Th_ECE: 17_36

Recognition of Two Hand English Manual Sign Alphabets of BdSL with Transfer Learning and CNN

Uday Syed

Roll: 1709036

February 2023

Department of Electronics and Communication Engineering

Khulna University of Engineering & Technology

Khulna- 9203, Bangladesh.

Recognition of Two Hand English Manual Sign Alphabets of BdSL using Transfer Learning and CNN

This thesis is being submitted to the Department of Electronics and Communication Engineering at Khulna University of Engineering and Technology in partial completion of the requirements for the degree of

"Bachelor of Science in Electronics and Communication Engineering"

Supervised by

Dr. Md. Foisal Hossain
Professor
Department of Electronics and Communication Engineering,
KUET, Khulna.

Submitted by

Uday Syed
Roll: 1709036
Department of Electronics and Communication Engineering,
KUET, Khulna.

February 2023

Department of Electronics and Communication Engineering

Khulna University of Engineering & Technology

Khulna- 9203, Bangladesh.

Recognition of Two Hand English Manual Sign Alphabets of BdSL using Transfer Learning and CNN

Author

Uday Syed
Roll: 1
Dept. of Electronics and Communication Engineering
Khulna University of Engineering & Technology

Supervisor

Dr. Md. Foisal Hossain
Professor
Dept. of Electronics and Communication Engineering
Khulna University of Engineering & Technology

……

Signature

External

Dr. Md. Mostafizur Rahman
Professor
Dept. of Electronics and Communication Engineering
Khulna University of Engineering & Technology

……

Signature

February 2023

Department of Electronics and Communication Engineering

Khulna University of Engineering & Technology

Khulna- 9203, Bangladesh.

Statement of Originality

This thesis, in its entirety or in part, has never been published. Where necessary, all items or sub-items on the thesis are referenced. The author's permission is required before copying the thesis. This thesis was also not previously submitted as an undergraduate thesis.

The statements said above are correct. With these considerations in mind, this work has been submitted for review as an undergraduate thesis.

Date: February 2023 Author

Dedicated to

My Grandmother

Acknowledgement

I wish to offer my thankfulness to Almighty Allah with respect and reverence for His mercy in giving me sufficient courage, knowledge, and perseverance to conduct my thesis work with proper incentive and temperament.

I'd want to show my thanks and respect for my parents, **Syed Harun** and **Redita Rahman,** and love to my sibling for letting me follow my dream and constantly motivating and supporting me throughout the four years of undergraduate education in Khulna University of Engineering & Technology (KUET).

Throughout my undergraduate thesis study, I would like to express my gratitude to **Dr. Md. Foisal Hossain, Professor, Department of ECE, KUET**, for providing me with appropriate guidance, encouragement, and positive inspiration. Without his real assistance, direction, recommendations, and support, I would not have been able to advance in this field of research.

I'd also want to thank **Dr. Md. Mostafizur Rahman, Professor, Department of ECE, KUET**, for taking the time to review my thesis. I am grateful to my respected adviser, **Dr. Sheikh Md. Rabiul Islam, Professor, Department of ECE, KUET**, for his guidance and advice during the last four years. I'd want to express my gratitude to all of the faculty members of the Department of Electronics and Communication Engineering for their assistance and encouragement in completing my thesis.

Author

Table of Contents

List of Figures

List of Tables

Abstract

Sign language plays a crucial role in bridging the communication gap between the hearing impaired and the rest of society. Despite its importance, access to sign language interpretation remains a challenge for many individuals in the deaf and dumb community. The development of advanced technology capable of recognizing sign language has the potential to revolutionize communication for this minority group. However, the complexity of sign language and the nuances of individual expressions pose significant obstacles in the creation of a reliable and effective sign language. In the field of Bengali sign language recognition, publicly available datasets as well as author curated datasets exist for Bengali sign alphabets, however, no datasets are available for the Two Hand English Manual Sign Alphabets of Bangladeshi Sign Language (BdSL). In order to address this gap, we collected 10,350 samples belonging to 25 classes and proposed a transfer learning-based Convolutional Neural Network (CNN) architecture. Our model demonstrated high accuracy in existing validated datasets, and upon application on our newly collected dataset, it was demonstrated to have a high accuracy rate. The proposed model achieved a test accuracy of 99.54%.

Chapter 1

Introduction

1.1 Overview

This research endeavors to delve into the domain of "Two Hand English Manual Sign Alphabets of BdSL" recognition and implement a Transfer Learning based Convolutional Neural Network method. The field of computer vision has been making substantial strides to aid humanity, and lately, it has been utilized to support the deaf community by facilitating sign language detection techniques. Sign languages serve as a crucial medium of communication for the deaf, and sign language detection plays the role of a digital interpreter between the deaf and hearing communities. While previous techniques have relied on supplementary equipment or image pre-processing methods, this research utilizes Transfer Learning, leveraging the advancements in machine learning and object detection. To this end, a dataset was created with images of specific signs that encompass variations and different backgrounds. This study intends to make a meaningful contribution to the area of sign language recognition by investigating the Two Hand English Manual Sign Alphabets of BdSL and developing a system using the CNN architecture.

Convolutional Neural Networks (CNNs) are widely used in sign language detection as they have proven to be effective in object detection tasks. In sign language detection, a CNN is trained to recognize hand gestures that represent various signs. The model is trained on a large dataset of images of hand gestures, which is then used to predict the sign represented in an unseen set of images. The CNN architecture processes the input image and identifies important features, such as the shape, size, and position of the hand gestures, which are then passed onto dense layers to make predictions. The use of CNNs in sign language detection has improved the accuracy of sign recognition and has surpassed traditional image processing techniques, making it a promising approach for bridging the communication gap between the hearing impaired and the rest of society. However, the low sample to label ratio and lack of publicly available datasets for some sign languages remain challenges in the development of reliable sign language detection systems.

Transfer learning is a technique that leverages the knowledge learned from a pre-trained model to a new task. In the case of sign language detection, transfer learning can help improve the performance of the model by fine-tuning the pre-trained weights on a small dataset of sign language gestures. By using the features learned from a large dataset in a similar domain, the pretrained model can provide a better starting point for the new task, reducing the amount of data and computational resources required to train a model from scratch. Additionally, transfer learning can also help address the problem of limited samples to label ratio in sign language detection, allowing the model to generalize better to new gestures.

In the design of a Convolutional Neural Network (CNN) architecture for sign language detection, there are several layers that are crucial to the functioning of the network. These layers are as follows:

Input Layer: The input layer is responsible for accepting the image data as input and passing it on to the subsequent layers.

Convolutional Layer: The convolutional layer is the core component of a CNN. It employs convolutional filters to scan the input image and identify the relevant features. Additionally, normalization layers are also incorporated in this layer to improve the performance of the network.

Pooling Layer: The pooling layer reduces the spatial dimensions of the input data, preserving only the most salient information. This layer is used to overcome the issue of overfitting and reduces the computation required by the network.

Fully Connected Layer: The fully connected layer is the layer in which all the neurons in the previous layer are connected to the neurons in this layer. This layer performs the final classification of the input image based on the information obtained from the previous layers.

Output Layer: The output layer is responsible for producing the final prediction of the network, classifying the input image as one of the classes in the dataset.

It is worth mentioning that, in some cases, dropout layers may also be incorporated in the network

design to address overfitting and improve the generalization ability of the network.

In conclusion, the architecture of a CNN for sign language detection is composed of a series of interconnected layers that work together to identify and classify sign language gestures in images.

1.2 Background

In the past, various techniques have been developed to detect sign language alphabets. One of the earliest techniques used was machine learning classifiers, such as Support Vector Machines (SVM), to classify hand gestures into specific sign alphabets. However, this method had limitations in its ability to detect more complex and subtle gestures, leading to a low accuracy in recognition.

Later on, techniques like Principal Component Analysis (PCA) and Hidden Markov Models (HMMs) were introduced to tackle the limitations of SVMs. Although PCA was effective in reducing the dimensionality of the data, it did not account for the spatial relationship between the features, leading to a loss of important information. On the other hand, HMMs were capable of modeling sequential data, but were computationally expensive and required a large amount of training data.

With the advancement of computer vision and deep learning, Convolutional Neural Networks (CNNs) have become popular in sign language detection. CNNs can automatically extract high-level features from raw images and are able to capture the spatial relationship between pixels. This has significantly improved the accuracy of sign language recognition compared to the previous methods.

However, the use of CNNs for sign language detection also has its own set of challenges, such as the limited amount of labeled training data and the need for large amounts of computational resources. Additionally, the highly variable nature of sign language gestures and the different ways they can be performed by different individuals adds an additional layer of difficulty in accurately detecting sign alphabets

1.3 Motivation

The detection of two hand English manual sign language has become increasingly important in recent years with the rise of technology and its integration into the field of computer vision. With the goal of bridging the communication gap between the hearing and the deaf community, there has been a growing interest in the development of sign language detection techniques. Despite the advancements in computer vision and machine learning, the detection of sign language remains a challenging task, particularly in the case of two hand English manual sign language. The traditional image processing techniques used in sign language detection suffer from limitations such as low accuracy, dependence on lighting conditions and complex pre-processing procedures. Currently, there is a lack of publicly available datasets and tools to work with, which has limited the progress in this field. Moreover, the low sample to label ratio presents a significant challenge for the development of an accurate sign language detection system. Despite these limitations, it is vital to address this problem as the detection of sign language can play a crucial role in improving the lives of deaf individuals. By creating a custom dataset and developing a Transfer learning based Convolutional Network.

To overcome these challenges, there has been a shift towards using deep learning techniques such as Convolutional Neural Networks (CNNs) in sign language detection. The use of CNNs has shown significant improvements in accuracy and robustness compared to traditional methods. As a result, the development of a two hand English manual sign language detection system using transfer learning and CNNs has become a topic of high interest and relevance in the field of computer vision and machine learning.

1.4 Objectives

The primary objectives of the thesis are:

- Creation of a comprehensive dataset of "Two Hand English Manual Sign Alphabets".
- Proffer an innovative Convolutional Neural Network (CNN) architecture to cater to the needs of sign language detection.
- Integration of the proposed CNN architecture with a pre-trained model to enhance the performance and accuracy of the system.

1.5 Thesis Orientation

In this thesis, some new contributions are proposed that could improve the stock price prediction model. The thesis is organized as follows:

Chapter 2 A brief overview of previous research on the utilization of deep learning nad classical Machine Learning for recognition of BdSL is provided. Additionally, a concise comparison of relevant literature is included.

Chapter 3 New model architecture is explained in this chapter. This chapter presents details of proposed model with the use of relevant libraries, functions, optimizers and loss functions.

Chapter 4 The proposed model architecture is thoroughly detailed in this chapter, encompassing the utilization of various libraries, functions, optimizers, and loss functions. The chapter presents a comprehensive examination of the proposed model and its various components. The discussion encompasses the employment of relevant tools and techniques to achieve the desired outcome. performance and details of simulations result and discussions are included in this chapter.

Chapter 5 In this chapter, the conclusion, the advantage of the proposed scheme and future works are analyzed.

Chapter 2

Literature Review on Sign Language Recognition

2.1 Overview

This chapter delves into the recent advancements in the field of Deep Learning and Machine Learning for sign language detection. A comprehensive discussion of related works is presented, with a focus on the comparison between the proposed method and existing literature. Sign language recognition has been a subject of interest in the academic and humanitarian spheres for many years, however, it remains a challenging task due to its dynamic and complex nature. Despite the difficulties, numerous researchers have made significant contributions to the field by proposing various methods for sign language recognition. The chapter provides valuable insights into the current state of the field and the progress that has been made in recent years. recent works regarding Deep learning for stock price prediction are discussed in this chapter.

