

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590 018.



A PROJECT REPORT

on

“AI CRICKET SCORE”

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

Bachelor of Engineering

in

Computer Science and Engineering

by

P RADHA RAI : **1VI19CS066**

PRAKHAR KUMAR CHANDRAKER : **1VI19CS070**

SHAIK NOWSHEEN : **1VI19CS098**

SHANTANU KUMAR SINGH : **1VI19CS100**

Under the supervision of

Ms. VEENA G
Assistant Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VEMANA INSTITUTE OF TECHNOLOGY

BENGALURU – 560 034

2022 - 23

Karnataka ReddyJana Sangha®
VEMANA INSTITUTE OF TECHNOLOGY
(Affiliated to Visvesvaraya Technological University, Belagavi)
Koramangala, Bengaluru-34.



Department of Computer Science and Engineering

Certificate

Certified that the project work entitled “**AI CRICKET SCORE**” carried out jointly by **P Radha Rai (1VI19CS066)**, **Prakhar Kumar Chandraker (1VI19CS070)**, **Shaik Nowsheen (1VI19CS098)**, **Shantanu Kumar Singh (1VI19CS100)**, are bonafide students of **Vemana Institute of Technology** in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2022-23. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the said degree.

Supervisor

Ms. Veena G

HOD

Dr. M. Ramakrishna

Principal

Dr. Vijayasimha Reddy. B. G.

Submitted for the university examination (viva-voce) held on

External Viva

Internal Examiner

External Examiner

- 1.
- 2.

DECLARATION BY THE CANDIDATES

We the undersigned solemnly declare that the project report “AI CRICKET SCORE” is based on our own work carried out during the course of our study under the supervision of ‘Ms. Veena G’.

We assert the statements made and conclusions drawn are an outcome of our project work. We further certify that,

- a. The work contained in the report is original and has been done by us under the general supervision of my supervisor.
- b. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.
- c. We have followed the guidelines provided by the university in writing the report.
- d. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and their details are provided in the references.

Date:

Place:

Project Team Members:

P Radha Rai	1VI19CS066
Prakhar Kumar Chandraker	1VI19CS070
Shaik Nowsheen	1VI19CS098
Shantanu Kumar Singh	1VI19CS100

ACKNOWLEDGEMENT

First we would like to thank our parents for their kind help, encouragement and moral support.

We thank **Dr. Vijayasimha Reddy. B. G**, Principal, Vemana Institute of Technology, Bengaluru for providing the necessary support.

We would like to place on record our regards to **Dr. M. Ramakrishna**, Professor and Head, Department of Computer Science and Engineering for his continued support.

We would like to thank our project coordinators **Mrs. Mary Vidya John**, Assistant Professor and **Mrs. J Brundha Elci**, Assistant Professor, Dept. of CSE for their support and coordination.

We would like to thank our project guide **Ms. Veena G**, Assistant Professor, Dept. of CSE for her continuous support, valuable guidance and supervision towards successful completion of the project work.

We also thank all the Teaching and Non-teaching staff of Computer Science and Engineering Department, who have helped us to complete the project in time.

Project Team Members:

P RADHA RAI	(1VI19CS066)
PRAKHAR KUMAR CHANDRAKER	(1VI19CS070)
SHAIK NOWSHEEN	(1VI19CS098)
SHANTANU KUMAR SINGH	(1VI19CS100)

CONTENTS

<u>Content Details</u>	<u>Page No.</u>
Title Page	i
Bonafide Certificate	ii
Declaration	iii
Acknowledgement	iv
Contents	v
Abstract	ix
List of Abbreviation	x
List of Figures	xi
List of Tables	xiii
Chapter 1	Introduction
	1 – 4
1.1	Introduction
	1
1.2	Scope
	2
1.3	Objectives
	2
1.4	Organization of the project work
	2
Chapter 2	Literature Survey
	4 – 22
2.1	Low cost approach for Real Time Sign Language Recognition
	4
2.2	Static Hand Gesture Recognition Based on Convolutional Neural Networks
	5
2.3	Hand Gesture Recognition Systems with the Wearable Myo Armband
	7
2.4	Gesture Recognition to Make Umpire Decisions
	9
2.5	Automatic Labeling of Sports Video Using Umpire Gesture Recognition
	10

