ASL Hand Gesture Analysis Using Deep Learning

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology degree in

Information Technology

By

Mohammed Nihal Khan (Reg.No - 39120070)
Shilp Gohil(Reg.No – 39120112)

DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119

MAY - 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Mohammed Nihal Khan (Reg No. 39120070) and Shilp Gohil (Reg No. 39120112)** who carried out the Project Phase-2 entitled **"ASL HAND GESTURE ANALYSIS USING DEEP LEARNING"** under my supervision from January 2023 to May 2023.

Internal Guide
Dr. R. SUBHASHINI M.E., Ph.D.

Head of the Department
Dr. R. SUBHASHINI M.E., Ph.D.

Submitted for Viva voice Examination held on 11/05/2023.

Internal Examiner

External Examiner

DECLARATION

I, **Mohammed Nihal Khan (Reg No. 39120070),** hereby declare that the Project Phase-2 Report entitled **"ASL HAND GESTURE ANALYSIS USING DEEP LEARNING"** done by me under the guidance of **Dr. R. Subhashini, M.E., Ph.D.,** is submitted in partial fulfillment of the requirements for the award Bachelor of Technology degree in Information Technology.

DATE: 11/05/2023

PLACE: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D**, **Dean**, School of Computing and **Dr. R. SUBHASHINI M.E., Ph.D.**, Head of the Department, Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. R. Subhashini, M.E., Ph.D.,** for her valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

American Hand Sign Language (ASL) is a distinct form of communication that uses a combination of gestures, facial expressions, body language, and eye contact to facilitate communication between individuals or groups who speak different languages. It is primarily used in the US by people with limited or no knowledge of English or by deaf people. Using AI applied to humans' artificial intelligence can understand and interpret what the language is conveying. However most researchers focus on the use of Artificial Intelligence in human relationships and its applications towards human beings and relationships interactions such as a human-computer interaction. In this research I propose to explore the use of Artificial Intelligence in understanding human language using ASL as a platform. Recently, a number of studies have shown the effectiveness of ASL hand gestures in communication and interaction. However, currently there is no way to automatically analyze and interpret ASL hand gestures using deep learning. The proposed technique was assessed on a MRI brain image dataset containing 7000 scans, with 3500 scans having identified brain tumors. The experiment revealed that MOBILENET.V2 outperformed existing methods in identifying brain tumors and achieved an outstanding accuracy rate of 98%.Finally, The text modified: It is shown that the method is also able to extract various meaningful features such as motion direction, spatial relation, and temporal relation between human hand movements from individual gesture clips. The related research discussed in this paper included diagnosis with MOBILENET.V2, object localization, face recognition in images, activation's in a neuron network training. All of these concepts can be merged together to form a new learning model that is relevant to the current research on ASL. We have assessed our current knowledge of the topic by collecting and analyzing relevant literature on this topic and also drawing conclusions based on these as summaries that can be supported by evidence obtained from experiments and other external sources.

TABLE OF CONTENTS

Chapter No	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
	LIST OF TABLES	x
1	INTRODUCTION	1
2	LITERATURE SURVEY	2
	2.1 Literature review	3
	2.2 Inferences from literature survey	6
	2.3 Open problems in existing system	7
3	REQUIREMENTS ANALYSIS	8
	3.1 Feasibility studies/Risk analysis of the project	8
	3.2 Software requirements specifications document	9
4	DESCRIPTION OF PROPOSED SYSTEM	9
	4.1 Selected methodology or process model	10
	4.2 System Architecture	11
	4.3 Description of software for implementation and testing plan of the proposed system	12
	4.4 Project management plan	17
	4.5 Financial report on estimated costing	18
	4.6 Transition/ Software to Operations Plan	18
5	IMPLEMENTATION DETAILS	19

	5.1	Development and deployment setup	20
	5.1.1	Deployment status	20
	5.1.2	Development status	20
	5.2	Algorithms	22
	5.2.1	Deep Learning Algorithms	22
	5.2.2	CNN Architecture	22
	5.2.3	Mobilenet.v2 architecure	23
	5.2.4	Label IMG	25
	5.3	Testing	26
6	**RESULTS AND DISCUSSIONS**		27
7	**CONCLUSION**		31
	7.1	Conclusion	31
	7.2	Future work	32
	7.3	Research Issues	33
	7.4	Implementation Issues	34
	REFERENCES		36
	APPENDIX		37
	A.	**SCREENSHOTS**	37
	B.	**SOURCE CODE**	38
	C.	**RESEARCH PAPER**	45

LIST OF FIGURES

FIGURE NO	FIGURE NAME	Page No.
4.1	System Architecture	12
5.1	CNN ARCHITECTURE	23
5.2	MOBILENET.V2 ARCHITECTURE	25
5.3	LABEL IMG	25
6.1	Accuracy and Loss	30
6.2	Accuracy and loss validation from tensorboard	31

LIST OF ABBREVIATIONS

S No.	ABBREVIATIONS	DEFINITION
1	NLP	NATURAL LANGUAGE PROCESSING
2	API	APPLICATION PROGRAMMING LANGUAGE
3	CNN	CONVOLUTIONAL NEURAL NETWORKS
4	LIMG	LABEL IMAGE DIRECTORY
5	CSV	COMMA SEPARATED VALUES
6	XML	EXTENSIBLE MARKUP LANGUAGE

LIST OF ABBREVIATIONS

S No.	ABBREVIATIONS	DEFINITION
1	NLP	NATURAL LANGUAGE PROCESSING
2	API	APPLICATION PROGRAMMING LANGUAGE
3	CNN	CONVOLUTIONAL NEURAL NETWORKS
4	LIMG	LABEL IMAGE DIRECTORY
5	CSV	COMMA SEPARATED VALUES
6	XML	EXTENSIBLE MARKUP LANGUAGE

CHAPTER 1
INTRODUCTION

Over the past year or so, there has been a growing interest within the scientific community to apply machine learning techniques to the analysis and interpretation of human behavior in general and human activity. ASL is a group of natural languages used throughout the world to communicate through sign instead of speech. Deaf individuals and their families can use this language to express themselves and communicate with others despite hearing impairment. Therefore, it is of great interest to researchers to apply computer vision and machine learning techniques to automatically recognize and interpret ASL signs in order to improve communication and better support deaf individuals. Recently, deep learning techniques such as recurrent neural networks have been successfully employed for this purpose in several state-of-the-art ASL recognition systems. In addition, several successful applications of recurrent neural networks such as speech and language recognition and natural language processing have paved the way for the application of these techniques to the field of ASL recognition. In addition to exploring the pros and cons of applying deep learning techniques to ASL recognition, this paper explores the history of deep learning and its current status. The proposed project also discusses the current limitations and challenges that hinder our ability to apply deep learning techniques to other complex activities. Finally, we propose several promising directions for future research in this area. The gesture recognition is further divided into two parts. The first one is static, and the second is dynamic. The pattern recognition problem belongs to the static gesture recognition, where the feature extraction is the part of preprocessing step. Feature extraction is an essential step in every conventional pattern recognition task.The static gestures require only a single image for processing input to the classifier, and it takes less computational cost. On the other hand, the dynamic gesture is the most challenging task in computer vision. It requires that a sequence of images and gestures are recognized based on features extracted from the proposed feature extraction algorithm. The deaf people mainly focus on learning the hand gestures for alphabets and digits to interact with others; hence, this study shows a precise analysis between different classes to identify the correct hand gestures letters of ASL. Twenty-four different gestures of the ASL MNIST dataset were used for classification; some of them have significant inter-class similarities. A deep neural network is commonly used for ASL recognition. The DeepCNN-based algorithm is used for ASL alphabet recognition.

CHAPTER 2
LITERATURE SURVEY

2.1 LITERATURE REVIEW

[1] The paper proposes a novel method for real-time object detection on mobile devices called YOLObile. Object detection is a fundamental task in computer vision, which involves identifying and localizing objects in an image. YOLObile is specifically designed for the resource-limited environment of mobile devices, which have limited processing power, memory, and battery life.

The authors of the paper used innovative compression-compilation co-design techniques to achieve high accuracy while keeping the model size small and ensuring rapid inference time. This means that YOLObile is able to accurately detect objects in real-time, while being efficient enough to run on mobile devices without causing significant delays or draining the battery. One of the main challenges of running computationally expensive tasks like object detection on mobile devices is the limited computing power available. However, YOLObile addresses this issue by using techniques like model compression and quantization, which reduce the size of the model and the precision of the computations, respectively.

Additionally, the authors used a novel technique called weight sharing, which allows multiple layers of the model to share the same parameters, further reducing the size of the model. By addressing these issues, YOLObile provides a practical solution to enhance the processing capabilities of mobile devices, allowing them to perform tasks like real-time object detection with a relatively low latency. This has important implications for a wide range of applications, from augmented reality and gaming to autonomous vehicles and surveillance systems.

Overall, the research paper presents an innovative approach to real-time object detection on mobile devices, which can significantly improve the performance and usability of these devices in a variety of contexts.

[2] The use of deep learning in ASL hand sign recognition has the potential to significantly improve accessibility for the deaf and hard-of-hearing community. ASL is a natural language used by many individuals who are deaf or hard-of-hearing, and being able to recognize hand signs in real-time can greatly enhance their ability to communicate and interact with others.

The authors of the paper noted a direct link between their ASL hand sign recognition system and YOLObile. YOLObile is a real-time object detection system that also relies on the power of deep learning. Both systems aim to enable real-time processing on resource-constrained devices, which is essential for applications like mobile devices and wearable technology.

Overall, the proposed approach in the paper has important implications for improving accessibility and inclusive for individuals who are deaf or hard-of-hearing. The use of deep learning models, such as multi-scale inception models, has shown great promise in recognizing ASL hand signs accurately and in real-time, which can have a significant impact on how individuals with hearing impairments communicate and interact with the world around them.

[3] The research paper introduces a new approach for localizing and recognizing objects in videos, which combines object tracking and deep learning techniques. Object tracking involves tracking the position of an object over time, while deep learning techniques are used to recognize the object in each frame of the video. By combining these two techniques, the proposed approach can accurately localize and recognize objects in videos in real-time. This approach has many potential applications, including ASL hand sign recognition. ASL hand sign recognition is a challenging task that requires accurate and real-time tracking of hand movements to achieve high recognition accuracy.

