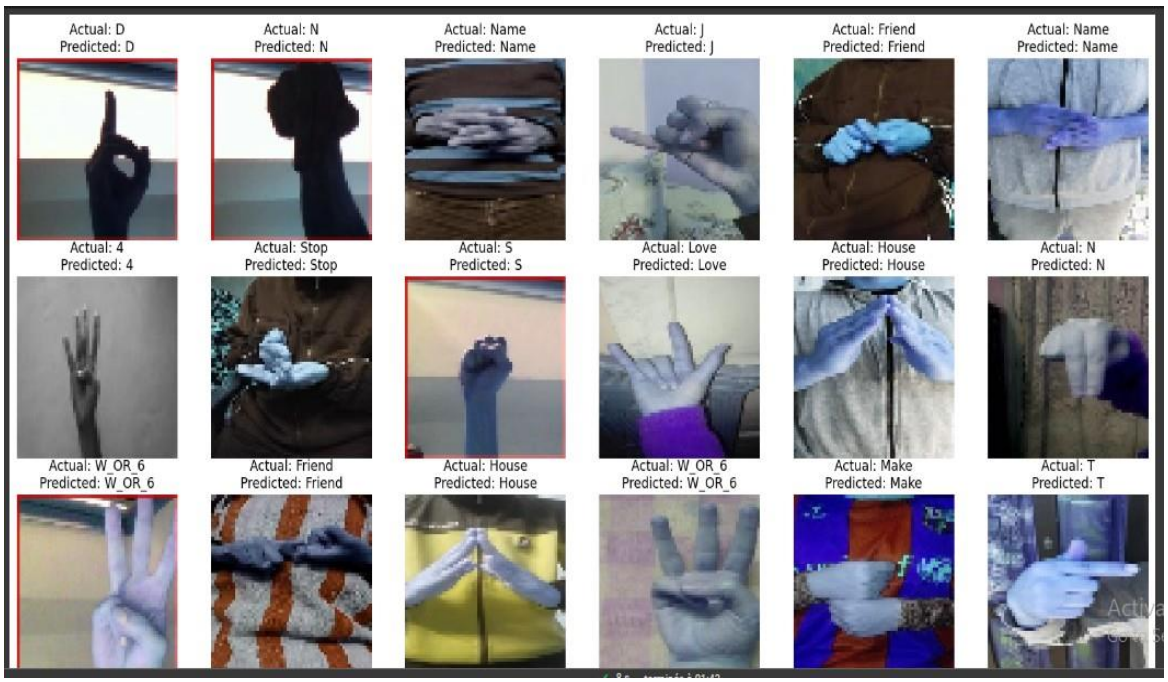


Figure 43 Comparison between the actual value and the expected value.

4.7.8 Prediction:

We made a prediction for a group of images to test the accuracy of the model in recognizing sign language, and it was discovered that the model predicted it correctly.

As we tested the model on a group of images consisting of 51 images, from each classification we took an image to test the model on. These images had not been seen by the model before during the training or verification phase, and the model was able to predict them very correctly. These images illustrate the model's prediction.

```
plt.figure(figsize=(20,20))
for i in range(0,len(test)):
    plt.subplot(10,10,i+1)
    plt.imshow(test[i])
    plt.axis('off')
    plt.title(getcode(y_pred_test[i]))# Predict testing data
y_pred_test = KerasModel.predict(np.array(test)).argmax(axis=1)
```

Figure 44 Predict data.

This code explains how prediction works:

- plt.figure(figsize=(20,20)): Creates a new figure for displaying the images with a specified size of 13x8 inches.
- 'for i in range(0,len(test)):' :Iterates through the images in the test set.
- plt.subplot(10,10,i+1): Specifies the subplot in the created figure to display the current image. It defines a grid of 4 rows and 7 columns, and uses the index i+1 to position the image in the grid.

- plt.imshow(test[i]): Displays the current image from the test set.

- plt.axis('off'): Turns off the axis display for the image.

- plt.title(getcode(y_pred_test[i])): Displays the prediction for the current image as the title of the image. It is assumed that the getcode() function returns the appropriate label for each prediction.

- y_pred_test=KerasModel.predict(np.array(test)).argmax(axis =1): Uses the model to predict the classes for the test set test, then extracts the predicted class for each image using argmax(axis=1).

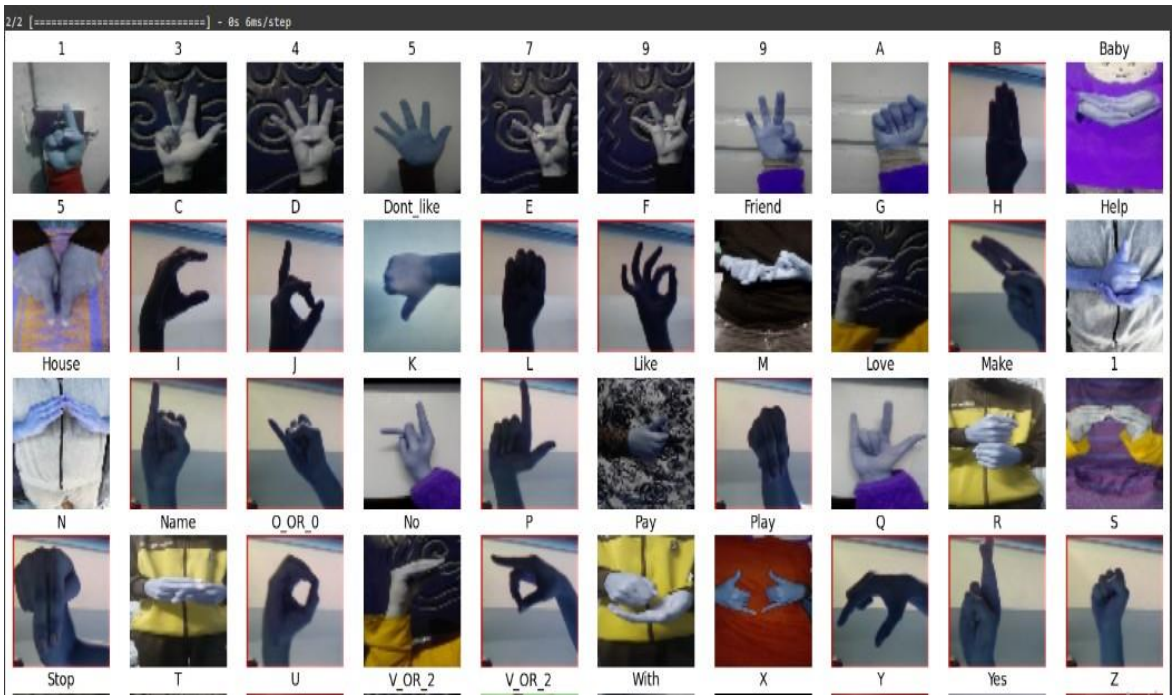


Figure 45 Prediction result.

4.8 Pre Train-Model

A pretrained model is a deep learning model someone else built that's trained on large datasets to accomplish a specific task and solve some problem, and it can be used as is or customized to suit application requirements across multiple industries.

Instead of building a model from scratch, developers can use pretrained models and customize them to meet their requirements. For example, you could re-purpose a deep learning model built to identify dog breeds to classify dogs and cats, instead of building your own.

This could save you the pain of finding an effective neural network architecture, the time you spend on training, the trouble of building a large corpus of training data and guarantee good results.

You could spend ages coming up with a fifty layered CNN for perfectly differentiating your cats from your dogs or you could simply re-purpose one of the many pre-trained image classification models available online.

There are mainly three different ways in which a pre-trained model can be re-purposed:

1-Feature extraction.

This mechanism is called a fixed feature extraction. They re-train only the new output layer they added and retain the heights of every other layer.

2-Copy the architecture of a pre-trained network.

They follow this approach when they have a large corpus of data to train on, but it isn't very similar to the data the pre-trained model was trained on.

3-Freeze some layers and train the others.

This approach is adopted when our dataset is small, and the data similarity is also low.

The other layers focus on the most basic information that can be extracted from the data and hence this could be used as it is on another problem, as often the basic level information would be the same.

4.8.1 ASL Alphabet Dataset:

The dataset is a collection of images representing the alphabets of American Sign Language. It is divided into 29 folders, each representing a different class. The training dataset consists of 87,000 images, each with dimensions of 200x200 pixels. There are 29 classes in total, including the letters A-Z, as well as three additional classes for SPACE, DELETE, and NOTHING.