2.6	Vision-Based Sign Language Translation Device	12
2.7	RGB-H-CbCr Skin Color Model for Human Face Detection	14
2.8	A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition	15
2.9	A Dataset and Preliminary Results for Umpire Pose Detection Using SVM Classification of Deep Features	17
2.10	An Approach to Automate the Scorecard in Cricket with Computer Vision and Machine Learning	19
2.11	Comparative Analysis	21

Chapter 3	System Analysis	23-24
------------------	------------------------	--------------

3.1	Existing System	23
	3.1.1 Drawbacks	23
3.2	Proposed System	23
3.3	Feasibility Study	24
	3.3.1 Technical Feasibility	24
	3.3.2 Operational Feasibility	24
	3.3.3 Economical Feasibility	24

Chapter 4	System Specification	25-27
------------------	-----------------------------	--------------

4.1	Hardware Requirements	25
4.2	Software Requirements	25
4.3	Functional Requirements	25
4.4	Non-Functional Requirements	26

Chapter 5	Project Description	28–45
5.1	Problem Definition	28
5.2	Overview of the Project	28
5.3	System Architecture	28
5.4	Module Description	29
	5.4.1 Create Dataset of Umpire Gestures	30
	5.4.2 Preprocessing Gesture dataset	30
	5.4.3 Train and test gesture dataset	31
	5.4.4 Preprocessing umpire dataset	36
	5.4.5 Train and test umpire dataset	37
	5.4.6 Capturing and detecting umpire	41
	5.4.7 Detecting hand gesture of umpire	42
5.5	Data Flow Diagram	43
5.6	Use Case Diagram	44
5.7	Sequence Diagram	45
 Chapter 6	 System Testing	 46–48
6.1	Introduction	46
6.2	Test Cases	46
 Chapter 7	 System Implementation	 49–52
7.1	Introduction	49
7.2	Screen Shots	49

Chapter 8	Conclusions and Future Enhancements	53
8.1	Conclusions	53
8.2	Future Enhancements	53
	References	54
	Appendix A	55-89
A.1	Front End	55
A.2	Back End	56
A.3	Source Code	58
A.4	Installation Procedure	84

ABSTRACT

Gesture Recognition pertains to recognizing meaningful expressions of motion by a human, involving the hands, arms, face, head, and/or body. It is of utmost important in designing an intelligent and efficient human–computer interface. The applications of gesture recognition are manifold, ranging from sign language through medical rehabilitation, monitoring patients or elder people, surveillance systems, sports gesture analysis, human behavior analysis etc., to virtual reality. In recent years, there has been increased interest in video summarization and automatic sports highlights generation in the game of Cricket. In Cricket, the Umpire has the authority to make important decisions about events on the field. The Umpire signals important events using unique hand on signals and gestures. The primary intention of our work is to design and develop a new robust method for Umpire Action and Non-Action Gesture Identification and Recognition based on the Umpire Segmentation and the proposed Histogram Oriented Gradient (HOG) feature Extraction oriented Non-Linear Support Vector Machine (NL-SVM) classification of Deep Features. Primarily the 80% of Umpire action and non-action images in a cricket match, about 1, 93, 000 frames, the Histogram of Oriented Gradient Deep Features are calculated and trained the system having six gestures of Umpire pose.