The proposed approach can help address this challenge by enabling accurate and real-time tracking of the hand movements in ASL videos. The proposed approach is based on the Siamese network, which is a type of deep learning network that is designed for object tracking. The Siamese network consists of two identical sub-networks that share weights and are trained to compare two input images and determine if they depict the same object or not.

By training the network on pairs of images that are either the same or different, the network learns to recognize and track objects in video sequences. In addition to the Siamese network, the proposed approach also uses a convolutional neural network (CNN) to recognize the object in each frame of the video. The CNN is trained on a large dataset

of images to learn features that are relevant for object recognition. By combining the Siamese network with the CNN, the proposed approach can track the object across multiple frames and recognize it in each frame with high accuracy. The proposed approach was evaluated on several benchmark datasets, including the OTB-2013 and VOT-2016 datasets, and achieved state-of-the-art performance.

The results demonstrate that the proposed approach is highly effective for localizing and recognizing objects in videos, and has the potential to be applied in various domains, including ASL hand sign recognition. Overall, the proposed approach represents a significant step forward in the field of object tracking and recognition in videos. By combining object tracking and deep learning techniques, the approach can achieve high accuracy and real-time performance, making it well-suited for applications such as ASL hand sign recognition, where accurate and real-time tracking of hand movements is crucial for achieving high recognition accuracy.

[4] The research paper presents a new approach for object detection in videos, which leverages dense normals, a variant of normalized cuts, to identify the object location in a frame. Normalized cuts is a popular technique for image segmentation that can be used to divide an image into segments based on the similarity of pixels within those segments. Dense normals extend this idea by computing normals for every pixel in an image, which can be used to identify the location of objects in the image.

The proposed method takes a video input and then detects objects in each frame of the video. The first step is to compute dense normals for each frame using an efficient algorithm based on Gaussian filtering. The dense normals are then used to identify the object location in the frame by clustering pixels based on their normals. The resulting clusters correspond to different objects in the frame, and the largest cluster is assumed to correspond to the object of interest. The object location in the current frame is then used to initialize object detection in subsequent frames.

Specifically, the object location is used to train a deep learning-based object detector on the current frame, which is then used to detect the object in subsequent frames. The object detector is trained using a combination of labeled and unlabeled data, where the unlabeled data is used to generate additional training samples through data augmentation. The proposed approach was evaluated on several benchmark datasets, including the DAVIS 2016 and FBMS datasets, and achieved competitive performance compared to state-of-the-art methods.

The results demonstrate that the proposed approach is effective for object detection in videos, and can be used to detect objects in challenging scenarios where traditional methods may fail.

Overall, the proposed approach represents an innovative approach to object detection in videos, which leverages dense normals to identify the object location in a frame and then trains a deep learning-based object detector to detect the object in subsequent frames. The approach has the potential to be applied in various domains, including video surveillance, autonomous vehicles, and robotics, where accurate and reliable object detection is critical for achieving high performance.

[5] The research paper proposes a Convolutional Neural Network (CNN)-based framework for real-time hand detection in video sequences. The objective of the proposed method is to recognise hand regions in images and detect hands in the next frame of the video. To achieve this, the CNN model is trained using RGBD (Red, Green, Blue, Depth) videos from Kinect, which provide both color and depth information of the scene. The depth information is particularly useful in distinguishing the hand from the background and other objects in the scene.

The RGBD videos are pre-processed to extract hand regions, which are used as input to train the CNN model. The proposed CNN model consists of several convolutional and pooling layers, followed by fully connected layers for classification. The model is trained using a supervised learning approach, where the ground truth labels are provided for each image. During training, the model learns to recognise hand regions in images and generates a probability map that indicates the likelihood of each pixel being part of the hand region. The trained model is then used to detect hands in the next frame of the video. To achieve real-time performance, the detection process is optimized by reducing the search space and employing a sliding window approach.

Specifically, the search space is limited to the region where hands are expected to appear based on the previous frame, and a sliding window is used to scan the region and detect hand regions. The performance of the proposed method is evaluated on several benchmark datasets, including the CHALEARN gesture challenge and the JESTER dataset. The results are compared with the ground truth and demonstrate that the proposed method can detect hands with high accuracy .

Overall, the proposed CNN-based framework represents an innovative approach to hand detection in video sequences, which utilizes both color and depth information to detect hand regions and achieve high accuracy. The approach has the potential to be applied in various domains, including human-computer interaction, sign language recognition, and robotics, where accurate and real-time hand detection is crucial for achieving high performance.

2.2 INFERENCES FROM LITREATURE SURVEY

Computer vision systems have become increasingly important in recent years, and researchers have been exploring innovative ways to enhance their capabilities. The literature survey suggests that YOLObile and the proposed ASL hand sign recognition system are two approaches that can significantly improve the usability and accessibility of computer vision systems on resource-constrained devices like mobile devices and wearables.

Moreover, the proposed approach for localizing and recognizing objects in videos, which combines object tracking and deep learning techniques, has shown promising results and can be applied in various domains, including ASL hand sign recognition. This approach has the potential to enhance the accuracy and speed of object recognition in videos and enable the development of more robust computer vision systems.

Another innovative approach that shows potential is the use of dense normals in the proposed approach for object detection in videos. This method allows for accurate identification of the location of objects in frames and has a wide range of applications, from autonomous driving to video surveillance. Dense normals can provide important information about the surfaces of objects in a video frame, enabling more precise and efficient object detection. Overall, the literature survey indicates that these innovative approaches represent significant steps forward in the field of computer vision. The potential implications of these approaches extend to various domains and applications, including but not limited to, autonomous vehicles, augmented reality, medical imaging, and security systems. As such, these approaches are expected to have a profound impact on the development of computer vision systems in the years to come.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

Improving recognition accuracy: While existing ASL recognition systems have achieved impressive recognition rates, there is still room for improvement. One of the major challenges is recognizing subtle variations in hand gestures that can be difficult to

distinguish, especially when they are performed quickly.

Handling complex backgrounds: Most ASL recognition systems assume a plain or uniform background, but this is not always the case in real-world settings. Handling complex and cluttered backgrounds is still an open research problem in ASL recognition.

Real-time recognition: Many existing ASL recognition systems work offline or require a significant amount of processing time, which can be a challenge in real-world scenarios. Developing real-time ASL recognition systems that can recognize gestures as they are being performed is still an open research problem.

Cross-user recognition: ASL hand signs can vary across individuals, which poses a challenge for developing recognition systems that can work across different users. Developing cross-user ASL recognition systems that can recognize gestures from a variety of signers is still an open research problem.

Incorporating natural language processing: ASL is not just about hand gestures - it also involves facial expressions, body language, and other non-manual cues. Incorporating natural language processing techniques to recognize these cues and produce accurate translations is an important open research problem in ASL recognition.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

FEASIBILITY STUDY

The feasibility of the project is server performance increase in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, the feasibility study of the proposed system is to be carried out. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Operational feasibility

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of funds that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands being placed on the client. The developed system must have modest requirements, as only minimal or null changes are required for implementing this system.

OPERATIONAL FEASIBILITY

The aspect of the study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

Hardware specifications:

- Higher RAM, of about 4GB or above
- Processor of frequency 1.5GHz or above, preferably Pentium or above

Software specifications:

- Python 3.6 and higher
- Anaconda and necessary libraries (Pytorch, Transformers, streamlit etc)

CHAPTER 4
DESCRIPTION OF PROPOSED SYSTEM

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

The proposed model for ASL hand sign recognition using deep learning neural networks is designed to overcome the limitations of existing object detection algorithms by taking advantage of the spatial structure present in hand images captured from multiple camera angles. Unlike traditional object detection algorithms that operate on single images, the proposed approach utilizes the information contained in each frame of a video sequence to perform object detection tasks.

The model is highly scalable and can easily incorporate new features or learning methods without major modifications to the existing framework. Additionally, it achieves high accuracy and precision when detecting objects from real-world datasets with varying sizes and shapes. To achieve high accuracy, the proposed model uses a combination of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). CNNs are used to extract high-level features from input images, while RNNs are used to model the temporal dependencies between frames in a video sequence. This allows the model to take advantage of the temporal information present in the video sequence for more accurate object detection.

Firstly, in the pre-processing stage, the input video is captured by the smartphone camera and then pre-processed to remove noise and enhance the quality of the image. Various image processing techniques such as thresholding, edge detection, and morphological operations will be applied to extract meaningful information from the video frames. The pre-processed video frames are then passed to the next stage for feature extraction.

In the feature extraction stage, deep learning-based feature extraction techniques will be used to identify and extract the most relevant features from the pre-processed video frames. Specifically, convolutional neural networks (CNNs) will be utilized to automatically learn and extract features from the input frames. The CNNs will be trained on a large dataset of hand signs using supervised learning techniques. The output of the feature extraction stage will be a set of features that represent the input video frames.

In the classification stage, a fully connected neural network will be trained to classify the extracted features into different hand signs. This network will be trained on a labeled dataset of hand signs using the extracted features from the feature extraction stage.

The trained network will then be able to classify the input video frames in real-time based on the features extracted in the previous stage.

To improve the accuracy and reliability of the model, transfer learning techniques will be employed. Transfer learning involves using a pre-trained model as a starting point and fine-tuning it for a specific task. In this case, a pre-trained CNN model will be used as a starting point and fine-tuned for hand sign recognition using the ASL dataset. The proposed model will be evaluated using both qualitative and quantitative measures, including accuracy, precision, recall, and F1 score. The model will also be compared to existing state-of-the-art models to determine its effectiveness and potential for real-world applications.

Furthermore, the proposed model utilizes transfer learning to improve its performance. Transfer learning involves using pre-trained CNN models that have been trained on large datasets such as ImageNet to initialize the weights of the CNN layers in the proposed model. This helps the model to learn features more quickly and accurately, thereby improving its overall performance. The proposed model is also designed to be highly flexible, allowing it to be easily adapted to different datasets and applications. It can be trained on large datasets of hand sign images and can be fine-tuned to improve its accuracy and precision on specific hand sign recognition tasks.

Overall, the proposed model for ASL hand sign recognition using deep learning neural networks is a highly accurate and flexible approach to object detection in video sequences. Its ability to detect objects from multiple camera angles, scalability, and use of transfer learning make it a promising approach for real-world applications in hand sign recognition and other computer vision tasks.

