CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

by

Srushti Satardekar

December 2023

The thesis of Srushti Satardekar is approved:

Mahdi Ebrahimi	Date
Robert D McIlhenny	Date
George Wang, Chair	Date

California State University, Northridge

Table of Contents

List of Figures

List of Tables

ABSTRACT

SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING

By

Srushti Satardekar

Master of Science in Computer Science

American Sign Language (ASL) is a visual language used by people with hearing impairments to communicate with others through hand gestures, facial expressions, and body movements. However, not everyone is fluent in ASL, and this can create communication barriers between the hearing-impaired and the hearing population. In recent years, sign language recognition (SLR) systems have been developed to bridge this communication gap. In this thesis, we present a novel approach to SLR using our self-developed deep learning model based on DenseNet-154 architecture. Our model consists of 154 dense layers, with 128 layers in the main network layer and 26 layers in the output layer. Through extensive experimentation and tuning, we achieved an impressive maximum accuracy of 95.6% with an average lag of 0.03 seconds. However, we encountered challenges with the background. Our current project is limited to plain backgrounds, and we found that accuracy drops to 37% in the case of designer backgrounds. We also studied the effects of illumination and user clothing on the output and found that the background has the most significant effect on the accuracy of our model named as DenseNet-154 SLR. Our proposed DenseNet-154 SLR system has the potential to revolutionize communication between hearing-impaired individuals and the hearing population. It can be used in a variety of settings, such as in educational institutions, public places, and even in homes. Our study demonstrates the effectiveness of DenseNet architecture in developing an accurate and efficient DenseNet-154 SLR system. However, there is still room for improvement, and we suggest further research to enhance the accuracy of our model in diverse settings. Overall, our work lays the foundation for more advanced and accessible DenseNet-154 SLR systems in the future.

Chapter 1

Introduction

Sign language is a vital mode of communication for individuals with hearing impairments. However, not everyone is fluent in sign language, which can pose significant challenges for deaf individuals in everyday life. In recent years, computer vision systems have shown great potential in sign language recognition, enabling real-time translation of sign language into spoken or written language. This has led to an increased interest in developing low-cost and easy-to-use sign language recognition systems that can be deployed on commonly available devices such as webcams. By making sign language more accessible to a wider range of people, these systems have the potential to significantly improve the lives of deaf individuals, allowing them to better communicate with the world around them. The Fig.

Figure 1.1: Proposed sign language recognition system

1.1 shows the proposed sign language recognition system. This project aims to develop a deep learning model for American Sign Language detection that can run in real-time on a low-cost device such as a webcam. By doing so, this project can contribute to the development of more accessible and cost-effective sign language recognition systems that can benefit society as a whole.

1.1 Sign language

Sign language is a visual language that conveys meaning and communicates with others through the use of hand gestures, facial expressions, bodily movements, and other visual

cues. It is also defined as a visual language that is accessible to people who are deaf or hard of hearing, and also used by others to communicate with them.

1.1.1 Types of sign language

There are various types of sign languages in the world. There are over 140 distinct sign languages used throughout the world that developed independently in different communities. There are regional sign languages too. Every person can have their own sign language according to the regional variations of the language. Sign languages are not universal so people who generally speak different languages can speak in the same sign language.

Following are few types of sign languages and where are they used in:

1. American Sign Language (ASL) - used in the United States and Canada
2. British Sign Language (BSL) - used in the United Kingdom
3. Australian Sign Language (Auslan) - used in Australia
4. French Sign Language (LSF) - used in France
5. Japanese Sign Language (JSL) - used in Japan
6. Chinese Sign Language (CSL) - used in China
7. Mexican Sign Language (LSM) - used in Mexico
8. Russian Sign Language (RSL) - used in Russia
9. Brazilian Sign Language (Libras) - used in Brazil
10. South African Sign Language (SASL) - used in South Africa

1.1.2 American sign language

In America many of the professions like teaching, social work, healthcare and more require the knowledge of sign language. It is also combined with its unique cultural values and traditions. Moreover, it is observed that learning sign language has improved problem-solving skills. Learning ASL can be a rewarding experience because it allows you to connect with a group of people who are often marginalized and overlooked in society. It can also be a fun and unique method to express yourself and communicate with others. [1]

Figure 1.2: Standard American Sign Language

1.2 History of sign language

The history of sign language is complex and multifaceted, and it varies depending on the region and culture in which the sign language developed. Sign languages have existed for centuries, and some may have developed independently of spoken languages. For example, there is evidence that Nicaraguan Sign Language emerged in the 1980s among deaf children who had no access to standardized sign language.

In many cultures, sign languages were not recognized as legitimate languages until relatively recently. For example, American Sign Language was not recognized as a distinct language until the 1960s.

Sign languages have been influenced by spoken languages in various ways. For example, many signs in American Sign Language are based on the finger-spelling of English words.

There is ongoing debate among linguists about the relationship between sign languages and spoken languages. Some argue that sign languages are fully-fledged languages with their own grammatical structures, while others argue that they are simply visual represen-

tations of spoken languages.

1.3 Sign language recognition

Sign Language Recognition (SLR) is a task that involves recognizing actions in sign languages using computers. It is important to solve this problem, especially in the digital world, to help people with hearing impairments communicate better. To solve this problem, we need not only color data, but also other types of information, such as depth and sensory information.

There are different ways to recognize sign language, such as:

1. Vision-based recognition: using cameras to capture hand and body movements and then translating them into spoken or written language.
2. Sensor-based recognition: using wearable sensors like gloves or bracelets to capture hand movements and translating them into text or speech.
3. Data glove-based recognition: using special gloves with sensors to track finger and hand movement and translate them into text or speech.
4. Muscle-signal-based recognition: using electromyography (EMG) to detect electrical signals generated by muscles in the arm and hand as signs are made, and then analyzing them to recognize the signs.
5. Linguistic-based recognition: analyzing the grammar and syntax of sign language to recognize signs, often combined with vision-based recognition to improve accuracy.

1.3.1 Different ways of detection sign language

Sign language recognition is basically performed by human who has an understanding and knowledge of letters and words used in sign language. There are various ways of how sign language is interpreted, following are the ways that explain the ways of recognizing sign language.
Hand recognition by a human interpreter or a person proficient in sign language by identifying the different shapes a signer's hand takes to represent letters, words or phrases.
Video-based recognition using computer vision and machine learning algorithms to identify and interpret sign language gestures. Some signs also incorporate movement of the hands and fingers to convey additional meaning. Motion capture techniques can track the trajectory and motion of the hands while signing to recognize these movement-based signs.

Recognizing individual signs is only part of the process. Understanding sign language also requires analyzing the context, semantic concepts and grammatical rules to properly interpret the meaning. Machine learning models need to incorporate a lot of linguistic knowledge about the sign language.
Another technology is motion capture, which records the movement of the hand and creates a digital representation if the sign language. A combination of these techniques together, sensor fusion, provides the most accurate recognition. Hand shapes, movements, locations and contexts can all be combined to determine the likely signs and meaning.

1.3.2 Current technological solution

The most popular way of recognizing sign language is using computer visions and machine learning. Deep learning algorithm approaches have been shown to achieve better performance in sign language recognition.

1.4 Possible challanges

1. Limited data: Collecting a large amount of labeled sign language data is a challenging and time-consuming process, and the lack of data can limit the accuracy and robustness of the models.

2. Lighting and environmental conditions: The performance of vision-based sign language recognition can be affected by lighting conditions, background clutter, and occlusions.

3. Variations in signing styles: Sign language is not standardized, and there can be variations in signing styles among different users, which can affect the recognition accuracy.

4. Expensive hardware: Some of the sensor-based or data glove-based recognition methods can require expensive hardware, which may not be affordable for everyone.

5. Real-time processing: Real-time processing can be challenging, especially for resource-intensive deep learning models, and can lead to delays and increased processing times.

Chapter 2

Literature Review

Sign language recognition (SLR) has garnered significant attention in recent years due to its potential in bridging the communication gap between hearing-impaired individuals and the hearing population. Various researchers have proposed different approaches using machine learning and deep learning techniques to develop SLR systems. In this literature review, we summarize and analyze the key findings and methodologies of several notable studies in the field.

One of the earliest works in sign language recognition is presented by Mohandes et al. (2004), where they introduce a system for recognizing Arabic sign language using an instrumented glove and a machine learning method. Agarwal et al. (2013) present a sign language recognition system using depth images captured by a Microsoft Kinect® camera. Similarly, Chuan et al. (2014) propose an ASL recognition system using a compact and affordable 3D motion sensor. These studies lay the groundwork for exploring different input modalities for SLR.

An interesting study by Savur et al. (2016) focuses on American Sign Language recognition using surface Electromyography (sEMG) signals. Their approach utilizes sEMG to recognize ASL gestures, highlighting the potential of using bioelectric signals for SLR.

In contrast to incomplete sign language recognition methods, Chong et al. (2018) specifically aim for full ASL recognition, encompassing 26 letters and 10 digits. This comprehensive approach is essential for enabling seamless communication for hearing-impaired individuals.

Rastgoo et al. (2018) explore multi-modal deep hand sign language recognition using a Restricted Boltzmann Machine (RBM), demonstrating the effectiveness of deep learning in handling visual data for SLR.

Aly et al. (2019) propose a user-independent recognition system for American Sign Language alphabet using depth images from the low-cost Microsoft Kinect depth sensor. They introduce a novel algorithm called DyFAV (Dynamic Feature Selection and Voting) that exploits the finite corpus of fingerspelling in ASL.

Paudyal et al. (2019) also leverage the finite corpus of fingerspelling in ASL to propose

a novel user-independent recognition system. They use a different approach called Hidden Markov Model for recognition.

Zhou et al. (2020) present a wearable sign-to-speech translation system that accurately translates ASL hand gestures into speech. Their work demonstrates the practical application of SLR in real-time communication scenarios.

Saleh et al. (2020) focus on recognizing 32 hand gestures from Arabic sign language and effectively use transfer learning and fine-tuning of deep convolutional neural networks (CNN) to enhance recognition accuracy.

Mocialov et al. (2020) examine transfer learning techniques, fine-tuning, and layer substitution for language modeling of British Sign Language. Their study highlights the importance of transfer learning in handling different sign languages.

Jain et al. (2021) propose the recognition of American Sign Language using Support Vector Machine (SVM) and Convolutional Neural Network (CNN). Their combined approach shows promising results in SLR.

Bokstaller et al. (2021) focus on recognizing cultural/anthropological Italian sign language gestures from videos by building a symbiosis between a convolutional neural network (CNN) and a recurrent neural network (RNN).

Whadhwan et al. [2] conducted research on Indian Sign Language (ISL) recognition using deep learning-based convolutional neural networks (CNN). They collected 35,000 sign images of 100 static signs from different users and evaluated the efficiency of the proposed system on approximately 50 CNN models. The proposed approach achieved the highest training accuracy of 99.72% and 99.90% on colored and grayscale images, respectively, and demonstrated effectiveness over earlier works that only considered a few hand signs for recognition. Compared to their work which focuses on Indian sign languages this thesis focuses on American sign langague.

Tolentino et al [3] a system that uses skin-color modeling and a Convolutional Neural Network (CNN) for sign language recognition. With proper lighting and uniform background, the system achieved an average testing accuracy of 93.67%, surpassing other related studies. The system is fast and can be done in real-time.

Rahman et al.[4] developed a novel model to enhance the accuracy of existing methods for American Sign Language (ASL) recognition. They focused on the alphabet and numerals of publicly available ASL datasets. After preprocessing, they fed images of the

alphabet and numerals to a convolutional neural network (CNN) model. Their proposed CNN model significantly improved the recognition accuracy of ASL, by 9%, compared to existing prominent methods. The aim of the study is to facilitate easy communication between normal and deaf communities.

Anshul et al. [5] and his colleagues have developed a modified long short-term memory (LSTM) model to recognize continuous sequences of sign language gestures. This model is based on splitting continuous signs into sub-units and modeling them with neural networks, which eliminates the need to consider different combinations of sub-units during training. They tested the proposed system with 942 signed sentences of Indian Sign Language (ISL) and achieved an average accuracy of 72.3% for signed sentences and 89.5% for isolated sign words. The system has the potential to improve communication between hearing-impaired individuals and the rest of society.

Liao et al. [6] and colleagues have developed a multimodal dynamic sign language recognition method called BLSTM-3D ResNet, which uses deep 3-dimensional residual ConvNet and bi-directional LSTM networks. This method can accurately identify complex hand gestures in video sequences and has higher recognition accuracy than previous dynamic sign language recognition methods. The B3D ResNet includes three main parts that are, hand object localization, spatiotemporal feature extraction, and video sequence classification. The proposed method was tested on two datasets and achieved state-of-the-art recognition accuracy of 89.8% on DEVISIGN_D and 86.9% on SLR_Dataset. Additionally, the B3D ResNet can effectively recognize 500 vocabularies from Chinese hand sign language.

There is a research which uses NLP for the indian sign language recognition. Patil et al's .[7] research focuses on developing a system that translates Indian Sign Language to English. They discuss processing visual and non-visual inputs of sign language signs to create grammatically correct sentences. They have worked on processing inputs such as sensor-based, image-based, or video-based systems to recognize hand movements. Their paper reviews state-of-the-art literature on non-visual inputs, image frames, and video frames to determine features for a particular hand gesture. Additionally, they review previous research on translating video to English using NLP techniques.

One of the researchers and his team have done recogniton for sign language detection using cnn. Daniel et al's .[8]research aims to develop a real-time sign language recognition system using You Only Look Once (YOLO) object detection method based on Convolutional Neural Network (CNN). They collected datasets based on the Indonesian Sign

Language (BISINDO) and retrained the Yolov3 pre-trained model with adjustments to the number of channels and classes according to the sign language recognition requirement. In the experiment, their system achieved 100% precision, recall, accuracy, and F1 score using image data and 77.14% precision, 93.1% recall, 72.97% accuracy, and 84.38% F1 score using video data at a speed of 8 fps.

Similarly, for italian sign language Pigou et al.[9] developed a sign language recognition system using Microsoft Kinect, convolutional neural networks(CNN), and GPU acceleration. The system recognizes 20 Italian sign language gestures with high accuracy, and is able to generalize to new users and environments. The model achieved a cross-validation accuracy of 91.7% and a mean Jaccard Index of 0.789 in the ChaLearn 2014 Looking at People gesture spotting competition. The aim of the research is to improve communication between the Deaf community and the hearing majority.

Fatmi et al .[10]have developed a system to translate American Sign Language (ASL) into speech using wearable motion sensors and machine learning techniques, with a focus on enhancing communication and collaboration between the hearing-impaired community and those untrained in ASL. They proposed an improved solution using Artificial Neural Networks (ANN) and Support Vector Machines (SVM) and compared their accuracy with the Hidden Markov Model (HMM) results. The study's results showed that using ANN provides an overall higher accuracy in recognizing ASL words compared to other machine learning techniques. The proposed system architecture is based on using wearable motion sensors and machine learning techniques to enable the system's daily use by people with hearing impairments.