Keywords: Python, Artificial Intelligence, Jupyter, Cricket Score, Sign Language Recognition, Umpire, Histogram of Oriented Gradients, Non-Linear SVM classifier

LIST OF ABBREVIATIONS

Abbreviation	Description
ANN	: Artificial Neural Network
AI	: Artificial Intelligence
CNN	: Convolutional Neural Network
CCTV	: Closed Circuit Television
DL	: Deep Learning
GB	: Giga Byte
HOG	: Histogram Oriented Gradient
KNN	: K-Nearest Neighbor
ML	: Machine Learning
NLP	: Natural Language Processing
OCR	: Optical Character Recognition
RGB	: Read Green Blue
SVM	: Support Vector Machine
VR	: Virtual Reality

LIST OF FIGURES

Fig No.	Name	Page No.
1.1	Timeline chart	3
5.1	System Architecture	29
5.2	Number of images per category	31
5.3	Percent Distribution of gesture across categories	32
5.4	Random images	32
5.5	DenseNet121 model	33
5.6	Loss history	34
5.7	Visualizing accuracy history	34
5.8	Visualizing precision history	35
5.9	Visualizing recall history	35
5.10	Confusion matrix	36
5.11	Number of umpire images per category	37
5.12	Percent Distribution of umpire across categories	38
5.13	Normal and umpire random image	38
5.14	Categorical accuracy	39
5.15	Precision	40
5.16	Recall	40
5.17	Confusion matrix	41
5.18	Data flow diagram (level 1)	43
5.19	Data flow diagram (level 2)	43
5.20	Use case diagram	44

5.21	Sequence diagram	45
7.1	Anaconda command prompt	49
7.2	Camera feed to capture the dataset	50
7.3	Preprocessed Dataset	50
7.4	Umpire Detection	51
7.5	Saving the Result of Umpire detection	51
7.6	Checking if umpire is detected or not	52
7.7	If umpire is detected then recognizing the gestures	52
A.1	Anaconda Navigator	57
A.2	Jupyter Notebook home page	58
A.3	Anaconda website	85
A.4	Locate your download and double click it	85
A.5	Click on next	86
A.6	Anaconda License Agreement	86
A.7	Anaconda Installation type selection	87
A.8	Choose Installation Location	87
A.9	Advanced Installation Options	88
A.10	Click on next	88
A.11	Installation Complete	89

LIST OF TABLES

Table No.	Name	Page No.
6	Comparative Analysis	21
6.1	Test case specifications of Gestures	46

CHAPTER 1

INTRODUCTION

Gestures are known as expressive, meaningful body movement which involves physical motion of the fingers, hands, arms, head, face, or body with the intent of assigning meaningful information or interacting with the environment. In recent years' gestures are widely used by humans to interact with computers and machines. Many common everyday equipment like TV, Smartphone, Car dashboard etc. can now be controlled by simple hand gestures. Gestures are also applied in many fields like developing aids for the hearing impaired, enabling very young children to interact with computers, recognizing sign language, medically monitoring patients' emotional states or stress levels, lie detection, monitoring automobile driver's alertness/drowsiness levels, etc. Involvement of gesture recognition technology in sports will make the gameplay fairer and proficient.

The gestures performed by the sports officials which indicate what is going on in the game. Which also can provide something meaningful about a player or the entire game. If the gestures of these officials are able to be recognized, meaningful information can be derived. We refer to a gesture as an intentional action by a person whereby part of the body is moved in a predefined way to indicate a specific event. Detecting these events enables automatic generation of highlights, contextual labeling of video and more importantly helps in decision making and automatic score update.

Cricket is the most popular sport in India. It is the 2nd most famous sports in Asia, 4th in Europe and also 2nd most famous sport in the whole world. But from the 19th century the same old manual method is being use to update the scoreboard, which is a great burden for the scorekeeper. The way of viewing the score has changed a lot in time, but the basic score updating process is still the same and performed by a person. So, in the 21st century an automatic system is very much needed at this sector. Automatic systems are taking places of many boring manual task which was performed by humans 5 or 10 years ago. So, developing a fully operational automated system like this is in dire need. Using modern equipment's and machine learning algorithms we can increase the accuracy of the decision and provide flawless result what the naked eye misses.