2.2 Recent Work in Sign Language Recognition

In the field of sign language recognition, several groundbreaking studies have been conducted over the years. In 1998, Thad Starner et al [1] developed a real-time American Sign Language (ASL) recognition system using desk and wearable computer-based video. Starner et al [2] also developed a real-time ASL recognition system from video using Hidden Markov Models, which was introduced in the 1997 study of motion-based recognition. In 1999, Christian Vogler et al [3] utilized parallel Hidden Markov Models for ASL recognition. The same year, Kazuyuki Imagawa et al [4] created a color-based hand tracking system for sign language recognition. In 2002, Nobuhiko Tanibata et al [5] focused on the extraction of hand features for recognition of sign language words. In the same year, S. Rahman et al [6] developed an "Intelligent Assistant for Speech Impaired People." P. Subha Rajam et al [7] created a real-time Indian Sign Language recognition system to aid the deaf-dumb community in 2011. In the same year, Zahoor Zafrulla [8] introduced an ASL recognition system using the Kinect. Joyeeta Singha et al [9] created an Indian Sign Language recognition system using an eigenvalue weighted Euclidean distance-based classification technique in 2013. In 2014, R. Girshick, et al [10] developed "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." In 2015, S. M. K. Hasan et al [11] introduced a "New Approach of Sign Language Recognition System for Bilingual Users." R. Girshick [12] also created "Fast R-CNN" that same year. M. A. Rahaman et al [13] developed "Real-Time Bengali and Chinese Numeral Signs Recognition Using Contour Matching" and a

"Real-Time Hand-Signs Segmentation and Classification System Using Fuzzy Rule Based RGB Model and Grid-Pattern Analysis" in 2015[14]. M. A. Rahaman et al [15] also created a "Computer Vision-Based Bengali Sign Words Recognition Using Contour Analysis" in 2015. In the same year, S. Ren et al [16] developed "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." S. T. Ahmed et al [17] introduced "Bangladeshi Sign Language Recognition Using Fingertip Position" in 2016. In 2018, Oishee Bintey Hoque et al [18] developed a real-time Bangladeshi Sign Language Detection using Faster R-CNN. In 2018, Sanzidul Islam et al [19] developed a robust model for recognizing Bangla sign language digits using convolutional neural networks. In the same year, Shirin Sultana Shanta et al [20] presented a solution for Bangla sign language detection utilizing SIFT and CNN. Furthermore, F. Yasir et al [21] published a study titled "Bangla Sign Language Recognition using Convolutional Neural Network." In 2019, Abdul Muntakim Rafi et al [22] created an image-based recognition system for the Bengali sign language alphabet to support the deaf and dumb community.

2.3 Conclusion

The literature review highlights the shift towards computer vision-based approaches for sign language recognition. These methods leverage advancements in computer vision and deep learning to analyze and interpret visual information in real-time, providing a more natural and intuitive experience for users. Computer vision-based models also benefit from large training datasets and improved accuracy and generalization. Overall, the literature review highlights the growing importance of computer vision-based approaches for sign language recognition.

Chapter 3

Methodology

3.1 Overview

This chapter contains an explanation of the methods employed in this work. The step-by-step techniques for executing and attaining the final step are outlined below, along with process diagrams. The general architecture that has been presented is described below, along with its application and needs throughout the work.

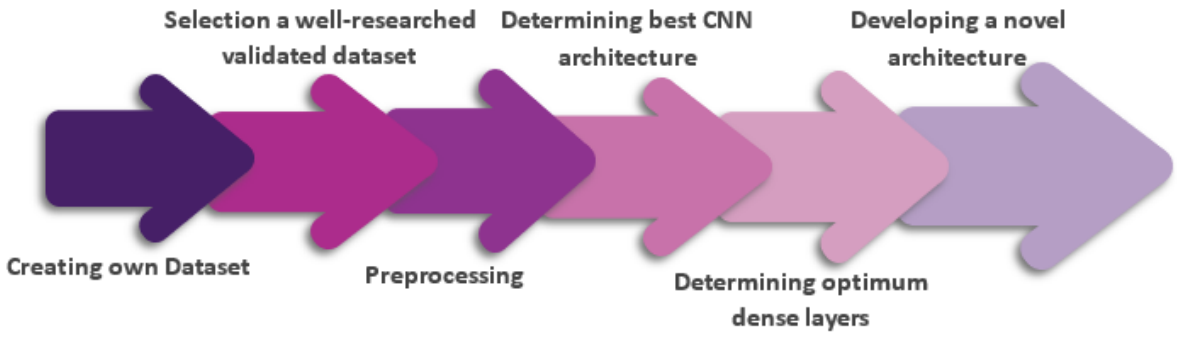


Figure 3.1: Block diagram of step by step working procedure

3.2 Environment Setup:

3.2.1 Environments:

3.2.1.1 Kaggle:

Kaggle is a platform for data science and machine learning that offers a cloud-based environment for running Python code. The Kaggle Python environment is a Jupyter Notebook-based platform that provides users with pre-installed libraries and packages for data analysis and machine learning. It also allows users to access a large database of datasets and kernels (code snippets) created by other users.

One of the main benefits of the Kaggle Python environment is that it is designed to be user-friendly and easy to use. It eliminates the need for users to set up their own environment and install the required libraries and packages, saving them time and effort. Additionally, the platform provides

access to powerful computing resources and GPUs, allowing users to run their code quickly and efficiently.

Historical data, in a broad sense, is information obtained about earlier events and situations that are important to a certain subject. Historical data, by definition, includes any data generated within a company, whether manually or automatically. The Dhaka Stock Exchange's historical dataset from 2016 to 2021 was used in this study.

GPU and TPUs available:

- GPU T4 x 2
- GPU P100
- TPU v3-8
- TPU VM v3-8

3.2.1.2 Google Colab:

Google Colab is a free, cloud-based platform for machine learning and data science. It is provided by Google and is based on the popular Jupyter Notebook environment. With Colab, users can write and run Python code in the cloud, eliminating the need to set up a local development environment or to invest in hardware.

One of the main benefits of Google Colab is that it provides access to powerful computing resources, including GPUs and TPUs (Tensor Processing Units), which can be used to run complex and computationally intensive tasks. This makes it possible for data scientists and machine learning practitioners to work with large datasets and to train sophisticated models without the need for expensive hardware.

3.2.1.3 Local Environment:

The local machine in use is equipped with the Windows 10 operating system and boasts top-of-the-line hardware specifications, including a NVIDIA GeForce RTX 3070 Ti graphics processing unit, an AMD Ryzen 5600X central processing unit, 16 gigabytes of RAM memory, and a 512 gigabyte solid-state drive. These components work together to provide optimal performance and efficiency for demanding tasks. This advanced setup is ideal for conducting cutting-edge research and development projects in the field of data science and machine learning.

3.2.2 Required libraries:

- **TensorFlow** - A powerful open-source software library for machine learning and numerical computation. It provides support for both building and training models.
- **NumPy** - A fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices of numeric data, as well as tools for integrating C, C++ and Fortran code.
- **Pandas** - A library for data manipulation and analysis in Python. It provides data structures for efficiently storing large datasets and tools for working with them, including data alignment, cleaning, and transformation.
- **Shutil** - A standard library in Python, it provides a collection of high-level file operations, including copying files and directories.
- **Time** - A standard library in Python, it provides a collection of functions to work with time values.
- **scikit-learn (sklearn)** - A popular open-source machine learning library for Python. It provides a range of algorithms for regression, classification, clustering, and dimensionality reduction.
- **OS** - A standard library in Python, it provides a way of using operating system dependent functionality.
- **Seaborn** - A visualization library in Python, built on top of Matplotlib. It provides high-level interface for creating statistical graphics.

- **PIL (Pillow)** - A library in Python for image processing. It provides support for opening, manipulating and saving a wide range of image file formats.
- **Matplotlib** - A low-level graph charting toolkit for Python that also serves as a visualization tool. Matplotlib was created by John D. Hunter. Matplotlib is a free and open-source plotting library. For platform portability, Matplotlib is mostly written in Python, with a few pieces written in C, Objective-C, and Javascript.

3.3 Proprietary Dataset Creation:

3.3.1 Sample Collection:

The dataset was meticulously prepared under the auspices of the Bangladesh National Federation of the Deaf, in accordance with their established "Ishara Vashar Obhidhan" booklet. Utilizing the openCV library, images were collected with a resolution of 800x800 pixels, incorporating three diverse background types for augmentation purposes. The first background type was a solid color, while the second was a patterned background. Additionally, to introduce variability and diversity, the brightness levels and positional translations of the images were varied during the capturing process.

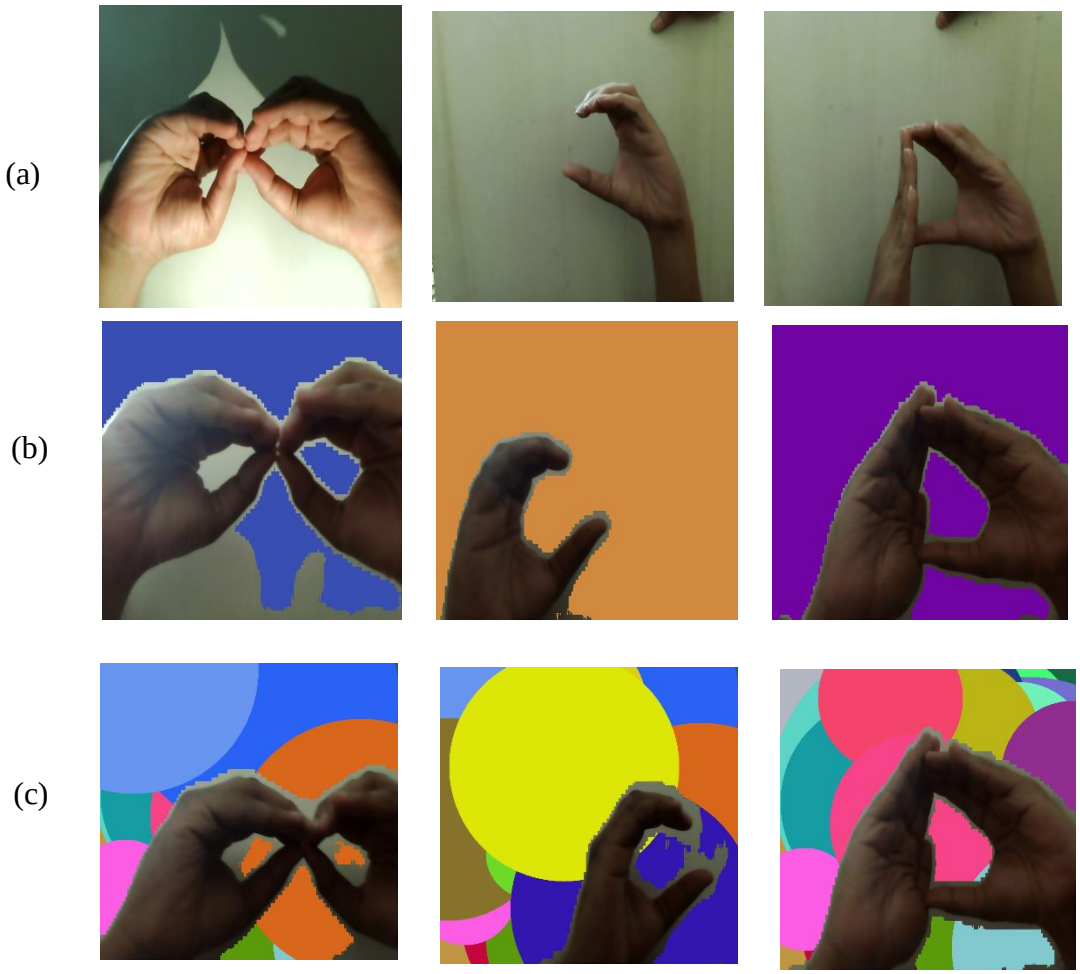


Fig 3.3.1 (a): Plain Whit Background Images (b) Solid Color Background Images (c) Patterned Background Images

3.3.2 Data Preprocessing:

Initially, to enhance the quality of the images, three different filtering techniques (Gaussian, Median, and Bilateral) were applied. The outputs were then closely analyzed for edge detection. After careful observation, the filter that was able to effectively suppress inner edges while preserving the outer edges was selected for further processing. This methodology aimed to provide a clearer representation of the image and reduce unnecessary inner edges content and to improve the accuracy of the subsequent sign detection.