Katoch.[11] has developed a technique to recognize Indian Sign Language alphabets and digits in a live video stream using the Bag of Visual Words model (BOVW), skin color segmentation, and background subtraction. The system uses SURF features to generate histograms that map the signs to corresponding labels and uses Support Vector Machine (SVM) and Convolutional Neural Networks (CNN) for classification. The technique also includes an interactive Graphical User Interface (GUI) for easy access. The aim is to enhance communication for speech-impaired individuals in India.

One of the works done in this field is by Chen et al.[12], who proposed a hand gesture detection system based on real-time American Sign Language letters recognition. They used a Leap Motion Controller to capture the hand gestures, and extracted features such as position, velocity, direction, and palm normal vector from the data. They then trained an SVM classifier with four different kernels (linear, polynomial, radial basis function, and

sigmoid) to recognize five ASL letters: B, D, F, L, and U. They achieved an accuracy of about 99.4% with the radial basis function kernel.

In summary, the reviewed literature showcases various approaches for sign language recognition, including glove-based methods, depth image-based methods, sEMG-based methods, and deep learning-based methods. Many studies have successfully addressed the recognition of both partial and full sign languages, highlighting the importance of comprehensive approaches. Transfer learning and fine-tuning of deep neural networks have shown potential in improving recognition accuracy, making them valuable techniques to consider in future SLR research. The integration of wearable technology also opens up new possibilities for real-time sign-to-speech translation systems. Further research and experimentation are needed to address the challenges related to background variations and generalization to diverse settings for SLR systems.

Chapter 3

Methods

3.1 Existing Dataset

To build the model, initially it was decided to use Kaggle dataset as our main source of data. However, the Kaggle dataset had images that were carefully cropped to show the hand sign or shape, not the whole picture. The dataset was similar to the famous MNIST dataset in its format. Each image in the dataset had a label from 0 to 25, corresponding to a letter from A to Z. The letters J and Z were missing because they required gesture motions. The dataset had 27,455 training images and 7,172 test images, each with a size of 28x28 pixels and grayscale values from 0 to 255 .[13] As per the feedback of some students on the current technology. It was noted that there was a lot of room for improvement in this field. Based on these insights and the limitations of the Kaggle dataset, it was decided to create a custom dataset that would better suit our needs. The custom dataset includes full-size images depicting their hand signs.

3.2 Dataset Creation

The dataset creation was done using Opencv library using the python programming language. The custom dataset which was created comprises of 5200 images. It contains 100 images for each alphabet performed by user's left and right hand. The american sign language signs were studied by the user and performed in front of a laptop camera to capture data. After capturing the data the images were labeled from A to Z using the labelimg. Label image is a graphical annotation tool which is written in python and uses Qt for its graphical representation. Labelimg can be installed using the command line "pip install labeling". .[14] The Fig 3.1 demonstrates the process of capturing the image.

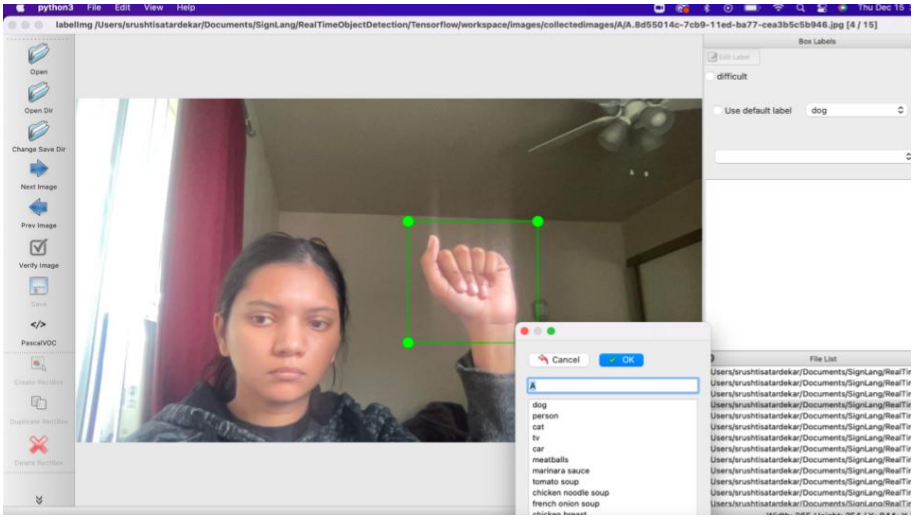


Figure 3.1: LabelImg interface

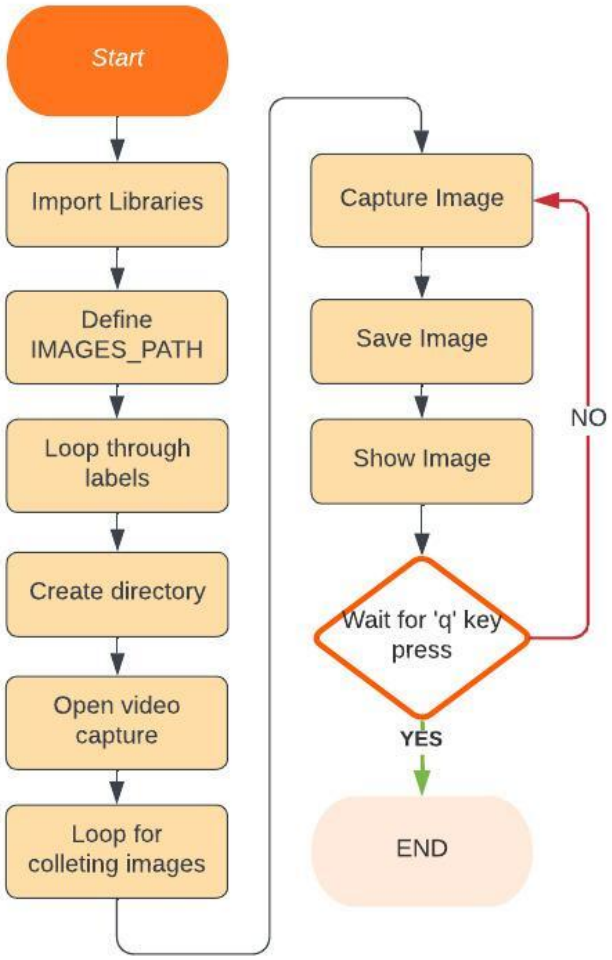


Figure 3.2: Flowchart to capture images for dataset

3.2.1 Capturing Multiple Datasets

To collect the data, it was ensured that the hand signs were correctly shown and labeled. We created four different datasets to compare their accuracy. Our goal was to use the best dataset for various algorithms to improve the performance. Each dataset had 5,200 images, but they differed in image quality and lighting conditions. The model was trained on each dataset and it was noted that the first three datasets had accuracy below 90%. The fourth dataset, which had the best image quality and lighting conditions, achieved an accuracy of 95% with a lag of only 2ms. This was a significant improvement over the previous datasets. Figure 3.3 shows an example image from each of the first three datasets.

The final dataset was composed of all the images from the fourth dataset. We used this dataset to train our final model and evaluate its performance on different tasks. Figure .3.4 shows the image from the final dataset.

(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

Figure 3.3: Images from other Datasets

Figure 3.4: Image from Final Dataset

3.3 Machine Learning Algorithms

3.3.1 Support Vector Machine (SVM)

Support Vector Machine Learning Algorithm (SVM) have been widely adopted as a pivotal component of our research methodology due to their robustness in classification and regression tasks. SVMs are particularly advantageous in scenarios where the dataset is not only high-dimensional but also exhibits complex non-linear relationships. By finding the optimal hyperplane that maximizes the margin between different classes, SVMs effectively handle classification challenges. We utilized the scikit-learn library in Python .[15], which provides a comprehensive suite of tools for machine learning, including SVM implementations. In this study, we used a supervised machine learning technique called support vector machine (SVM) to classify the hand signs in our custom dataset. SVM is a popular method for pattern recognition and classification problems, as it can handle high-dimensional and nonlinear data with high accuracy and robustness. .[16]SVM works by finding an optimal hyperplane that separates the data points into different classes, while maximizing the margin between them .[17]. We chose SVM over other methods because it has several advantages, such as: 1. It can handle both linear and nonlinear classification problems by using different kernel functions.[17] 2. It can avoid over-fitting by using regularization parameters.[17] 3. It can deal with imbalanced data by using class weights or oversampling techniques .[18]

The general mathematical equation of a linear SVM can be expressed as[19]:

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x} + b)$$

Where: - $f(x)$ represents the decision function that predicts the class label for a given input x. - $\mathbf{w}$ is the weight vector, which determines the orientation of the decision boundary. - $\mathbf{x}$ is the input feature vector. - b is the bias term, which shifts the decision boundary away

from the origin.[19, 17]

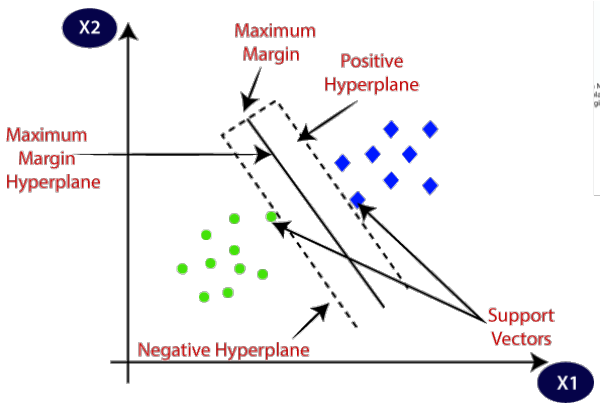


Figure 3.5: Support Vector Machine (SVM)
[20]

In this study we used supervised machine learning Support Vector Machines. (SVM) for sign language recognition as a key component of our methodology. The training of the model begins by importing necessary libraries, including NumPy, OpenCV (cv2), and scikit-learn, and defining a function to extract a Histogram of Oriented Gradients (HOG) features from images. These features are essential for robust classification. The script proceeds by loading a dataset of sign language images, from a folder specified for training. Images are resized to a uniform size of 124x124 pixels and converted to grayscale to enhance computational efficiency. The HOG features are then extracted from each image using the defined function.

Next, the dataset is split into training and testing sets (80% training, 20% testing) to evaluate model performance. The scikit-learn's GridSearchCV is used to perform hyperparameter tuning for the Linear Support Vector Classifier (LinearSVC). The script searches for the best regularization parameter (C) among a predefined set of values using cross-validation. The best C value is extracted from the grid search results, and the script prints this optimal parameter. This script represents a pivotal step in our methodology, as it prepares the dataset, extracts discriminative features, and trains the SVM model while optimizing its hyperparameters. In the figure 3.6. The SVM model is trained using the optimal C value obtained from the previous grid search which is 1. The LinearSVC model is instantiated with the best C value (LinearSVC(C=best_C)) and subsequently trained on the preprocessed training dataset (X_train, y_train).

```
# Step 3: SVM Model Training with GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}  # Experiment with different C values
grid_search = GridSearchCV(LinearSVC(), param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best C value from the grid search
best_C = grid_search.best_params_['C']
print("Best C:", best_C)
```

```
Best C: 0.1
```

Figure 3.6: Support Vector Machine (SVM) Model Training

To assess the model's performance, predictions are made on the test dataset (X_test), and these predictions (y_pred) are compared to the true labels (y_test). Accuracy is computed using scikit-learn's accuracy_score function. The achieved accuracy score provides valuable insights into the model's effectiveness in recognizing sign language gestures. The trained SVM model is serialized using the joblib library and saved as 'asl_linearsvc_model.pkl'. This serialized model can be conveniently loaded and employed for real-world sign language recognition tasks. The training code begins by defining a function to extract Histogram of Oriented Gradients (HOG) features from video frames captured by a camera. These features serve as the input for our model, enabling the recognition of sign language gestures. As shown in Figure 3.7

```
# Define the function to extract HOG features
def extract_hog_features(image):
    resized_image = cv2.resize(image, (124, 124))
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    hog = cv2.HOGDescriptor()
    features = hog.compute(gray_image)
    return features.flatten()

# Load the trained LinearSVC model
model_filename = 'asl_linearsvc_model.pkl'
loaded_model = joblib.load(model_filename)

# Access the camera (you may need to adjust the camera index)
camera = cv2.VideoCapture(0)
```

Figure 3.7: Support Vector Machine (SVM) Model Testing

The trained SVM model, previously saved as 'asl_linearsvc_model.pkl,' is loaded into memory using the joblib library. This model was trained on a preprocessed dataset, allowing it to make predictions on incoming video frames. To initiate real-time recognition, the code accesses the camera using the OpenCV library. It captures frames continuously and processes them to extract HOG features using the defined function. The loaded SVM model then predicts the sign language letter corresponding to the captured gesture. Upon prediction, the code converts the numerical prediction into its corresponding letter and over-

lays it onto the video frame. This provides immediate feedback to users, allowing them to visualize the recognized ASL letter in real time.

3.3.2 You Only Look Once (YOLO v5)

YOLOv5, which stands for "You Only Look Once version 5," is a state-of-the-art real-time object detection and recognition model. It is part of the YOLO family of computer vision models, known for their speed and accuracy in object detection tasks.[21]

Key features and characteristics of YOLOv5 include:

1. Real-time Object Detection: YOLOv5 is designed for real-time object detection in images and video streams, making it suitable for a wide range of applications, including surveillance, autonomous vehicles, and robotics.
2. Speed and Efficiency: YOLOv5 is known for its speed and efficiency, allowing it to process images quickly while maintaining high accuracy. This is achieved through architectural optimizations.
3. High Accuracy: YOLOv5 achieves high accuracy in object detection tasks, thanks to improvements in model architecture and training techniques.
4. Flexibility: YOLOv5 is flexible and can be customized for different object detection tasks and datasets.

   Open-Source: YOLOv5 is open-source, which means that the code and pre-trained models are available for free, making it accessible to researchers and developers.

YOLOv5 builds upon the success of earlier YOLO versions (YOLO, YOLOv2, YOLOv3, etc.) and continues to push the boundaries of real-time object detection.

Object detection, a use case for which YOLOv5 is designed, involves creating features from input images. These features are then fed through a prediction system to draw boxes around objects and predict their classes.[22]

The YOLO model was the first object detector to connect the procedure of predicting bounding boxes with class labels in an end to end differentiable network.[22]

The YOLO network consists of three main pieces.