On the other hand, in sensor-based system the person who perform the gesture wears sensor, and when the gesture is performed the sensor data taken and used to identify the gesture. In the area of sports and other sector, several attempts have been made for gesture recognition using both sensor-based and vision-based technique. [10]

1.1 Scope

Technology can act as a flexible medium for hearing and speech impaired people to communicate amongst themselves and with other individual to enhance their education. Another expectation is to use the research outcome as learning tool of sign language where learner can practice. It can be implemented in all types of sports like football, tennis, baseball, hockey, kabaddi etc.

1.2 Objective

The primary aim of the proposed system is to develop a mechanism that can identify the Umpire based on his dress code and then detect the various gestures and postures made by the Umpire during the game to update the score. This system is designed to identify and recognize all the relevant and meaningful gestures made by the Umpire, which will then be displayed on the score board. By using this system, the Umpire's actions and decisions can be accurately tracked and recorded, which can help to ensure the accuracy and fairness of the game. Additionally, this system can provide valuable information to players and spectators, as it can help them to better understand the Umpire's decisions and actions during the game.

1.3 Organization of the project work

Chapter 1: This chapter gives the introduction about the project by defining the objective and scope of the project.

Chapter 2: This chapter explains about literature survey which gives information which we gain from each paper.

Chapter 3: This chapter gives information about System Analysis in which existing system proposed system and feasibility study.

Chapter 4: This chapter gives information about Hardware and Software requirements, Functional and non-functional requirements.

Chapter 5: This chapter gives an overview of the project and the methods used.

Chapter 6: This chapter gives detailed information about the results of the test cases implemented.

Chapter 7: This chapter speaks about working of the final implemented project.

Chapter 8: This chapter gives summary about the Phase 2 implementation of the project.

The Fig 1.1 refers the timeline chart of the project:

Month/ Activity	September				October				January				February				March				April			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Problem Formulation																								
Synopsis Submission																								
Research, Tools and Equipment Procurement																								
Building the Prototype																								
Testing the Prototype for the required Specifications																								
Customizing the Prototype																								
Testing the Prototype for the required Specifications																								
Report Submission																								

Fig 1.1 Timeline Chart

CHAPTER 2

LITERATURE SURVEY

2.1 Title: Low cost approach for Real Time Sign Language Recognition [1]

Author: Matheesha Fernando, Janaka Wijayanayaka

Identify the signs and convert them into text and speech using appearance based approach with a low cost web camera. A series of image processing techniques with Hub-moment classification was identified as the best approach. Uses fast Convex Hull Algorithm for Binary image for pattern recognition, Histogram based classification. During the research available human computer interaction approaches in posture recognition were tested and evaluated. A series of image processing techniques with Hu-moment classification was identified as the best approach. The system is able to recognize selected Sign Language signs with the accuracy of 76% without a controlled background with small light adjustments. With the development of information technology new areas of human computer-interaction are emerging. There, human gesture plays a major role in the field of human computer interaction. As sign language is a collection of gestures and postures any effort in sign language recognition is in the field of human computer interaction. there are two types of approaches commonly used to interpret gestures for human computer interaction. First category, Data Glove based approach relies on electromechanical devices attached to a glove for digitizing hand and finger motions into multipara metric data. The major problem with that approach is it requires wearing the devices and will cause fewer natural behaviors. And also, these devices are quite expensive. Appearance based method was selected for the research as main objective of the research is to identify a low-cost method for sign language recognition. In appearance-based method feature extraction and classification are the major components. Feature extraction methods are used to reduce the number of dimensions of an image. A descriptor can be used for that. A descriptor describes an image and if properly used, image can be represented less with dimensions than the image itself. Also, it can introduce some useful properties like scale and rotation invariance. Classification includes a broad range of decision theoretic approaches to the identification of images. It analyses the numerical properties of various image features and organizes data in to categories. The classification represents the task of assigning a feature vector or a set of features to some predefined classes in order to recognize the hand gesture. The research is focused on an application that can convert a video signal (processed as sequence of images) into a sequence of written words (text) and speech in real time. In the real-world, visual information could be very rich, noisy, and incomplete, due to changing illumination and dynamic backgrounds .