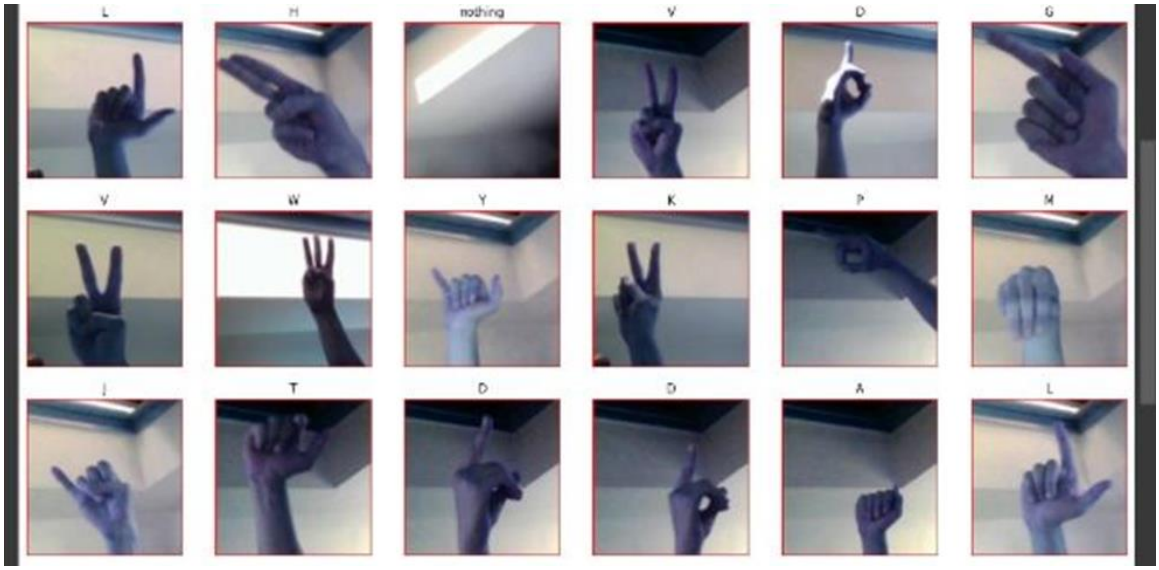


Figure 46 samples from the dataset.

4.8.2 Load dataset:

```
[ ] files.upload()

    Choose Files  kaggle.json
    • kaggle.json(application/json) - 67 bytes, last modified: 10/18/2023 - 100% done
    Saving kaggle.json to kaggle.json
    {'kaggle.json': b'{"username":"amirahashim","key":"f9e5237fb8ddf962c49226486f263f16"}'}

    !mkdir ~/.kaggle

[ ] !cp kaggle.json ~/.kaggle/

    ! kaggle datasets download -d grassknoted/asl-alphabet

    Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
    Downloading asl-alphabet.zip to /content
    100% 1.03G/1.03G [00:30<00:00, 43.6MB/s]
    100% 1.03G/1.03G [00:30<00:00, 36.1MB/s]
```

Figure 47 Data collection.

We obtained a ready-made dataset from the Kaggle website. It was convenient to download the JSON file from the Kaggle website and upload it to Colab after creating an account on Kaggle. By accessing the dataset in this manner.

```
training_dir="/content/asl_alphabet_train/asl_alphabet_train"
content=sorted(os.listdir(training_dir))
print(content)
len(content)
```

```
['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', 'del',
29
```

Figure 48 Preparing directories.

Here is a code snippet that reads the names of subdirectories, which. contain images of hand signs, in the specified directory "training Dir". It sorts the names alphabetically and prints the list of directory names along with the total number of directories, which is equivalent to the number of hand signs in the dataset.

4.8.3 Preprocessing:

Model (Inception V3):

```
data_generator = ImageDataGenerator(
    samplewise_center=True,
    samplewise_std_normalization=True,
    brightness_range=[0.8, 1.0],
    zoom_range=[1.0, 1.2],
    validation_split=0.1
)

train_generator = data_generator.flow_from_directory(training_dir, target_size=(200,200), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=64, subset="training")

validation_generator = data_generator.flow_from_directory(training_dir, target_size=(200, 200), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=64, subset="validation")
```

```
Found 78300 images belonging to 29 classes.
Found 8700 images belonging to 29 classes.
```

Figure 49 Inception V

The code snippet above is used to create data generators for image classification tasks using the Keras library in Python. These data generators are created using the ImageDataGenerator class, which enables data augmentation and preprocessing of image data. The generators are then utilized to feed data into a neural network for training and testing purposes.

Here is a detailed explanation of the code, line by line:

```
data_generator = ImageDataGenerator(
    samplewise_center=True,
    samplewise_std_normalization=True,
    brightness_range=[0.8, 1.0],
    zoom_range=[1.0, 1.2],
    validation_split=0.1
)
```

➢ The "ImageDataGenerator" is a class from Keras that generates batches of tensor image data with real-time data augmentation. In this code, an instance of this class is created and assigned to the variable "data generator".

➢ The "samplewise_center" parameter is set to True, indicating that the mean pixel value of each image in the batch will be subtracted from the image.

➢ Similarly, "samplewise_std_normalization" is set to True, which means that the pixel values of each image in the batch will be. divided by the standard deviation of all pixels in the image.

➢ The "brightness_range" parameter is set to [0.8, 1.0], resulting in

random adjustments to the brightness of each image between

80% and 100% of its original brightness.

➢ The zoom_range parameter is set to [1.0, 1.2], enabling random

zooming in or out of each image by a factor between 1.0 and

1.2.

➢ Lastly, the testing_split parameter is set to 0.1, indicating that

10% of the training data will be allocated for testing purposes.

```
)

train_generator = data_generator.flow_from_directory(training_dir, target_size=(200,200), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=64, subset="training")

validation_generator = data_generator.flow_from_directory(training_dir, target_size=(200, 200), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=64, subset="validation")
```

In this step, you began dividing your dataset into training and testing sets. You also specified the dimensions of your images and applied zooming and brightness adjustments to your pictures as inputs.

The dataset was divided into two types: training and testing.

Training: This section consists of 78,300 images, organized into 29

classes.

Testing: This section contains 8,700 images, also grouped into 29

classes.

The division was performed by allocating 10% of the dataset for testing and the remaining 90% for training.

```
train_generator = data_generator.flow_from_directory(training_dir, target_size=(200,200), shuffle=True, seed=13,
                                                     class_mode='categorical', batch_size=64, subset="training")
```

- Flow from directory is a method of the ImageDataGenerator class that generates batches of image data from a directory.
- training Dir represents the path to the directory that contains the training images.
- Target size is set to (200, 200), which means that all images will be resized to a height and width of 200 pixels.
- Shuffle is set to True, indicating that the order of the images in each batch will be randomly shuffled.
- The seed is set to 13, which is a random seed used for shuffling the images.
- The class mode is set to categorical, which means that the labels for each image are one-hot encoded.
- The batch size is set to 64, which means that each batch will contain 64 images.

- The subset is set to "training", which means that this data generator will only generate batches from the training set, and not include the testing set.

```
validation_generator = data_generator.flow_from_directory(training_dir, target_size=(200, 200), shuffle=True, seed=13,
                                                          class_mode='categorical', batch_size=64, subset="validation")
```

- Like the train generator, the testing generator is also created using the Flow from directory method of the ImageDataGenerator class.
- The subset parameter is set to "testing," indicating that this data generator will only generate batches from the testing set.

The remaining parameters for the testing generator are the same as those used for the train generator.

Summary: The provided code creates two data generators, namely a train generator and testing generator, for image classification tasks.

These data generators are created using the ImageDataGenerator. class, allowing for data augmentation and preprocessing of the images. The flow from directory method is utilized to generate. batches of image data from specific directories. These generators will serve as inputs for training and validating a neural network.

4.8.4 Choosing a Model:

In this step, the model selection process begins by considering the type of data you are dealing with. Different models are designed to handle specific data types such as images, sound, text, or even 3D images. The choice of the most suitable model depends on the nature of the data.

For your specific case, a CNN (Convolutional Neural Network) model was chosen. CNNs are specifically designed for image-related tasks and are a type of deep learning algorithm. Deep learning is used because your model operates on a large dataset, and deep learning excels in handling complex and extensive datasets. This highlights the difference between traditional machine learning approaches and deep learning.

By selecting a CNN model, you can leverage its algorithms tailored for image processing and take advantage of the capabilities offered by deep learning techniques to effectively analyze your dataset.

We used the InceptionV3 model, which is a deep learning model used in the field of image classification and computer vision.

```
WEIGHTS_FILE = './inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

inception_v3_model = keras.applications.inception_v3.InceptionV3(
    input_shape = (200, 200, 3),
    include_top = False,
    weights = 'imagenet'
)

inception_v3_model.summary()
```

Figure 50 pre-trained model.

The provided code segment defines WEIGHTS_FILE as the path to the pre-trained weights file for the InceptionV3 model. These weights were trained on the ImageNet dataset and can be downloaded from the Keras website.

- The next line creates a new instance of the InceptionV3 model named inception_v3_model with the following arguments:
  input shape = (200, 200, 3): This argument specifies the shape of the input tensor for the model. In this case, the input shape is set to (200, 200, 3), indicating that the model expects images with a width and height of 200 pixels and 3 color channels (RGB).
- include top = False: This argument indicates that the fully connected layers at the top of the InceptionV3 model should not be included. Instead, these layers will be replaced with custom classification layers suited for the specific task.
- weights = 'imagenet': This argument specifies that the pretrained weights trained on the ImageNet dataset should be used.

The final line prints a summary of the InceptionV3 model, providing information about its layers, the shape of output tensors from each

layer, and the number of parameters in each layer. This summary helps in understanding the model's architecture and verifying that it was initialized correctly.

```
inception_output_layer = inception_v3_model.get_layer('mixed7')
print('Inception model output shape:', inception_output_layer.output_shape)

inception_output = inception_v3_model.output
```

```
Inception model output shape: (None, 10, 10, 768)
```

Figure 51 pre-trained model

The provided code segment defines inception_output_layer as the layer with the name 'mixed7' in the pre-trained InceptionV3 model.

'mixed7' refers to the 7th mixed convolutional layer in the InceptionV3 model, which combines convolutions of different sizes to capture features at multiple scales.

The line print ('Inception model output shape:', inception_output_layer. output_shape) prints the shape of the output tensor of the inception_output_layer. This information clarifies the dimensions of the features extracted by the layer.