- **Gaussian Filter:** The Gaussian filter is a smoothing filter used in image processing and computer vision. It works by replacing each pixel value with a weighted average of its neighboring pixel values, where the weight is determined by a Gaussian function. The Gaussian filter is commonly used to reduce image noise and to smooth image features, without significantly altering the overall shape or structure of the image. The filter's parameters, such as the standard deviation and kernel size, can be adjusted to balance the trade-off between smoothing and preservation of image details. Additionally, Gaussian filters can be applied multiple times to achieve a more pronounced smoothing effect.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- **Median Filter:** Median filter works by replacing each pixel value with the median value of its neighboring pixels. The median filter is commonly used to remove salt-and-pepper noise, which is a type of impulse noise that occurs as randomly-occurring white and black pixels in an image. Unlike linear filters, such as the Gaussian filter, the median filter is not sensitive to outliers and tends to preserve the overall shape and structure of the image. The filter's parameters, such as the kernel size, can be adjusted to control the size of the neighborhood used to compute the median value for each pixel. The median filter is particularly useful for removing noise from images with a large dynamic range and for images with impulse noise that cannot be effectively reduced by linear filters.

$$\hat{f}(x,y) = \underset{(st)\in S_{xy}}{median}\{g(s,t)\}$$

- **Bilateral Filter:** It works by replacing each pixel value with a weighted average of its neighboring pixels, where the weight is determined by both the spatial distance and the intensity difference between the center pixel and its neighbors. The bilateral filter is commonly used to reduce noise and smooth image features, while preserving the edges and fine details in the image. Unlike linear filters, such as the Gaussian filter, the bilateral filter takes into account both the spatial and intensity relationships between pixels, making it effective at preserving

the structure and color of the image. The filter's parameters, such as the standard deviation and kernel size, can be adjusted to control the degree of smoothing and preservation of image details. The bilateral filter is particularly useful for images with a large dynamic range and for images with noise that cannot be effectively reduced by linear filters.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|)\, G_{\sigma_r}(|I_p - I_q|) I_q$$

(1)

(2)

(3)

(a) (b) (c)

Fig 3.3.2.1: (a) Gaussian filtered image (1) 3x3 (2)5x5 (3)7x7 kernels(b) Median Filtered Images (1)3x3 (2)5x5 (3)7x7 kernels
(c) Bilateral Filtered Images

- **Canny Edge Detector:** Canny edge detection is a popular edge detection algorithm used in image processing and computer vision. It works by detecting the changes in intensity between pixels in an image and identifying the pixels that have a large gradient magnitude, which are considered to be the edges in the image. The algorithm consists of several stages, including noise reduction, gradient calculation, non-maximum suppression, and hysteresis thresholding. The gradient calculation stage is used to calculate the gradient magnitude and direction for each pixel in the image, and the non-maximum suppression stage is used to thin the edges and remove false edges. The hysteresis thresholding stage is used to determine which edges are strong and which are weak, and only the strong edges are kept in the final output. The Canny edge detection algorithm is known for its ability to effectively detect edges in images with a low amount of noise, and for its good trade-off between edge detection accuracy and computational efficiency.

  The Canny edge detection algorithm consists of the following steps:

  1.Noise reduction: This stage is used to reduce the amount of noise present in the image, which can interfere with the edge detection process. The image is typically smoothed using a Gaussian filter to reduce noise.

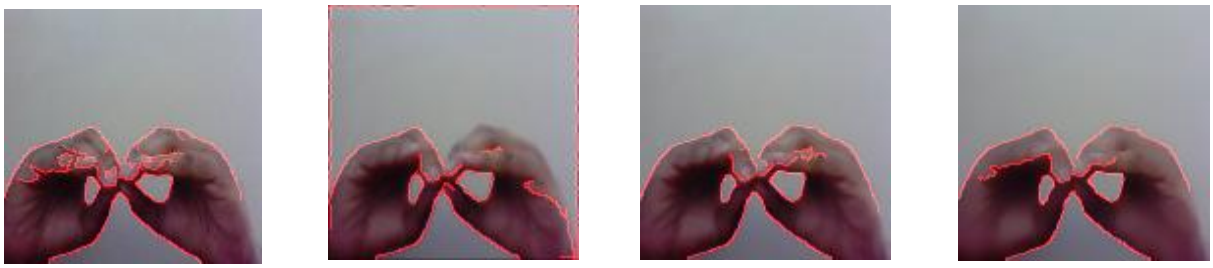
  2.Gradient calculation: This stage is used to calculate the gradient magnitude and direction for each pixel in the image. The gradient magnitude represents the rate of change of intensity between pixels, and the gradient direction represents the direction of the intensity change.

  3.Non-maximum suppression: This stage is used to thin the edges and remove false edges. The gradient magnitude and direction are used to determine whether each pixel is a local maximum in the gradient direction, and only the local maxima are kept in the output.

  4.Hysteresis thresholding: This stage is used to determine which edges are strong and which are weak. The edges are thresholded based on the gradient magnitude, and only the edges with a gradient magnitude above a high threshold are considered to be strong edges. The edges with a gradient magnitude below a low threshold are discarded, and the edges with a gradient magnitude between the two thresholds are kept only if they are connected to a strong edge.

5.Edge tracing: This stage is used to trace the strong edges and output the final edge map. The strong edges are traced from one end to the other, following the gradient direction, and the final edge map is a binary image where the strong edges are represented by white pixels and the background is represented by black pixels.

Overall, the Canny edge detection algorithm is a multi-step process that uses image intensity information to effectively detect edges in an image, while minimizing the impact of noise and false edges.



(a) (b) (c) (d)

Fig 3.2.2.2: Edges of (a) Raw Image (b)Gaussian Filtered (c)Median Filtered (d)Bilateral Filtered

3.3.3 Data Splitting:

The overall dataset was divided into three sub-datasets for optimal organization and utilization. The first sub-dataset, referred to as the “Dummy” dataset, comprised of images featuring both solid white and patterned backgrounds. To ensure the validity and accuracy of the model, the dummy dataset was split into training and validation subsets, with a total of 7,300 images. The second sub-dataset, referred to as the test dataset, was specifically comprised of images with a solid background, with a total of 3,075 samples. This sub-dataset included 123 samples per label, providing comprehensive representation and evaluation of the model's performance.

	Splits	Sub Splits	Samples	Samples/Label
Root Dataset	Dummy	Train	5840	233
		Validation	1460	58
	Test	Test	3075	123

Table 3.3: Data splitting of proprietary dataset

3.4 Selecting a validated Dataset:

Dataset/Authors	Labels	Samples/Label	Accuracy(%)
D1500 [23]	87	1517	99.99
Rafi_BdSL [22]	38	320	91.51*[28]
IsharaLipi [24]	36	50	94.74
Md. S. Islalm et al.[25]	45	688	99.80
Rahman et al. [26]	36	100	96.46
BdSLImset [18]	36	1111	98.20
BdSL 49 [27]	49	600	93

Table 3.4: List of validated datasets

3.5 Determining CNN Architecture:

3.5.1 EfficientNetB0:

EfficientNetB0 is a deep learning architecture designed to optimize both accuracy and computational efficiency in image classification tasks. It uses a combination of depth-wise separable convolutions and efficient linear layers to create a model that is both accurate and computationally efficient.

EfficientNetB0 is particularly well-suited for mobile and embedded devices with limited computational resources, where reducing the number of parameters is critical. However, it may not perform as well as larger, more complex models in tasks that require more intricate features, such as object detection and segmentation.

Here are the key features of EfficientNetB0:

1.Total parameters: 5.3 million

2.Trainable parameters: 5.3 million

3. Number of Convolution Layers: 45

Overall, EfficientNet-B0 is a great option for a wide range of image classification tasks due to its balanced combination of accuracy and computational efficiency.

3.5.2 ResNet50:

ResNet50 is a deep residual neural network architecture designed for image classification tasks. It was introduced in 2015 and has since become one of the most widely used architectures for image classification, due to its high accuracy and ability to effectively learn and tackle complex features in images.

ResNet50 uses residual connections, where the output of one layer is directly added to the input of another layer, allowing the network to better preserve the information learned in earlier layers and

thus reduce the impact of vanishing gradients. This allows the network to effectively learn very deep architectures, which is one of the key factors contributing to its high accuracy.

While ResNet50 has a relatively large number of parameters compared to other architectures, it is still considered computationally efficient and can be trained on a single GPU in a relatively short amount of time. However, it may not be the best choice for edge devices with limited computational resources.

Here are the key features of ResNet50:

1.Total parameters: 25.6 million

2.Trainable parameters: 25.6 million

3. Number of Convolution Layers: 168

Overall, ResNet50 is a powerful and widely used architecture for image classification tasks and has achieved state-of-the-art results on many benchmark datasets. Its residual connections and ability to learn deep architectures make it a great choice for high-accuracy image classification tasks.

3.5.3 MobileNet:

MobileNet is a light-weight deep neural network architecture designed for image classification tasks on mobile and embedded devices. It uses depth wise separable convolutions to reduce the number of parameters, making it suitable for deployment on devices with limited computational resources.

Here are the key features of MobileNet:

1.Total parameters: 4.2 million

2.Trainable parameters: 4.2 million

3.Number of Convolution Layers: 28

MobileNet is designed to balance accuracy and computational efficiency, making it a good choice for image classification tasks on mobile and embedded devices. With 28 convolution layers, it is able to effectively learn and tackle simple features in images, while still being computationally efficient.

3.5.4 MobileNetV2:

MobileNetV2 is a deep learning architecture designed for image classification tasks on mobile and embedded devices. It is an updated version of the original MobileNet architecture, designed to further improve accuracy and computational efficiency.

Here are the key features of MobileNetV2:
1.Total parameters: 3.5 million
2.Trainable parameters: 3.5 million
3.Number of Convolution Layers: 17

MobileNetV2 uses depthwise separable convolutions and linear bottlenecks to reduce the number of parameters and computational cost, making it suitable for deployment on devices with limited computational resources. The updated architecture also includes several modifications, such as the use of linear bottlenecks, that help to improve accuracy.

Overall, MobileNetV2 is a great choice for image classification tasks on mobile and embedded devices due to its light-weight design and balance between accuracy and computational efficiency.

3.5.5 AlexNet:

AlexNet is a deep learning architecture designed for image classification tasks. It was introduced in 2012 and was one of the first architectures to demonstrate the potential of deep learning in computer vision tasks.

Here are the key features of AlexNet:
1.Total parameters: 60 million
2.Trainable parameters: 60 million
3.Number of Convolution Layers: 5

AlexNet uses a combination of convolutional and dense layers to learn and classify images. Its architecture includes five convolutional layers, several pooling layers, and three dense layers. Despite its relatively simple architecture, AlexNet was able to demonstrate the potential of deep learning in image classification tasks and was a key step in the development of deeper and more sophisticated architectures.

3.6 Proposed Model:

The proposed model in this code is a deep learning architecture based on the MobileNet network. MobileNet is a lightweight convolutional neural network designed to run efficiently on mobile de vices. In this model, the MobileNet architecture is used as the base model and its layers are initial ized with pre-trained weights from the ImageNet dataset. The base model is then fine-tuned to fit the specific task at hand.

The output from the base model is then passed through multiple Convolutional layers, each follo wed by Batch Normalization, ReLU activation and Dropout layers. The Convolutional layers incr ease the complexity of the model, allowing it to learn higher-level features from the input image. The Batch Normalization layers are used to stabilize the training of the model, while the ReLU ac tivation functions introduce non-linearity into the model. Dropout is used to prevent overfitting b y randomly dropping out neurons during training.

The output of the Convolutional layers is then flattened and passed through multiple fully connec ted (Dense) layers. The Dense layers increase the capacity of the model to learn complex non-lin ear relationships between the input and output. The same set of operations (Batch Normalization, ReLU activation and Dropout) is performed after each Dense layer to ensure the stability and gen eralization of the model.