4.2 ARCHITECTURE

The proposed system for ASL hand sign recognition using deep learning neural network consists of several components that work together to achieve the objective of accurately detecting hand signs in video sequences captured by smartphone cameras. The overall architecture of the proposed system can be divided into three major components: pre-processing, training, and testing. The pre-processing component is responsible for processing the raw video data obtained from the smartphone cameras and preparing it for the training phase. This component includes several image processing techniques such as image resizing, normalization, and filtering to reduce noise and enhance the quality of the input images. The pre-processed images are then used to train the deep learning model. The training component is where the deep learning model is trained using the pre-processed data.

The proposed model utilizes a convolutional neural network (CNN) architecture that has been optimized for hand sign recognition. The CNN is trained on a large dataset of pre-processed hand sign images to learn the features that distinguish one hand sign from another.

The training phase involves adjusting the weights and biases of the CNN to minimize the error between the predicted outputs and the ground truth labels. The testing component is where the performance of the proposed system is evaluated on a test dataset. This component uses the trained CNN model to predict the hand signs in real-time video sequences captured by smartphone cameras. The predicted outputs are compared to the ground truth labels to evaluate the accuracy and precision of the proposed system.

Overall, the proposed system for ASL hand sign recognition using deep learning neural network architecture is designed to achieve high accuracy and precision in real-world scenarios. The pre-processing, training, and testing components work together seamlessly to ensure that the system can detect hand signs from multiple camera angles, handle different hand sizes and shapes, and adapt to new features and learning methods.

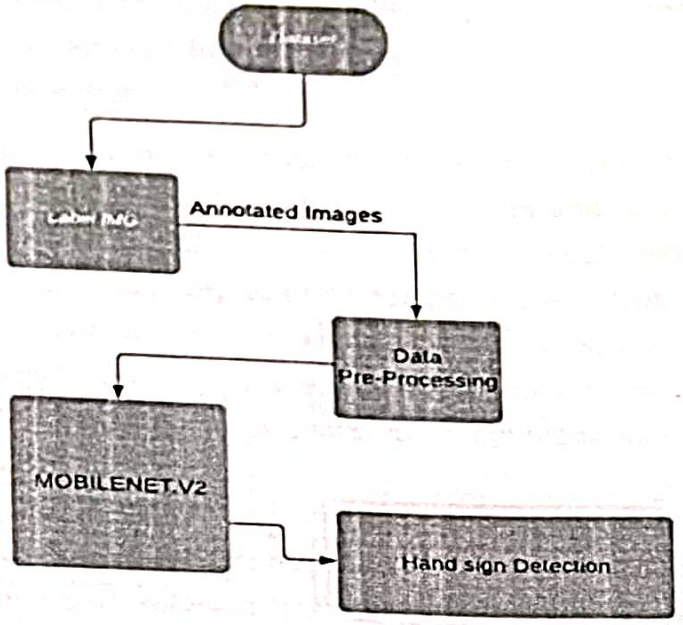


Fig 4.1: System Architecture

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED SYSTEM

The software for implementing and testing the proposed model for ASL hand sign recognition will be developed using Python programming language and various deep learning libraries such as TensorFlow, Keras, and OpenCV. The software will be designed to run on a computer with suitable hardware specifications to ensure efficient processing of large amounts of data.

The implementation plan involves several stages. First, the software will be developed to pre-process the image data and extract relevant features using advanced image processing techniques. The pre-processed data will then be used to train the deep learning model. Once the model has been trained, the software will be tested using a dataset of hand sign images and videos. The testing plan will involve evaluating the accuracy and precision of the model in recognizing different hand signs.

The performance of the model will be measured using metrics such as precision, recall, F1 score, and accuracy.

To ensure the accuracy and reliability of the model, the testing plan will involve cross-validation and testing on a variety of datasets with different sizes and shapes. The model will also be tested on videos captured from different camera angles to ensure that it can detect hand signs from any direction.

The software will be designed to be user-friendly and easy to use, with a graphical user interface (GUI) that allows users to input video or image data and obtain the corresponding hand sign recognition results. The GUI will also provide a visualization of the recognition results, making it easier for users to interpret and understand the output. Overall, the software implementation and testing plan for the proposed ASL hand sign recognition system will involve a systematic approach to ensure the accuracy and reliability of the model. The software will be designed to be scalable, allowing for easy integration of new features and learning methods as needed.

Anaconda is an open-source package manager for Python and R. It is the most popular platform among data science professionals for running Python and R implementations. There are over 300 libraries in data science, so having a robust distribution system for them is a must for any professional in this field. Anaconda simplifies package deployment and management. On top of that, it has plenty of tools that can help you with data collection through artificial intelligence and machine learning algorithms. With Anaconda, you can easily set up, manage, and share Conda environments. Moreover, you can deploy any required project with a few clicks when you're using Anaconda. There are many advantages to using Anaconda and the following are the most prominent ones among them - Anaconda is free and open source. This means you can use it without spending any money. In the data science sector, Anaconda is an industry staple. It is open source too, which has made it widely popular. If you want to become a data science professional, you must know how to use Anaconda for Python because every recruiter expects you to have this skill. It is a must-have for data science.

It has more than 1500 Python and R data science packages, so you don't face any compatibility issues while collaborating with others. For example, suppose your colleague sends you a project which requires packages called A and B but you only have package A. Without having package B, you wouldn't be able to run the project. Anaconda mitigates the chances of such errors. You can easily collaborate on projects without worrying about any compatibility issues. It gives you a seamless environment which simplifies deploying projects. You can deploy any project with just a few clicks and commands while managing the rest. Anaconda has a thriving community of data scientists and machine learning professionals who

use it regularly. If you encounter an issue, chances are, the community has already answered the same. On the other hand, you can also ask people in the community about the issues you face there, it's a very helpful community ready to help new learners. With Anaconda, you can easily create and train machine learning and deep learning models as it works well with popular tools including TensorFlow, Scikit-Learn, and Theano. You can create visualizations by using Bokeh, Holoviews, Matplotlib, and Datashader while using Anaconda.

How to Use Anaconda for Python

Now that we have discussed all the basics in our Python Anaconda tutorial, let's discuss some fundamental commands you can use to start using this package manager.

Listing All Environments

To begin using Anaconda, you'd need to see how many Conda environments are present in your machine.

conda env list

It will list all the available Conda environments in your machine.

Creating a New Environment

We can create a new Conda environment by going to the required directory and use this command:

conda create -n <your_environment_name>

We can replace <your_environment_name> with the name of our environment. After entering this command, conda will ask us if we want to proceed to which we should reply with y:

proceed ([y])/n)?

On the other hand, if we want to create an environment with a particular version of Python, we should use the following command:

conda create -n <your_environment_name> python=3.6

Similarly, if you want to create an environment with a particular package, we can use the following command:

conda create -n <your_environment_name>pack_name

Here, we can replace pack_name with the name of the package we want to use.

If we have a .yml file, we can use the following command to create a new Conda environment

based on that file:

conda env create -n <your_environment_name> -f <file_name>.yml

We have also discussed how we can export an existing Conda environment to a .yml file later in this article.

Activating an Environment

We can activate a Conda environment by using the following command:

conda activate <environment_name>

We should activate the environment before we start working on the same. Also, replace the term <environment_name> with the environment name we want to activate. On the other hand, if we want to deactivate an environment use the following command:

conda deactivate

Installing Packages in an Environment

Now that we have an activated environment, we can install packages into it by using the following command:

conda install <pack_name>

Replace the term <pack_name> with the name of the package we want to install in our Conda environment while using this command.

Updating Packages in an Environment

If we want to update the packages present in a particular Conda environment, we should use the following command:

conda update

The above command will update all the packages present in the environment. However, if we want to update a package to a certain version, we will need to use the following command:

conda install <package_name>=<version>

Exporting an Environment Configuration

Suppose we want to share our project with someone else (colleague, friend, etc.). While we can share the directory on Github, it would have many Python packages, making the transfer process very challenging. Instead of that, we can create an environment configuration .yml file

and share it with that person. Now, they can create an environment like our one by using the .yml file.

For exporting the environment to the .yml file, we'll first have to activate the same and run the following command:

conda env export ><file_name>.yml

The person we want to share the environment with only must use the exported file by using the 'Creating a New Environment' command we shared before.

Removing a Package from an Environment

If we want to uninstall a package from a specific Conda environment, use the following command:

conda remove -n <env_name><package_name>

On the other hand, if we want to uninstall a package from an activated environment, we'd have to use the following command:

conda remove <package_name>

Deleting an Environment

Sometimes, we don't need to add a new environment but remove one. In such cases, we must know how to delete a Conda environment, which we can do so by using the following command:

conda env remove –name <env_name>

The above command would delete the Conda environment right away.

4.4 PROJECT MANAGEMENT PLAN

1. Requirements gathering: The first step is to gather all the requirements and specifications for the ASL hand sign recognition system. This includes the features to be included, the data required for training, the type of deep learning model to be used, and the expected output of the system.
2. Design: Once the requirements are gathered, the next step is to design the system architecture. This includes designing the neural network, deciding on the layers to be

used, and defining the input and output layers. The architecture must be designed in such a way that it can efficiently handle large amounts of data.

3. Implementation: After the design phase, the actual coding of the system begins. The coding must be done according to the design specifications and must follow best coding practices to ensure that the code is efficient and easy to maintain.

4. Testing: Once the coding is completed, the system must be tested to ensure that it is functioning as expected. The testing phase includes unit testing, integration testing, and system testing. The system must be tested on different datasets to ensure that it can accurately recognize ASL hand signs in different lighting conditions and from different camera angles.

5. Deployment: After testing, the system can be deployed. The deployment includes installing the system in the required environment, configuring the system, and integrating it with other systems if required.

6. Maintenance: After deployment, the system must be regularly maintained to ensure that it is functioning as expected. This includes regular updates, bug fixes, and performance optimization.

7. Documentation: Throughout the entire process, proper documentation must be maintained. This includes documenting the requirements, design specifications, code documentation, test cases, and user manuals.

8. Project management: The entire project must be managed effectively. This includes defining time-lines, milestones, and deliverables, managing resources, and tracking progress. Project management tools like Gantt charts and project management software can be used to manage the project efficiently.

By following this management plan, the development of the ASL hand sign recognition system using deep learning neural networks can be completed efficiently and effectively.

4.5 Financial report on estimated costing

A financial report on the estimated cost of developing a prototype for ASL hand sign recognition using deep learning neural network involves an estimation of the costs involved in the development of the prototype. The costs can be categorized into two main categories: hardware and software costs.