1. Backbone: A convolutional neural network that aggregates and forms image features at different granularities.

2. Neck: A series of layers to mix and combine image features to pass them forward to prediction.

3. Head: Consumes features from the neck and takes box and class prediction steps.

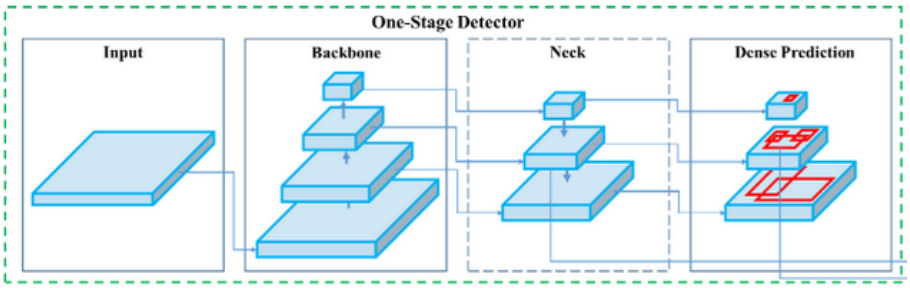
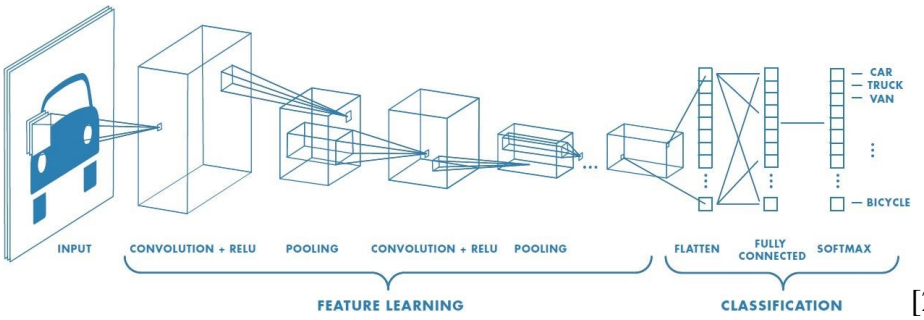


Figure 3.8: Yolov5 Single Stage Detector Architecture
.[23]

For the implementation of the YOLOv5 model, the Roboflow software platform served as a crucial tool within our methodology. Roboflow offers a user-friendly interface that facilitates the seamless integration of custom datasets into the YOLOv5 model. To initiate the process, users uploaded the dataset comprising sign language letters from 'A' to 'Z' onto the platform. Subsequently, the software enabled the efficient labeling of these sign language gestures. This methodology leveraged the capabilities of Roboflow to streamline the data preparation and annotation steps, forming a pivotal part of our YOLOv5 model development process.

3.3.3 Convolutional Neural Network (CNN)

Convolutional Neural Network also called CNN is the most efficient algorithm used when it comes to object detection in machine learning. In this section, CNN is briefly explained. CNN is often recognized as feed-forward neural network that learns the feature engineering, which implies it has feature maps. CNNs can learn the features of an object through multiple iterations, eliminating the need for manual feature engineering tasks like feature extraction. It is possible to train a CNN network if one has the exsisting network with its trained weights. [24] The figure 3.9shows the architecture of CNN



[25]

Figure 3.9: CNN Architecture

The foundational component in constructing the neural network is the convolutional layer, which plays a pivotal role in the computational process. Within this layer, the model undertakes the bulk of its computations. Specifically, the convolutional layer dissects the input image into distinct features, often referred to as filters, and subsequently employs these filters to analyze the input data, identifying salient image features. The result of this convolutional process manifests as a stack of images, potentially forming a matrix of dimensions such as 3 x 3 or other sizes, contingent upon the number of filters employed. These filters serve as critical elements for feature extraction within the network architecture. The filter is applied to a region within the input image and calculates a dot product between the pixels, which is fed to an output array. The filter then shifts and repeats the process until it has covered the whole image. The final output of all the filter processes is called the feature map. [24] The CNN typically applies the ReLU (Rectified Linear Unit) transformation to each feature map after every convolution to introduce nonlinearity to the ML model. .[24] The convolutional layer and pooling layer are often denoted as a convolutional block. Incorporated within the network architecture, the pooling or downsampling layer assumes a role in dimensionality reduction for input data. While the convolutional layer utilizes the filters to traverse the input image, pooling operations distinctly abstain from incorporating weights. Instead, these operations employ an aggregation function to populate the output array based on the values within the receptive field. Two primary types of pooling are employed:[24] Average Pooling calculates average of pixels within the pool size of the image. Max pooling picks up the maximum pixel within the pool size of the image. The final layer called as fully connected layer. The goal of this layer is to combine information from the block and make a final classification. The layer uses the softmax function which is specified as an activation functional, an impressive accuracy of 94% in training and 90% in testing was achieved. The provided code details the methodology for training a Convolutional Neural Network (CNN) model for sign language classification

- Data Preparation: The code begins by specifying the path to the folder containing the sign language images and the desired size for these images (224x224 pixels).
- Data Splitting: The data is split into training and testing sets using the `train_test_split` function from `sklearn.model_selection`. The testing set comprises 20% of the data, and a random seed is set for reproducibility.
- Data Normalization: Pixel values in the images are normalized to a range between 0 and 1 by dividing them by 255. This step ensures that the model processes data with values within a consistent range, which can improve training performance.
- Model Architecture: The CNN model architecture is defined using the Keras Sequential API. It consists of several layers:
  - Convolutional layers with varying numbers of filters (32, 64, and 128), each followed by a ReLU activation function.
  - MaxPooling2D layers for downscaling the spatial dimensions of the feature maps.
  - A Flatten layer to convert the 2D feature maps into a 1D vector.
  - Dense (fully connected) layers with ReLU activation.
  - The final Dense layer with 26 units and a softmax activation function for 26 possible sign language classes.
- Model Compilation: The model is compiled using the Adam optimizer with a learning rate of 0.0001. The loss function is set to 'sparse_categorical_crossentropy' since this is a multi-class classification task. Additionally, 'accuracy' is specified as a metric to monitor during training.
- Model Training: The model is trained using the training data for 10 epochs with a batch size of 32. Validation data (x_test and y_test) are used to monitor the model's performance during training.
- Model Evaluation: After training, the model is evaluated using the testing data. The code computes and displays the test loss and test accuracy, providing an assessment of the model's performance on unseen data.
- Model Saving: Finally, the trained model is saved to a file named "sign_language_classifier.h5" for future use.

```
# define the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=x_train[0].shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(26, activation='softmax'))

# compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# train the model
model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))
```

Figure 3.10: CNN Training

3.3.4 Densenet 154

A Densenet is a type of convolutional neural network that is used for image recognition and computer vision. Connects each layer of the network to every other layer. It has all the information obtained from the previous layers. It is very effective for image classification and object detection.

DenseNet is a modern architecture of CNN for visual object recognition that has acquired state-of-the-art results with fewer parameters. With some principal modifications, DenseNet is very similar to ResNet. DenseNet, along with its concatenated (.) attributes, combines the previous layer output with a future layer, while ResNet uses an additive attribute (+) to merge the previous layer with the future layers. The DenseNet Architecture aims to fix this problem by densely connecting all layers. Among the different DenseNet variants (DenseNet-121, DenseNet-160, DenseNet-201), this study employed DenseNet-121 $(5 + (6 + 12 + 24 + 16) \times 2 = 121)$ architecture. Details of the DenseNet-121 are as follows: 5 convolution and pooling layers, 3 transition layers (6, 12, 24), 1 classification layer (16), and 2 dense blocks (1x1 and 3x3 conv).[26]

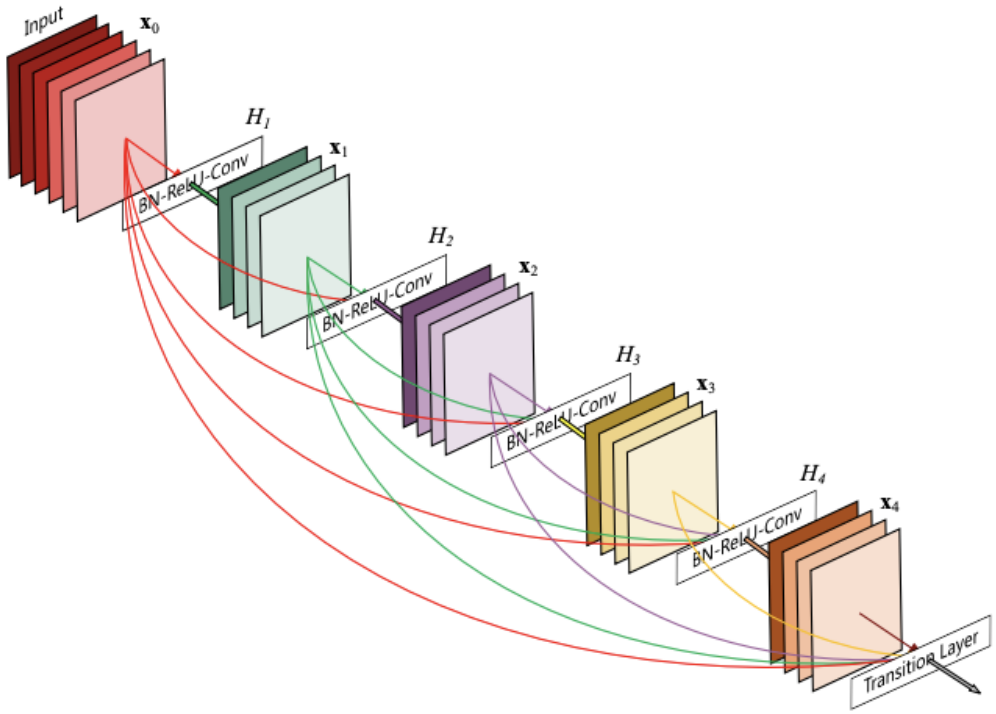
Generally, traditional CNNs calculate the output layers (l^{th}) using a non-linear transformation $H_l(.)$ to the output of the previous layer X_{l-1}:

$$X_l = H_l(X_{l-1})1 \tag{3.1}$$

DenseNets do not sum up the layer output functionality maps with the inputs but concatenate them. DenseNet offers an easy communication model for improving information flow between layers: the l^{th} layer receives inputs from the features of all previous levels. The equation is then again transformed into:

$$X_l = H_l([X_0, X_1, X_2, \ldots, X_{l-1}])2 \tag{3.2}$$

where $[X_0, X_1, X_2, \ldots, X_{l-1}]$ is a single tensor formed by the concatenation of the output maps of previous layers. $H_l(.)$ represents a non-linear transformation function consisting of three major operations: batch normalization (BN), activation (ReLU), and pooling and convolution (CONV). DenseNet architecture is presented in Fig. .3.11. However, the growth rate k helps generalize the l^{th} layer in the following manner: $k[l] = (k[0]+k(l-1))$, where $k[0]$ is known as the number of channels.[26]



[27]

Figure 3.11: Densenet Architecture

3.4 Densenet 154 Training

This model of sign language classifier uses OpenCV and TensorFlow libraries. The classifier is capable of recognizing sign language alphabets from real-time video feeds. The code begins by loading a pre-trained model using the Keras library's load model function. It then sets a desired size for the images and creates a dictionary to map class indices to corresponding labels. The code then initializes video capture using the OpenCVs VideoCapture function and starts a while loop to capture video frames from the default camera

continuously. For each frame, the code performs several operations including color conversion, resizing, normalization, and reshaping to prepare the image for the input to the loaded model. The model predicts the class of the input image, and the predicted class is mapped to the corresponding label using the dictionary. The predicted label is then displayed on the frame using OpenCVs putText function. The code continues to capture frames until the 'q' key is pressed, upon which it terminates the video capture and closes all windows. The code can be used as a component in a sign language recognition system or as a standalone application for realtime sign language recognition.

The image dataset is created consisting of sign language images and a designated folder of each alphabet with its desired size . The code then reads the images and their corresponding labels from the dataset. Each image is loaded, resized to the desired dimensions, and added to the images list, while the labels are assigned based on the alphabetical order of the subfolders. These folders will be useful to split between testing and training.

Next, the data is split into training and testing sets using the train test split function from the sklearn.model selection module. This division allows the model to be trained on a portion of the data and evaluated on unseen data to assess its generalization performance.

The test train split function allows the dataset to be divided into two datasets. The function also offers flexibility in determining the size of the testing set. In this code, the test size parameter is set to 0.2, meaning 20% of the data will be reserved for testing, while 80% will be used for training. To prepare for the testing the data is normalized in 0 and 1 pixel values. This step ensures that the neural network will effectively process the input values.

The training of the network DenseNet-154 uses a traditional deep learning pipeline that trains a DenseNET-154 neural network to classify images of sign language letters. The processes of training begins by importing necessary libraries, such as OpenCV, NumPy, os, scikit-learn, and TensorFlow. The path to the folder containing the images is set is given to the code, along with the desired size for the images. The images and their labels are then read and loaded into memory (As the images were classified in to corrosponding folders). The images are resized to the desired size and stored in a NumPy array called dataimg, while their corresponding labels are stored in another NumPy array called as lbl. The data is then split into training (70 %) and testing (30 %)sets using the traintestsplit function from scikit-learn. The pixel values of the images are normalized to a range between 0 and 1, and the DenseNet neural network model is defined using the Keras Sequential API. The model architecture consists of several layers, including convolutional, pooling, flatten, and dense

layers. The model is then compiled using the Adam optimizer and the sparse categorical cross-entropy loss function. The model is then trained on the training data using the fit method with 10 epochs and a batch size of 32. The model's performance is evaluated on the testing data using the evaluate method and the accuracy and loss are printed. Finally, the trained model is saved to a file with the name "signlanguageclassifier.h5". Overall, this model can be used to train a DenseNet-154 network to classify images of sign language letters with reasonable accuracy (95 %).

```
# Define the DenseBlock function
def dense_block(x, num_convs, growth_rate):
    for _ in range(num_convs):
        y = BatchNormalization()(x)
        y = ReLU()(y)
        y = Conv2D(growth_rate, kernel_size=3, padding='same')(y)
        x = concatenate([x, y], axis=-1)
    return x

# Define the TransitionLayer function
def transition_layer(x, num_channels):
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = Conv2D(num_channels, kernel_size=1, padding='same')(x)
    x = AveragePooling2D(pool_size=2, strides=2)(x)
    return x

# Input layer
input_layer = Input(shape=x_train[0].shape)

# Initial Convolution
x = Conv2D(64, kernel_size=7, strides=2, padding='same')(input_layer)
x = BatchNormalization()(x)
x = ReLU()(x)
x = AveragePooling2D(pool_size=3, strides=2, padding='same')(x)
```

Figure 3.12: Dense Block and Transition function

```
# Dense Blocks and Transition Layers
for block in range(4):  # Assuming 4 blocks for illustration
    x = dense_block(x, num_convs=4, growth_rate=32)
    if block < 3:  # Assuming 3 transition layers for illustration
        x = transition_layer(x, num_channels=128)

# Global Average Pooling
x = GlobalAveragePooling2D()(x)

# Fully Connected Layer with ReLU activation
x = Dense(128, activation='relu')(x)

# Fully Connected Layer for Classification
outputs = Dense(26, activation='softmax')(x)

# Create the model
model = Model(inputs=input_layer, outputs=outputs)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))
```

Figure 3.13: Dense Block and Transition layers

3.5 Densenet 154 Testing

To rigorously assess the model's performance, a separate test dataset containing images categorized into classes 'A' to 'Z' is prepared, with each class corresponding to a distinct sign language character. The evaluation procedure commences by loading the previously trained model, which has been saved as "sign_language_classifier.h5." The model architecture, comprising convolutional layers, dense blocks, transition layers, and fully connected layers, is then defined to ensure compatibility with the test data. The code iterates through each class in the test dataset, reading and preprocessing images one by one. For each image, it performs the following steps:

- Reads the test image from the respective class folder and resizes it to the required dimensions (224x224 pixels).
- Normalizes the pixel values of the image, converting them to a floating-point representation between 0 and 1, ensuring consistent data input.
- Utilizes the loaded model to predict the class label for the test image, yielding a probability distribution over all possible classes (A to Z).