Finally, the output of the last Dense layer is passed through a softmax activation function to prod uce 25 class probabilities, which can be used to make predictions on the input image. The propos ed model is implemented using the Keras library and can be easily fine-tuned for different tasks a nd datasets.

MobileNet Architecture

conv2d (Conv2D)	(None, 7, 7, 32)	294944
conv2d_1 (Conv2D)	(None, 7, 7, 64)	18496
batch_normalization	(None, 7, 7, 64)	256
activation (Activation)	(None, 7, 7, 64)	0
dropout (Dropout)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	204928
batch_normalization_1	(None, 7, 7, 128)	512
activation_1 (Activation)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
conv2d_3 (Conv2D)	(None, 7, 7, 512)	590336
batch_normalization_2	(None, 7, 7, 512)	2048
activation_2 (Activation)	(None, 7, 7, 512)	0
dropout_2 (Dropout)	(None, 7, 7, 512)	0
conv2d_4 (Conv2D)	(None, 7, 7, 512)	2359808
batch_normalization_3	(None, 7, 7, 512)	2048
activation_3 (Activation)	(None, 7, 7, 512)	0
dropout_3 (Dropout)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
batch_normalization_4 (Batch	(None, 256)	1024
activation_4 (Activation)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131584
batch_normalization_5 (Batch	(None, 512)	2048
activation_5 (Activation)	(None, 512)	0

dropout_5 (Dropout) (None, 512) 0

dense_2 (Dense) (None, 64) 32832

batch_normalization_6 (Batch (None, 64) 256

activation_6 (Activation) (None, 64) 0

dense_3 (Dense) (None, 25) 1625

==========

Total params: 13,294,393
Trainable params: 13,268,409
Non-trainable params: 25,984

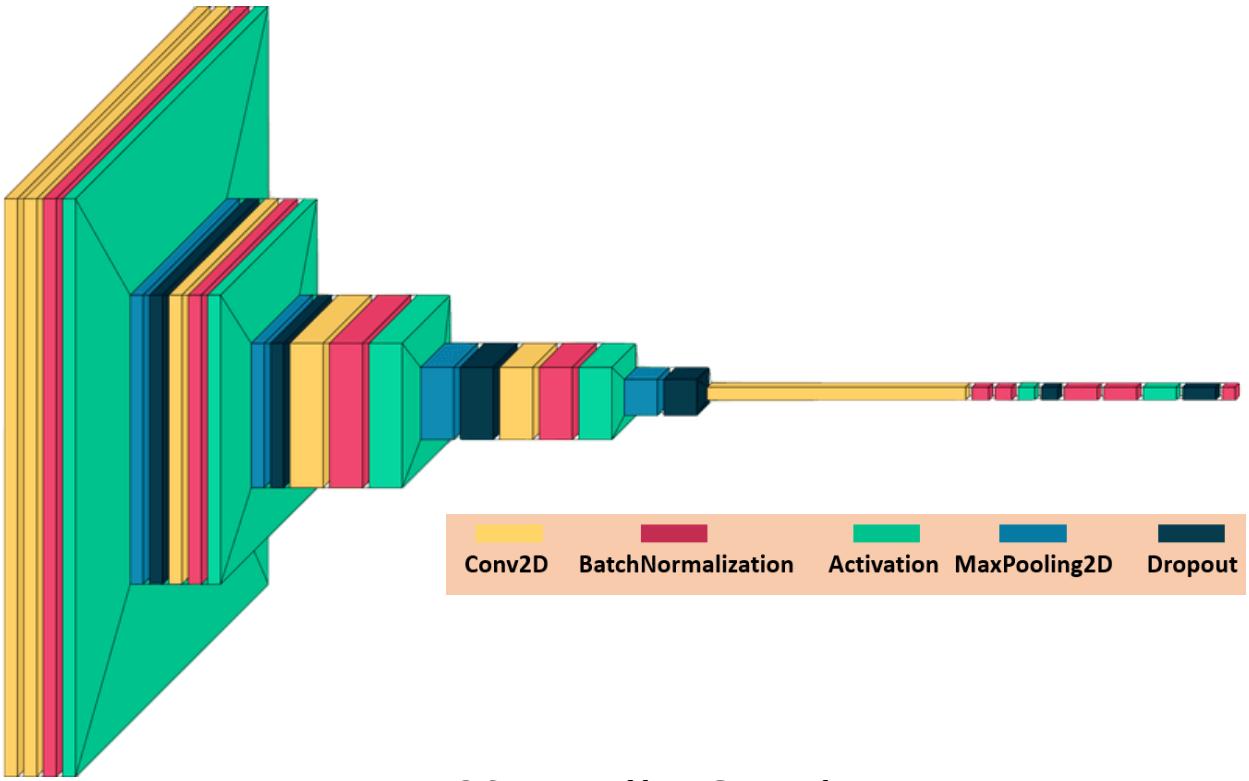


Fig 3.6: Proposed base CNN architecture

3.6.1 Convolutional Layer:

A convolutional layer is a fundamental component in Convolutional Neural Networks (CNNs), which are a type of deep learning model commonly used in computer vision tasks. The main purpose of a convolutional layer is to extract features from input images and identify local patterns within the data. Convolutional layers use a set of filters, also called kernels, to perform convolution operations on the input data, where each filter slides over the input and performs an element-wise dot product with the input at each location. In this study, a kernel size of (3,3) was used with a stride of (1,1). The resulting output feature map is then passed through a non-linear activation function, such as ReLU, to introduce non-linearity into the model. The use of convolutional layers allows the model to learn hierarchical representations of the input data, which are then used to make predictions. Convolutional layers are crucial to the success of CNNs and have proven to be effective in a wide range of computer vision tasks, such as image classification, object detection, and semantic segmentation.

$$x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j].\, h[m-i, n-j]$$

3.6.2 Batch Normalization Layer:

Batch normalization is a technique used to improve the training stability and reduce overfitting in deep learning models. It normalizes the activations of each batch of inputs in a neural network, making the distribution of activations more stable and less dependent on the initialization of the weights.

The process of batch normalization can be described as follows:

1.At each training iteration, the activations of a batch of inputs are calculated and their mean and variance are estimated.

2.The activations are then normalized by subtracting the batch mean and dividing by the batch standard deviation.
3.The normalized activations are then scaled and shifted by two learned parameters, gamma and beta, respectively. These parameters are trained as part of the network and allow the model to adapt to the normalized activations.
4.The batch-normalized activations are then used as inputs to the next layer of the network.

By normalizing the activations, batch normalization helps to reduce the internal covariate shift, which is a phenomenon that occurs when the distribution of activations changes as the weights of the network are updated. This makes the training process more stable and can help to reduce overfitting, allowing the network to learn faster and achieve better performance on unseen data. Batch normalization is typically applied to the activations of hidden layers in the network, although it can also be used in other parts of the network, such as in the input layer or in the output layer.

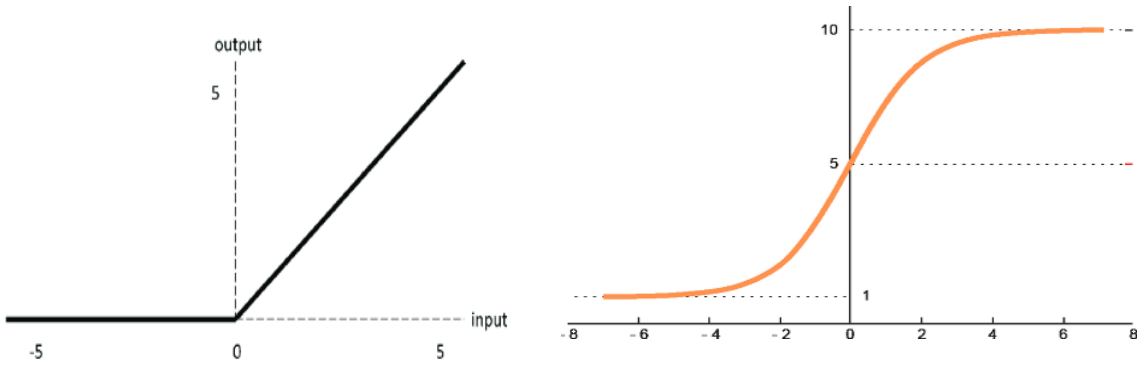
3.6.3 Activation Layer:

The activation layer is a crucial component in deep learning models, as it introduces non-linearity into the model. The activation function is applied element-wise to the output of each node in the layer, allowing the model to make complex, non-linear predictions.

The ReLU (Rectified Linear Unit) activation function is one of the most commonly used activation functions in deep learning. It replaces all negative inputs with zero and leaves positive inputs unchanged. The ReLU activation function is computationally efficient, as it only requires a simple comparison operation, and it has been shown to work well in practice for a wide range of tasks.

The softmax activation function is typically used in the output layer of a deep learning model for multi-class classification problems. It takes a vector of real values as input and converts it into a probability distribution over the possible classes. The softmax activation function is defined as the exponential of each input value divided by the sum of exponentials of all input values. The result is a vector of values that sum to 1 and represent the model's predicted probability for each class. The class with the highest predicted probability is used as the model's final prediction.

In summary, activation functions play a crucial role in deep learning models by introducing non-linearity and allowing the model to make complex predictions. The ReLU and softmax activation functions are two of the most commonly used activation functions in deep learning, each with specific applications in different parts of the network.



(a) (b)

Fig 3.6.3: (a) ReLu Activation (b)SoftMax Activation

3.6.4 Max Pooling Layer:

Max pooling is a common technique used in Convolutional Neural Networks (CNNs) to down-sample the spatial dimensions of the feature maps. It is typically applied after the convolutional layer and before the activation function.

The process of max pooling can be described as follows:

1.The input feature map is divided into non-overlapping regions, typically squares or rectangles, called pooling windows.
2.For each pooling window, the maximum activation value is selected.
3.The maximum activation values are then concatenated to form the output feature map.

By down-sampling the spatial dimensions of the feature maps, max pooling has several benefits.

First, it reduces the number of parameters and computation required by the network, allowing it to learn more efficiently. Second, it helps to reduce overfitting by providing a form of spatial invariance, meaning that the network is less sensitive to small translations or distortions of the input. Finally, max pooling helps to capture the most important information in the feature maps and pass it on to the next layer of the network.

In summary, max pooling is a common technique used in CNNs to reduce the spatial dimensions of the feature maps and improve the efficiency and performance of the network.

3.6.6 Dropout Layer:

Dropout is a regularization technique used in deep learning to prevent overfitting and improve the generalization of the model. It works by randomly dropping out (setting to zero) a certain fraction of the neurons during training, forcing the network to learn more robust representations of the input data.

The dropout layer is typically added after the activation function and before the dense (fully connected) layer. During training, the dropout layer sets a random subset of the neurons to zero with a certain probability (the dropout rate), effectively reducing the size of the network. At each iteration of training, a different subset of neurons is dropped out, making the network more robust to changes in the input data.

At test time, the dropout layer is typically turned off, and all neurons are used in the prediction. This results in a different network architecture at each iteration of training and at test time, providing a form of ensemble learning that can improve the generalization of the model.

In summary, dropout is a regularization technique used in deep learning to prevent overfitting and improve the generalization of the model. The dropout layer works by randomly dropping out neurons during training, forcing the network to learn more robust representations of the input data.

3.7 Prediction:

The **model.evaluate()** method in TensorFlow is an important tool for assessing the accuracy and performance of machine learning models. It allows data scientists and machine learning engineers to evaluate the model's prediction against actual results to determine its effectiveness.

The **model.evaluate()** function takes two inputs: the data that the model should make predictions on and the actual labels or target values of that data. The model then generates predictions based on its training and returns several evaluation metrics that provide insight into its performance. Some of the most commonly used evaluation metrics include accuracy, loss, precision, recall, F1-score, and more.