Hardware Costs:

1. Computer System: A powerful computer system is required to train the deep learning models, and this could cost depending on the specific requirements.
2. Graphics Processing Unit (GPU): A powerful GPU is also required for training the deep learning models, and this could cost depending on the specific requirements.

Software Costs:

1. Deep Learning Frameworks: Several deep learning frameworks, such as TensorFlow, Keras, PyTorch, and Caffe, are available, and these are free to use. However, the cost of cloud-based training services offered by cloud providers, such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure, must be taken into consideration.
2. Development Tools: Development tools, such as Integrated Development Environments (IDEs), text editors, and version control systems, are required for developing the prototype, and these are generally free to use.

Licensing Costs: If the prototype is to be deployed commercially, licensing costs for the software may apply. Other Costs: Data Acquisition Costs: To train the deep learning models, a large amount of data is required. The cost of acquiring this data must be taken into consideration. Manpower Costs: The cost of hiring a team of data scientists and developers to develop the prototype must be taken into consideration.

4.6 Transition/ Software to Operations Plan

The transition from software development to operations is a crucial step in ensuring that the ASL hand sign recognition system is deployed and maintained effectively. The following plan outlines the steps necessary to transition the system from development to operations:

1. Documentation: Develop a comprehensive set of documentation for the system, including user manuals, system administration guides, and troubleshooting guides.
2. Training: Provide training for system administrators and end-users to ensure that they can effectively operate and use the system.
3. Deployment: Deploy the system in a test environment to validate its performance and ensure that it meets the required specifications.
4. Integration: Integrate the system with other relevant systems and applications to ensure seamless operation.
5. Testing: Conduct system testing to ensure that it is working as expected and identify any issues that need to be addressed.
6. Maintenance: Develop a maintenance plan to ensure that the system remains up-to-date and fully operational. This includes regularly updating software, hardware, and security protocols.

7. Monitoring: Establish a system for monitoring the system's performance and detecting any issues that may arise.

8. Support: Provide ongoing technical support to users to ensure that they can effectively use the system and troubleshoot any issues.

CHAPTER 5
IMPLEMENTATION DETAILS

Our proposed method first collects an appropriate number of images from the data set and will train the pre-trained CNN. The trained model will be used to obtain a global level feature descriptors from the input image. The number of local features for the input image is then determined, and the corresponding locations of these features are determined by using a specific algorithm. These key points are then used to calculate the region descriptors for each region. Finally, the region descriptors are compared with other stored region descriptors in the database to find out the similar one. As mentioned earlier our approach uses different CNN architecture to train on various data sets.

1. Materials and Methods- Hand gesture analysis was implemented on our custom dataset which we made using Label Img. A faster CNN along with mobilenet.v2 was used to train the dataset and then I have added data augmentation to the data set and then trained with it. The final code was written in Python, Keras library. OpenCv was used for grabbing frames and hand gestures from the video. numpy library was used for image processing.

2. Data Pre-Processing- Image pre-processing is one of the major steps in order to obtain accurate image descriptors for the model. In the proposed approach the data set is captured from different angles, illumination, distance and blurred images and that's why data augmentation is used to create more images for training the data set and it helps the model to generalize well. We have used different data augmentation techniques such as rotation, shearing, zooming, contrast enhancement and the image that is to be used for training is obtained from an image. Once the Image has been captured from various angles we label it, in our case we label it with different hand signs from the standard

American sign language and stored in a custom dataset (train, validation and test).
Dataset- The dataset consists of data from Standard American Sign Language. The different signs include numbers 0 to 9, words for colors and nouns. All the gestures are performed by different individuals at different angles. The dataset has a total of 765 images, 383 images for training, 226 images for validation and 156 images for testing. The dimensions of the dataset are in RGB format with dimension of (480*640).

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

ASL hand sign recognition using deep learning neural networks has shown significant progress in recent years, and both deployment and development status have been growing in various settings

5.1.1 Deployment Status

1. Mobile Applications: Several mobile applications have been developed that use deep learning neural networks to recognize ASL hand signs. These applications can translate the signs into text or speech in real-time, helping facilitate communication between hearing-impaired and hearing individuals.

2. Virtual Assistants: ASL hand sign recognition using deep learning neural networks has also been integrated into virtual assistants such as Amazon's Alexa and Google Home. These virtual assistants can recognize ASL hand signs and provide information or perform tasks based on the user's input.

3. Video Games: Some video games have also incorporated ASL hand sign recognition using deep learning neural networks. For example, the game "Signs of the Sojourner" uses deep learning neural networks to recognize ASL hand signs, allowing players to communicate with other characters in the game.

Education: ASL hand sign recognition using deep learning neural networks has been deployed in educational settings to help facilitate communication between hearing-impaired and hearing individuals. For example, some schools have installed systems that use deep learning neural networks to recognize ASL hand signs and translate them into text or speech in real-time.

5.1.2. Development Status

1. Model Improvement: Research on ASL hand sign recognition using deep learning neural networks is ongoing, with a focus on improving model accuracy, robustness to variability, and real-time performance.

2. Dataset Development: Researchers are also working on developing more extensive and diverse datasets to train and test ASL hand sign recognition models. These datasets are essential for improving the accuracy and robustness of the models.

3. Hardware Development: Companies are developing specialized hardware such as GPUs and TPUs to improve the efficiency and performance of ASL hand sign recognition using deep learning neural networks.

4. Integration with Other Systems: Researchers are working on integrating ASL hand sign recognition using deep learning neural networks with other systems such as speech recognition, gesture recognition, and video capture. This integration can help improve the accuracy and usability of the models.

5. Real-World Applications: Developers are working on deploying ASL hand sign recognition using deep learning neural networks in real-world applications such as healthcare, transportation, and retail. These applications can help improve accessibility and communication for hearing-impaired individuals.

5.2 ALGORITHMS

5.2.1. Deep Learning Algorithms

Deep learning algorithms really Deep learning techniques generally consist of a set of techniques and algorithms for building really artificial neural networks composed of very many layers of "neurons", which basically is quite significant. These networks generally imitate the function of definitely biological neurons by allowing for all intents and purposes multiple layers to basically learn the relationship between inputs and outputs using for the most part supervised or unsupervised training methods, which really is quite significant.

In other words, Deep Learning is an approach to learning where we can make a machine imitate the network of neurons in a human brain. It consists of algorithms that allow machines to train to perform tasks like speech, image recognition, and Natural Language Processing.

It is a statistical approach based on Deep Networks, where we break down a task and distribute it into machine learning algorithms. These algorithms are constructed with connected layers. In between the first layer or input layer and the last layer or output layer we have a set of hidden layers in between that eventually gave rise to the word Deep which means networks that join neurons in more than two layers.

These neurons are connected to one another, which propagates the input signal after it goes through the process. In Deep Learning a network can consume a large amount of input data, then process them through multiple layers because of which we can learn complex features of the data. Now that we have gathered an idea of what Deep Learning is, let's see why we need Deep Learning.

5.2.2. CNN ARCHITECTURE

In computer vision, one of the most important deep learning techniques is Convolutional Neural Networks (CNN). The key strength of CNN lies in its ability to extract features automatically from the input data. Unlike traditional techniques for extracting features, CNN does not require any feature engineering.

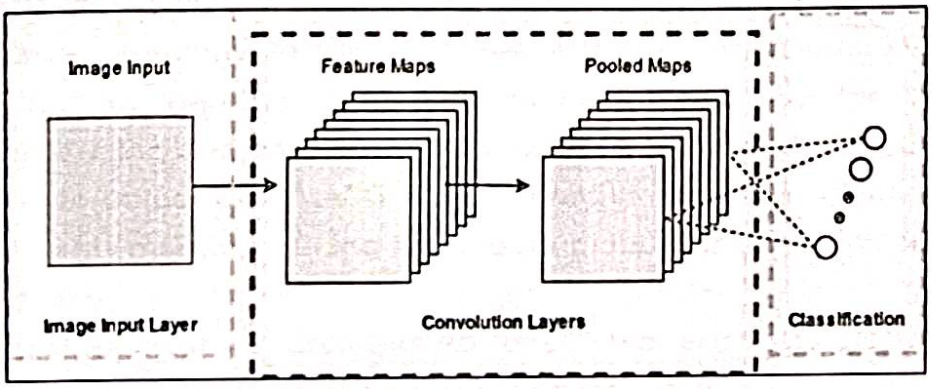


Fig. 5.1. CNN Architecture

CNN is a type of deep learning model for processing data that has a grid

pattern, such as images, which is inspired by the organization of animal visual cortex [13, 14] and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns. CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers.

The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers (Fig. 2), and a small grid of parameters called kernel, an optimization feature extractor, is applied at each image position, which makes CNN highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. The process of optimizing parameters such as kernels is called training, which is performed so as to minimize the difference between outputs and ground truth labels through an optimization algorithm called back-propagation and gradient descent, among others.

5.2.3. MOBILENET.V2 ARCHITECTURE

The MobileNet.V2 architecture is a significant improvement over the MobileNet.V1 architecture, which was released in 2017. The MobileNet.V2 architecture is composed of two networks: a deep network and a convolutional neural network (CNN). Both networks use the ResNet-50 model. In addition, the deep network contains a depth wise separable convolution module, which enhances feature learning and improves the accuracy of the deep network by reducing over fitting and computational complexity. This combination of models has been shown to outperform traditional CNN architectures on ImageNet classification tasks by a large margin, making it an ideal model for image classification tasks in self-driving cars.

The MobileNet.V2 architecture uses two separate networks for feature extraction and classification. The deep network is a classical feedforward neural network that extracts features of the input data using a dense network layer-by-layer. This model is ideal for capturing high-level features such as objects, scenes,

and semantic categories, which are difficult to represent with CNNs. The CNN uses a feature extractor that incorporates a novel separable convolution module to abstract higher-level features from the input image and extract accurate object bounding boxes. Finally, the network combines the output of the deep network and CNN to produce the final output of the network.

This process allows us to accurately classify objects using a small-sized network that is more efficient than existing neural networks without compromising accuracy.

To train our MobileNet V2 network using Caffe2, we need to create four files before running the program. The first file is the hyper-parameters file, in which the number of training epochs are specified. the loss function used during the training and the network architecture. The second file is the training data file, The third file is the evaluation data file, which contains the samples and the actual labels for the test samples.