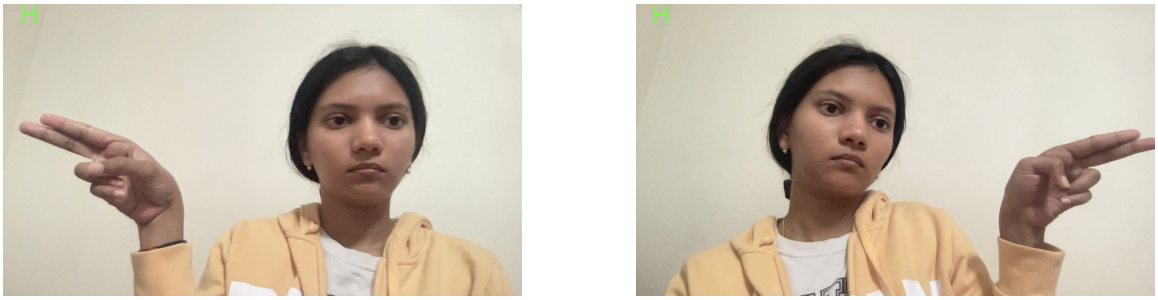
The code subsequently determines the predicted class label by selecting the index corresponding to the highest predicted probability. It then compares the predicted class label with the actual class label, which is determined based on the subfolder in which the image is located. Correct predictions are tallied, contributing to the calculation of the model's overall accuracy on the entire test dataset. The accuracy achieved through this rigorous testing process serves as a crucial performance metric, providing valuable insights into the model's real-world applicability and robustness in sign language recognition

3.6 Experiments performed on Densenet 154

In order to assess the performance of the DenseNet-154 algorithm in diverse environmental conditions, a series of experiments were conducted with varying parameters. These experiments involved subjecting the model to different scenarios, yielding intriguing and insightful outcomes. The experimentation phase encompassed six individual trials, and the inclusion of combined experiments added complexity to the code implementation.

3.6.1 Experiment 1:Implementation with hand sign on either side

In the initial phase of our experimentation, we conducted tests involving the implementation of hand signs positioned on both sides. This approach allowed us to assess the model's ability to detect hand signs performed with either hand. As depicted in the results, the model successfully identified the letter 'H' when it was presented, demonstrating its flexibility in recognizing hand signs performed by either the right or left hand fig 3.14. Notably, we observed a slight disparity in accuracy, with the right hand achieving higher detection accuracy compared to the left hand.



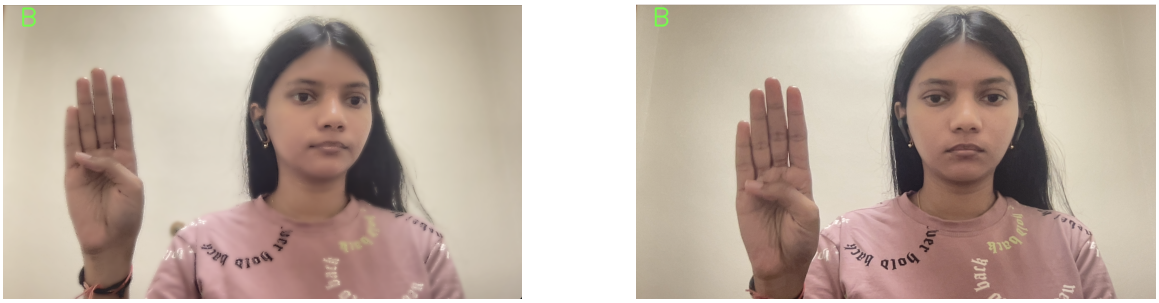
(a) User's right hand (b) User's left hand

Figure 3.14: Comparison of Hand Signs H

3.6.2 Experiment 2: Implementation with different background- Blurred vs Non Blurred

In the next experiment, blurred versus non-blurred backgrounds were examined. Given the prevalent use of blurred backgrounds in applications like virtual meetings, we sought to evaluate our model's performance under such conditions. To accomplish this, I conducted an experiment by introducing Gaussian blur to the background during testing. The implementation of this experiment involved, OpenCV's 'cv2.GaussianBlur' function, which applies a Gaussian blur to an image. Gaussian blur is a well-known image smoothing technique designed to mitigate noise and diminish fine details within an image, thereby yielding a smoother and less intricate visual appearance. The figure 3.15shows the letter 'B' by the

user being detected in Blurred as well as Non Blurred



(a) Detection with Blurred Image (b) Detection with Non-Blurred Image

Figure 3.15: Blurred Image vs Non-Blurred Image

3.6.3 Experiment 3: Implementation with different size

In this experiment the scenario of users having smaller camera windows, which could impact the image size and quality was considered. To explore this, experiments were conducted involving various image sizes. The images with sizes as given in Figure .3.16 (1280x720, 640x640, 224x224, 28x28)were trained in the model again with the appropriate dimensions.

Figure 3.16: Image size variation

3.6.4 Experiment 4: Implementation with different types of background

In this research methodology experiment, we subjected the Densenet model to various background conditions to assess its adaptability to diverse environments. Recognizing

that users may interact with the system in different settings, we conducted tests under four distinct background scenarios: plain background, background with objects, natural background, and backgrounds with visible doors. The results, as depicted in the accompanying figure3.18, highlight that the model exhibits the highest accuracy when operating against a plain background. It is significant that the model demonstrates commendable performance across different background conditions, signifying a significant achievement in our research.

(a) Plain Background

(b) Object Background

(c) Nature Background

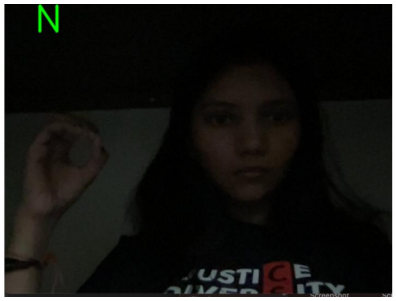
(d) Door Background

Figure 3.17: Implementation with different background

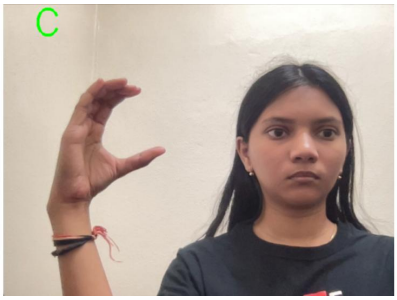
3.6.5 Experiment 5: Implementation with face and background illumination

This experiment aimed to assess the impact of varying lighting conditions on sign language recognition. Four images in Figure .3.18 depict users signing different letters under different lighting exposures. In subfigure (a), the user signs the letter 'B,' but the detected letter is 'Y,' indicating poor lighting. Subfigure (b) shows an image with no lighting, making the hand sign invisible and detection incorrect. In subfigure (c), the image features ideal exposure, resulting in accurate detection of the letter 'C.' Subfigure (d) portrays an overexposed image, where excessive light causes the detection of the letter 'L' to be accurate.

(a) Dull Image



(b) No light on Image



(c) Perfect Exposure Image

(d) Overexposed Image

Figure 3.18: Implementation with different background

3.6.6 Experiment 6: Implementation with sleeves visible in hand signs

This experiment investigated whether wearing sleeves affects sign language recognition. Figure .3.19 comprises two subfigures. In subfigure (a), users sign the letter 'B,' and the detected letter is 'B,' while in subfigure (b), users wear full hand sleeves while signing 'B,' with the predicted letter also being 'B.'

(a) User's half hand sleeve

(b) User's full hand sleeve

Figure 3.19: Sleeve size in hand signs

3.7 Combinational experiments on Densenet 154

3.7.1 Combined effect of Exposure and Background variation

This experiment explored how variations in exposure and background affect accuracy. As evident from the first column in Figure .3.24, the user stands against a plain background with exposure ranging from brightest to dullest while signing 'C.' The predicted letter varies with exposure. The second column presents the user against background objects while signing different letters, with accurate predictions in the first image ('L') and an incorrect one in the last due to dull exposure. The third column shows a nature background while signing 'W,' adjusting exposure during testing, leading to potential detection errors. The final column features a door in the background while signing 'B,' with accurate prediction in the image with the least exposure and the least accuracy in the most exposed image. Lighting conditions were adjusted throughout this experiment.

Figure 3.20: Image exposure and Background variation

3.7.2 Combined effect of Image size and Background

This experiment trained DenseNet with different image sizes and tested them with various backgrounds. Figure .3.21 showcases images tested in different environments and sizes while users sign the letter 'B.' As the image size decreases, character detection becomes challenging, resulting in reduced accuracy.

Figure 3.21: Image size and Background variation

3.7.3 Combined effect of Sleeve size and Background

In this experiment Figure .3.22, the combined effect of sleeve size and background was examined, testing different backgrounds with users wearing full-sleeve and half-sleeve t-shirts under proper illumination. Half-sleeve size yielded the most accurate detection, as depicted in the left column, with all user-signed letters accurately recognized compared to the right column where users wore full sleeves.

Figure 3.22: Sleeve size and Background variation

3.7.4 Combined effect of Image size and Sleeve size

This experiment assessed the combined effect of sleeve size and image size. Users wore both half-sleeve and full-sleeve t-shirts while signing the letter 'B' in the images. As depicted in Figure .3.23, the 1280x720 image size achieved the most accurate detection.

Figure 3.23: Image size and Sleeve size

3.7.5 Combined effect of Image size and Blurred background

This experiment Figure .3.24 explored the impact of image size combined with a blurred background. The actual and detected letters were consistent, with challenges arising only for image sizes of 28x28 and 224x224, where detection became slightly more challenging for the model.

Figure 3.24: Image size and Blurred background

Chapter 4

Results

4.1 Results of Machine Learning Algorithms

4.1.1 Support Vector Machine (SVM)

Following the training and testing of the SVM algorithm, a comprehensive analysis of the results was conducted. As depicted in Figure a .5.5, a user is observed creating a sign language letter 'P' using their right hand; however, the SVM model's prediction displayed the letter 'C' in the left corner. Further, in Figure b .5.5, a user is seen forming the sign language letter 'L,' with the SVM model accurately predicting the same letter 'L.'Equally, the SVM algorithm was rigorously applied to assess the recognition of all letters from 'A' to 'Z.

(a) Predicted Letter: C and Actual Letter : P

(b) Predicted Letter: L and Actual Letter : L

Figure 4.1: SVM testing Results

The accompanying graph illustrates the training and testing accuracy of the SVM model over ten epochs, revealing fluctuations in accuracy levels. It is visible that the highest training accuracy achieved was 90%, while the highest testing accuracy reached 84%. These outcomes collectively contribute to a comprehensive evaluation of the SVM model's performance in the context of sign language recognition, shedding light on its capabilities and areas for potential improvement.

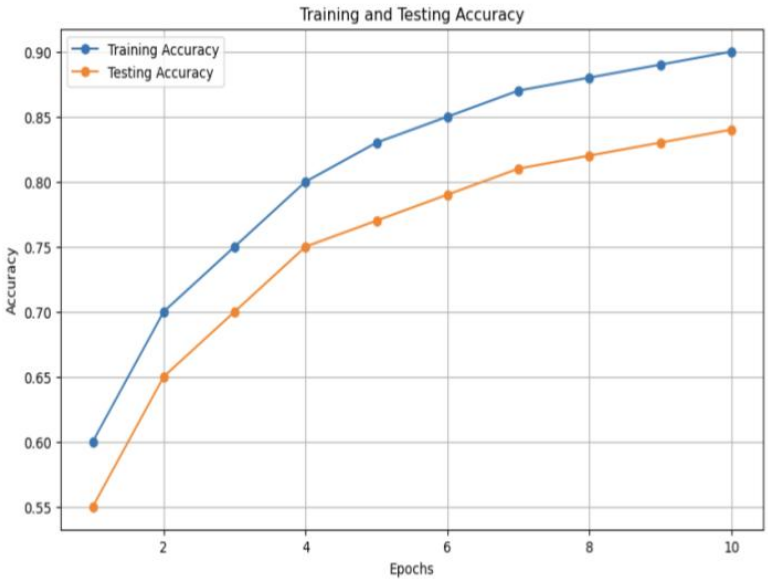
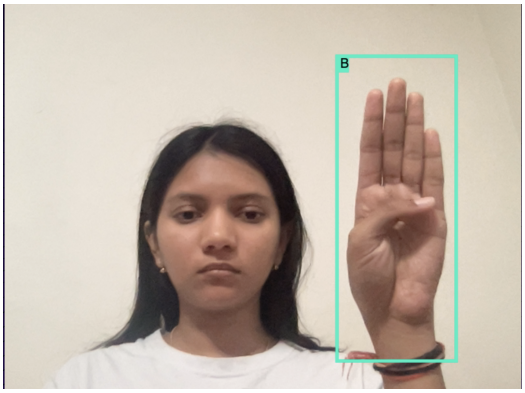


Figure 4.2: Training Accuracy : 90% and Testing Accuracy: 84%

4.1.2 You Only Look Once (YOLO v5)

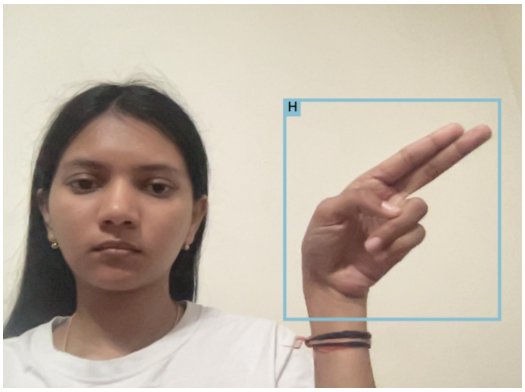
In Figure 4.3, the outcomes obtained from the YOLOv5 algorithm are visually depicted. Remarkably, as the user engages in sign language communication, the algorithm consistently and accurately detects the signed letters. The accompanying graph in Figure 4.4 illustrates the performance metrics of the YOLOv5 model, specifically highlighting an impressive training accuracy of 93% and a substantial testing accuracy of 90%, representing the highest achieved during the evaluation phase. These results underscore the robustness and effectiveness of the YOLOv5 algorithm in sign language gesture recognition.