The line inception_output = inception_v3_model. output assigns the output tensor of the InceptionV3 model to the variable inception output. This tensor represents the activations of the last layer in the InceptionV3 model. It will be used as the input to the additional layers that will be added for the specific classification task.

```
from tensorflow.keras import layers
x = layers.GlobalAveragePooling2D()(inception_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense(29, activation='softmax')(x)

model = Model(inception_v3_model.input, x)

model.compile(
    # optimizer=SGD(lr=0.0001, momentum=0.9),
    optimizer=SGD(learning_rate=0.0001, momentum=0.9),
    loss='categorical_crossentropy',
    metrics=['acc']
)
for layer in model.layers[:249]:
    layer.trainable = False
for layer in model.layers[249:]:
    layer.trainable = True
```

Figure 52 Tensor flow layers & activation layers.

- layers.GlobalAveragePooling2D() (inception_output): This layer applies global average pooling to the output of the InceptionV3 model, which reduces the spatial dimensions of the feature map while preserving the channel information. This results in a fixedlength vector for each image, regardless of its size.
- layers.Dense(1024, activation='relu')(x): This layer is a fully connected layer with 1024 nodes and a RELU activation function. It takes the output of the global average pooling layer and applies a non-linear transformation to it.

- layers.Dense(51, activation='softmax')(x): This layer is the final fully connected layer with 51 nodes and a softmax activation function, which produces the probability distribution over the 51 possible classes.

- Model (inception_v3_model.input, x): This line defines the model architecture by specifying the input and output layers.
- model.compile(...): This line compiles the model by specifying the optimizer, loss function, and evaluation metrics.

- for layer in model.layers[:249]: layer.trainable = False: This loop sets the first 249 layers of the model to be non-trainable, which means their weights will not be updated during training.

- for layer in model.layers[249:]: layer.trainable = True: This loop sets the remaining layers to be trainable, which means their weights will be updated during training. By default, all layers in a Keras model are trainable, so we need to explicitly set the layers we want to freeze.

4.8.5 Training and Testing:

Training and testing are two crucial phases in the development and evaluation of deep learning models.

During the data collection phase, the dataset was divided into two types: Training and Testing.

Training data: Consists of 78,300 images, which accounts for 90% of the dataset.

Testing data: Consists of 8,700 images, which accounts for 10% of the dataset.

Training is the process of using a labeled set of input data to train a neural network model. The objective is to enable the model to recognize patterns and make accurate predictions. In the training process, the model adjusts its weights and biases based on the errors it encounters while processing the training data. The aim is to minimize the overall error and improve the model's performance.

Testing, on the other hand, involves evaluating a trained model using a separate testing dataset. The testing dataset is a distinct subset of the original dataset that was not used during the training phase. By assessing the model's performance on the testing dataset, we can gauge its ability to generalize and make accurate predictions on unseen data. Testing helps in assessing the model's performance, identifying potential issues such as overfitting or underfitting, and fine-tuning the model for better results.

```
LOSS_THRESHOLD = 0.2
ACCURACY_THRESHOLD = 0.95

class ModelCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if logs.get('val_loss') <= LOSS_THRESHOLD and logs.get('val_acc') >= ACCURACY_THRESHOLD:
      print("\nReached", ACCURACY_THRESHOLD * 100, "accuracy, Stopping!")
      self.model.stop_training = True

callback = ModelCallback()
```

Figure 53 Training Model

The provided code defines a custom callback function for a TensorFlow Keras model. This callback function is designed to monitor the model's performance during training and potentially stop the training process early based on specified criteria.

The custom callback class, named "ModelCallback," contains a single method called "on_epoch_end." This method is automatically called by Keras at the end of each training epoch. The method takes the "logs" parameter, which is a dictionary containing various metrics, such as loss and accuracy, computed during the current epoch .

By implementing this callback function, we can incorporate early stopping functionality into our model training. Early stopping is a technique used to prevent overfitting and improve training efficiency.

It allows the model to stop training if certain performance criteria are met on the testing set. This can save computational resources and prevent the model from continuing to train unnecessarily.

The specific implementation of the early stopping criteria is not provided in the given code snippet, but it can be customized based on the specific needs of the project. Common criteria include monitoring the testing loss or accuracy and stopping training if there is no improvement or if the performance starts to degrade.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 200, 200, 3)]     0         []

 conv2d (Conv2D)             (None, 99, 99, 32)        864       ['input_1[0][0]']

 batch_normalization (Batch  (None, 99, 99, 32)        96        ['conv2d[0][0]']
 Normalization)

 activation (Activation)     (None, 99, 99, 32)        0         ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)           (None, 97, 97, 32)        9216      ['activation[0][0]']

 batch_normalization_1 (Bat  (None, 97, 97, 32)        96        ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, 97, 97, 32)        0         ['batch_normalization_1[0][0]'
                                                                 ]

 conv2d_2 (Conv2D)           (None, 97, 97, 64)        18432     ['activation_1[0][0]']

 batch_normalization_2 (Bat  (None, 97, 97, 64)        192       ['conv2d_2[0][0]']
 chNormalization)

 activation_2 (Activation)   (None, 97, 97, 64)        0         ['batch_normalization_2[0][0]'
                                                                 ]

 max_pooling2d (MaxPooling2  (None, 48, 48, 64)        0         ['activation_2[0][0]']
 D)

 conv2d_3 (Conv2D)           (None, 48, 48, 80)        5120      ['max_pooling2d[0][0]']

 batch_normalization_3 (Bat  (None, 48, 48, 80)        240       ['conv2d_3[0][0]']
 chNormalization)

 activation_3 (Activation)   (None, 48, 48, 80)        0         ['batch_normalization_3[0][0]'
```

Figure 54 Model summary (Inception)

```
activation_91 (Activation)  (None, 4, 4, 384)          0        ['batch_normalization_91[0][0]
                                                                 ']

activation_92 (Activation)  (None, 4, 4, 384)          0        ['batch_normalization_92[0][0]
                                                                 ']

batch_normalization_93 (Ba  (None, 4, 4, 192)          576      ['conv2d_93[0][0]']
tchNormalization)

activation_85 (Activation)  (None, 4, 4, 320)          0        ['batch_normalization_85[0][0]
                                                                 ']

mixed9_1 (Concatenate)      (None, 4, 4, 768)          0        ['activation_87[0][0]',
                                                                  'activation_88[0][0]']

concatenate_1 (Concatenate  (None, 4, 4, 768)          0        ['activation_91[0][0]',
)                                                                 'activation_92[0][0]']

activation_93 (Activation)  (None, 4, 4, 192)          0        ['batch_normalization_93[0][0]
                                                                 ']

mixed10 (Concatenate)       (None, 4, 4, 2048)         0        ['activation_85[0][0]',
                                                                  'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

global_average_pooling2d (  (None, 2048)               0        ['mixed10[0][0]']
GlobalAveragePooling2D)

dense (Dense)               (None, 1024)               2098176  ['global_average_pooling2d[0][
                                                                 0]']

dense_1 (Dense)             (None, 29)                 29725    ['dense[0][0]']

==================================================================================================
Total params: 23930685 (91.29 MB)
Trainable params: 13242781 (50.52 MB)
Non-trainable params: 10687904 (40.77 MB)
```

Figure 55 Model summary (Inception)

The ‘model.summary()’ function is typically used in machine learning frameworks like TensorFlow and Keras to display a summary of the architecture and parameters of a neural network model. It provides a concise overview of the layers in the model, the output shape of each layer, and the total number of trainable parameters in the model.

This summary is useful for understanding the structure of the model and verifying that it has been built correctly. It can also help in diagnosing any issues related to the model's architecture or parameter configuration.

```
# history = model.fit_generator(
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    steps_per_epoch=200,
    validation_steps=50,
    epochs=25,
    callbacks=[callback]
)
```

Figure 56 Training and prediction.

The provided code trains a Keras model using data generators for both the training and testing data. It includes various training parameters and utilizes a custom callback function to monitor testing performance and potentially stop training early.

The training process is initiated using the model.fit function. The train generator and testing generator parameters represent the data generators for the training and testing data, respectively. The steps per epoch parameter determine the number of batches to use in each training epoch, which is set to 200 steps. Similarly, the testing steps parameter specifies the number of batches to use in each testing epoch, set to 50 steps.

The epochs parameter specifies the total number of training epochs, set to 25 epochs in this case.

During the training process, Keras records the training and testing loss and accuracy for each epoch in a history object. The history variable is used to store this object, which can be later used to visualize the performance using graphs or charts.

By utilizing the custom callback function and monitoring the testing

performance, the training process can be stopped early if certain criteria are met. The specific implementation of the callback function and its criteria are not provided in the given code snippet and would need to be customized according to the requirements of the specific project.