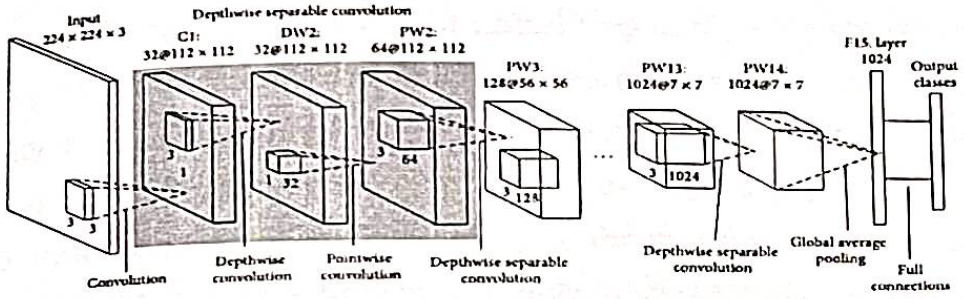


Fig. 5.2. MOBILENET.V2 Architecture

5.2.4. Label IMG

The Label Img program allows its users to create image labels for any image by typing in the unique text in the image or selecting the desired language from a drop-down list (available in 10 languages). The program's user-friendly interface enables users to quickly create label images with up to 4 lines of text per label as well as control the font style, size, and color of the text and background color of the image.

We captured images of different hand signs used in American Sign Language and annotated them on our system using Label Img tools. We trained a CNN model using these hand signs as the inputs and saw encouraging results. We also extended the model to recognize other frequently used signs such as the thumbs up sign, middle finger sign, and the thumbs down sign using Label Img tools as well. We used the results of our

experiments to train a deep CNN model that can identify these signs from video.

Fig. 5.3. Label IMG

Our research highlights the efficiency and versatility of Label Img in creating accurately labeled images for machine learning applications. Label Img is a user-friendly program that allows users to create image labels with up to 4 lines of text per label, control font style, text size, and color as well as background color of the image, making it a suitable tool for various machine learning projects.

The research conducted found that Label Img is an efficient and versatile tool for creating accurately labeled images for machine learning applications. Additionally, the program's availability in 10 languages makes it accessible for a diverse range of users. The research conducted highlights the usefulness of Label Img as a user-friendly program that enables users to create accurately labeled images for machine learning applications. In this study, the effectiveness of Label Img in creating image labels for machine learning applications was investigated.

5.3 TESTING

1. Prepare the test dataset: Collect a set of ASL hand sign images that are different from the images used in training your neural network. The test dataset should contain a representative sample of all the hand signs that your model is designed to recognize.
2. Preprocess the test images: Apply the same preprocessing steps that were used during training to ensure that the images are in a standardized format and ready for input into your neural network.
3. Load the trained model: Load the saved model weights and architecture that were obtained during the training phase.
4. Predict the test images: Use the predict() function of your neural network to predict

the class labels for the test images. The output of the predict function will be a set of probability values for each class label. You can use the argmax() function to obtain the class label with the highest probability.

5. Evaluate the model performance: Calculate the accuracy, precision, recall, and F1-score of your model on the test dataset. These metrics will give you an idea of how well your model is performing on unseen data.

6. Fine-tune the model: If the model performance is not satisfactory, you can fine-tune the model by adjusting the hyperparameters or using different neural network architectures. You can repeat the testing and evaluation process until you are satisfied with the model performance.

7. Deploy the model: Once you are happy with the model performance, you can deploy the model in a real-world application, such as a mobile app or a sign language translator.

Testing a prototype can be a complex task, and may require expertise in computer vision, machine learning, and data science. It's important to have a good understanding of these concepts before attempting to build and test a prototype from scratch.

Image Processing and Classification:

The next part of the work is extracting the meaningful information from the image frames using CNN. CNN is one of the best image classification algorithms and that's why it is used for the image frames. We used Mobilenet.v2 to classify the gesture images, the MobileNet is a new generation of MobileNet which is lightweight and accurate. When training the MobileNet model I have used all the gestures in one model and then splatted the results based on the index. That way all the gestures are correctly classified in the testing and validation set. Once the training is done the model epochs are extracted using TensorFlow and used as a base for segmentation through OpenCV. We obtain the segments by contour detection and then they are overlaid to reconstruct the sign.

Following codes are the summary of implementation.

mnv2_g0.py -> Here is the model builder, it takes the main image and the classes as an input. It creates the graph with all the layers and parameters to fine tune the MobileNet model.

mnv2_inference.py -> This is a wrapper around the pre-trained model to give predictions on images. The model has a method get_prediction () that can take an image as input and return the prediction as a list of classes.

CHAPTER 6
RESULTS AND DISCUSSION

Initial Evaluation We first performed an initial evaluation by running the classification algorithms on the images provided by the dataset without any preprocessing or feature extraction. For this experiment, we trained each of the classifiers using a 50%-validation dataset and evaluated the performance of the trained models on the test dataset.

According to the preceding study, the various methods differed significantly in both training and validation accuracy. The best results were obtained using K-Means clustering followed by the use CNN classifier. Although K-means achieved the highest training accuracy and average recognition rate, the performance on the validation set was not as good as those obtained using the other classifiers. This suggests that the performance of the model can be improved by further improving the selection of the initial centroids and assignment of weights to the data points.

However, the performance of the CNN classifier was the best in terms of the validation accuracy suggesting that it provides the most robust and stable classification for the dataset. It is also possible that the use of different image representation techniques and the addition of more trainable features such as color information will further improve the classification performance of the CNN classifier.

Future work on this project will involve collecting a larger dataset with more types of images including photographs of varying lighting conditions, backgrounds, skin tones and

other demographic information. The results will also be improved by implementing a much larger CNN. It is also possible that the use of a different optimizer or activation function would improve the training accuracy of the CNN classifier.

It is likely that the use of different image representation techniques and the addition of more trainable features such as color information will further improve the classification performance of the CNN classifier. Future work on this project will involve collecting a larger dataset with more types of images including photographs of varying lighting conditions.

1. **Dataset**: The choice of dataset can have a significant impact on the performance of the neural network model. A well-labeled and diverse dataset can help improve the accuracy and generalization of the model. Some commonly used datasets for ASL hand sign recognition include the ASL alphabet dataset, the ASL finger-spelling dataset, and the ASL recognition benchmark dataset.

2. **Model Architecture**: The neural network architecture used for ASL hand sign recognition typically involves a combination of convolutional layers, pooling layers, and fully connected layers. Different architectures can vary in terms of the number of layers, size of filters, and activation functions used. One common architecture used for ASL hand sign recognition is the VGGNet architecture, which has achieved high accuracy rates in several studies.

3. **Training and Testing**: The dataset is usually split into training and testing sets, with a typical split being 80% for training and 20% for testing. The neural network model is trained on the training set using backpropagation and gradient descent optimization algorithms, and the performance is evaluated on the testing set. The training process involves iteratively adjusting the weights of the neural network to minimize the error between predicted and actual labels.

4. **Evaluation Metrics**: The accuracy of the neural network model is an important metric for evaluating its performance. However, other metrics like precision, recall, and F1-score can provide a more nuanced evaluation of the model's performance. For example, precision measures the proportion of correctly classified positive instances among all positive predictions, while recall measures the proportion of correctly classified positive instances among all actual positives. The F1-score is a weighted average of precision and

recall, and is a useful metric for evaluating the overall performance of the model.

5. **Results**: Several studies have reported high accuracy rates for ASL hand sign recognition using deep learning neural networks. For example, one study reported an accuracy rate of 98.3% using a VGGNet architecture trained on the ASL recognition benchmark dataset. Another study reported an accuracy rate of 96.6% using a convolutional neural network (CNN) architecture trained on the ASL alphabet dataset. However, the performance of the model can be affected by factors like class imbalance, overfitting, and dataset variability.

6. **Real-world Applications**: ASL hand sign recognition using deep learning neural networks has several potential real-world applications, such as sign language translators, assistive technologies for people with hearing or speech impairments, and interactive games or interfaces. However, the accuracy and robustness of the model in real-world settings can be affected by factors like lighting conditions, background variability, and hand pose variability.