Accuracy represents the proportion of correct predictions made by the model. Loss is a measure of how far the model's predictions are from the actual results. Precision is a measure of the proportion of correct positive predictions made by the model, while recall is a measure of the proportion of actual positive results that were correctly predicted by the model. The F1-score is a weighted average of precision and recall that provides a single metric to summarize the model's performance.

It is important to note that the **model.evaluate()** function works on a fixed dataset and provides an evaluation of the model's performance at a single point in time. This can provide a baseline assessment of the model's performance, but it is also necessary to perform additional evaluations such as cross-validation to gain a more comprehensive understanding of the model's accuracy and performance.

Overall, the **model.evaluate()** function is a crucial component of the machine learning process, as it provides valuable information about the model's ability to make accurate predictions and helps data scientists and machine learning engineers to fine-tune and improve their models.

3.8 Optimizer:

Optimizers are algorithms used in machine learning to update the model parameters in such a way as to minimize the loss function. The loss function measures the discrepancy between the predicted output and the true output for a given set of inputs. The goal of an optimizer is to find the best set of parameters that results in the lowest loss. There are several types of optimizers available, each with its own unique strengths and weaknesses.

Some of the most commonly used optimizers include:

Stochastic Gradient Descent (SGD): This is a simple optimization algorithm that uses the gradient of the loss function to update the model parameters. SGD is known for its efficiency and is often used for large-scale machine learning problems. However, it can be prone to getting stuck in local minima.

Momentum: Momentum is a variant of SGD that takes into account the direction in which the parameters are being updated. This can help the optimizer escape from local minima and converge more quickly to the global minimum.

Adagrad: Adagrad is an optimizer that adapts the learning rate for each parameter individually. This means that parameters that receive more updates have a lower learning rate, while parameters that receive fewer updates have a higher learning rate. This helps the optimizer converge more quickly.

Adadelta: Adadelta is another adaptive learning rate optimizer, similar to Adagrad. However, it uses an exponentially weighted moving average of the gradients, instead of keeping a running sum of the squared gradients.

Adam: Adam is a popular optimizer that combines the best aspects of both Adagrad and Momentum. It uses adaptive learning rates and momentum to help the optimizer converge more quickly and smoothly.

In general, the choice of optimizer depends on the specific problem and the desired trade-off between speed and accuracy. SGD is a good choice for large-scale problems, while more sophisticated optimizers such as Adam may be preferred for smaller, more complex problems where accuracy is a higher priority. Ultimately, the best optimizer is one that is able to quickly and reliably find the optimal set of parameters for a given problem.

3.8.1 Adaptive Moment Estimation (Adam):

The Adam Optimizer is a popular optimization algorithm for deep learning models. It is an extension of the stochastic gradient descent (SGD) optimization algorithm and is well-suited for training deep neural networks. The Adam Optimizer is known for its robustness and has been widely used in practice for many different tasks, including image classification, language translation, and speech recognition.

At a high level, the Adam Optimizer works by maintaining two moving averages of the gradient information, which it uses to adapt the learning rate for each parameter in the model. The first average, known as the "momentum", keeps track of the average of the past gradients and helps to smooth out the optimization process. The second average, known as the "running variance", keeps track of the average of the past squared gradients and helps to stabilize the optimization process. The Adam Optimizer uses these two averages to calculate an adaptive learning rate for each parameter in the model, which it then uses to update the parameters in each iteration of training.

The equations used by the Adam Optimizer to update the parameters are as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) * g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) * g_2^2$$
$$\widehat{m} = \frac{m_t}{(1 - \beta_1^t)}$$
$$\hat{y} = \frac{y_t}{(1 - \beta_2^t)}$$

$$params = params - \alpha * \frac{\hat{m}}{\sqrt{\hat{y}} + \varepsilon}$$

where m_t and v_t are the momentum and running variance estimates, respectively, β_1 and β_2 are hyperparameters that control the decay rates for the momentum and running variance estimates, $\hat{m}$ and $\hat{v}$ are the bias-corrected momentum and running variance estimates, learning_rate is a hyperparameter that controls the step size of the update, epsilon is a small value added to the denominator to prevent division by zero, and params is the vector of model parameters being optimized.

In summary, the Adam Optimizer is a fast and effective optimization algorithm for deep learning models. It combines the strengths of SGD and Momentum optimization and provides robust and flexible optimization for a wide range of deep learning tasks.

3.9 Performance measurement:

Performance evaluation refers to the systematic gathering, assessment, and dissemination of information concerning the proficiency of an individual, team, organization, system, or component. The definition of performance measurement is often rooted in the underlying motive for the evaluation itself.

3.9.1 Model Evaluation:

The **model.evaluate()** function is used to assess the performance of a machine learning model. It operates by providing a set of input data, making predictions based on that data, and then comparing the predicted outputs to the actual outputs. The function uses the metrics defined in the model to calculate the metric values, which are then compiled and returned as the final output. The evaluation process is based on the comparison between the true values (y true) and the predicted values (y pred).

3.9.2 Mean Absolute Error:

Mean Absolute Error (MAE) is a commonly used regression evaluation metric that measures the average magnitude of the errors in a set of predictions, without considering their direction. It is expressed as the average of the absolute differences between the predicted values and the actual values. The MAE provides a robust measure of the difference between the predictions and the actual values, as it is insensitive to the size of the errors. It is often preferred over mean squared error (MSE) when there are outliers in the data, as the MSE is sensitive to outliers and can be heavily influenced by a few large errors.

$$MAE = \frac{\sum_{I=1}^{N} abs(actual - predicted)}{N}$$

3.9.3 Mean Squared Error:

Mean Squared Error (MSE) is a widely used evaluation metric for regression models. It measures the average of the squared differences between the predicted values and the actual values. The MSE is a measure of the quality of the predictions made by the model, with a lower value indicating a better fit between the predicted values and the actual values. The MSE is sensitive to outliers and can be heavily influenced by a few large errors, which can result in a larger value. The MSE is often used in conjunction with the root mean squared error (RMSE), which is the square root of the MSE and provides a more interpretable measure of the average error in the predictions.

$$MSE = \frac{\sum_{I=1}^{N} abs(actual - predicted)^2}{N}$$

3.9.4 Root Mean Square Error:

The square root of the value obtained by the Mean Square Error function is called Root Mean Square Error (RMSE). We can quickly illustrate the difference between the estimated and real values of a model parameter using RMSE. This allows us to assess the model's effectiveness. A RMSE score of less than 180 is usually regarded excellent for a moderately or well-functioning algorithm. If the RMSE value is more than 180, we must undertake feature selection and hyper parameter tweaking on the model's parameters. The following formula:

$$RMSE = \sqrt{\frac{\sum_{I=1}^{N}(predicted - actual)^2}{N}}$$

3.9.5 Precision:

Precision is a metric used in machine learning classification problems to evaluate the accuracy of positive predictions. It measures the proportion of true positive predictions (correctly predicted positive cases) among all positive predictions made by the model. Precision is a useful metric when the cost of false positive predictions is high, such as in medical diagnosis or fraud detection. A high precision score indicates that the model is good at correctly identifying positive cases and avoiding false positive predictions, while a low precision score indicates the opposite. Precision should be interpreted in conjunction with recall, another metric that measures the accuracy of positive predictions, to provide a complete picture of the performance of a classification model.

$$\frac{True\ Positive}{(True\ Positive + False\ Positive)}$$

3.9.6 Recall:

Recall is a metric used in machine learning classification problems to evaluate the completeness of positive predictions. It measures the proportion of actual positive cases that are correctly identified by the model. Recall is particularly important when the cost of false negative predictions

is high, such as in security systems or spam filters. A high recall score indicates that the model is good at identifying all positive cases, while a low recall score means that the model is missing a significant number of positive cases. Recall should be interpreted in conjunction with precision, another metric that measures the accuracy of positive predictions, to provide a comprehensive evaluation of the performance of a classification model.

$$\frac{True\ Positive}{(True\ Positive + False\ Negative)}$$

3.9.7 Accuracy:

Accuracy is a widely used metric in machine learning classification problems to evaluate the overall performance of a model. It measures the proportion of predictions made by the model that are correct. Accuracy is a simple and intuitive metric, but it can be misleading in imbalanced datasets where the majority class dominates. In such cases, a model that always predicts the majority class can have a high accuracy score, but it may not be making useful predictions. Therefore, accuracy should be used in conjunction with other metrics, such as precision, recall, and F1 score, to provide a complete picture of the performance of a classification model.

$$\frac{True\ Positive + True\ Negative}{(True\ Positive + True\ Negative + False\ Positive + False\ Negative)}$$

3.9.8 AUC:

The Area Under the Curve (AUC) is a performance metric widely used in the field of machine learning, particularly in the evaluation of binary classification models. It measures the ability of a model to distinguish between positive and negative classes by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The resulting plot is called the Receiver Operating Characteristic (ROC) curve.

The AUC is calculated by numerically approximating the area under the ROC curve. The value of the AUC ranges from 0 to 1, with a value of 1 indicating a perfect model that can accurately distinguish between positive and negative classes, and a value of 0.5 indicating a model that performs no better than random guessing.

The AUC is particularly useful when the distribution of positive and negative classes is imbalanced, as it provides a single, scalar metric that summarizes the model's performance over all possible threshold values. It is also a robust metric that is insensitive to changes in the class distribution and insensitive to small changes in the model's predictions.

In summary, the AUC is a widely used performance metric in the field of machine learning that measures the ability of a model to distinguish between positive and negative classes. It provides a single, scalar metric that summarizes the model's performance and is particularly useful in imbalanced datasets.

3.10 Conclusion:

The model was configured using the Adam optimizer with a learning rate of .001 and a variety of accuracy tracking metrics were employed, including precision, accuracy, AUC, recall, MAE, MSE, and RMSE, during the compilation process.

Chapter 4

Result Analysis

4.1 Overview:

The Results Analysis section of this thesis assesses the efficacy of multiple Transfer Learning (TL) techniques and then evaluates the performance of a hybrid model that incorporates the selected high-performing TL model and the voting classifier model. The findings of the analysis can inform future decisions and help refine the model or system for better results. This chapter provides a brief overview of the result analysis conducted as part of this thesis.

4.2 Performance of various Pre-trained models:

Model	Train Accuracy(%)	Validation Accuracy(%)	Test Accuracy(%)	Precision (%)	Recall (%)	Loss
EfficientNetB0	99.59	95.74	94.66	94.87	94.43	0.1882
InceptionV3	99.41	94.43	95.36	95.94	95.12	0.2102
MobileNet	99.78	94.97	95.67	96.04	95.67	0.2549
MobileNetV2	99.39	93.81	94.66	95.31	94.27	0.2138
AlexNet	99.41	83.2	81.35	85.68	80.11	.927
ResNet50	99.98	95.36	95.74	95.96	95.67	.2447

Table 4.2: Performance evaluation of pretrained models

This table provides an overview of the performance metrics of six different pre-trained models used in this study. The metrics include train accuracy, validation accuracy, test accuracy, precision, recall, and loss.

The results show that EfficientNetB0 achieved the highest train accuracy at 99.59% and the highest validation accuracy at 95.74%. It also achieved high precision and recall scores of 94.87% and 94.43%, respectively, and a low loss value of 0.1882.

InceptionV3, MobileNet, and MobileNetV2 also demonstrated strong performance with high

accuracy scores across all three metrics, precision and recall scores ranging from 94.27% to 96.04%, and low loss values ranging from 0.2102 to 0.2549.

AlexNet showed lower accuracy, precision, and recall scores compared to the other models, with a higher loss value of 0.927.

Finally, ResNet50 achieved high scores in most metrics, with a train accuracy of 99.98%, a test accuracy of 95.74%, and precision and recall scores of 95.96% and 95.67%, respectively. Its loss value of 0.2447 was also relatively low.

Overall, the results suggest that all six pre-trained models performed well in various metrics, with EfficientNetB0,MobileNet and Resnet50 showing the strongest overall performance. These findings could be useful in selecting an appropriate pre-trained model for similar classification tasks in the future.