Vision-based systems should be user independent and robust against all these factors. The suggested solution for the communication problem requires real-time facility, with effective as well as cost efficient techniques/algorithms. Therefore, robustness, computational efficiency and uses Tolerance were the important challenges need to be considered here. In this approach the movement of the hand is recorded by a camera and the input video is decomposed into a set of features by taking individual frames into account. The video frames contain background pixels other than the hand, as the hand will never fill a perfect square. These pixels have to be removed as they contain arbitrary values that introduce noise into the process. The basic idea is to get the real time image and then extract the predefined features and then compare the feature vectors against the features extracted from stored template sign images. In the feature extraction phase what is most important is to get possible precise features as output. A very simple method is to compare each pixel location with each other and sum all the differences. This method is not realistic as it is not going to work on images that are not the same size or orientation. And also, for the same hand posture there will be different images with small variations. Also, it is computationally expensive for larger images. For an image of 100 by 100 pixels there are already 10,000 dimensions. Therefore, features selected for classification are hand contour, orientation histogram, convex hull, convexity defects and moments.

Advantages:

- Helps in identifying a low cost, affordable method that can facilitate hearing and speech impaired people to communicate with the world in more comfortable way where they can easily get what they need from the society and also can contribute to the well-being of the society.

Disadvantages:

- This project only looks at the hand postures not on hand gestures.

2.2 Title: Static Hand Gesture Recognition Based on Convolutional Neural Networks [2]

Author: Raimundo F. Pinto Jr., Carlos D. B. Borges, Antonio M. A. Almeida, and Ialis C. Paula Jr.

Proposes a gesture recognition method using convolutional neural networks. The procedure involves the application of morphological filters, contour generation, polygonal approximation, and segmentation during preprocessing, in which they contribute to a better feature extraction. Segmentation algorithms can be implemented to separate, colors, textures, points, lines, discontinuities, borders, among others. Training and testing are performed with different

convolutional neural networks, compared with architectures known in the literature and with other known methodologies. All calculated metrics and convergence graphs obtained during training are analyzed and discussed to validate the robustness of the proposed method. The images are obtained from the database. The images go through an image processing stage, in which the following operations occur: color segmentation using an MLP network, morphological operations of erosion and closing, contour generation, and polygonal approximation, to remove image noise. After segmentation, binary images are obtained, so a logical AND operation is performed between these images and the originals, in order to preserve the information contained in the fingers and the surface of the hand. After these steps, the images are used to train a CNN and assess the performance of the technique with cross validation. Finally, the validation results are analyzed. Proposed architectures presented good results, with average rates of success of 96%. For CNN 1, with only two layers of convolution, it presented an accuracy rate of 94.7%, and for CNN 2, 3 and 4 accuracies remained above 96%. This is possible due to the proposed image processing methodology, in which unnecessary information is removed, allowing improved feature extraction by the CNN. Proposed methodology and CNN architecture open the door to a future implementation of gesture recognition in embedded devices with hardware limitations. One of the problems in gesture recognition is dealing with the image background and the noise often present in the regions of interest, such as the hand region. Use of neural networks for color segmentation, followed by morphological operations and a polygonal approximation, presented excellent results as a way to separate the hand region from the background and to remove noise. This step is important because it removes image objects that are not relevant to the classification method, allowing the convolutional neural network to extract the most relevant gesture features through their convolution and pooling layers and, therefore, to increase network accuracy. Proposal to make a logical AND operation with the segmentation masks and the original images provided the relevant information of the palms and fingers. The proposed CNN architectures achieved high success rates at a relatively low computational cost. In addition, the proposed architectures reached accuracies very similar to the architectures already defined in the literature, although they are much simpler and have a lower computational cost. It is possible due to the proposed image processing methodology, in which unnecessary information is removed, allowing improved feature extraction by the CNN. It is clear that starting from 3 layers of convolution, together with pooling layers, modifying this type of neural network architecture does not increase the feature extraction and classification capacities of the network. Therefore, there is no significant increase in accuracy, but only in the computational cost of the network. After analyzing the network convergence times during training, it is seen that increasing the number of convolution layers reduces the number of epochs necessary for the network to converge, thus