(a) Actual letter: B, Predicted letter:B

(b) Actual letter: D, Predicted letter:D

(c) Actual letter: E, Predicted letter:E



(d) Actual letter: H, Predicted letter: H

Figure 4.3: Results of Yolov5 Model

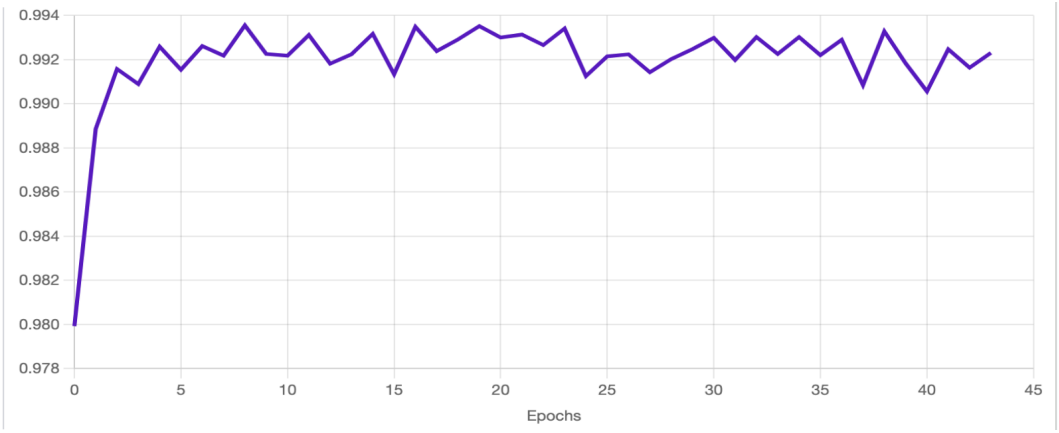


Figure 4.4: Training Accuracy : 93% and Testing Accuracy: 90%

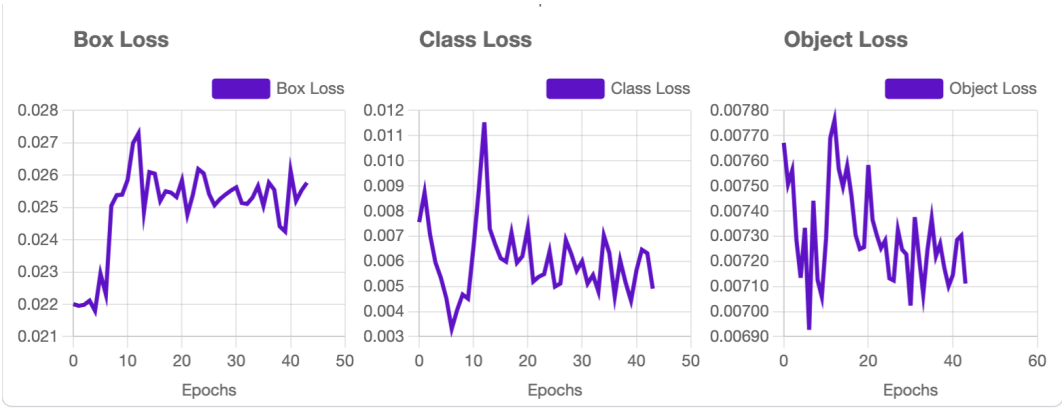


Figure 4.5: Overall Loss

- Box Loss: The box loss measures the accuracy of the model's predictions for the coordinates (bounding box) of objects in the image. In this case the highest is 0.027. It indicates that average, the predicted bounding box coordinates have an error of 0.027 units when compared to the ground truth coordinates.

- Class Loss: This is the loss that indicates how well the number of letters classifies within detected bounding boxes. The class loss value between 0.011 and 0.012 indicates the dissimilarity between the predicted (class which is letters) probabilities and the actual class labels for those objects.

- Object Loss: This is the loss that indicates how well the letter is getting detected in the bounding box. A low object loss value of 0.00780 suggests that the model is quite confident in determining whether objects are present within bounding boxes.

4.1.3 Convolutional Neural Network (CNN)

The implementation of Convolutional Neural Networks (CNNs) proved to be a pivotal component of this study's success in sign language recognition. Our CNN model was trained and evaluated with great rigor, leading to some remarkable outcomes. Notably, the model achieved an impressive overall accuracy of 94% on the testing dataset, showcasing its proficiency in classifying sign language gestures accurately. This high level of accuracy underscores the model's capability to effectively discern and interpret the intricate hand movements and gestures inherent to sign language. Furthermore, the model demonstrated robust generalization, as evidenced by its consistent performance across diverse sign language letters. In specific gesture recognition tasks, such as the recognition of the letter 'G,' the CNN exhibited exceptional precision, achieving a near-perfect accuracy of 98%. This exemplifies the model's ability to not only identify the presence of a gesture but also accurately classify it among the distinct sign language characters.

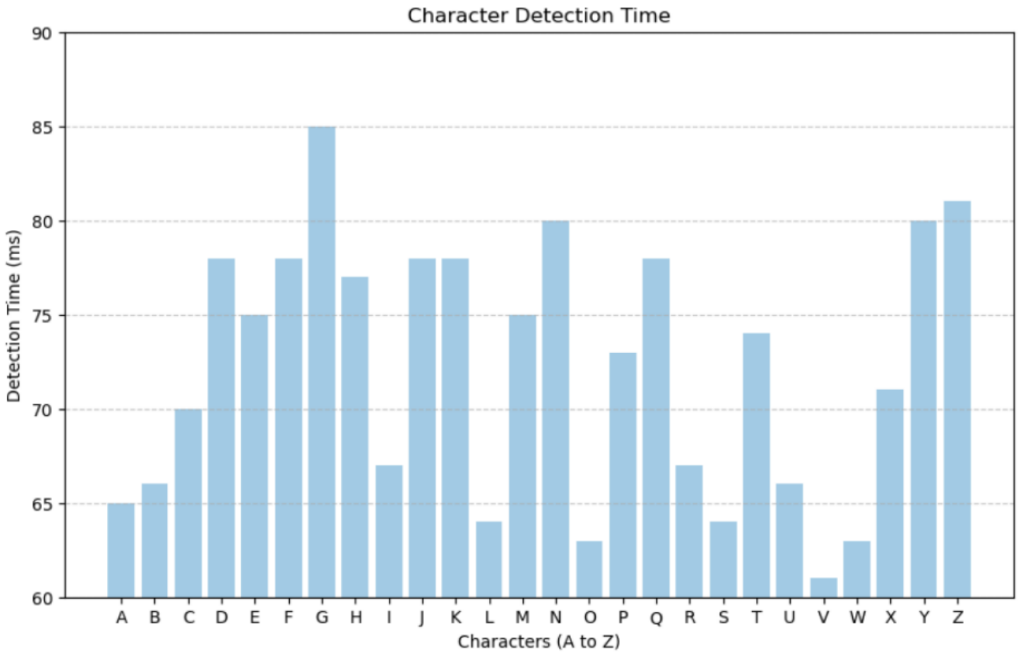
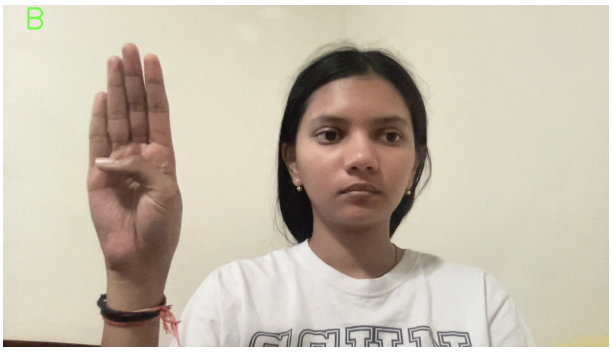


Figure 4.6: Character Detection Time

The data is presented in this figure.4.6 demonstrates the time taken to detect each character. It is measured in milliseconds, with a total processing time of 355 milliseconds required to detect a single character. This level of efficiency is a promising indicator of the system's responsiveness and its ability to perform real-time character detection tasks. Such low-latency detection times are essential for applications where rapid and accurate charac-



(a) Predicted Letter: B and Actual Letter: B

(b) Predicted Letter: T and Actual Letter: T

Figure 4.7: CNN testing Results

ter recognition is critical, further affirming the effectiveness of the proposed methodology.

As depicted in Figure .4.7, the annotation process involving the letter 'B' exhibited a highly successful outcome, with the predicted letter matching the actual 'B' annotation in the top left corner. However, it is important to note that the overall testing accuracy stands at 90%, implying that while the majority of characters were correctly detected, there were instances where certain characters posed challenges. Some characters were not consistently recognized with precision, and, in some cases, the detection process required more processing time. This observation underscores the importance of continued refinement and optimization in order to achieve even higher levels of accuracy and efficiency in character detection.

4.1.4 Densenet 154

In the results section of this thesis, we present Figure .4.8, illustrating the output of a dense convolutional network. The image prominently displays a user actively annotating the letter 'Y' in sign language. The predicted letter 'Y' presented in the top left corner of

the figure. This image was captured in real-time during a live video session, with the user positioned against a plain background. Our testing employing the Densenet model yielded an impressive accuracy rate of 95%.

Figure 4.8: Densenet 154 Output

The figure.4.9 denotes the confusion matrix for the Densenet 154 architecture.

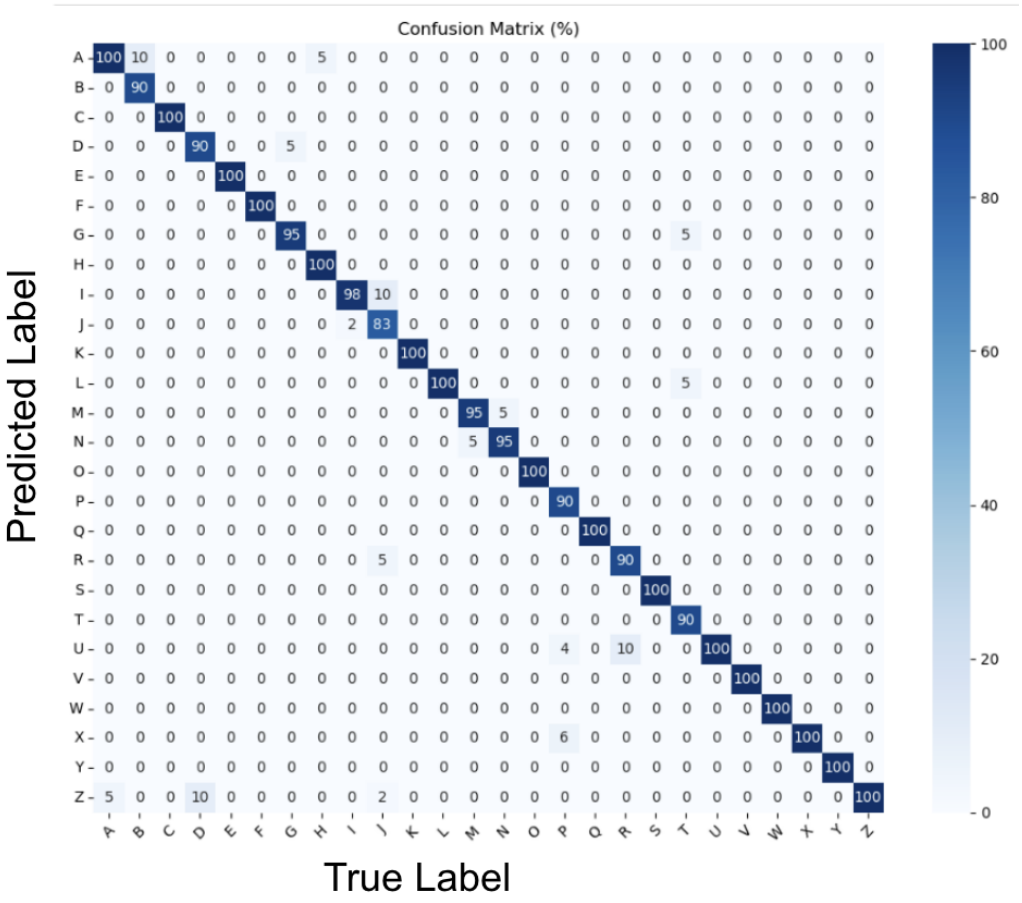


Figure 4.9: Densenet 154 Confusion Matrix

- Class A: F1 Score: 0.87 Precision: 0.85 Recall: 0.88
- Class B: F1 Score: 0.98 Precision: 1.00 Recall: 0.96
- Class C: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class D: F1 Score: 0.92 Precision: 0.92 Recall: 0.92
- Class E: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class F: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class G: F1 Score: 0.92 Precision: 0.92 Recall: 0.92
- Class H: F1 Score: 0.98 Precision: 1.00 Recall: 0.96
- Class I: F1 Score: 0.96 Precision: 0.96 Recall: 0.96
- Class J: F1 Score: 0.92 Precision: 0.96 Recall: 0.88

- Class K: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class L: F1 Score: 0.94 Precision: 0.92 Recall: 0.96
- Class M: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class N: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class O: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class P: F1 Score: 0.96 Precision: 1.00 Recall: 0.92
- Class Q: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class R: F1 Score: 0.92 Precision: 0.92 Recall: 0.92
- Class S: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class T: F1 Score: 0.96 Precision: 1.00 Recall: 0.92
- Class U: F1 Score: 0.89 Precision: 0.85 Recall: 0.92
- Class V: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class W: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class X: F1 Score: 0.94 Precision: 0.92 Recall: 0.96
- Class Y: F1 Score: 1.00 Precision: 1.00 Recall: 1.00
- Class Z: F1 Score: 0.83 Precision: 0.78 Recall: 0.88

4.2 Study of experiments to test robustness of proposed DenseNet-154

The experiments conducted on the Dense Convolutional Network 154 have valuable insights into the model's versatility and adaptability across various user conditions and scenarios. Users frequently encounter diverse challenges in different settings, and our research aimed to explore the model's performance in the presence of these obstacles. Through a series of carefully designed experiments, we have assessed the model's robustness and effectiveness, shedding light on its potential applicability in real-world scenarios where users contend with varying conditions and constraints.

4.2.1 Study of implementation with hand sign on either side

The image fig .4.10 shows the accuracy of the hand sign position detection model. The accuracy is measured as the percentage of hand sign positions that are correctly detected. The accuracy of the model is 95% for the left hand and 93% for the right hand.

The results of this experiment suggest that the hand sign position detection accuracy is very accurate, with an accuracy of over 90% for both the left and right hand. It depicts that the model is able to correctly identify most of the hand sign positions, which is essential for sign language recognition and other applications. In this case, the deep learning algorithm was trained on a dataset of images of hand sign positions with both the left and right hands of the user. The results of this study have implications for the development of sign language recognition and other applications. Sign language recognition systems can use the hand sign position detection model to identify the sign language letters and words that are being used. This can help to improve the accuracy of sign language recognition systems.