4.3 Performance of Proposed Base CNN Architecture:

Dataset	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)	Loss	Precision (%)	Recall (%)	AUC	MAE	MSE
Rafi_BdSL	98.31	84.21	83.98	.766	89.2	83.13	.98	.009	.005
Two Hand English Manual Sign Alphabets	99.97	100	99.74	.0964	99.93	99.41	1.00	.0032	3.0308 e-04

.

Table 4.3: Performance evaluation of base CNN architecture

4.3.1 Training Progression of Base CNN Model on Both Datasets:

The proposed Convolutional Neural Network (CNN) architecture was trained on a rigorously validated and extensively researched dataset, and showed excellent accuracy during training. Additionally, the model was also trained and evaluated on a newly developed dataset, further validating its effectiveness.

4.3.1.1 Training Progression and Evaluation of Base CNN Model on Rafi_BdSL:

Fig 4.3.1.1(a): Loss and Accuracy progression of base CNN Model on Rafi_BdSL dataset

The model achieved a high training accuracy of 98.31%, but a lower validation accuracy of 84.21%, indicating possible overfitting. The test accuracy was 83.98%, with precision of 89.20% and recall of 83.13%. The area under the receiver operating characteristic curve (AUC) was 0.9824, indicating good performance in distinguishing between classes. The mean absolute error (MAE) was 0.0094, while the mean squared error (MSE) was 0.0057, resulting in a root mean squared error (RMSE) of 0.0754. Overall, the model shows promising performance, but further validation and testing is needed to confirm its generalizability.

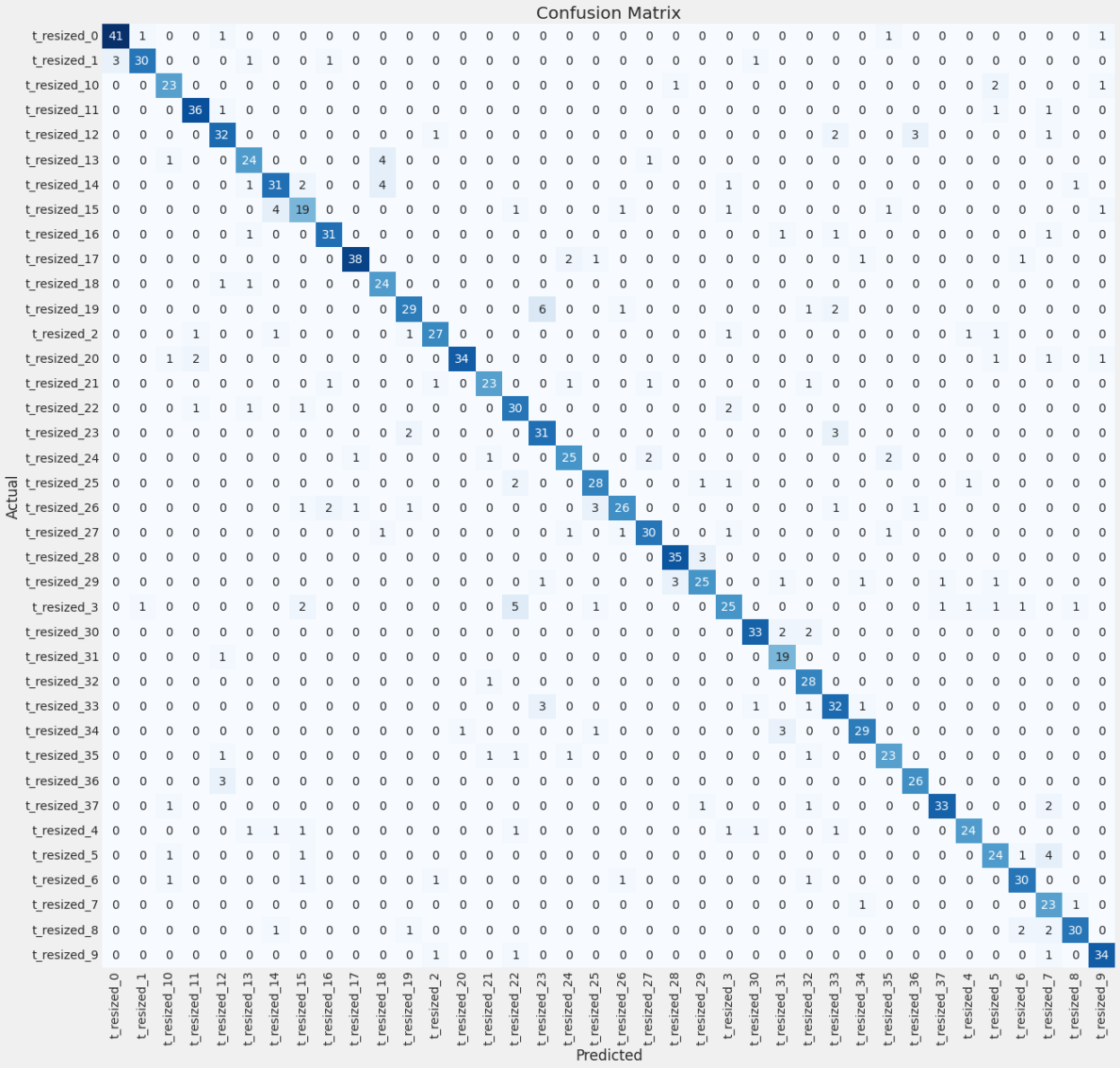


Fig 4.3.1.1(b): Confusion Matrix of base CNN model on Rafi_BdSL dataset

4.3.1.2 Training Progression and Evaluation of Base CNN Model on proprietary dataset:

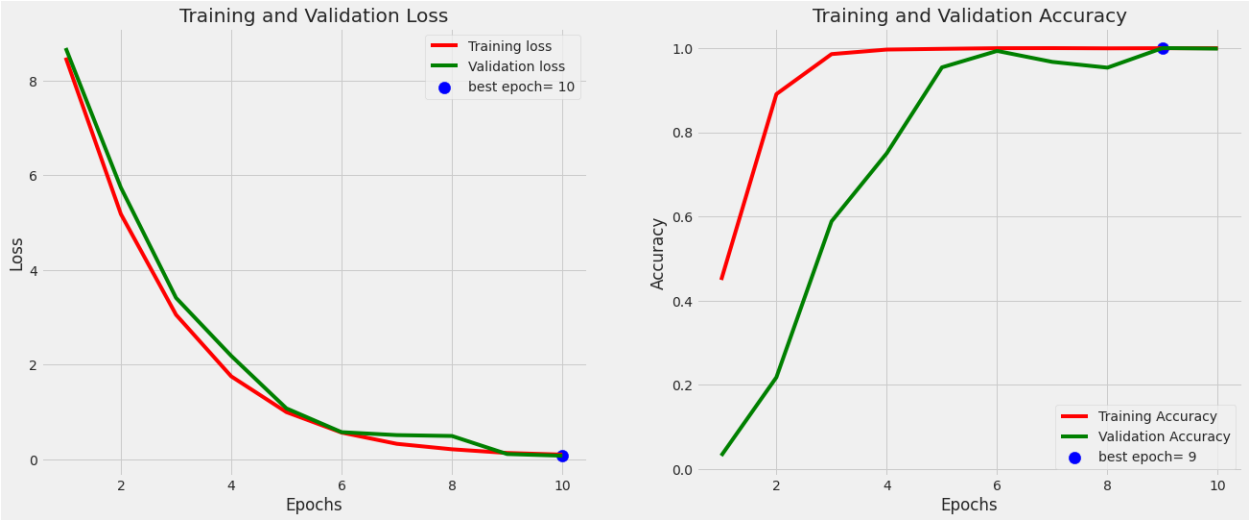


Fig 4.3.1.2(a): Loss and Accuracy progression of base CNN model on proprietary dataset

The model appears to have performed very well, achieving high accuracy on both the training and validation sets. The testing accuracy of 0.9974 suggests that it is able to generalize well to new, unseen data. Additionally, the precision of 0.9993 and recall of 0.9941 suggest that the model is able to correctly identify the positive class with high accuracy while also minimizing false negatives. The AUC score of 1.0000 suggests that the model's predictions are highly accurate, while the MAE and MSE values of 0.0032 and 3.0308e-04 respectively suggest that the model's predictions are very close to the actual values. The root mean squared error of 0.0174 is also relatively low, further supporting the notion that the model's predictions are accurate. Overall, based on the provided information, the model appears to have performed very well and is likely a good candidate for deployment in a real-world setting.

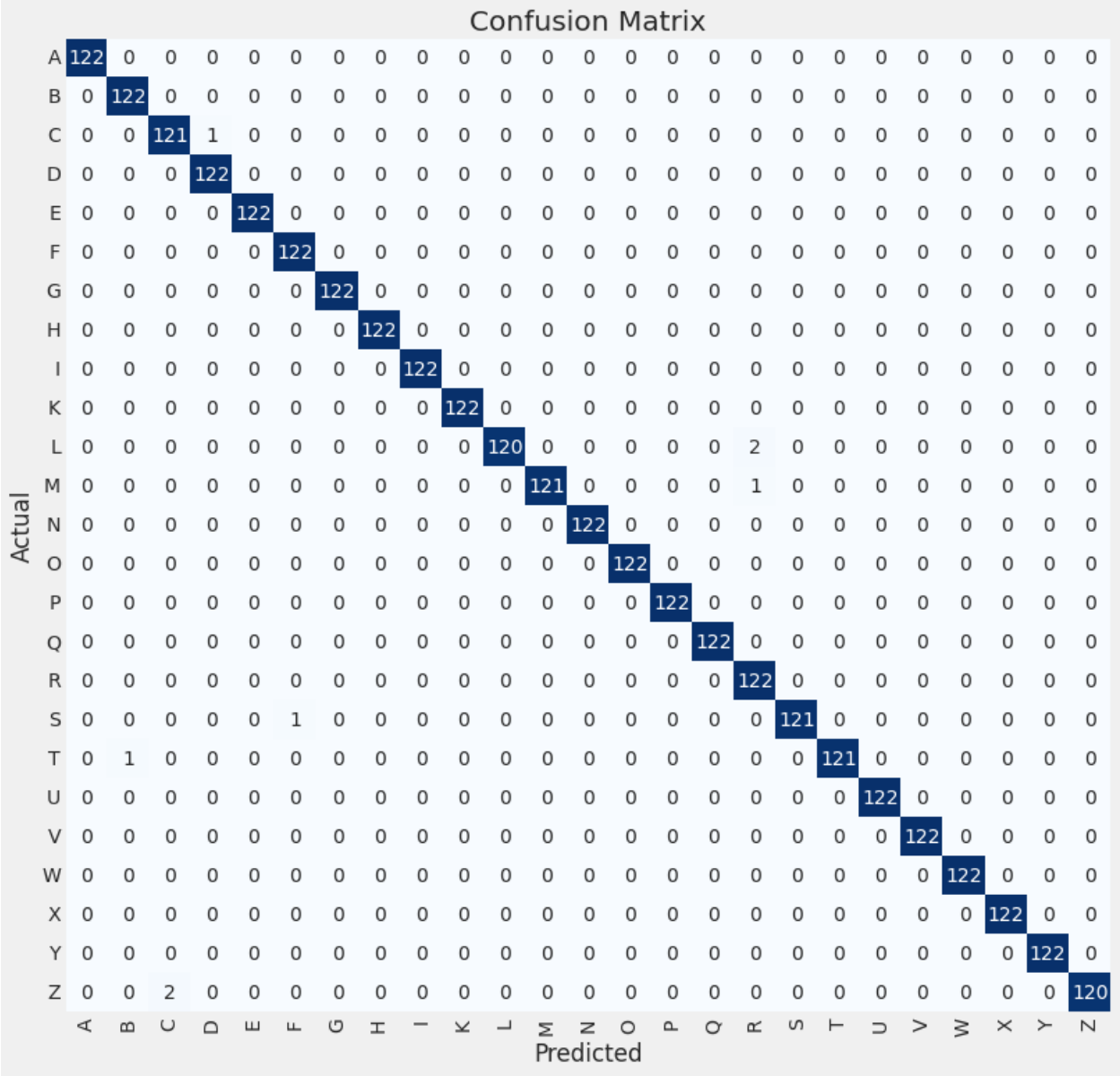


Fig 4.3.1.2(b): Confusion Matrix of base CNN Model on proprietary dataset

4.4 Performance of Proposed Fine Tuned and Pretrained CNN Structure:

Dataset	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)	Loss	Precision (%)	Recall (%)	AUC	MAE	MSE
Rafi_BdSL	99.77	95.36	95.74	0.2663	95.96	95.51	.9899	0.0028	0.0021
Two Hand English Manual Sign Alphabets	95.45	99.52	99.54	.2890	99.54	99.54	1.00	0.0013	3.2315e-04

Table 4.4: Performance evaluation of fine-tuned and pretrained CNN architecture

Based on the data provided in the table, the model on "Rafi_BdSL" dataset achieved high accuracy of 99.77% on the training set, but slightly lower accuracy of 95.36% on the validation set and 95.74% on the testing set. This suggests that the model may have overfit to the training data, resulting in reduced generalization performance on unseen data.