```
Epoch 1/25
200/200 [==============================] - 254s 1s/step - loss: 3.3029 - acc: 0.0862 - val_loss: 3.1568 - val_acc: 0.1369
Epoch 2/25
200/200 [==============================] - 218s 1s/step - loss: 2.8124 - acc: 0.3765 - val_loss: 2.7581 - val_acc: 0.3828
Epoch 3/25
200/200 [==============================] - 218s 1s/step - loss: 2.1958 - acc: 0.6314 - val_loss: 2.1805 - val_acc: 0.5975
Epoch 4/25
200/200 [==============================] - 218s 1s/step - loss: 1.5656 - acc: 0.7873 - val_loss: 1.5935 - val_acc: 0.7350
Epoch 5/25
200/200 [==============================] - 219s 1s/step - loss: 1.0668 - acc: 0.8719 - val_loss: 1.1486 - val_acc: 0.8253
Epoch 6/25
200/200 [==============================] - 262s 1s/step - loss: 0.6862 - acc: 0.9224 - val_loss: 0.7704 - val_acc: 0.8750
Epoch 7/25
200/200 [==============================] - 217s 1s/step - loss: 0.4602 - acc: 0.9438 - val_loss: 0.5410 - val_acc: 0.9125
Epoch 8/25
200/200 [==============================] - 217s 1s/step - loss: 0.3284 - acc: 0.9575 - val_loss: 0.4094 - val_acc: 0.9284
Epoch 9/25
200/200 [==============================] - 217s 1s/step - loss: 0.2369 - acc: 0.9688 - val_loss: 0.3204 - val_acc: 0.9397
Epoch 10/25
200/200 [==============================] - 217s 1s/step - loss: 0.1851 - acc: 0.9778 - val_loss: 0.2504 - val_acc: 0.9578
Epoch 11/25
200/200 [==============================] - 219s 1s/step - loss: 0.1449 - acc: 0.9817 - val_loss: 0.2153 - val_acc: 0.9631
Epoch 12/25
200/200 [==============================] - ETA: 0s - loss: 0.1185 - acc: 0.9852
Reached 95.0 accuracy, Stopping!
200/200 [==============================] - 215s 1s/step - loss: 0.1185 - acc: 0.9852 - val_loss: 0.1993 - val_acc: 0.9553
```

Figure 57 Epochs and final accuracy.

And the goal here is that you try your model on what you know, see its accuracy, and test it on the Testing Set, so if it doesn't work well, you won't waste the whole Evaluation Set. In the Cross Testing, it turned out fine. Start worrying about the Testing Set and try it too, and see our model accuracy percentage also, and see how much it has reached to make sure that my model is working properly or not.

```
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.show()
```

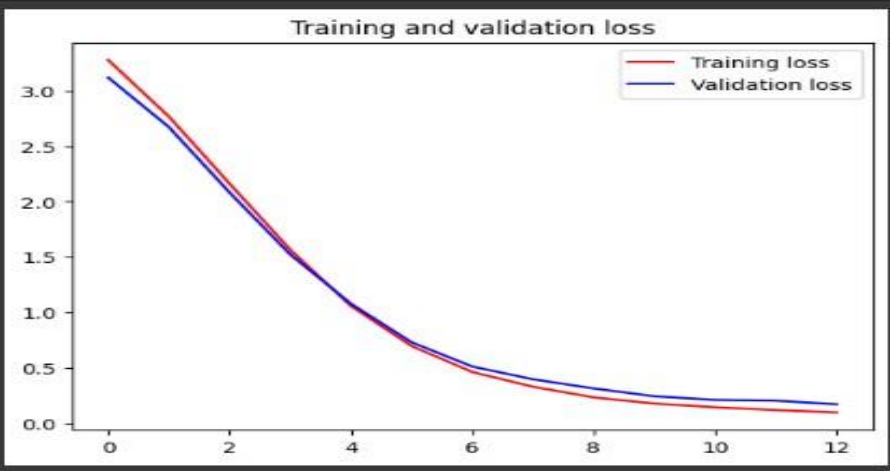


Figure 58 Visualize training result (loss)

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.show()
```

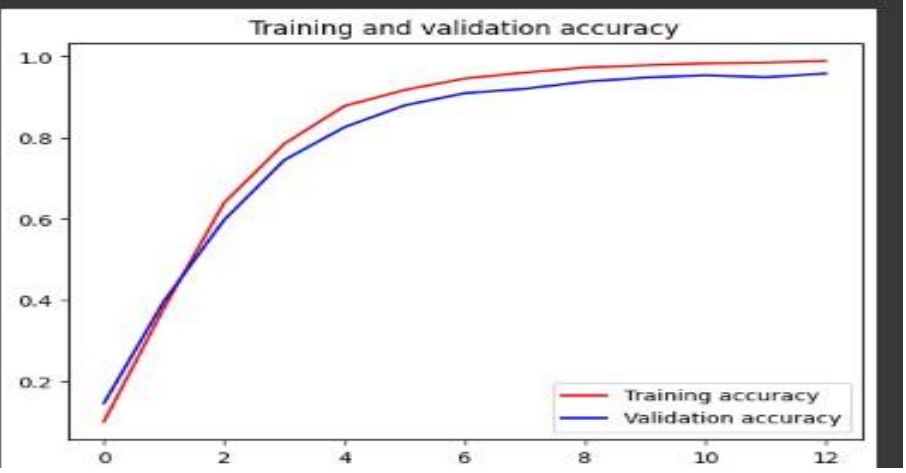


Figure 59 Visualize training result (accuracy)

Here, if the result of the evaluation was bad, you start to see where your problems were exactly, modify the parameters of the Dataset, and do Training more than once to improve the Model.

But it is as shown in the screenshot taken from our model, which is that the model Ours works very well and there are no errors.

4.8.6 Evaluation:

Here the model's accuracy is shown after training, its training accuracy reaches 99.5% and validation accuracy reaches 95.5%.

```
final_train_loss = history.history['loss'][-1]
final_train_accuracy = history.history['acc'][-1]

print("Final Train Loss:", final_train_loss)
print("Final Train Accuracy:", final_train_accuracy)

Final Train Loss: 0.11854083091020584
Final Train Accuracy: 0.9852343797683716
```

Figure 60 Evaluate the model on the training data.

```
final_val_loss = history.history['val_loss'][-1]
final_val_accuracy = history.history['val_acc'][-1]

print("Final Validation Loss:", final_val_loss)
print("Final Validation Accuracy:", final_val_accuracy)

Final Validation Loss: 0.19925682246685028
Final Validation Accuracy: 0.9553124904632568
```

Figure 61 Evaluate the model on the validation data.

4.8.7 Prediction:

In Our sign language recognition model, prediction refers to the process of using the trained model to classify new or unseen sign language gestures or signs. Once the model has been trained on a dataset of labeled sign language images, it can be used to predict the label or class of new images by passing them through the trained model.

The prediction process typically involves passing the new image through the model's layers, which extract features and map them to a final output layer that provides a probability distribution over the

possible classes. The class with the highest probability is then selected as the predicted class for the input image.

The accuracy of the prediction depends on various factors such as the quality and diversity of the training data, the complexity and

architecture of the model, and the accuracy of the prediction algorithm used in the model.

As we tested the model on a group of images consisting of 29 images, from each classification we took an image to test the model on. These images had not been seen by the model before during the training or verification phase, and the model was able to predict them very correctly. These images illustrate the model’s prediction.

Figure 62 Prediction result.

4.9 YOLOv8

4.9.1 Object Detection

Object detection technology is an important field in the field of machine learning and computer vision, as it aims to develop systems capable of identifying and classifying objects in images or videos. Object detection is an important technical challenge due to the complexity of different environments and the variety of shapes and sizes of objects.

The object detection process involves several technical challenges, including:

- Localization: In this step, the locations of objects within the image or video are determined accurately. Objects are usually identified using a Bounding Box, and the center and size of the frame are known to determine the exact location and size of the object. (bx, by, bh, bw).

- Classification of objects: After locating objects, they are classified into different categories. For example, if the image contains cars, people, and animals, this step aims to correctly identify and classify the type of each object.

The success of an object detection system depends on its ability to achieve high accuracy in identification and classification, and at the same time the system must work quickly and efficiently with huge data.

Object detection techniques are mainly based on deep neural networks, especially closed-loop neural networks (CNNs). These networks are used to extract features from images and videos and accurately classify objects.

Hence the final y matrix is:

PC

bx

by

bh

bw

c1

c2

c3

…

PC indicates the probability of the presence of an object. If there is an object (a car, an animal, a person, or a sign), the value is 1, and if there is no object or part of the background, the value is 0.

bx, by, bh, bw They are the values that specify the location or dimensions of the boundary box.

bx and by: The coordinates of the upper-left corner of the boundary box surrounding the object in the image, where bx indicates the location on the horizontal (X) axis, and by indicates the location on the vertical (Y) axis.

bh and bw: represent the height and width of the boundary box, respectively. These values are used to determine the size and shape of the object within the image.

c1, c2, c3: represent the different object classes that are classified in the image. Each category is represented by a specific number or

symbol. define what the specific thing is and express the classification.

Deep Learning-based Object Detection:

This technique relies on using deep neural networks such as closed-loop neural networks (CNNs) to detect objects in images. This category includes popular models such as Faster R-CNN, YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), Landmarks, Sliding Windows and others.

4.9.2 YOLO (You Only Look Once):

is an object detection model that uses deep neural networks. YOLO takes an innovative approach to the object detection process, dividing the image into a grid of cells and then analyzing the image as a whole in one go, making it more efficient and faster than traditional models that divide the image into small sectors and analyze them separately.

It divides the image into several square sections, such as (3*3), and each square examines the image to determine whether it contains the required part or not. Per-cell prediction where for each cell in the network, a set of predictive results is produced. Objects in the image are predicted by these cells, including their locations determined by the boundary frame and their classification into different classes.

can detect and classify several different objects in a single image without having to repeat the process. This makes it ideal for applications that require more than one object to be detected at the same time.

4.9.3 ASL Alphabet Dataset:

The dataset is a collection of images of alphabets from the American Sign Language, separated in 26 folders for the letters A-Z which represent the various classes.