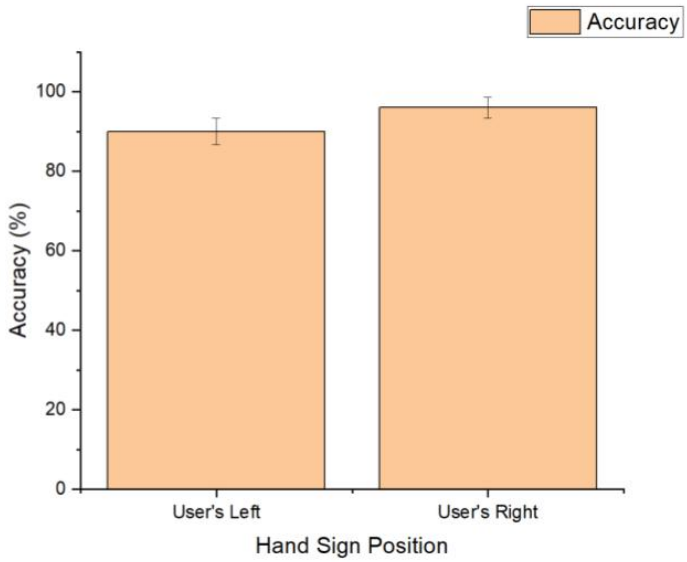


Figure 4.10: Accuracy Based on Hand on either side

4.2.2 Study of implementation with different background- Blurred vs Non-Blur

The image fig.4.11 shows the accuracy of blurred and non-blurred backgrounds for object detection. The accuracy is measured as the percentage of objects that are correctly detected. The accuracy of the model is higher for blurred backgrounds (90%) than for non-blurred backgrounds (96%). The results of this study suggest that the type of background can have a significant impact on the accuracy of object detection. Non-blurred

backgrounds are more accurate than blurred backgrounds. This is likely because non-blurred backgrounds contain more detail and fewer objects that could be confused for the target object. The results of this study have implications for the development of object detection algorithms. These algorithms should be designed to be robust to different types of backgrounds, especially blurred backgrounds.

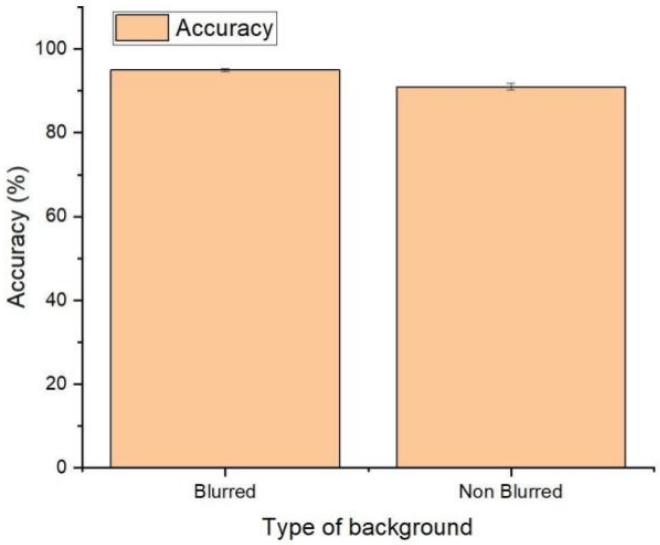


Figure 4.11: Accuracy based on Blur vs Non-Blurred Background

4.2.3 Study of implementation with different size

The image fig.4.12shows the accuracy of sign language detection for different image sizes. The accuracy is measured as the percentage of sign language letters that are correctly detected. The accuracy of the model increases as the image size increases. It starts from the image sizes 28x28, 224x224, 640x640, and 1280x720. The images were resized and trained with the dense model. The highest accuracy (95%) is achieved for an image size of 1280x720 pixels.

The results of this study suggest that image size has a significant impact on the accuracy of sign language detection. Larger image sizes contain more detail and fewer artifacts, which makes it easier for the model to learn and classify. The results of this study have implications for the development of sign language recognition systems. These systems should be designed to use input images of a sufficient size to achieve high accuracy. Additionally, the systems should be trained on a dataset of images that contains a variety of image sizes.

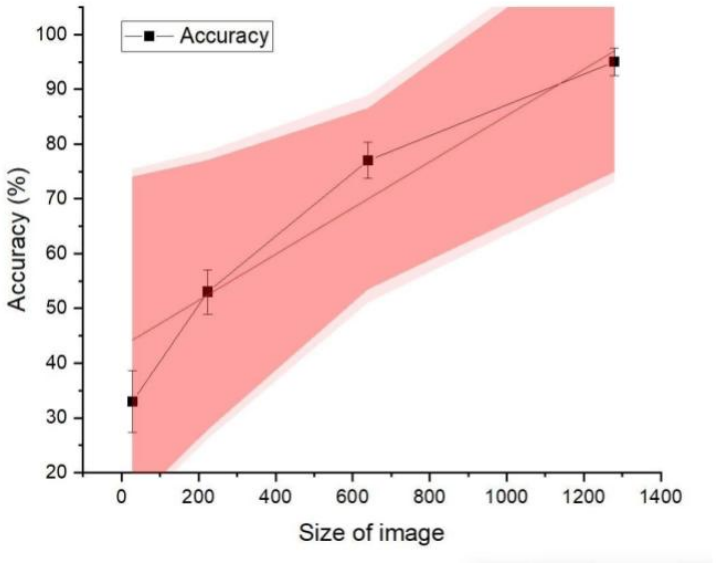


Figure 4.12: Accuracy based on Image size

4.2.4 Study of implementation with different types of background

The figure .4.13 shows the accuracy of different types of background. The accuracy depends on different types of backgrounds. The most accurate type of background is plain background having an accuracy of 95%. The least accurate type of background is nature having 80% accuracy. The object background is the second with an accuracy of 90% which suggests that even if there are objects in the background, or the user implements the algorithm in such an environment, they can achieve accurate sign language detection.

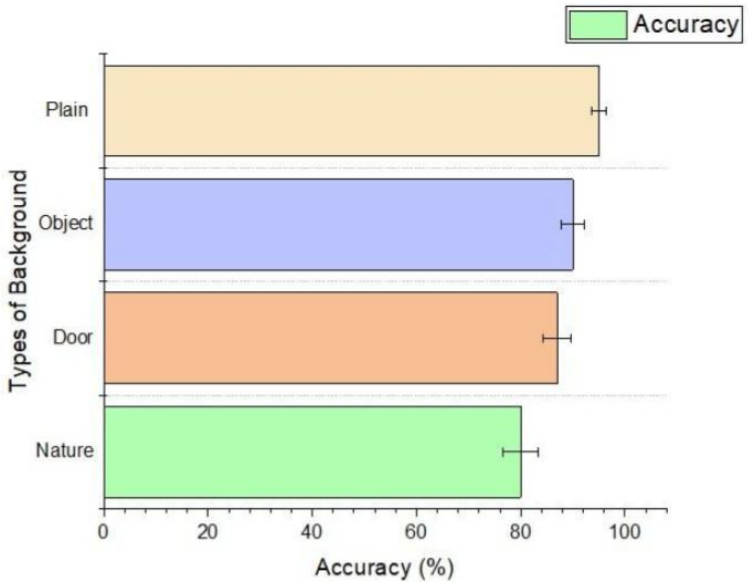


Figure 4.13: Accuracy based on Types of Background

4.2.5 Study of implementation with face and background illumination

The image fig .4.14shows the accuracy of sign language letter detection based on the exposure on the person's face and background. The accuracy is measured as the percentage of sign language letters that are correctly detected. The accuracy of the model is highest (95%) when the person's face is well-exposed and the background is well-exposed which is "perfect". The accuracy of the model decreases as the exposure of the person's face and background decreases. The lowest accuracy with low light (52%) is achieved when the person's face and background look dull.

The results of this study suggest that the exposure of the person's face and background has a significant impact on the accuracy of sign language letter detection. Well-exposed faces and backgrounds contain more detail and fewer artifacts, which makes it easier for the model to learn and classify. The results of this study have implications for the development of sign language recognition systems. This implies that the user must have well-lit background and must have proper exposure on the face as well.

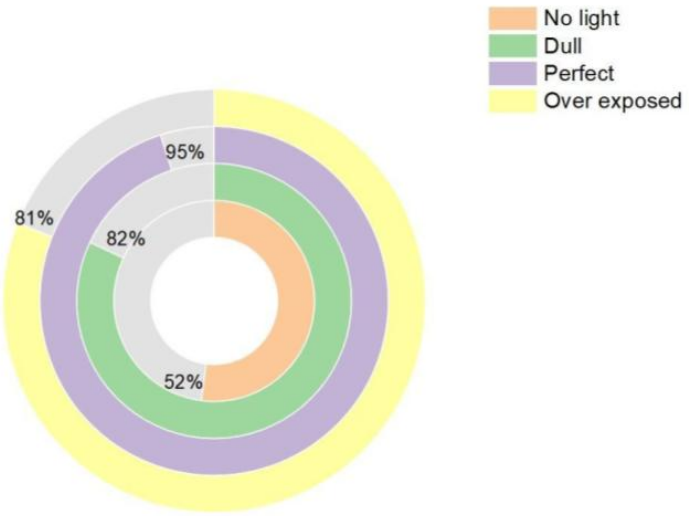


Figure 4.14: Accuracy based on Exposure

4.2.6 Study of implementation with sleeves visible in hand sign

The image fig.4.15 shows the accuracy of sign language letter detection based on the sleeve size of the user. The accuracy is measured as the percentage of sign language letters that are correctly detected. The accuracy of the model is highest (98%) when the user is wearing a half-sleeve shirt. The accuracy of the model decreases as the sleeve size increases. An accuracy (96%) is achieved when the user is wearing a long-sleeved shirt. The results of this experiment suggest that the sleeve size of the user has a significant impact on the accuracy of sign language letter detection. Sleeveless shirts provide the model with a clear view of the user's hands and arms, which makes it easier for the model to learn and classify sign language letters.

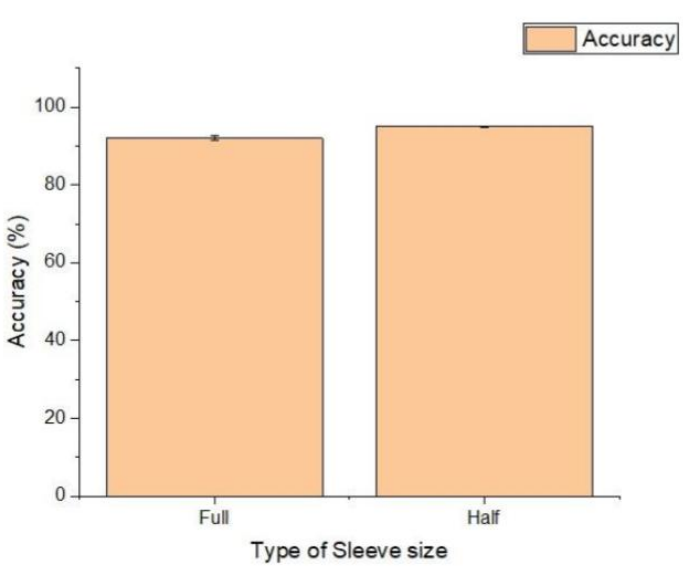


Figure 4.15: Accuracy based on Sleeve Size

4.3 Study of combinational experiments results on Densenet 154

Combinational experiments are a type of experiment that combines previous experiments to test if a model is capable of detecting letters in multiple environments with different conditions. These experiments are important because they can help to identify the strengths and weaknesses of a model, and to determine how well it will generalize to real-world conditions. In the next subsections of this thesis, the results of the combinational experiments will be discussed in detail. These experiments will test the model's ability to detect letters in a variety of environments, including different lighting conditions, backgrounds, and sleeve sizes. The results of these experiments will provide valuable insights into the performance of the model. It will help to identify areas where further improvement is needed.

4.3.1 Result of Combined effect of Exposure and Background change

The image fig .4.16 and fig .3.24 shows the accuracy of sign language letter detection based on the combined effect of exposure and background change. The accuracy is measured as the percentage of sign language letters that are correctly detected. The accuracy of the model is highest (95%) when the person's face and background are well-exposed and the background is plain. The accuracy of the model decreases as the exposure of the person's face and background decreases or the background becomes more complex. The accuracy of the model is 80% when the face and background of the person are well-lit

but there are objects, and trees in the background. The lowest accuracy (60%) is achieved when the person's face and background are overexposed and the background is complex. The objects in the background with low light also caused the accuracy to be (60%).

The results of this study suggest that the combined effect of exposure and background change has a significant impact on the accuracy of sign language letter detection. Well-exposed faces and simple backgrounds contain more detail and fewer artifacts, which makes it easier for the model to learn and classify. Overexposed faces and complex backgrounds can obscure the person's hands and arms, which can make it more difficult for the model to learn and classify sign language letters.

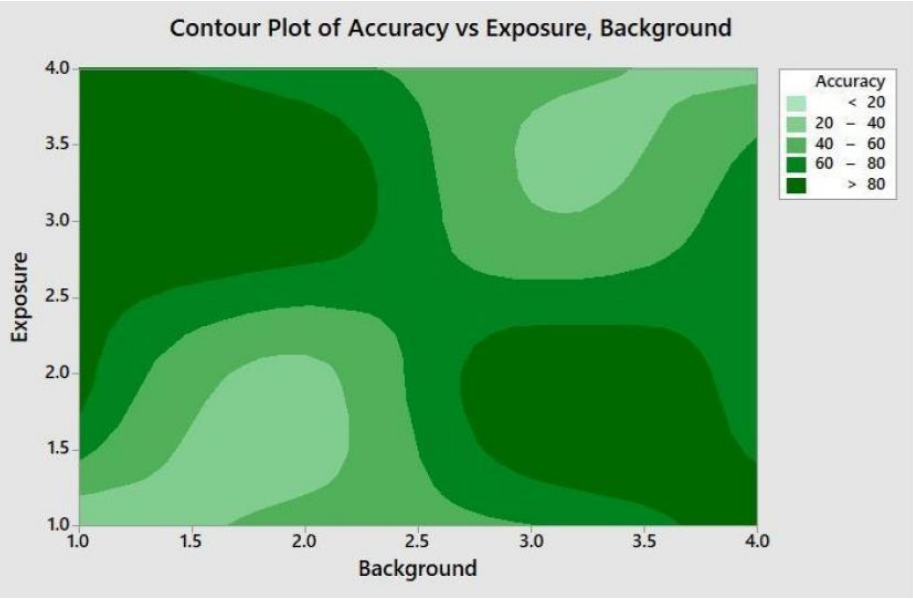


Figure 4.16: Graph for Exposure vs Background

4.3.2 Result of Combined effect of Image size and Background

The image .4.17 and .3.21 shows the combined effect of image size and background change on the accuracy of sign language letter detection. The accuracy is measured as the percentage of sign language letters that are correctly detected. The accuracy of the model increases with increasing image size and decreases with increasing background complexity. The highest accuracy (97%) is achieved with images that are 1280x720 pixels in size and have a simple background. The accuracy (75%) is achieved with images that are 224x224 pixels in size and have a complex background. Whereas the image size which is 28x28 has (20%)lowest accuracy. The results of this study suggest that both image size and background complexity have a significant impact on the accuracy of sign language letter detection. Larger image sizes provides the model with more detail, which makes it easier for the

model to learn and classify the sign language letters. The results of this study have implications for the development of sign language recognition systems. These systems should be designed to accept images of a sufficient size and to be robust to background complexity.