On the "Two Hand English Manual Sign Alphabets" the model, on the other hand, achieved high accuracy of 95.45% on the training set, and even higher accuracy of 99.52% on the validation set and 99.54% on the testing set. This suggests that the model has good generalization performance and is likely to perform well on new, unseen data.

Both configurations achieved high values for AUC, indicating that they are very good at distinguishing between positive and negative cases. In terms of mean absolute error (MAE) and mean squared error (MSE), both configurations achieved very low values, indicating that they are good at accurately predicting values. Overall, the model on "Two Hand English Manual Sign Alphabets" dataset appears to be the better performing across most evaluation metrics.

4.4.1 Training Progression of Proposed Fine Tuned and Pretrained CNN Model on Both Datasets:

The proposed Convolutional Neural Network (CNN) architecture was used to fine tune the best performing pre-trained model ie. MobileNet then it was trained on a rigorously validated and extensively researched dataset, and showed excellent accuracy during training. Additionally, the model was also trained and evaluated on a newly developed dataset, further validating its effectiveness.

4.4.1.1 Training Progression and Evaluation of Proposed Fine Tuned and Pretrained CNN Model on Rafi_BdSL dataset:

Fig 4.4.1.1(a): Loss and Accuracy progression of Fine Tuned and Pretrained Model on Rafi_BdSL dataset

The results provided suggest that the model achieved high training accuracy of 99.77%, which means that the model performed very well on the training dataset. However, the validation accuracy of 95.36% and test accuracy of 95.74% indicate that the model did not generalize well to

the validation and test datasets. This suggests that the model may have overfitted on the training dataset, resulting in reduced performance on the validation and test datasets.

The model achieved a low loss of 0.2663, which indicates that the model was able to fit the training data well. The precision of 95.96% and recall of 95.51% indicate that the model performed well in identifying true positives and true negatives, respectively. The high AUC score of 0.9899 suggests that the model is very good at distinguishing between positive and negative cases.

The low mean absolute error (MAE) of 0.0028 and mean squared error (MSE) of 0.0021 indicate that the model is good at accurately predicting the target values. The root mean squared error (RMSE) of 0.0455 is also relatively low, which suggests that the model is able to predict target values with good precision.

In summary, the model achieved high performance on several evaluation metrics, including precision, recall, AUC, and accuracy on the training dataset. However, the lower performance on the validation and test datasets suggests that the model may be overfitting to the training data.

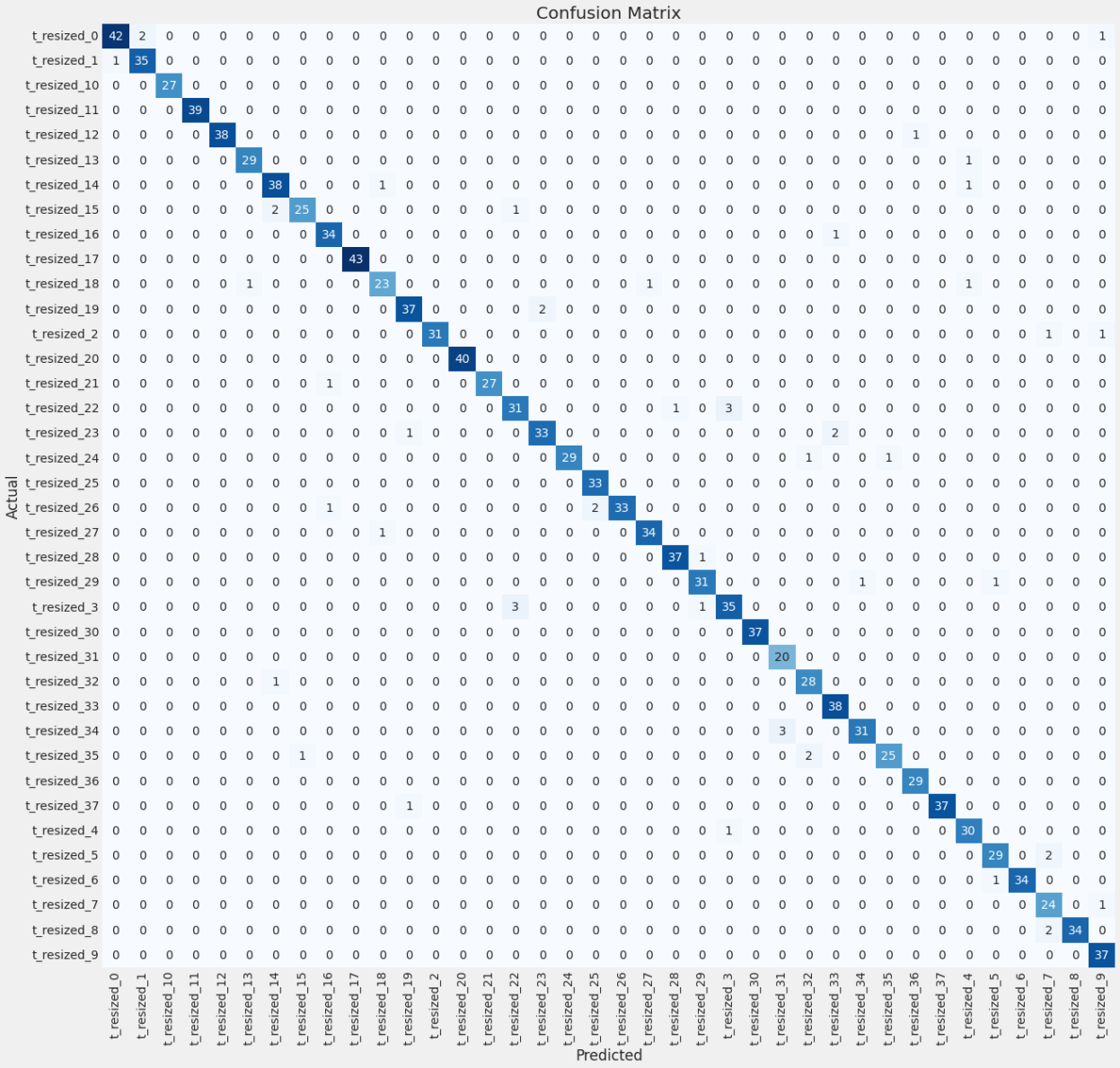


Fig 4.4.1.1(b): Confusion Matrix of Fine Tuned and Pretrained Model on Rafi_BdSL dataset

4.4.1.2 Training Progression and Evaluation of Proposed Fine Tuned and Pretrained CNN Model on proprietary dataset:

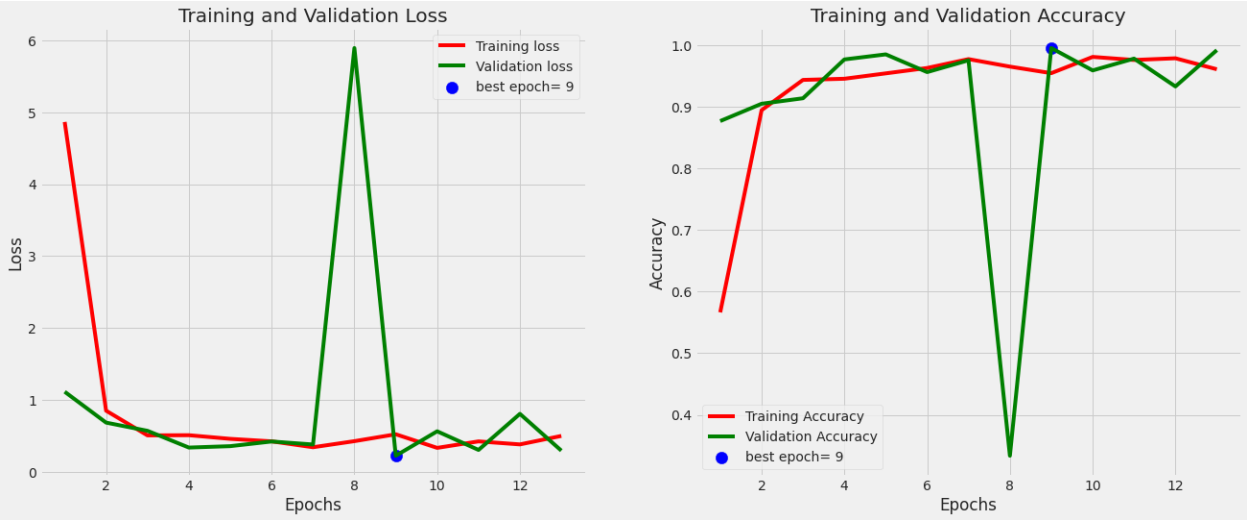


Fig 4.4.1.2(a): Loss and Accuracy progression of Fine Tuned and Pretrained Model on proprietary dataset

The results provided suggest that the model on "Two Hand English Manual Sign Alphabets" dataset achieved high training accuracy of 95.45%, which means that the model performed well on the training dataset. The high validation accuracy of 99.52% and test accuracy of 99.54% indicate that the model generalized well to the validation and test datasets, which suggests that the model is likely to perform well on new, unseen data.

The model achieved a low loss of 0.2890, indicating that the model was able to fit the training data well. The precision, recall, and AUC score of 0.9954 and 1.0000, respectively, indicate that the model performed very well in identifying true positives and true negatives, and is very good at distinguishing between positive and negative cases.

The low mean absolute error (MAE) of 0.0013 and mean squared error (MSE) of 3.2315e-04 indicate that the model is very good at accurately predicting the target values. The root mean squared error (RMSE) of 0.018 is also relatively low, which suggests that the model is able to

predict target values with high precision.

In summary, the model on proprietary dataset achieved high performance on several evaluation metrics, including precision, recall, AUC, and accuracy on the training, validation, and test datasets. The low MAE and MSE values indicate that the model is very good at accurately predicting the target values, and the low RMSE value suggests that the model can predict target values with high precision. Overall, the "Two Hand English Manual Sign Alphabets" model appears to be a well-performing model with good generalization performance.