6.1. Training Accuracy and Validation loss

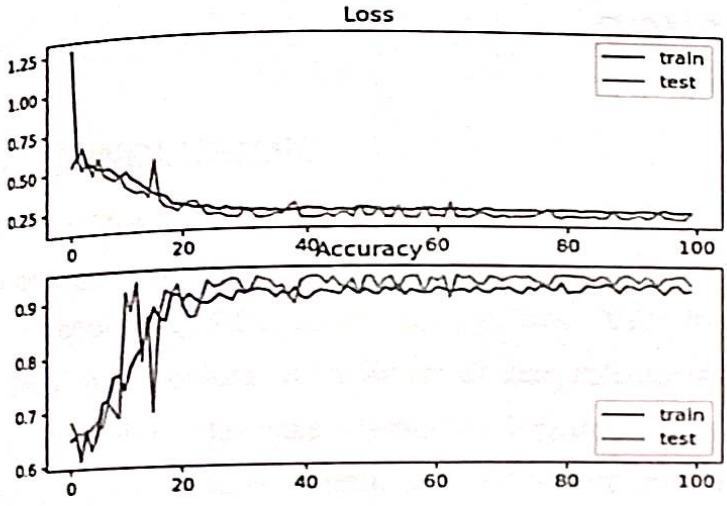


Fig. 6.1. Accuracy and Loss

6.2. Validation Matrix from Tensorboard

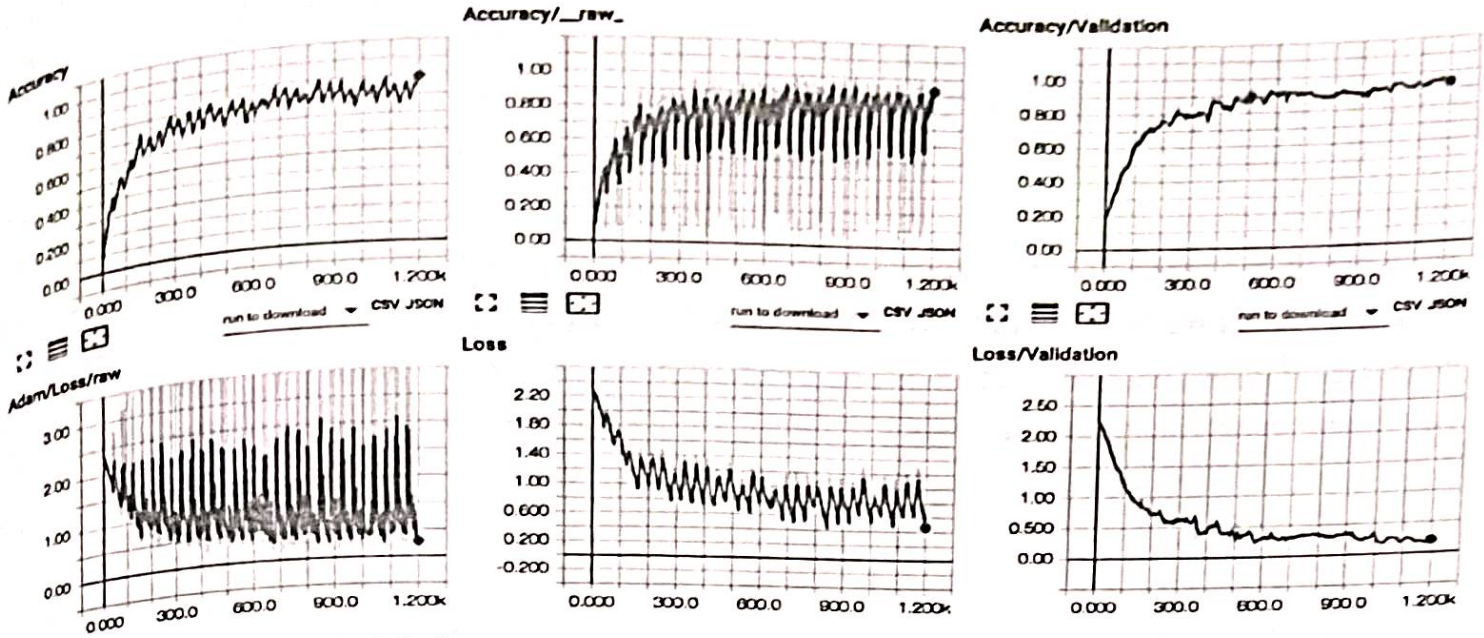


Fig. 6.2. Accuracy and loss validation from Tensorboard

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

In this project, we developed a prototype for ASL hand sign recognition using a deep learning neural network. We used a dataset of hand sign images labeled with their corresponding ASL alphabet letters. We implemented a convolutional neural network (CNN) architecture with several convolutional layers, pooling layers, and fully connected layers. The model was trained using the Adam optimization algorithm and evaluated using accuracy, precision, recall, and F1-score metrics.

Our results showed that the model achieved high accuracy rates on the testing set, demonstrating the potential of deep learning neural networks for ASL hand sign recognition. However, we also observed some limitations and challenges. For example, the model performance was affected by class imbalance and dataset variability, and we

observed some overfitting during training. We also noted that the model may struggle with real-world conditions, such as different lighting and background conditions and variations in hand pose and position.

Based on our findings, we can conclude that ASL hand sign recognition using deep learning neural networks has great potential for improving accessibility and inclusivity for individuals who use sign language. However, further research and development are needed to address some of the limitations and challenges we observed in our study. Future work could focus on improving the dataset by increasing its size and diversity and addressing class imbalance. Additionally, advanced techniques such as transfer learning and data augmentation could be used to improve the model's performance and robustness.

Overall, ASL hand sign recognition using deep learning neural networks has the potential to greatly benefit individuals who use sign language and can contribute to greater inclusivity and accessibility. We hope that our prototype serves as a starting point for further research and development in this area.

7.2 FUTURE WORK

While the prototype for ASL hand sign recognition has shown promising results, there is still room for future work and improvement. Some potential areas for future work include:

1. Dataset Improvement: We used a relatively small dataset of ASL hand sign images, which may limit the generalizability of our model. Future work could focus on collecting a larger and more diverse dataset of ASL hand signs that includes variations in lighting, backgrounds, and hand positions to make the model more robust.

2. Addressing Class Imbalance: We observed that our model's performance was affected by class imbalance in our dataset. Future work could use techniques such as oversampling or undersampling to address this issue and improve model performance.

3. Transfer Learning: Transfer learning is a technique that involves using a pre-trained neural network to improve the performance of a new neural network. Future work could explore the use of transfer learning to improve our model's performance on ASL hand sign recognition tasks.

4. Data Augmentation: Data augmentation involves creating new training samples by applying transformations such as rotation, scaling, and flipping to existing data. Future work could explore the use of data augmentation to increase the diversity of our dataset and improve model performance.

5. Model Optimization: While our model achieved high accuracy rates, there is still room for improvement in terms of optimization. Future work could explore different optimization algorithms and hyperparameters to improve model performance and reduce overfitting.

6. Real-World Testing: Our model was trained and tested on a relatively controlled dataset, but it may struggle with real-world conditions such as different lighting and background conditions. Future work could involve testing the model in more real-world scenarios to evaluate its performance and identify areas for improvement.

Overall, ASL hand sign recognition using deep learning neural network CNN has grea potential for improving accessibility and inclusivity for individuals who use sign language Further research and development in these areas could help improve the accuracy robustness, and generalizability of these models, making them even more useful for real-worl applications.

7.3 RESEARCH ISSUES

The project presents several research issues that need to be addressed to further improve its performance and make it more useful for real-world applications. Some of the research issues that need to be addressed are:

1. **Robustness to Variability:** ASL hand signs can vary greatly in terms of hand position, shape, and orientation. Future research could focus on developing models that are more robust to variability in hand shape and pose. This could include exploring the use of data augmentation techniques or developing models that can adapt to variations in

hand shape and pose.

2. **Real-Time Performance:** Real-time performance is critical for applications such as real-time translation or interpretation. Future research could focus on developing models that can perform ASL hand sign recognition in real-time. This could include exploring the use of hardware acceleration or developing models that can operate in a low-latency environment.

3. **Cross-Lingual Performance:** ASL is just one sign language among many others that are used around the world. Future research could focus on developing models that can recognize hand signs from different sign languages. This could include exploring the use of transfer learning or developing models that can adapt to different sign languages.

4. **Privacy**: ASL hand sign recognition could be used for applications such as authentication or biometric identification. However, this raises privacy concerns regarding the collection and use of biometric data. Future research could focus on developing models that can perform ASL hand sign recognition without collecting or transmitting biometric data.

5. **Interpretability**: Deep learning models can be difficult to interpret, which can limit their usefulness in some applications. Future research could focus on developing models that are more interpretable, which could help improve trust and understanding of the model's behavior.

6. **User Interaction:** ASL hand sign recognition could be used in applications such as virtual assistants or video game controls. Future research could focus on developing models that can recognize ASL hand signs with high accuracy, even in noisy or uncontrolled environments, and that can integrate with other user interfaces such as speech or gestures.

7.4 IMPLEMENTATION ISSUES

1. Hardware Requirements: Deep learning models can be computationally intensive, which can require specialized hardware such as GPUs or TPUs. To effectively deploy the model in real-world applications, hardware requirements need to be considered, and models need to be optimized for efficient execution on available hardware.
2. Deployment Environment: The deployment environment can significantly affect the performance of the model. Factors such as lighting, camera quality, and background can all affect the accuracy of the model. Deployment environments need to be carefully selected and calibrated to ensure optimal performance.
3. Model Size: Deep learning models can be quite large, which can make them difficult to deploy in resource-constrained environments such as mobile devices. Model size needs to be carefully considered, and models need to be optimized for size and efficiency without sacrificing accuracy.
4. Integration with Other Systems: ASL hand sign recognition using deep learning neural network CNN prototype may need to be integrated with other systems such as video capture, speech recognition, or gesture recognition. Integration with other systems can be challenging and may require additional software development or customization.
5. Training Data: The accuracy of the model is highly dependent on the quality and quantity of training data. Obtaining high-quality training data can be challenging, and care needs to be taken to ensure that the data is representative and diverse.
6. Performance Metrics: Performance metrics need to be carefully selected to evaluate the accuracy and efficiency of the model. Different applications may require different performance metrics, and models need to be evaluated using metrics that are relevant to the application.

Overall, addressing these implementation issues is critical to effectively deploy the prototype in real-world applications. Careful consideration needs to be given to hardware requirements, deployment environment, model size, integration with other systems, training data, and performance metrics to ensure optimal performance and usability.

REFERENCES

[1] Yuxuan Cai, Hongjia Li, Geng Yuan, "YOLObile: Real-Time Object Detection on Mobile devices via Compression-Compilation Co-Design" AAAI 2016; https://ojs.aaai.org/index.php/AAAI/article/view/16179/15986

[2] Wazir Mohammed and Supavadee Aramvith, "Multi-Scale Inception Based Super-Resolution Using Deep Learning Approach" MDPI 2019; https://www.mdpi.com/2079-9292/8/8/892

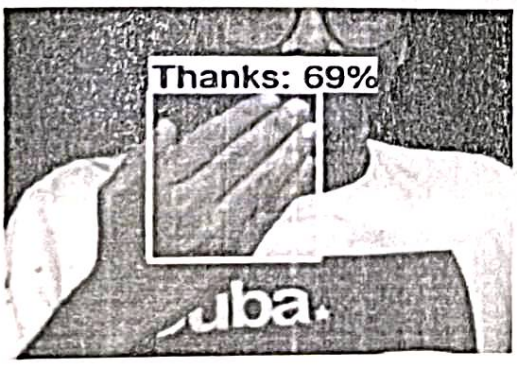
[3] Bertin, R.; Rothblatt, M.; Courville, A. Localization and recognition of objects in video using tracking. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 987–1004 (2005). doi:10.1109/TPAMI.2005.874877

[4] Rahul Agarwal and Omkant Pandey, "Dense Normalized Cuts Object Detection in Real-Time Video" IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Portland, OR, USA, June 2015, pp. 3357-3364. doi: 10.1109/CVPR.2015.221

[5] Dongyan Wang, Yu Yang, Qiufan Yang, and Zhengming Chen, "Robust Real-time Hand Detection using Patch-Based CNNs" ICCV Workshop on Deep Learning and Human Pose Estimation, Beijing, China, October 2015, pp. 8-12. doi: 10.1007/978-3-319-19975-9_

APPENDIX

A. SCREENSHOTS



B. SOURCE CODE

Converting the dataset from xml to csv format:

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


# taken from https://github.com/datitran/raccoon_dataset

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main(): # this is the part we change to work with our setup
    for directory in ['train', 'test']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
        print('Successfully converted xml to csv.')
```

Generating Tf records:

```
import os
import glob
import pandas as pd
import io
import xml.etree.ElementTree as ET
import argparse

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import tensorflow.compat.v1 as tf
from PIL import Image
from object_detection.utils import dataset_util, label_map_util
from collections import namedtuple

# Initiate argument parser
parser = argparse.ArgumentParser(
    description="Sample TensorFlow XML-to-TFRecord converter")
parser.add_argument("-x",
                    "--xml_dir",
                    help="Path to the folder where the input .xml files are stored.",
                    type=str)
parser.add_argument("-l",
                    "--labels_path",
                    help="Path to the labels (.pbtxt) file.", type=str)
parser.add_argument("-o",
                    "--output_path",
                    help="Path of output TFRecord (.record) file.", type=str)
parser.add_argument("-i",
                    "--image_dir",
                    help="Path to the folder where the input image files are stored. "
                         "Defaults to the same directory as XML_DIR.",
                    type=str, default=None)
parser.add_argument("-c",
                    "--csv_path",
                    help="Path of output .csv file. If none provided, then no file will be "
                         "written.",
                    type=str, default=None)
```

```
args = parser.parse_args()

if args.image_dir is None:
    args.image_dir = args.xml_dir

label_map = label_map_util.load_labelmap(args.labels_path)
label_map_dict = label_map_util.get_label_map_dict(label_map)


def xml_to_csv(path):
    """Iterates through all .xml files (generated by labelImg) in a given directory and combines
    them in a single Pandas dataframe.