extracting the data faster. CNN 1 converges at epoch 16, while CNN 3 converges at epoch 7. The architectures already defined in the literature presented accuracy rates with values up to 99%. However, they are more complex than the proposed architectures, some of them are more than 200 layers deep. Thus, these architectures are also capable of extracting characteristics of the images more quickly, presenting the smallest numbers of epochs for convergence, as is the case of InceptionV3 and ResNet50, which converged at epochs 5 and 7. Results were also generated using the individual image datasets. It is possible to conclude that the CNNs are able to extract the features and classify the patterns well enough to reach correct answers close to 100%. Demonstrating robustness of the methodology independent of the base used.

Advantages:

- Proposed methodology are much simpler and have a lower computational cost.

Disadvantages:

- The proposed methodology approaches only cases of gestures present in static images.

2.3 Title: Hand Gesture Recognition Systems with the Wearable Myo Armband [3]

Author: Engin Kaya, Tufan Kumbasar

The hand gesture recognition systems deal with identifying a given gesture performed by the hand. Utilized machine learning techniques to recognize the hand gestures. Seven different time domain features are extracted from the raw EMG signals using sliding window approach to get distinctive information. The performance of KNN, SVM and ANN algorithm will be compared. Then, the dimension of the feature matrix is reduced by using the principal component analysis to reduce the complexity of the deployed machine learning methods. The presented study includes the design, deployment and comparison of the machine learning algorithms that are k-nearest neighbor, support vector machines and artificial neural network. The results of the comparative comparison show that the support vector machines classifier-based system results with the highest recognition rate. The first step of the process is to extract meaningful and distinctive features from the raw data. Due to the complex and noisy characteristics of EMG, proper selection of features is essential for classification. In machine learning, working with directly raw data increases the complexity of implementation and processing cost. Also, some hidden patterns which cannot be visible in a single data point can be obtained in features. In this study, Authors have performed a sliding window method to record the data with windows length of 40 and sliding length of 20 as a pre-processing step. Then, the seven-time domain features are calculated which are the Mean Absolute Value (MAV), variance, waveform length, Root Mean Square (RMS), Willison amplitude, Zero Crossing (ZC) and Slope Sign Change (SSC) of the signals. It can be firstly observed that increasing the number of neurons in the

hidden layer has not significantly improved the recognition accuracy. For the handled data set, an optimal number of hidden of neurons can be set as 12. In comparison of the training methods, the trained method resulted with lowest accuracy; whereas, trainrp and trainscg provided significantly better performances. Structure recognition systems with the wearable armband to classify the numbers from 0 to 9 in Turkish Sign Language (TSL). First step is to introduce the armband that has 8 electrodes/channel and handled data set. Then, to utilize machine learning techniques to recognize the hand gestures. In this context, a sliding window approach will be applied for signals of each channel of the armband and seven-time domain features will be extracted from each obtained window. Hence, the feature matrix will have 56 dimensions that will be then reduced by applying Principal Component Analysis (PCA) algorithm to 15 dimensions. Then, using the processed feature set, design and deploy the machine learning algorithms KNN, SVM and ANN as classifiers and compare their recognition performances. In the comparative comparisons, the performance of the KNN algorithm will be examined for various distance metrics and number of neighbors while the SVM classifier will be tested for linear and radial basis function kernel methods. For the SVM, the performances of One Versus One (OVO) and One Versus All (OVA) binary classification methods will also be examined. The performance of the ANN for various numbers of neurons in hidden layer is analyzed and three different training algorithms. In the light of these analyses, the tuning parameters of the classification methods is adjusted to optimal values to perform overall performance comparison in hand gesture recognition. The results will show that highest recognition rate will be obtained when the SVM classifier is deployed. The first step of the process is to extract meaningful and distinctive features from the raw data. Due to the complex and noisy characteristics of EMG, proper selection of features is essential for classification. In machine learning, working with directly raw data increases the complexity of implementation and processing cost. Also, some hidden patterns which cannot be visible in a single data point can be obtained in features. In this study, Authors have performed a sliding window method to record the data with windows length of 40 and sliding length of 20 as a pre-processing step. Then, the seven-time domain features are calculated which are the Mean Absolute Value (MAV), variance, waveform length, Root Mean Square (RMS), Willison amplitude, Zero Crossing (ZC) and Slope Sign Change (SSC) of the signals. Dimension reduction is a process of reducing the number of variables in the feature vector set. In the classification problems, choosing the number of inputs has great importance to determine the time and space complexity of the classifier; therefore, working with less dimensional data provides simpler solution. In this study, the well-known technique PCA, has been employed to reduce the dimension of the feature matrix. It is observed that the first 15 principal components preserve about 96 percent of the variance; and thus reduced the feature matrix dimension.