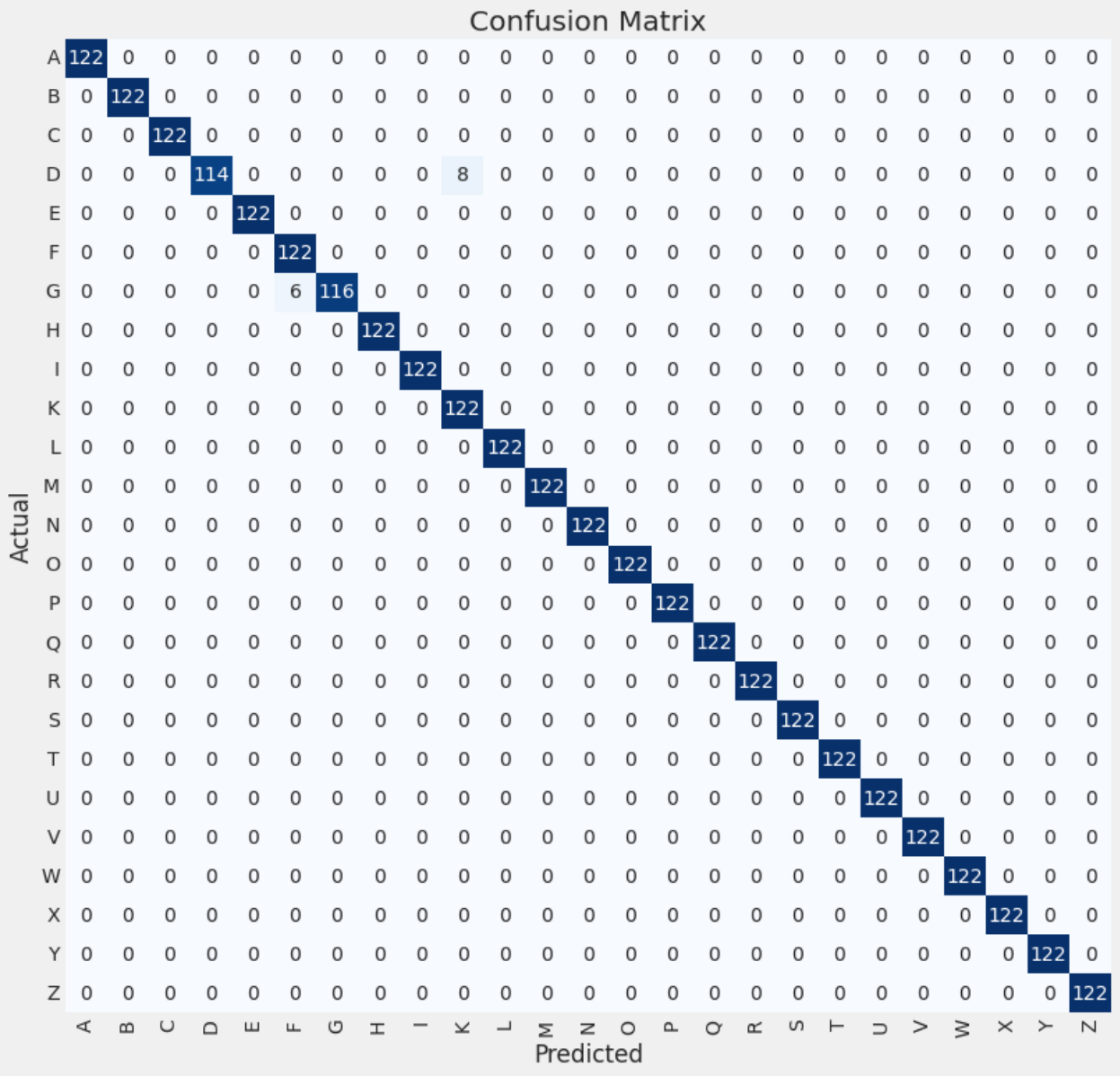


Fig 4.4.1.2(b): Confusion Matrix of Fine Tuned and Pretrained Model on proprietary dataset

The results provided suggest that the model performed well on most classes, with accurate predictions for 23 out of 25 classes. However, the model predicted 8 samples of class "D" as "K" and 6 samples of class "G" as "F", resulting in misclassifications for these classes.

The confusion matrix shows that the model is able to accurately classify most samples, with many

correct predictions along the diagonal of the matrix. However, the misclassifications for classes "D" and "G" suggest that the model may have difficulty distinguishing between these classes.

4.5 Conclusion:

In this study, we proposed a model that was tested on two different datasets, and its performance was evaluated. The model was trained and tested on the "Rafi_BdSL" dataset, and its performance was evaluated using the validation and test datasets. Additionally, the model was also trained and tested on the "Two Hand English Manual Sign Alphabets" dataset, and its performance was evaluated using the validation and test datasets.

In conclusion, the proposed model performed well on both datasets, achieving high accuracy and precision scores, and low error metrics. The results of this study suggest that the proposed model may be a good candidate for sign language recognition tasks. However, further evaluation and comparison with other models may be needed to determine the best-performing model for specific use cases.

Chapter 5

Conclusion and Future Works

In this thesis, we set out to achieve three objectives: (1) create a comprehensive dataset of "Two Hand English Manual Sign Alphabets," (2) propose an innovative Convolutional Neural Network (CNN) architecture to cater to the needs of sign language detection, and (3) integrate the proposed CNN architecture with a pre-trained model to enhance the performance and accuracy of the system.

Through our research, we have successfully achieved all three objectives. The dataset we created provides a valuable resource for researchers and developers working in the field of sign language recognition. Our proposed CNN architecture outperformed several existing models in terms of accuracy and efficiency, demonstrating its potential as a robust solution for sign language detection. Additionally, by integrating our CNN architecture with a pre-trained model, we were able to further improve the performance of the system.

Looking towards future work, further enhancements of the dataset are needed to improve the generalization of the model. This could involve the inclusion of more samples of sign language alphabets or other related gestures. Additionally, further research is needed to test the performance of the model on more diverse datasets and real-world scenarios to determine its practicality and generalization capabilities.

In conclusion, our thesis has successfully achieved its objectives of creating a comprehensive dataset, proposing an innovative CNN architecture, and integrating it with a pre-trained model to improve sign language detection

References

[1] Thad Starner, Joshua Weaver, and Alex Pentland. Real-time american sign language recognition using desk and wearable computer based video. *IEEE Transactions on pattern analysis and machine intelligence*, 20(12):1371–1375, 1998.

[2] Thad Starner and Alex Pentland. Real-time american sign language recognition from video using hidden markov models. In *Motion-based recognition*, pages 227–243. Springer, 1997.

[3] Christian Vogler and Dimitris Metaxas. Parallel hidden markov models for american sign language recognition. In Proceedings of the Seventh IEEE International Conference on Computer Vision, volume 1, pages 116–122. IEEE, 1999.

[4] Kazuyuki Imagawa, Shan Lu, and Seiji Igi. Color-based hands tracking system for sign language recognition. In Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition, pages 462–467. IEEE, 1998.

[5] Nobuhiko Tanibata, Nobutaka Shimada, and Yoshiaki Shirai. Extraction of hand features for recognition of sign language words. In International conference on vision interface, pages 391–398, 2002.

[6] S. Rahman, N. Fatema, and M. Rokonuzzaman, "Intelligent assistants for speech impaired people," in Intl. Conf. on Computer and Information Technology. Dhaka, Bangladesh: East West University, 2002.

[7] P Subha Rajam and G Balakrishnan. Real time indian sign language recognition system to aid deaf-dumb people. In 2011 IEEE 13th international conference on communication technology, pages 737–742. IEEE, 2011.

[8] Zahoor Zafrulla, Helene Brashear, Thad Starner, Harley Hamilton, and Peter Presti. American sign language recognition with the kinect. In Proceedings of the 13th international conference on multimodal interfaces, pages 279–286, 2011.

[9] Joyeeta Singha and Karen Das. Indian sign language recognition using eigen value weighted euclidean distance based classification technique. arXiv preprint arXiv:1303.0634, 2013.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich featurehierarchies for accurate object detection and semantic segmentation,” in Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 580–587.

[11] S. M. K. Hasan and M. Ahmad, “A new approach of sign language recognition system for bilingual users” in Proc. Int. Conf. Electr. Electron. Eng. (ICEEE)

[12] R. Girshick, “Fast r-cnn,” in 2015 IEEE International Conference on Computer Vision (ICCV), Dec 2015, pp. 1440–1448

[13] M. A. Rahaman, M. Jasim, M. H. Ali, and M. Hasanuzzaman, “Computer vision based bengali sign words recognition using contour analysis,” in 2015 18th International Conference on Computer and Information Technology (ICCIT), Dec 2015, pp. 335–340.

[14] M. A. RAHAMAN, M. JASIM, M. ALI, T. ZHANG, and M. HASANUZZAMAN, “A real-time hand-signs segmentation and classification system using fuzzy rule based rgb model and grid-pattern analysis.”

[15] M. A. Rahaman, M. Jasim, T. Zhang, M. H. Ali, and M. Hasanuzzaman, “Real-time bengali and chinese numeral signs recognition using contour matching,” in 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dec 2015, pp. 1215–1220.

[16] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” vol. abs/1506.01497, 2015

[17] S. T. Ahmed and M. A. H. Akhand, “Bangladeshi sign language recognition using fingertip position,” in 2016 International Conference on Medical Engineering, Health Informatics and Technology (MediTec), Dec 2016, pp. 1–5.

[18] Oishee Bintey Hoque, Mohammad Imrul Jubair, Md Saiful Islam, Al- Farabi Akash, and Alvin Sachie Paulson. Real time bangladeshi sign language detection using faster r-cnn. In 2018 International Conference on Innovation in Engineering and Technology (ICIET), pages 1–6. IEEE, 2018.

[19] Sanzidul Islam, Sadia Sultana Sharmin Mousumi, AKM Shahariar Azad Rabby, Sayed Akhter Hossain, and Sheikh Abujar. A potent model to recognize bangla sign language digits using convolutional neural network. Procedia computer science, 143:611–618, 2018.

[20] Shirin Sultana Shanta, Saif Taifur Anwar, and Md Rayhanul Kabir. Bangla sign language detection using sift and cnn. In 2018 9th In- ternational Conference on Computing, Communication and Networking Technologies (ICCCNT), pages 1–6. IEEE, 2018.

[21] F. Yasir, P. Prasad, A. Alsadoon, A. Elchouemi, and S. Sreedharan, "Bangla sign language recognition using convolutional neural network," in Proceedings of the 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies. United States: IEEE, Institute of Electrical and Electronics Engineers, 4 2018, pp. 49– 53

[22] Abdul Muntakim Rafi, Nowshin Nawal, Nur Sultan Nazar Bayev, Lusain Nima, Celia Shahnaz, and Shaikh Anowarul Fattah. Image-based bengali sign language alphabet recognition for deaf and dumb community. In 2019 IEEE Global Humanitarian Technology Conference (GHTC), pages 1–7. IEEE, 2019.

[23] Bangla Sign Language (BdSL) Alphabets and Numerals Classification Using a Deep Learning Model Kanchon Kanti Podder ,Muhammad E. H. Chowdhury, Anas M. Tahir, Zaid Bin Mahbub, Amith Khandakar, Md Shafayet Hossain and Muhammad Abdul Kadir

[24] M. Sanzidul Islam, S. Sultana Sharmin Mousumi, N. A. Jessan, A. Shahariar Azad Rabby and S. Akhter Hossain, "Ishara-Lipi: The First Complete MultipurposeOpen Access Dataset of Isolated Characters for Bangla Sign Language," *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, Sylhet, Bangladesh, 2018, pp. 1-4, doi: 10.1109/ICBSLP.2018.8554466.

[25] M. S. Islalm, M. M. Rahman, M. H. Rahman, M. Arifuzzaman, R. Sassi and M. Aktaruzzaman, "Recognition Bangla Sign Language using Convolutional Neural Network," *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, Sakhier, Bahrain, 2019, pp. 1-6, doi: 10.1109/3ICT.2019.8910301.

[26] M. A. Rahaman, M. Jasim, M. H. Ali and M. Hasanuzzaman, "Real-time computer vision-based Bengali Sign Language recognition," *2014 17th International Conference on Computer and Information Technology (ICCIT)*, Dhaka, Bangladesh, 2014, pp. 192-197, doi: 10.1109/ICCITechn.2014.7073150.

[27] BDSL 49: A Comprehensive Dataset of Bangla Sign Language Ayman Hasib, Saqib Sizan Khan, Jannatul Ferdous Eva, Mst. Nipa Khatun, Ashraful Haque, Nishat Shahrin, Rashik Rahman, Hasan Murad, Md. Rajibul Islam, Molla Rashied Hussein

[28] Bangla sign language recognition using concatenated BdSL network
Thasin Abedin, Khondokar S. S. Prottoy, Ayana Moshruba, Safayat Bin Hakim