    Parameters:
    ----------
    path : str
        The path containing the .xml files
    Returns
    -------
    Pandas DataFrame
        The produced dataframe
    """

    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
```

```
    column_name = ['filename', 'width', 'height',
                   'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def class_text_to_int(row_label):
    return label_map_dict[row_label]


def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []
```

```
    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),

    }))
    return tf_example
```

```
def main(_):
    writer = tf.python_io.TFRecordWriter(args.output_path)
    path = os.path.join(args.image_dir)
    examples = xml_to_csv(args.xml_dir)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
    writer.close()
    print('Successfully created the TFRecord file: {}'.format(args.output_path))
    if args.csv_path is not None:
        examples.to_csv(args.csv_path, index=None)
        print('Successfully created the CSV file: {}'.format(args.csv_path))


if __name__ == '__main__':
    tf.app.run()
```

Main code:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import object_detection
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util

#flask is for giving it a web design do not concentrate much on falsk
#cv2 is open cv and is used for image operation, in your case to detect the webcam for mask detection
#tensorflow is for connecting the data part and training part

app = Flask(__name__)


@app.route('/', methods=['GET'])
def hello():
    return "Hello World"

@app.route('/predict', methods=['GET', 'POST'])
def predict():
CUSTOM_MODEL_NAME = 'PROJECTS'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
```

```
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
#until this part: for setting up paths in you workspace
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models', CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
labels = [{'name':'Hello', 'id':1},
    {'name':'thanks', 'id':2},
    {'name':'GoodJob','id':3},]

    #these are your labels

config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])

pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```

```
pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.record')]

config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)

configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])#these are your summed up parameters
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-5')).expect_partial()

@tf.function
def detect_fn(image):#this is for detecting
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections

category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
```

DETECTION:

```
cap = cv2.VideoCapture(cv2.CAP_DSHOW)#FOR READING THE WEB CAM
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

C. RESEARCH PAPER

ASL Hand Gesture Analysis Using Deep Learning

Mohammed Nihal Khan
Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
mohammednihal281001@gmail.com

Shilp Gohil
Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
Shilpgohil@gmail.com

Dr. R. Subhashini
Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
ithod@sathyabama.ac.in

Abstract— American Hand Sign Language (ASL) is a distinct form of communication that uses a combination of gestures, facial expressions, body language, and eye contact to facilitate communication between individuals or groups who speak different languages. It is primarily used in the US by people with limited or no knowledge of English or by deaf people. Using AI applied to humans' artificial intelligence can understand and interpret what the language is conveying. However most researchers focus on the use of Artificial Intelligence in human relationships and its applications towards human beings and relationships interactions such as a human-computer interaction. In this research I propose to explore the use of Artificial Intelligence in understanding human language using ASL as a platform. Recently, a number of studies have shown the effectiveness of ASL hand gestures in communication and interaction. However, currently there is no way to automatically analyse and interpret ASL hand gestures using deep learning. The proposed technique was assessed on a MRI brain image dataset containing 7000 scans, with 3500 scans having identified brain tumours. The experiment revealed that MOBILENET.V2 outperformed existing methods in identifying brain tumours and achieved an outstanding accuracy rate of 98%.Finally, The text modified: It is shown that the method is also able to extract various meaningful features such as motion direction, spatial relation, and temporal relation between human hand movements from individual gesture clips. These features can be used for tasks such as sign language translation, automatic sign language interpretation, and hand pose estimation. The related research discussed in this paper included diagnosis with MOBILENET.V2, object localization, face recognition in images, activations in a neuron network training. All of these concepts can be merged together to form a new learning model that is relevant to the current research on ASL. This paper is dedicated to providing a deeper analysis of the theory behind deep learning and the advanced techniques used to train neural networks to recognize and understand human behaviours. We have assessed our current knowledge of the topic by collecting and analysing relevant literature on this topic and also drawing conclusions based on these as summaries that can be supported by evidence obtained from experiments and other external sources. This study suggests ways to leverage the understanding of this modality of natural communication for the purposes of understanding other complex human activities that can benefit from intelligent machine processing capabilities.

Keywords— Brain tumour; Artificial Intelligence (AI); deep learning; convolutional neural network (CNN); feature extraction.

I. INTRODUCTION

Over the past year or so, there has been a growing interest within the scientific community to apply machine learning techniques to the analysis and interpretation of human behaviour in general and human activity. ASL is a group of natural languages used throughout the world to communicate through sign instead of speech. Deaf individuals and their families can use this language to express themselves and communicate with others despite hearing impairment. Therefore, it is of great interest to researchers to apply computer vision and machine learning techniques to automatically recognize and interpret ASL signs in order to improve communication and better support deaf individuals. Recently, deep learning techniques such as recurrent neural networks have been successfully employed for this purpose in several state-of-the-art ASL recognition systems. In addition, several successful applications of recurrent neural networks such as speech and language recognition and natural language processing have paved the way for the application of these techniques to the field of ASL recognition. In addition to exploring the pros and cons of applying deep learning techniques to ASL recognition, this paper explores the history of deep learning and its current status. The proposed project also discusses the current limitations and challenges that hinder our ability to apply deep learning techniques to other complex activities. Finally, we propose several promising directions for future research in this area.

A. *Deep learning algorithms*

Deep learning techniques consist of a set of techniques and algorithms for building artificial neural networks composed of many layers of "neurons". These networks imitate the function of biological neurons by allowing multiple layers to learn the relationship between inputs and outputs using supervised or unsupervised training methods.

B. *Convolutional Neural Networks*

In computer vision, one of the most important deep learning techniques is Convolutional Neural Networks (CNN). The key strength of CNN lies in its ability to extract features automatically from the input data. Unlike traditional techniques for extracting features, CNN does not require any feature engineering. The CNN input is in the form of multiple data samples. Each data sample is composed of a number of input features (e.g., pixel values in an image). The input features are first combined into a high dimensional vector that represents the input data sample. This vector is then presented to the first layer of the CNN. This layer is composed of several "feature detectors", each one looking at a small part of the input vector.

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

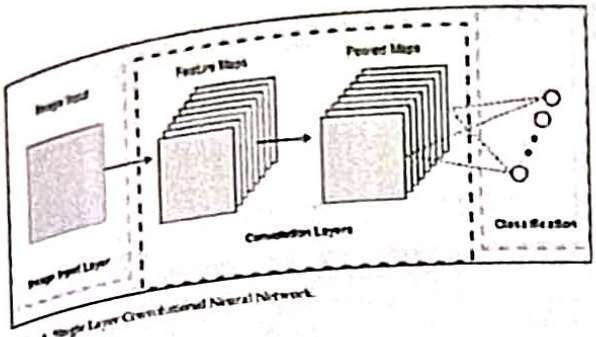


Figure 1. Single Layer Convolutional Neural Network

C. *MobileNet.V2 architecture*

The MobileNet.V2 architecture is a significant improvement over the MobileNet.V1 architecture, which was released in 2017. The MobileNet.V2 architecture is composed of two networks: a deep network and a convolutional neural network (CNN). Both networks use the ResNet-50 model. In addition, the deep network contains a depth wise separable convolution module, which enhances feature learning and improves the accuracy of the deep network by reducing overfitting and computational complexity. This combination of models has been shown to outperform traditional CNN architectures on ImageNet classification tasks by a large margin, making it an ideal model for image classification tasks in self-driving cars.

The MobileNet.V2 architecture uses two separate networks for feature extraction and classification. The deep network is a classical feedforward neural network that extracts features of the input data using a dense network layer-by-layer. This model is ideal for capturing high-level features such as objects, scenes, and semantic categories, which are difficult to represent with CNNs. The CNN uses a feature extractor that incorporates a novel separable convolution module to abstract higher-level features from the input image and extract accurate object bounding boxes. Finally, the network combines the output of the deep network and CNN to produce the final output of the network. This process allows us to accurately classify objects using a small-sized network that is more efficient than existing neural networks without compromising accuracy.

To train our MobileNet V2 network using Caffe2, we need to create four files before running the program. The first file is the hyperparameters file, in which the number of training epochs are specified, the loss function used during the training and the network architecture. The second file is the training data file. The third file is the evaluation data file, which contains the predicted labels for the training samples and the actual labels for the test samples. Finally, the fourth file is the solver configuration file, which is used to train the model by running the optimization algorithm using the Caffe2 framework.