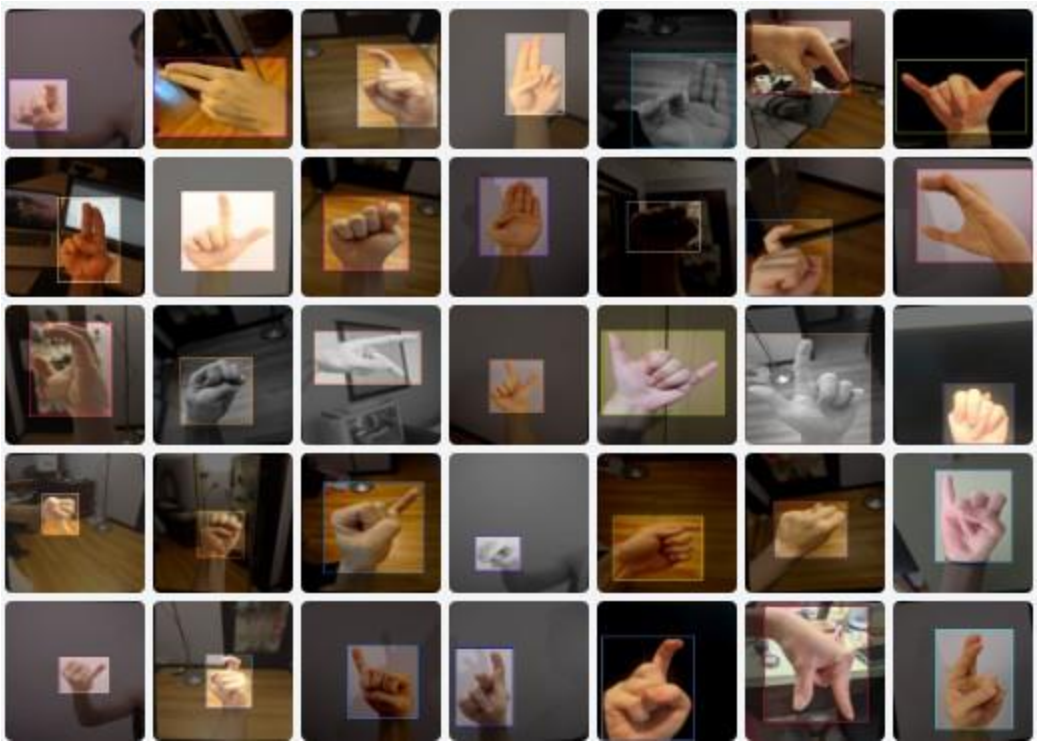


Figure 63 samples from the dataset.

4.9.4 Load dataset:

We obtained a ready-made dataset from the Roboflow website. The code uses the Roboflow Python library to load a specific dataset from a specific project into a Roboflow account. Using an API key for authentication, it is then used to identify a specific project in the workspace using the workspace and project names. Next, the desired version of the project is selected and the data format to be downloaded is selected. upload it to Colab after creating an account on Roboflow and the dataset is loaded and stored in a dataset variable by accessing the dataset in this manner.

Figure 64 Install roboflow.

Figure 65 Data collection.

Figure 66 Import YOLO.

This code is used to load a pre-trained model from YOLOv8.

"The pre-trained YOLOv8 model is loaded using the weight file 'yolov8n.pt'."

This file is used to load the parameters of the model that has been trained on a pre-defined dataset. Once the model is loaded, it can be used to perform object detection and classification tasks in images and videos without the need for retraining.

```
[ ]  #`Load a pretrained Yolov8 model
     model = YOLO('yolov8n.pt')
```

Figure 67 Pretrained YOLOV8.

4.9.5 Preprocessing:

At the preprocessing stage, the resizing process, the results of this process produce an image size of 640×640 pixels, from the original image that have different dimensions.

We made augmentations for the images to increase their size and trained the model on different shapes and positions of the images. They include:

Crop: 0% Minimum Zoom, 20% Maximum Zoom.

Rotation: Between -15° and +15°.

Shear: ±10° Horizontal, ±10° Vertical.

Brightness: Between -15% and +15%.

4.9.6 Model Training:

This code trains the model using YOLOV8 on the downloaded data set.

```
!yolo task = detect mode = train model = yolov8n.pt data = {dataset.location}/data.yaml epochs = 20 imgsz=640
```

Figure 68 train the model, use 20 epochs.

- ! yolo: Indicates the use of You Only Look Once.
- task = detect: Specifies the training task that will be performed.
- mode = train: Specifies that the process that will take place is training, where the model is trained on the given data set.
- model=yolov8n.pt: Specifies the name and location of the file containing the YOLOv8n model that will be used in training.
- data = {dataset.location}/data.yaml: Specifies the location and format of the training data. The data.yaml file is used to determine the locations of images and object labels in the dataset.
- epochs = 20: Specifies the number of epochs during which the model will be trained on the dataset.
- imgsz=640: Specifies the size of the images used in training with dimensions of 640 x 640 pixels.

4.9.7 Model Validation:

To evaluate the performance of the trained model by testing it on a set of images designated for verification, by downloading the best version of the trained model and using it to detect objects in images located in a specific data location, which helps in evaluating the model's ability to recognize and predict objects accurately and effectively.

```
# Model validation
!yolo task = detect mode = val model = /content/runs/detect/train/weights/best.pt  data = {dataset.location}/data.yaml
```

Figure 69 Model validation.

- ! yolo: Indicates the use of You Only Look Once object detection.
- task = detect: This defines the task we want to perform, which is the task of detecting objects in images.
- mode = val: Specifies the mode in which we want to execute the command, which is the form validation mode.
- model = /content/runs/detect/train4/weights/best.pt: This specifies the path containing the best model trained during the training process, which will be used in the validation process.
- data = {dataset.location}/data.yaml: Specifies the location of the data file used for verification, which contains information about image locations and object classifications.

4.9.8 Test Data:

We back test the data to evaluate the performance of the model trained on the dataset. During the testing process, data that was not used in training is used to determine the model's ability to handle new, unknown data. Testing helps determine how accurate and effective the model is in predicting or classifying unknown data.

```
# Model Test
!yolo task = detect mode = predict model =/content/runs/detect/train/weights/best.pt  data = {dataset.location}/data.yaml source = {dataset.location}/test/images save_txt = true save = true
```

Figure 70 Test data.

This code is designed to perform object detection testing using YOLOv8. It utilizes a trained YOLOv8 model to make predictions on a set of test images. The predictions include bounding box coordinates for detected objects, which can be saved in text files. Additionally, the command saves images with bounding boxes drawn around the detected objects, aiding in visualizing the model's performance on the test data. This process helps assess the accuracy and effectiveness of the trained model in detecting objects within the test images.

This is a detailed explanation of the code:

- !yolo: Indicates the use of You Only Look Once object detection.

- task = detect: Specifies that the task to be performed is object detection.

- mode = predict: Sets the mode of operation to prediction, meaning it's used for making predictions on new data.

- model = /content/runs/detect/train4/weights/best.pt: Specifies the path to the trained YOLOv8 model weights file.

- data = {dataset.location}/data.yaml: Specifies the path to the YAML file containing dataset configuration (e.g., class names, image paths).

- source = {dataset.location}/test/images: Specifies the directory containing the test images.

- save_txt = true: Indicates that the predicted bounding box coordinates should be saved in a text file.

- save = true: Indicates that the images with bounding boxes drawn around detected objects should be saved.

This command will run YOLOv8 in detection mode, using the specified model to make predictions on the test images. Detected objects will be saved in text files (in YOLO format) and images with bounding boxes drawn around them will also be saved.

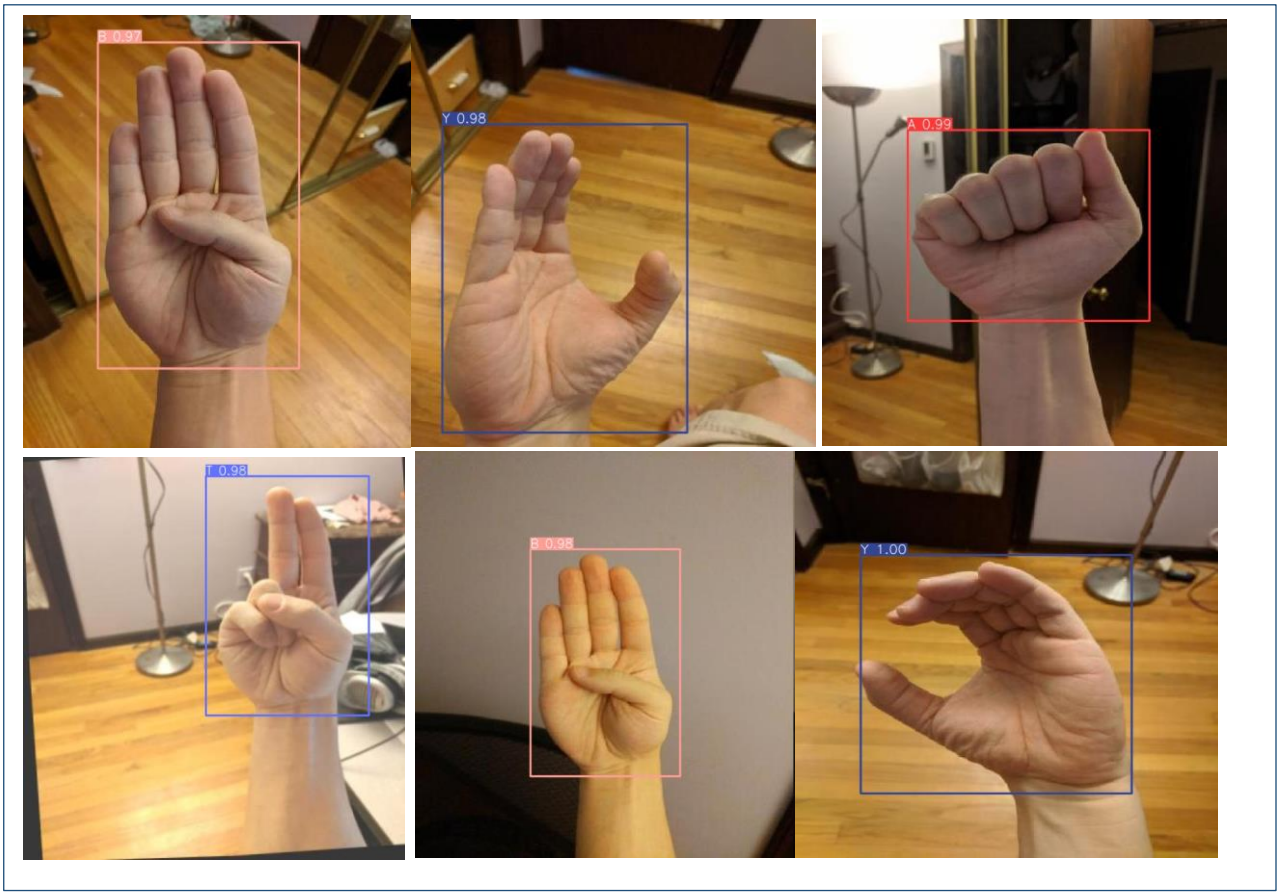


Figure 71 Prediction result.