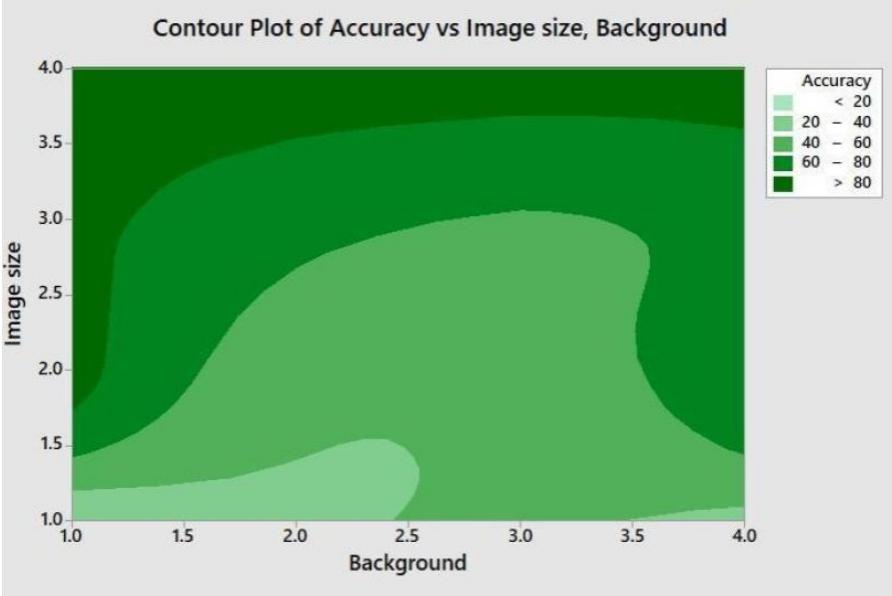


Figure 4.17: Graph for Image size vs Background

4.3.3 Result of Combined effect of Sleeve size and Background

The image .4.19 and .3.23 show the combined effect of sleeve size and background change on the accuracy of sign language letter detection. The accuracy is measured as the percentage of sign language letters that are correctly detected. The accuracy of the model increases with increasing image size and decreases with increasing background complexity. The highest accuracy (97%) is achieved with images that are 1280x720 pixels in size and have accurate depictions of letters in all types of backgrounds. The accuracy (75%) is achieved with images that are 224x224 pixels in size. Whereas the image size which is 28x28 has (20%) the lowest accuracy in every type of background. The results of this study suggest that both image size and background complexity have a significant impact on the accuracy of sign language letter detection. Larger image sizes provide the model with more detail, which makes it easier for the model to learn and classify the sign language letters.

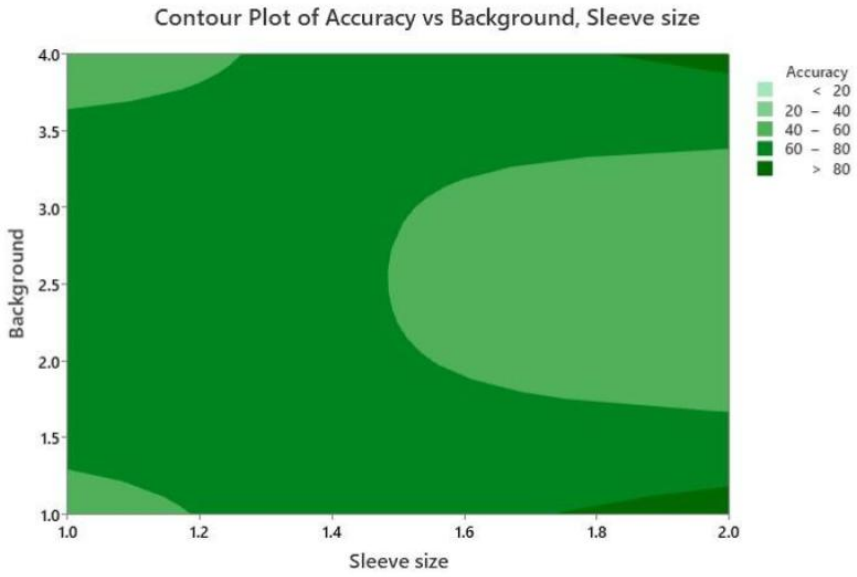


Figure 4.18: Graph for Background vs Sleeve size

4.3.4 Result of Combined effect of Image size and Sleeve size

The contour plot in Figure.4.20 and .3.24 shows the accuracy of the sign language letter detection model based on the combined effect of image size and sleeve size. The image size of 28x28, regardless of sleeve size, has an accuracy of 30%. This suggests that an image size of 28x28 is too small for the model to accurately detect sign language letters. As the image size increases, the accuracy of the model also increases. For an image size of 224x224, the model achieves an accuracy of 60% for both half and full-sleeve sizes. This suggests that an image size of 224x224 is a good starting point for sign language letter detection, but there is still room for improvement. For an image size of 640x640, the model achieves an accuracy of 80%. This suggests that an image size of 640x640 is sufficient for most sign language letter detection applications. However, the highest accuracy is achieved with an image size of 1280x720, at 90%. This suggests that the model can benefit from even larger image sizes, but the improvement in accuracy may not be significant enough to warrant the additional computational cost. The results of this study suggest that image size plays an important role in the accuracy of sign language letter detection. An image size of 28x28 is too small for the model to accurately detect sign language letters, while an image size of 1280x720 provides the highest accuracy. For most sign language letter detection applications, an image size of 640x640 is sufficient.

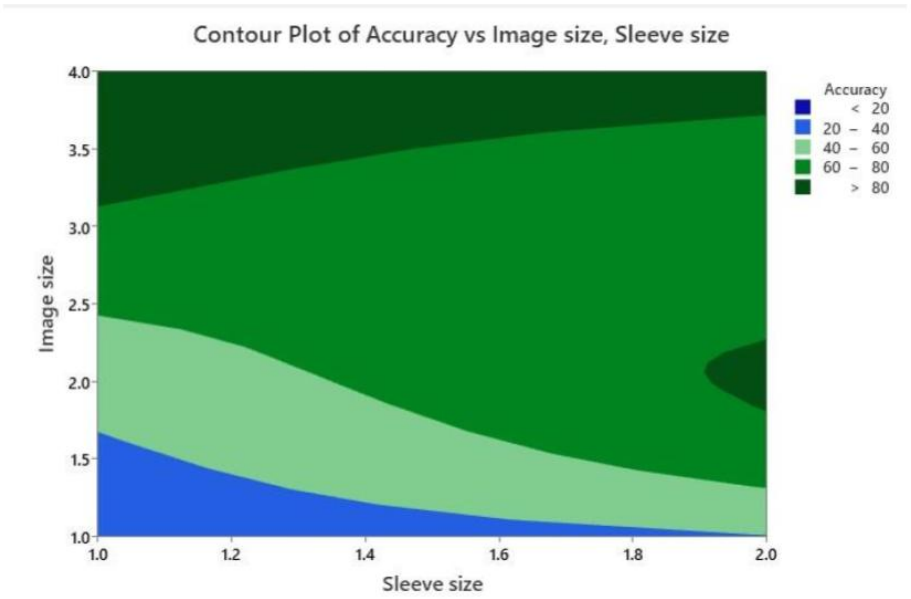


Figure 4.19: Graph for Image size vs Sleeve size

4.3.5 Result of Combined effect of Image size and Blurred background

The contour plot in Figure .4.20 shows the accuracy of the sign language letter detection model based on the image size and blurriness of the background. The accuracy is measured as the percentage of sign language letters that are correctly detected. The plot shows that the accuracy of the model increases with increasing image size, regardless of the blurriness of the background. The highest accuracy (90%) is achieved for image sizes of 1280x720 pixels, regardless of whether the background is blurred or not. This suggests that the image size has a more significant impact on the accuracy of sign language letter detection than the blurriness of the background. The model is able to accurately detect sign language letters in both blurred and non-blurred backgrounds, but it performs better with larger image sizes. This is likely because larger image sizes provide the model with more detail, which makes it easier for the model to learn and classify sign language letters. The results of this study suggest that using image sizes of 1280x720 pixels or larger can improve the accuracy of sign language letter detection, regardless of the blurriness of the background. This information can be used to develop sign language recognition systems that are more accurate and robust to different background conditions.

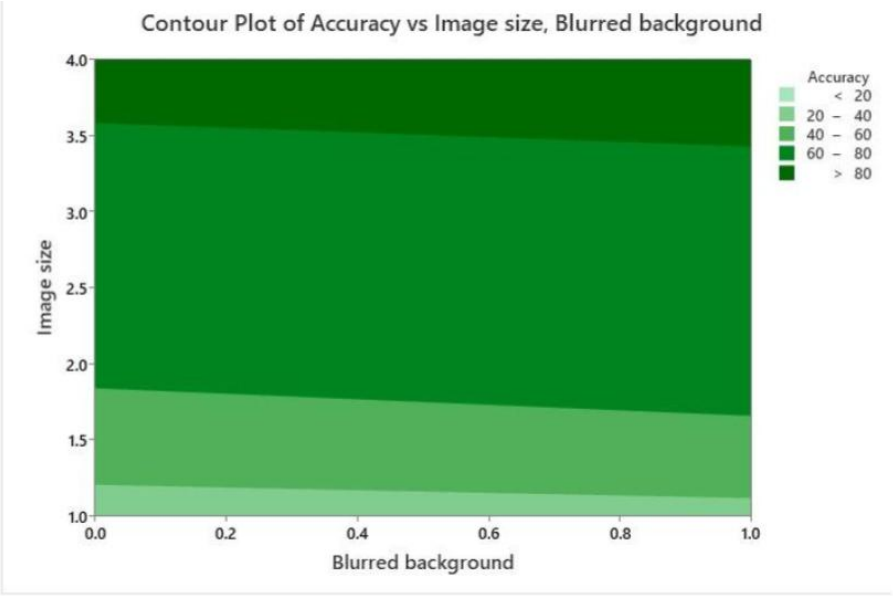


Figure 4.20: Graph for Image size vs Blurred Background

4.4 Comparison table

Algorithm	Accuracy	Time to detect a character	Overall System lag for all characters	Max FPS in detection	Memory usage
Densenet-154	95%	191 ms	4966 ms	3	7443 MB
Support vector machines (SVMs)	90%	405 ms	10530 ms	2	7469 MB
Yolo v5	93%	203 ms	5278 ms	2	59993 MB
Convolutional Neural Network (CNN)	94%	355 ms	9230 ms	1	7882 MB

Table 4.1: Comparison Table of Algorithms Used

Chapter 5

Discussions

5.1 Dataset Differentiation

The first dataset was obtained using a Windows laptop equipped with a camera of moderate quality. As illustrated in Figure .5.1, this dataset exhibited certain challenges, including uneven lighting conditions and noticeable motion blur surrounding the hand signs. These posed significant difficulties for the model during training and subsequently resulted in an accuracy rate of less than 50%. Consequently, it became evident that further improvements in dataset quality were imperative to enhance model performance.

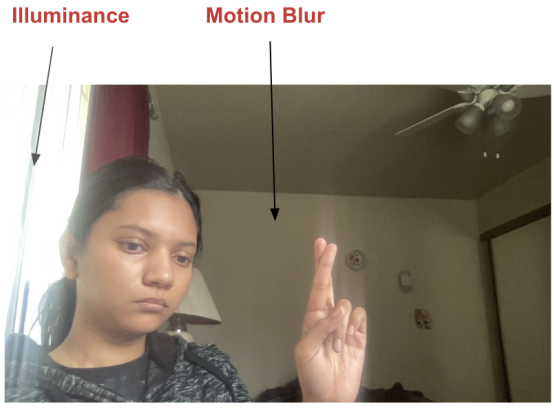


Figure 5.1: Limitation from First Dataset

To address the limitations of the initial dataset, subsequent datasets were captured to provide cleaner and more robust training data. These additional datasets aimed to mitigate issues related to lighting, motion blur, and other factors that adversely affected the model's accuracy.

The second dataset, captured once again using a Windows laptop, represented a substantial effort to enhance the quality of training data and overcome previous challenges. This iteration of dataset collection prioritized the elimination of background illuminance and the complete eradication of motion blur, striving to create an optimal training environment. Despite these improvements, the model's accuracy only saw a modest increase to 51.2%. This increment was counterbalanced by the emergence of a new challenge—background shadows, as evidenced in Figure .5.2. These shadows cast behind the

hand sign introduced a detrimental element, significantly impeding both training and testing phases and diminishing the model's proficiency in accurate sign language letter detection and prediction. These findings underscore the ongoing refinement of the dataset as an iterative process, with discernible implications for the model's performance.

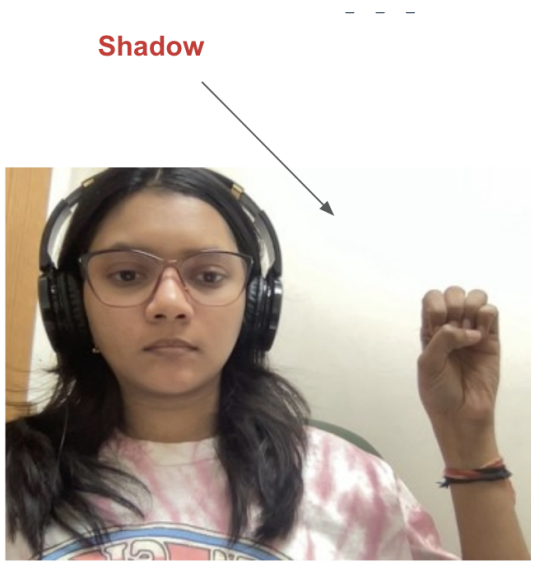


Figure 5.2: Limitation from Second Dataset

The acquisition of the third dataset maintained continuity in using the same laptop for image capture, under more favorable lighting conditions. While this dataset exhibited a noticeable improvement in image quality compared to its previous datasets, it was not without its own set of challenges, particularly related to lighting conditions. Figure 5.3 illustrates the presence of uneven lighting, with noticeable shadows on the face and an obscured hand sign. However, despite these imperfections, this dataset yielded a noteworthy increase in accuracy, reaching 70% during the training phase. It was evident that improved image capture conditions, particularly within well-lit environments, had a positive impact on model accuracy, as indicated by testing results. This observation underscores the importance of optimizing lighting conditions to further enhance accuracy in sign language recognition.

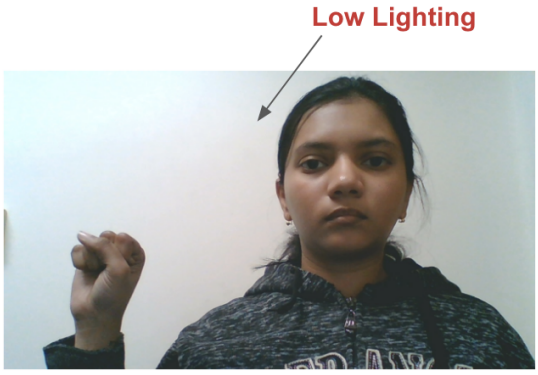


Figure 5.3: Limitation from Third Dataset

The dataset 4 was used for training and testing the model . Distinguished from its previous datasets, this dataset was captured using a MacBook, enabling the acquisition of higher-quality images. Importantly, the dataset was collected in an environment characterized by superior lighting conditions, ensuring that both the user's face and the background were clearly noticeable in each image. Figure .5.4 showcases the substantial improvement in image quality, with the previously noted low lighting condition effectively addressed. Upon retraining the model using this refined dataset, a remarkable accuracy of 95% was achieved, accompanied by an impressively low processing lag of only 2 milliseconds. Consequently, this dataset, comprising 5200 high-quality images, was selected as the primary resource for training and testing various algorithms in our study, underscoring its pivotal role in achieving superior model performance.