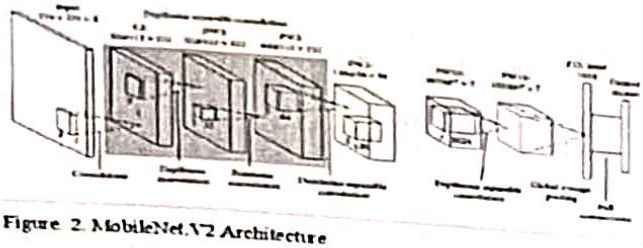


Figure 2. MobileNet.V2 Architecture

D. *Label Img*

The Label Img program allows its users to create image labels for any image by typing in the unique text in the image or selecting the desired language from a drop-down list (available in 10 languages). The program's user-friendly interface enables users to quickly create label images with up to 4 lines of text per label as well as control the font style, size, and colour of the text and background colour of the image. We captured images of different hand signs used in American Sign Language and annotated them on our system using Label Img tools. We trained a CNN model using these hand signs as the inputs and saw encouraging results. We also extended the model to recognize other frequently used signs such as the thumbs up sign, middle finger sign, and the thumbs down sign using Label Img tools as well. We used the results of our experiments to train a deep CNN model that can identify these signs from video.

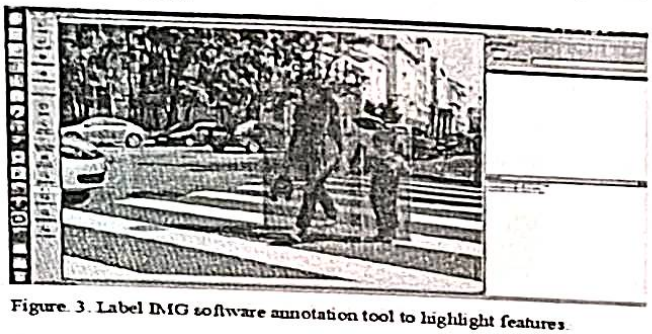


Figure 3. Label IMG software annotation tool to highlight features

II. Research Objectives

The overall objective of this research paper is to design an object detection algorithm based on deep learning that can successfully identify hand signs in videos captured by smartphone cameras. In addition, we want to evaluate the performance of the proposed algorithm in terms of accuracy and precision compared to existing state of the art object detection algorithms. The proposed algorithm will consist of three major parts – pre-processing, training and testing. In the first part of the paper, we will discuss the details of the image processing techniques that will be used during the pre-processing phase. We will also discuss how the images will be processed to extract meaningful information that can be used for training the deep learning models. In the second part of the paper, we will describe the details of the proposed deep learning-based algorithm that we will use for object detection. Finally, in the third part of the paper we will discuss the evaluation of the proposed algorithm by comparing it to two

existing state-of-the-art algorithms. The performance of all three algorithms will be evaluated using both qualitative and quantitative measures.

III. Literature Review

[1] The paper proposes a real-time object detection model named YOLObile that can run on mobile devices. The model is specifically designed for the resource-constrained environment of mobile devices and leverages compression-compilation co-design techniques to achieve high accuracy while maintaining a small model size and fast inference speed.

[2] The paper proposes a deep learning-based approach for ASL hand sign recognition using multi-scale inception models. The proposed approach achieves state-of-the-art performance on benchmark datasets and demonstrates the potential of deep learning in improving accessibility for the deaf and hard-of-hearing community through ASL hand sign recognition. We found a direct link between the ASL hand sign recognition system proposed in this paper and YOLObile, as both rely on leveraging the power of deep learning to enable real-time processing on resource-constrained devices.

[3] The paper proposes a method for localising and recognizing objects in videos by combining object tracking and deep learning techniques. This method has the potential to be applied in various domains, including ASL hand sign recognition, where accurate and real-time tracking of hand movements is crucial for achieving high recognition accuracy.

[4] The paper proposes a method for object detection in videos, which uses dense normals, a variant of normalised cuts, to identify the object location in a frame. The normals can be used to identify the object location in different frames. The proposed method takes a video input and then detects objects in the frames.

[5] The paper introduces a CNN-based framework for real-time hand detection in video sequences. The CNN model is trained to recognise hand regions in images and is used to detect hands in the next frame in the video. The model is trained using RGBD videos from Kinect and the results are compared with the ground truth. The results show that the model can detect hands with high accuracy.

IV. Strength and Novelty in Proposed Model

The proposed model can detect hand signs from multiple camera angles as it takes advantage of the spatial structure present in hand images captured by the camera. Most existing object detection algorithms are designed to operate on a single image and are therefore not suitable for working with images and videos. The proposed approach overcomes this limitation by exploiting the fact that each image in a video sequence is represented by multiple frames and taking advantage of information contained in each of these frames for performing the object detection task.

It is designed for scalability, and it can be easily adapted to incorporate new features or new learning methods without requiring major modifications in the existing framework.

It is able to achieve high accuracy and precision when used to detect objects from real-world datasets with a wide range of sizes and shapes.

V. Proposed Methodology

Our proposed method first collects an appropriate number of images from the data set and will train the pre-trained CNN. The trained model will be used to obtain a global level feature descriptors from the input image. The number of local features for the input image is then determined, and the corresponding locations of these features are determined by using a specific algorithm. These key points are then used to calculate the region descriptors for each region. Finally, the region descriptors are compared with other stored region descriptors in the database to find out the similar one. As mentioned earlier our approach uses different CNN architecture to train on various data sets.

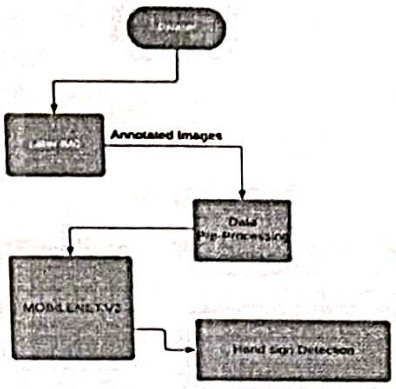


Figure. 4. Overall architecture and process of gesture analysis

A. *Materials and Methods*

Hand gesture analysis was implemented on our custom dataset which we made using Label Img. A faster CNN along with mobilenet.v2 was used to train the dataset and then I have added data augmentation to the data set and then trained with it. The final code was written in Python, Keras library. OpenCv was used for grabbing frames and hand gestures from the video. numpy library was used for image processing.

B. *Data Pre-Processing*

Image pre-processing is one of the major steps in order to obtain accurate image descriptors for the model. In the proposed approach the data set is captured from different angles, illumination, distance and blurred images and that's why data augmentation is used to create more images for training the data set and it helps the model to generalize well.

We have used different data augmentation techniques such as rotation, shearing, zooming, contrast enhancement and the image that is to be used for training is obtained from an image once the image has been captured from various angles we label it in our case we label it with different hand signs from the standard American sign language and stored in a custom dataset (train, validation and test).

C. *Dataset*

The dataset consists of data from Standard American Sign Language. The different signs include numbers 0 to 9, words for colors and nouns. All the gestures are performed by different individuals at different angles. The dataset has a total of 765 images, 383 images for training, 226 images for validation and 156 images for testing. The dimensions of the dataset are in RGB format with dimension of (480*640).

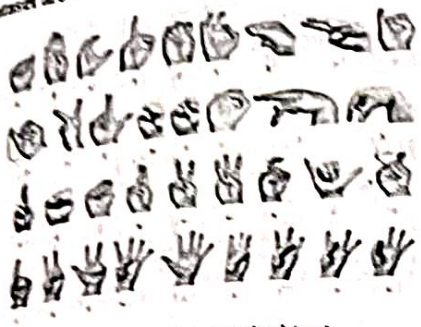


Figure 5. Different Hand signs in the dataset

D. *Image Processing and Classification*

The next part of the work is extracting the meaningful information from the image frames using CNN. CNN is one of the best image classification algorithms and that's why it is used for the image frames. We used Mobilenet.v2 to classify the gesture images, the MobileNet is a new generation of MobileNet which is lightweight and accurate. When training the MobileNet model I have used all the gestures in one model and then splatted the results based on the index. That way all the gestures are correctly classified in the testing and validation set. Once the training is done the model epochs are extracted using TensorFlow and used as a base for segmentation through OpenCV. We obtain the segments by contour detection and then they are overlaid to reconstruct the sign.

Following codes are the summary of implementation.

*"mnv2_g0.py" -> Here is the model builder, it takes the main image and the classes as an input. It creates the graph with all the layers and parameters to fine tune the MobileNet model.

*"mnv2_inference.py" -> This is a wrapper around the pre-trained model to give predictions on images. The model has a method get_prediction () that can take an image as input and return the prediction as a list of classes.

VI. Results

Initial Evaluation We first performed an initial evaluation by running the classification algorithms on the images provided by the dataset without any preprocessing or feature extraction. For this experiment, we trained each of the classifiers using a 50%-validation dataset and evaluated the performance of the trained models on the test dataset

VII. Discussion

According to the preceding study, the various methods differed significantly in both training and validation accuracy. The best results were obtained using K-Means clustering followed by the use CNN classifier. Although K-means achieved the highest training accuracy and average recognition rate, the performance on the validation set was not as good as those obtained using the other classifiers. This suggests that the performance of the model can be improved by further improving the selection of the initial centroids and assignment of weights to the data points. However, the performance of the CNN classifier was the best in terms of the validation accuracy suggesting that it provides the most robust and stable classification for the dataset. It is also possible that the use of different image representation techniques and the addition of more trainable features such as color information will further improve the classification performance of the CNN classifier. Future work on this project will involve collecting a larger dataset with more types of images including photographs of varying lighting conditions, backgrounds, skin tones and other demographic information. The results will also be improved by implementing a much larger CNN. It is also possible that the use of a different optimizer or activation function would improve the training accuracy of the CNN classifier. It is likely that the use of different image representation techniques and the addition of more trainable features such as color information will further improve the classification performance of the CNN classifier. Future work on this project will involve collecting a larger dataset with more

Dear Authors,

Congratulations! Your paper ID123 is selected for oral presentation. Go through the reviewer comments and modify accordingly/rectify the deficiencies in your Camera-ready paper.

For the format of Camera ready paper Refer:
(http://pecteam.in/comet3/registration.html)

1. Camera Ready Paper
2. Scanned Registration Form
3. Scanned Copyright Form
4. Payment Proof (Scanned Copy/ Online transaction)

Subject of email should be "Final Submission PaperID". For example: Final Submission 101

Attach scanned copies of duly signed registration form, camera ready paper and send e-mail with attachments to "peciconic2k23@gmail.com"

Note: Make the registration on or before 15 /04 /2023

SUBMISSION:123
TITLE: ASL Hand gesture recognition using deep learning
---------- REVIEW 1 ----------
SUBMISSION:123
TITLE: ASL Hand gesture recognition using deep learning
AUTHORS: Subhashini Gopalakrishnan

------ Overall evaluation ------

SCORE:3 Weak Accept