4.9.9 Live Object Detection:

Live Object Detection using YOLO Model refers to the utilization of the YOLO model for real-time object detection, meaning during the direct streaming of video. The model analyzes the live video stream and identifies the

objects within it. And It displays the detection results directly within the video. When the video stream is running, the model analyzes the frames at a high speed and identifies the objects present in each frame. Objects are identified by drawing bounding boxes around them, and often confidence scores are provided, indicating the model's confidence level in correctly identifying the object. This process allows the user to monitor real-time occurrences and quickly and accurately recognize the objects present in the surrounding environment.

After training and testing the model on Colab, we saved the model weights, which we will now utilize for making predictions in real-time using ‘Anaconda’.

```
model = YOLO("best (1).pt")
model.predict(source=(0),conf=0.7,show=True,save=True)
```

Figure 72 Live Object Detection using YOLO Model

This is a detailed explanation of the code:

- model = YOLO("best(1).pt"): Loads a pre-trained YOLO model from the file "best(1).pt".

- model.predict(source=(0), conf=0.7 , show=True, save=True): The loaded model performs real-time object detection. Details of the functionalities:

- source=(0):This parameter specifies the source of the images for object detection. Here, it's set to 0, indicating that the source is the camera with index 0, likely the default webcam.

- conf=0.7: This sets the confidence threshold to 0.7, meaning that the model will ignore objects with a confidence score lower than this threshold. In other words, detected objects will only be considered if the model is at least 70% confident in their identity.

- show=True: Enables the display of the live video stream with overlaid detected objects.

- save=True: Optionally saves the processed video stream with detected objects.

In essence, this code harnesses the capabilities of a pre-trained YOLO model to conduct real-time object detection and visualization from a live video stream, providing immediate insights into the contents of the scene captured by the camera.

Chapter5

PROTOTYPE OF APPLICATION

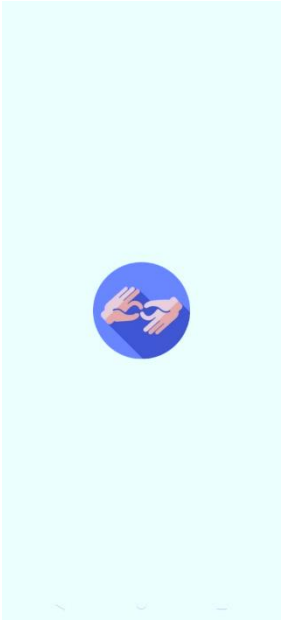


Figure 73 Prototype of Application.

Welcome to the sign talk application. a new application designed to bridge the communication gap between the deaf and hearing communities. Communication is essential for humanity, yet many deaf and hard of hearing individuals face barriers due to their inability to hear. SignTalk aims to break down these barriers through real-time sign language translation.

by getting into the application, we see a splash screen with

the logo of the application and then the application opens on the welcome page.

5.1.1 Screen of Welcome:

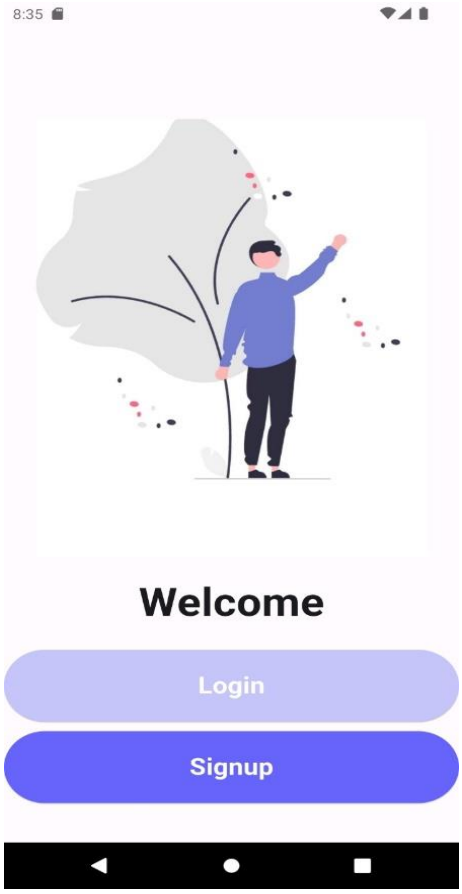


Figure 74 Screen of Welcome.

the welcome page contains 2 buttons. the first button is login button which directs you to login page. if you don't have an account, you can choose the sign it is second button.

5.1.2 Screen of Login:

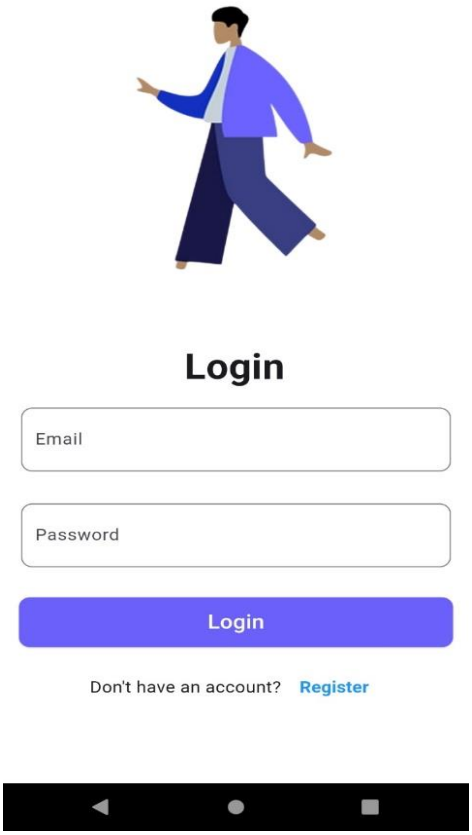


Figure 75 Screen of Login.

the user goes to the login page and enters his credentials to get access to the application. If the user entered his credentials wrong, the application wouldn't open for him. If the user entered his credentials correctly and pressed the login button, then he is routed to the home page.

if you didn't enter your credentials and press register button an error message will appear telling you to enter your credentials. once you enter your credentials correctly, the information you provided is then stored in firebase.

5.1.3 Screen of Register:

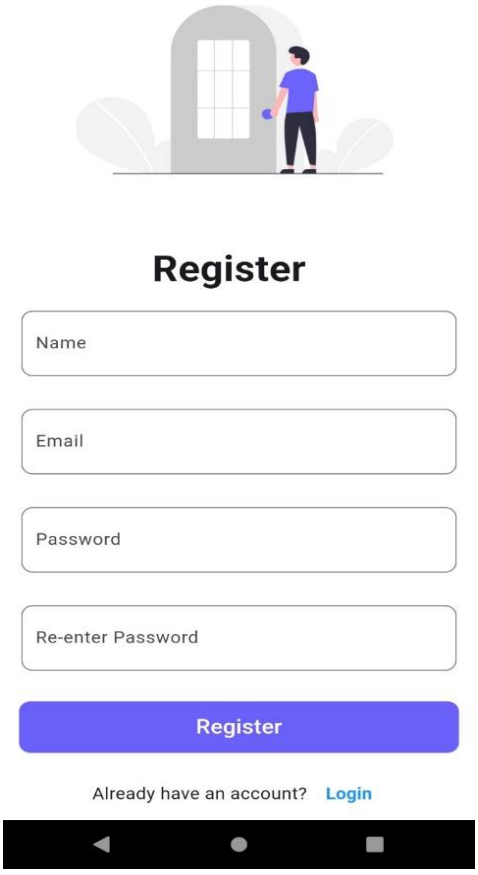


Figure 76 Screen of Register.

Once you are on the register page you can add your credentials by adding your name, email, password, and re-enter password. if you didn't enter your credentials and press the register button an error message will appear telling you to enter your credentials. once you enter your credentials right, the information you provided is then stored in Firebase.

in our app, the data is stored and once the user creates his account, he goes to the login page and enters his credentials to get access to the application. if the user entered his credentials wrong, the application wouldn't open for him. The user entered his credentials right and pressed the login button then he is routed to the home page.