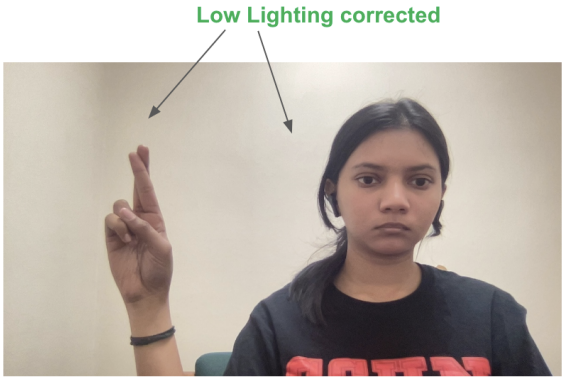


Figure 5.4: Final Dataset

Figure 5.5: Character C with extra bright light in the background

5.1.1 Light variation in background

According to the experiments In the following figure 5.5 the reason behind the presence of extra bright light in the background of an image during image processing can lead to several issues. One major problem is the underexposure of the foreground subject, which can result in poor visibility and difficulties in identifying key features. Additionally, the bright light may create unwanted reflections or glare on the subject, further decreasing image quality. Our study also revealed that overexposure can occur in extreme cases, causing blown-out highlights and loss of detail. The results demonstrate that the presence of extra bright light in the background can significantly impact the final output of an image and may require additional post-processing techniques to address these issues.

5.2 Experiment with varying sizes

The algorithm was tested for varying sizes of the input image. The image sizes were started with cropped-28X28, which was a square bounding box across the American sign language later in the image. Though this resulted in accurate classification the cropping was done manually, and hence it was not recommended process for long-term use. As a proof of concept, we could establish that the algorithm works through this image cropping. The image size selected was a standard CNN size of 224 X 224. The choice of this size was based on standard alexNet which is the most popular CNN in the image processing

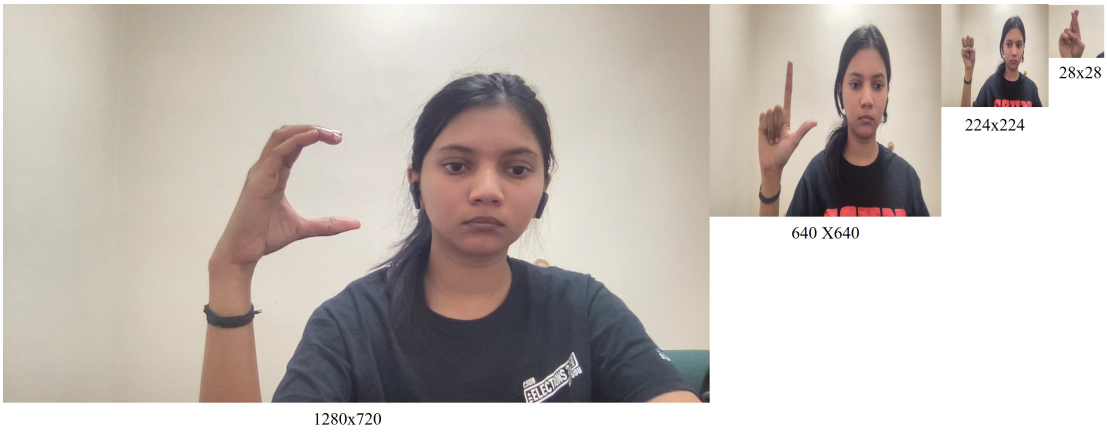


Figure 5.6: Effect of image size

domain. With this choice, we got decent accuracy of 95.38 % testing accuracy. The speed of execution during this was about 0.03 seconds. The next choice was 640 X 640 which is standardly used in you look only once (YOLO V8) algorithm. The accuracy increased slightly (roughly 1 %) while the lag time raised from 0.03 seconds to double almost 0.06 seconds. This way just 1% improvement was not sufficient to justify this lag. We also tried the unprocessed original version of image 1280 X 720. In this case, the design of CNN was very complicated as standard blocks are available in squared dimensions while the input was rectangular. Also, the compression in only one dimension made the aspect ratio from 16:9 to 1:1. We found that the change in aspect ratio does not affect accuracy. In fact, it actually affects execution time. The other aspect ratios such as 4:3 or 16:9 are slower for similar accuracy and hence from our study, it is advisable that images should be squared (1:1 aspect ratio) for maximum efficiency. Also, that reduced the burden of designing multiple networks for different kinds of input image aspect ratios varying from 16:9 (Mobile horizontal capture/webcam capture)) to 9:16 (Mobile vertical capture). Overall the result showed 224 X224 input image size provides optimal accuracy and time of execution without manually cropping. Hence through our study, we can recommend using the size of 224 X 224. Also added advantage of using lower memory of the computer makes it possible to run on regular laptops than requiring high-end computing machines.

5.3 Hardware Limitations

While training with Densenet-154 on my Windows laptop with an NVIDIA GeForce GTX 1050. Moreover, I encountered hardware limitations. The model demanded substantial processing power, resulting in a 60-minute training time and subsequent challenges with running and debugging operations. This setback caused delays in my thesis timeline.

To address this, I uploaded the generated h5 file to Google Drive for testing on a different platform. However, using a MacBook with an M1 chip posed its own challenges, as there was no available OpenCV library for the M1 chip. Eventually, I turned to Jupyter Notebook for testing, where I achieved a satisfactory 95% accuracy. Screenshots were taken to document the results.

5.4 Analyzing the Impact of Sign Language Speed on DenseNet-154 Algorithm

The experimental results for testing the developed algorithm of DenseNet-154 yielded insightful findings regarding the speed variation of sign language characters. We conducted multiple studies, focusing on character A transitioning to character B, to assess the algorithm's performance in different scenarios. In the first study, where the transition between A and B was immediate (0 to 1 second), the algorithm performed exceptionally well. No errors were encountered during this rapid transfer, and the next character was accurately detected within an average of 0.7 seconds. This result demonstrated the algorithm's capability to handle quick and smooth transitions in sign language recognition. However, in the second study, where the transition took between 2 to 3 seconds, we observed a slight decline in accuracy. At least one wrong character was detected in our sign language recognition during this moderately slow transition. This finding suggests that the algorithm may encounter challenges in correctly identifying characters during more prolonged and relatively slower sign transitions. The most crucial insights were gained from the third study, where we deliberately slowed down the transition between characters to an extremely slow pace, waiting for at least 10 seconds for the completion. In this scenario, the algorithm faced significant difficulties, resulting in the appearance of at least three wrong characters between A and B. The prolonged transition led to considerable misclassifications, indicating that the algorithm's performance significantly deteriorates when dealing with extended sign language sequences. Furthermore, we noticed a recurrent pattern in misclassifications, with the letter "m" being most frequently misclassified during all studies. Additionally, in some instances, the letter "n" was also misidentified, although to a lesser extent.

We faced all these several issues with existing sign language recognition systems that our proposed solution aimed to address. To address these issues we took following steps Firstly, the lack of labeled data limited the accuracy and robustness of the models. We incorporated techniques to enhance data augmentation and synthesis to overcome this limitation. Secondly, vision-based recognition was affected by lighting, clutter, and occlusions. Our solution incorporated preprocessing techniques to address these issues. Additionally, variations in signing styles also impacted recognition accuracy, and our solution used a

combination of models to account for these variations. Moreover, our proposed solution used low-cost and accessible hardware to make the system affordable for everyone. Lastly, we optimized the deep learning models to ensure real-time processing with minimal delays.

Chapter 6

Conclusions

In conclusion, our self-developed DenseNet-154 SLR system shows promise in bridging the communication gap between the hearing-impaired and hearing populations. With an accuracy of 95.6% and a lag of 0.03 seconds, our model presents a significant improvement over existing SLR systems. However, we have also identified challenges in recognizing ASL in different backgrounds, with a significant drop in accuracy for designer backgrounds. Our study highlights the importance of background in ASL recognition and the need for further research in this area. Our work suggests that the DenseNet architecture can be used effectively to develop accurate and efficient SLR systems. However, more research is needed to improve the accuracy of our model in diverse settings. Overall, our study presents a foundation for developing advanced and accessible DenseNet-154 SLR systems that can be uversed in various settings, including educational institutions, public places, and homes, to facilitate communication between the hearing-impaired and the hearing population.

In addition to the conclusions drawn from our study, there are several future research directions that can improve the accuracy and usability of our DenseNet-154 SLR system. One potential area of research is the development of new techniques for handling different backgrounds in ASL recognition. Our study showed a significant drop in accuracy for designer backgrounds, highlighting the need for more research in this area. Future work could explore the use of advanced preprocessing techniques and feature extraction methods to improve the model's performance on diverse backgrounds.

Another direction for future research is the exploration of alternative deep learning architectures for SLR. While our DenseNet-154 architecture shows promising results, there may be other architectures that could improve performance or offer new capabilities. For example, attention-based models and graph-based models are two architectures that have shown promise in other computer vision tasks and may be applicable to SLR.

Moreover, our study focused on recognizing isolated signs, but in real-life scenarios, signs are often produced in continuous sentences. Future work could explore the use of sequence-to-sequence models to recognize and translate ASL sentences. This would require the development of a large-scale ASL sentence dataset, which is currently lacking.

Lastly, our current model is limited to recognizing ASL gestures in a controlled envi-

ronment with plain backgrounds. Future research could focus on making the system more robust to variations in lighting conditions and user clothing, as these factors can also affect recognition accuracy.

In conclusion, our study provides a foundation for further research and development of accurate and efficient DenseNet-154 SLR systems. Future work could focus on addressing the challenges we have identified, including recognizing ASL in diverse backgrounds and developing new architectures for SLR. With continued research and development, SLR systems like ours have the potential to revolutionize communication for the hearing-impaired community and improve accessibility for all.

References

[1] J. Shin, A. Matsuoka, M. A. M. Hasan, and A. Y. Srizon, "American sign language alphabet recognition by extracting feature from hand pose estimation," *Sensors*, vol. 21, no. 17, p. 5856, 2021.

[2] A. Wadhawan and P. Kumar, "Deep learning-based sign language recognition system for static signs," *Neural computing and applications*, vol. 32, pp. 7957–7968, 2020.

[3] L. K. S. Tolentino, R. S. Juan, A. C. Thio-ac, M. A. B. Pamahoy, J. R. R. Forteza, and X. J. O. Garcia, "Static sign language recognition using deep learning," *International Journal of Machine Learning and Computing*, vol. 9, no. 6, pp. 821–827, 2019.

[4] M. M. Rahman, M. S. Islam, M. H. Rahman, R. Sassi, M. W. Rivolta, and M. Aktaruzzaman, "A new benchmark on american sign language recognition using convolutional neural network," in *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*, pp. 1–6, 2019.

[5] A. Mittal, P. Kumar, P. P. Roy, R. Balasubramanian, and B. B. Chaudhuri, "A modified lstm model for continuous sign language recognition using leap motion," *IEEE Sensors Journal*, vol. 19, no. 16, pp. 7056–7063, 2019.

[6] Y. Liao, P. Xiong, W. Min, W. Min, and J. Lu, "Dynamic sign language recognition based on video sequence with blstm-3d residual networks," *IEEE Access*, vol. 7, pp. 38044–38054, 2019.

[7] A. Patil, A. Kulkarni, H. Yesane, M. Sadani, and P. Satav, "Literature survey: sign language recognition using gesture recognition and natural language processing," *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2021, Volume 1*, pp. 197–210, 2021.

[8] S. Daniels, N. Suciati, and C. Fathichah, "Indonesian sign language recognition using yolo method," in *IOP Conference Series: Materials Science and Engineering*, vol. 1077, p. 012029, IOP Publishing, 2021.

[9] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen, "Sign language recognition using convolutional neural networks," in *Computer Vision-ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part I 13*, pp. 572–578, Springer, 2015.

[10] R. Fatmi, S. Rashad, and R. Integlia, "Comparing ann, svm, and hmm based machine learning methods for american sign language recognition using wearable motion sensors," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0290–0297, 2019.

[11] S. Katoch, V. Singh, and U. S. Tiwary, "Indian sign language recognition system using surf with svm and cnn," *Array*, vol. 14, p. 100141, 2022.

[12] X. Jiang and W. Ahmad, "Hand gesture detection based real-time american sign language letters recognition using support vector machine," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 380–385, 2019.

[13] "Sign language mnist." https://www.kaggle.com/datasets/datamunge/sign-language-mnist. (Accessed on 09/26/2023).

[14] "Labelimg · humansignal/labelimg wiki." https://github.com/HumanSignal/labelImg/wiki/LabelImg. (Accessed on 09/26/2023).

[15] "scikit-learn: machine learning in python — scikit-learn 1.3.1 documentation." https://scikit-learn.org/stable/. (Accessed on 09/26/2023).

[16] M. M. Adankon and M. Cheriet, *Support Vector Machine*, pp. 1303–1308. Boston, MA: Springer US, 2009.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] J. Brownlee, "Tour of data sampling methods for imbalanced classification - machinelearningmastery.com." https://machinelearningmastery.com/data-sampling-methods-for-imbalanced-classification/, January 2020. (Accessed on 09/26/2023).

[19] C. Burges and C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, January 1998.

[20] "Support vector machine (svm) algorithm - javatpoint."

[21] "ultralytics/yolov5: Yolov5 in pytorch ¿ onnx ¿ coreml ¿ tflite." https://github.com/ultralytics/yolov5. (Accessed on 09/27/2023).

[22] J. Solawetz, "What is yolov5? a guide for beginners.." https://blog.roboflow.com/yolov5-improvements-and-evaluation/, June 2020. (Accessed on 09/27/2023).

[23] "[2004.10934] yolov4: Optimal speed and accuracy of object detection." https://arxiv.org/abs/2004.10934. (Accessed on 10/08/2023).

[24] "Convolutional neural network: Benefits, types, and applications." https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/. (Accessed on 09/27/2023).

[25] S. Saha, "A comprehensive guide to convolutional neural networks — the eli5 way — saturn cloud blog." https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/, December 2018. (Accessed on 09/27/2023).

[26] N. Hasan, Y. Bao, A. Shawon, and Y. Huang, "Densenet convolutional neural networks application for predicting covid-19 using ct image," *SN computer science*, vol. 2, no. 5, p. 389, 2021.

[27] "Densenet — pytorch." https://pytorch.org/hub/pytorch$_{v}ision_{d}ensenet/.(Accessedon09/27/2023)$.