5.1.4 Screen of Home Page:

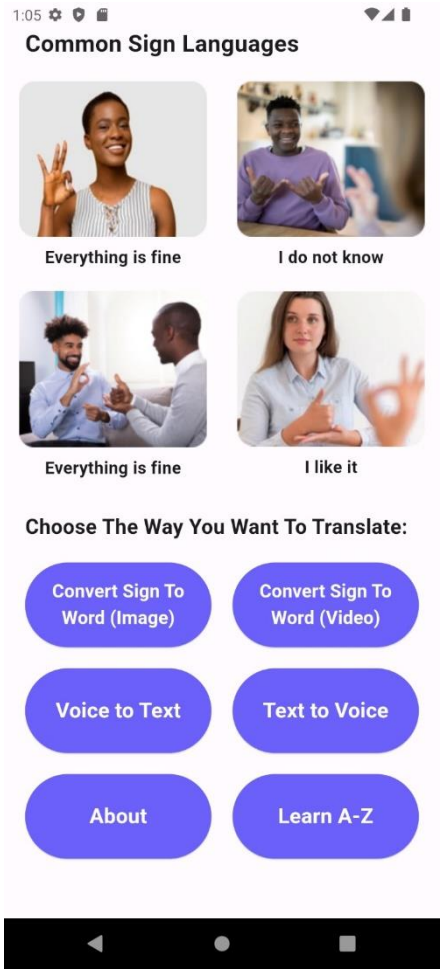


Figure 77 Screen of Home Page.

down these pictures we found 6 buttons. each button makes a suitable job, and We will explain them.

This page contains a small picture of people with disabilities that explain common language words or sentences.

5.1.5 Screen of Convert Sign To Word (Image) :

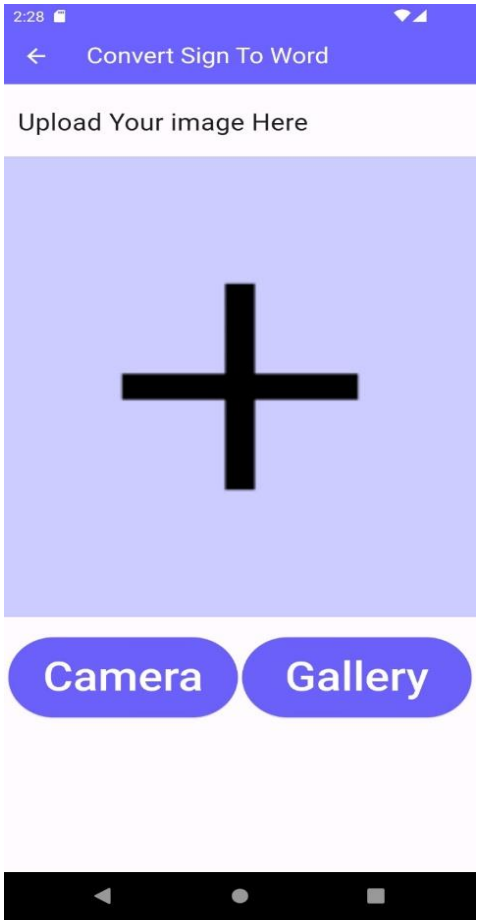


Figure 79 Screen of Convert Sign to Word.

This is the first button in the home page, convert sign to word. when accessing this button, a page appears to upload your image, you can upload your image by pressing the camera button and opens the camera, or by adding an existing picture found in your gallery.

5.1.6 Screen of Voice to Text:

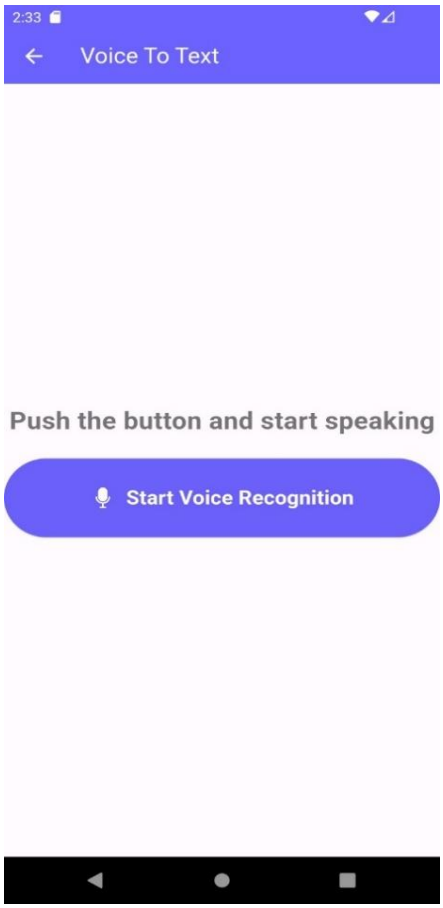


Figure 80 Screen of Voice to Text.

the third button is a voice-to-text. The normal user is directed to a page where he can find a microphone button that he can use to talk, and the application writes above what the user talks about as we can see.

5.1.7 Screen of Text to Voice:

Figure 81 Screen of Text to Voice.

the fourth button is text to voice. The normal user uses it where he can enter the text and press the speak button to convert the text written to voice.

5.1.8 Screen of Learn A To Z:

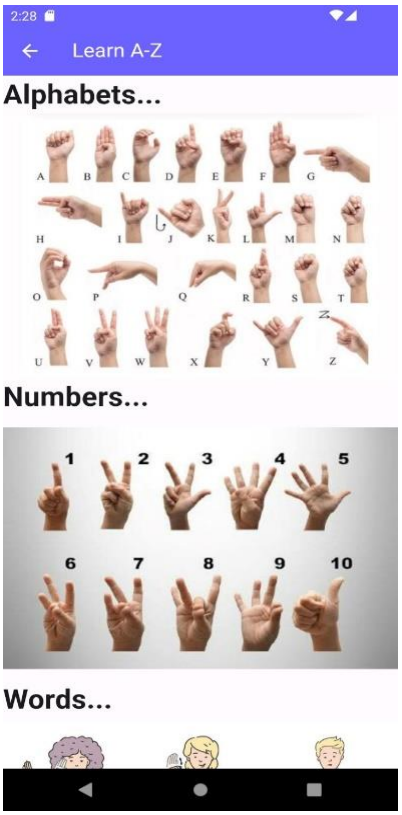


Figure 82 Screen of Learn A To Z and some Numbers.

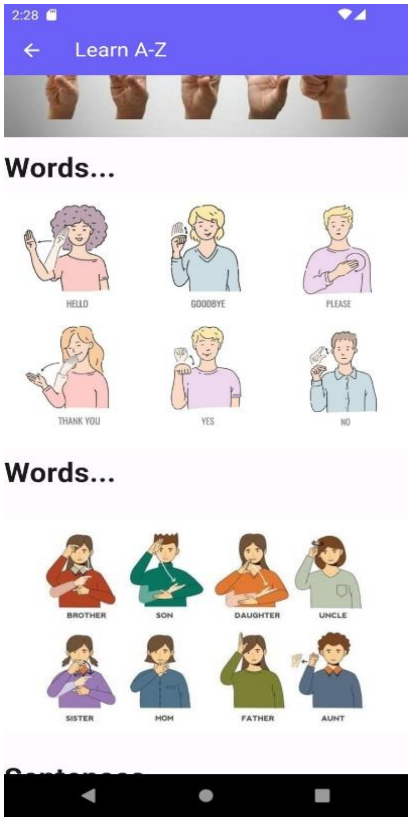


Figure 83 Screen of Learn some words.

the last button is the Learn A-Z button. The "Learn A-Z" button in the application serves as a valuable resource for users who want to expand their knowledge and understanding of sign language. When a user clicks on the "Learn A-Z" button,

they are directed to a dedicated section within the application focused on educational content related to sign language. This section may include:

Alphabet: An interactive guide to learning the sign language alphabet (A-Z)

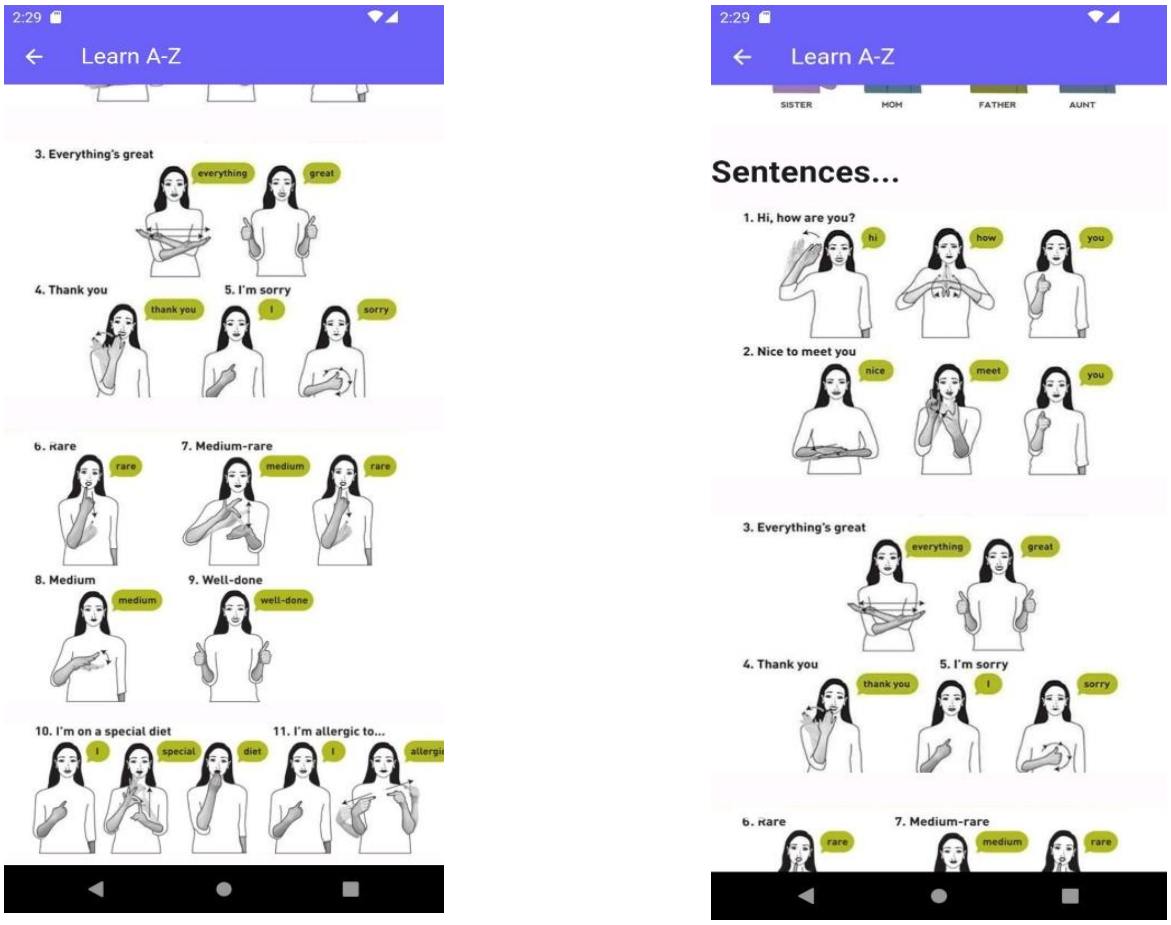


Figure 84 Screen of learn some common sentences.

5.1.9 Screen of About:

Finally, we reach the last page which is the about page. by accessing this page, the normal user can see a brief information about the application, how it works and contact us in case any inquiries. Thanks for visiting our application. Happy signing

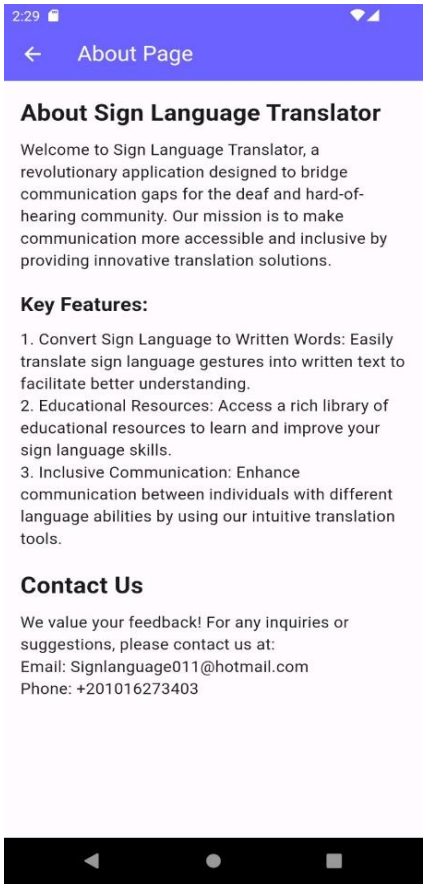


Figure 85 Screen of About.

Key Features:

1. Convert Sign Language to Written Words: Easily translate sign language gestures into written text to facilitate better understanding.

2. Educational Resources: Access a rich library of educational resources to learn and improve your sign language skills.

3. Inclusive Communication: Enhance communication between individuals with different language abilities by using our intuitive translation tools.

Contact Us We value your feedback! For any inquiries or suggestions, please contact us at:

Email: Signlanguage011@hotmail.com

Phone: +201016273403

Chapter 6

6.1 PROJECT SCOOP

6.1.1 Dependances

- Motivation and motivation: The implementing team and project participants must have a strong motivation to achieve their goals, whether these goals are related to learning sign language or community service.
- Planning and organization: A solid plan and effective organization must be developed to implement the project, including identifying the required resources and distributing responsibilities appropriately.
- Effective communication: There must be constant and effective communication between team members, partners, and the target community to ensure the project objectives are successfully achieved.
- React and adapt: The project must be flexible and able to adapt to challenges and changes that may come along the way, allowing for improved performance and success.

- Evaluation and feedback: Project performance should be regularly evaluated, and lessons learned from previous experiences drawn, and mechanisms should be provided to receive feedback from participants and other actors.

- We choose algorithms to be represented in solving technical problems, as they significantly impact system performance and efficiency. Therefore, choosing an algorithm requires a careful study of the project requirements. Therefore, sufficient emphasis must be placed on determining and improving the performance and accuracy of the results

6.1.2 Risk and Mitigation

Risk	Mitigation
The user has more than one language for the deaf.	We collected a set of data from 4 different languages, this data was studied, and then we began to choose the language most spoken among people.
It is possible that the user will choose a word that is not found in the data set.	We combined two data sets.
It is possible that the user may not be proficient in writing to communicate with a deaf person	Therefore, we added a microphone that converts speech into written text, making communication faster and easier.

Figure 86 Risk and Mitigation.

6.2 Project developments

1- We will enhance the application for faster and easier communication by converting speech into sign language, enabling

deaf individuals to understand. This makes the application bidirectional.

2- Additionally, we will convert text into sign language as well.

3- We will incorporate multiple languages, allowing any user to use the application in any country and serve a broader audience.

4- We will add a dataset containing sentences and more phrases to cover a larger portion of the language.

6.3 CONCLUSION

Using deep learning, we have created a model that recognizes hand gestures in sign language and translates them into expressive sentences and texts that express their needs and facilitate communication. This is done using convolutional neural networks that were trained on a set of properly classified data and tested to reach the highest accuracy of the model so that it is ready to be linked With a mobile application and used it to help them communicate in daily life.

Using hand gesture recognition and convolutional neural networks. It has the potential to revolutionize communication for people suffering from Speech disabilities. By improving accuracy and We can use previous solutions and hope to create valuable solutions A tool that can prevent communication and promotion barriers Read from the six people who rely on contract language Contribute to the project step by step to the result Power learning and computer vision techniques big chapter in the lives of people suffering from anaemia Persia and their societies. There is another very important aspect of this the project is to ensure that the resulting system is easy to use and It is accessible to a wide range of users, including those with minimal needs Technical expertise. To achieve this, we will focus on developing Intuitive user interface simplifies the signal capture process Language gestures and display the corresponding translation in it in real time. We will also look at integrating our system with

Popular communication platforms and devices, enabling user We seamlessly integrate technology into their daily lives.

Learn about the importance of validating our proposed solution Real world scenarios. To this end, we will conduct comprehensive testing and evaluate the system performance under different conditions, such as different lighting environments, the user Demographics, sign language dialects. We will also strive. Feedback from the speech and sign language disabled community Interpreters to ensure that our system meets their needs Effectively and accurately.

6.4 REFERENCES

[1].Alphapet_Dataset

[1]English

https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet/data,"

https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-aplhabet-dataset

https://www.kaggle.com/datasets/datamunge/sign-language-mnist/data

https://www.kaggle.com/datasets/grassknoted/asl-alphabet/code

[2]Arabic

https://www.kaggle.com/datasets/sabribelmadoui/arabic-sign-language-unaugmented-dataset

https://universe.roboflow.com/omdena-pemas/arabic-sl

[3]Spanish

https://www.kaggle.com/datasets/kirlelea/spanish-sign-language-alphabet-static"

[4]india

https://universe.roboflow.com/niladri-basu-roy-qnrm4/indian-sign-language-detection

https://www.kaggle.com/datasets/kartik2112/indian-sign-language-translation-letters-n-digits

https://www.kaggle.com/datasets/dodiyaparth/indian-sign-language

[2].Word_Dataset

[1]English

https://www.kaggle.com/datasets/risangbaskoro/wlasl-processed,

https://www.kaggle.com/datasets/kshitij192/action-recognition

https://www.kaggle.com/datasets/belalelwikel/asl-and-some-words/data

[2]Arabic

https://universe.roboflow.com/rania-hamada-xcozs/arabic-sign-language-words-detection

https://www.kaggle.com/datasets/mahmoudmsaafan/arabic-sign-language-dataset

[3]Spanish

https://www.kaggle.com/datasets/mguiralc03/spanishsignlanguagerecognitiontfg"

[4]india

https://www.kaggle.com/datasets/kshitij192/action-recognition

https://www.kaggle.com/datasets/daskoushik/include

https://www.kaggle.com/datasets/soumyakushwaha/indian-sign-language-dataset

[1].Sentence_Dataset

[1]English

https://www.kaggle.com/datasets/tasinalnahiankhan/phrase-level-asl-converted-in-numpy-array,

[2]Arabic

https://www.kaggle.com/datasets/mohamedlotfy50/arabic-sign-language

[4]india

https://data.mendeley.com/datasets/kcmpdxky7p/1

Paper_Link

https://sci-hub.se/10.1109/sti47673.2019.9067974

https://sci-hub.se/10.1109/iciccs51141.2021.9432296

https://link.springer.com/chapter/10.1007/978-981-15-7961-5_6

https://www.hindawi.com/journals/cin/2022/1450822/

https://link.springer.com/chapter/10.1007/978-981-99-3734-9_32

https://ieeexplore.ieee.org/abstract/document/9491897

https://www.sciencedirect.com/science/article/pii/S1877050922021846/pdf?

https://scholarworks.calstate.edu/downloads/dj52wb92k

https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/9179356/

https://www.hindawi.com/journals/wcmc/2020/3685614/