Advantages:

- Achieve good accuracy

Disadvantages:

- Need to testing the proposed method with recording signals from different people and for more complicated hand gestures.

2.4 Title: Gesture Recognition to Make Umpire Decisions [4]

Author: Lesha Bhansali ,Meera Narvekar

The Umpire Gesture Recognition System aims squarely to introduce a more robust technology to show Umpire choices with the assistance of Gesture Recognition and trailing of hand movement of the Umpire. This technology helps to alleviate the burden of the scorekeepers. Authors tested the subsequent six gestures particularly OUT, SIX, NEWBALL, NO-BALL, DEAD_BALL, FOUR. Edge detection algorithms which emphasize edges and transitions. The Umpire Gesture Recognition System aims squarely to introduce a more robust technology to show Umpire choices with the assistance of Gesture Recognition and trailing of hand movement of the Umpire. This technology helps to alleviate the burden of the scorekeepers. It conjointly minimizes errors in displaying Umpire choices therefore adding to a more robust viewing expertise. The steps concerned during this method are as follows: Authors Implemented the gradient magnitude calculation. The aim is to outline wherever within the image the most important gradient magnitudes are present. Then, it'll be used to determine a threshold within the gradients so as to filter the concerned area of interest (hands, palms and legs) and to discard all the background. They Created a gradient magnitude threshold that had to erase the lower levels gradients so as to keep the higher ones. This cut all the noise and regularized the background. Then, succeeding step was to calculate the geometric distance between the vectors of the various pictures analyzed. This half was formed to check the various photos, by scrutiny of histograms. With this, area of unit is calculated to acknowledge the various gestures. This technique is termed Eigen faces. It's a helpful applied mathematics technique that has found application in numerous fields (such as face recognition and image compression). Authors have conferred results on the recognition of Umpire's gestures during a cricket match. The novelty of their work is that they've had an inclination to use the recognition of gesture to display the output directly on screens. Gesture recognition is then performed among the device domain that avoids the problems associated with correct image segmentation. They have an inclination to use this approach to the popularity of various sports. Results show

that this recognition system is capable of recognizing a group of six umpire gestures from the game of cricket and performs best once using a feature set. Also, in addition authors want to show the performance of segmenting gestures from a stream of continuous gestures by selecting candidate gestures by the existence of movement. Then, to use mathematician filter to blur the image and have a homogeneous image. It allowed to get higher ends up in the gradient magnitude. The goal of this filter is to erase the background defects. It's very vital to obtain a regular background to avoid noise. Authors created a gradient magnitude threshold that had to erase the lower levels gradients so as to keep the higher ones. This cut all the noise and regularized the background. Then, succeeding step was to calculate the geometric distance between the vectors of the various pictures analyzed. This half was formed to check the various photos, by scrutiny of histograms. With this, they calculated area of unit to acknowledge the various gestures. The first side to put into thought is the style of underlying structure design. An important reason for choosing this framework is that it is user-friendly. With the assistance of MATLAB authors have done it in a simple manner. The very last step was to check by calculating the geometric distance between the coefficients that area unit before every eigenvector. As the game of cricket has evolved over the decades, so has the technology behind the game. The system "Umpire Hand Gesture Recognition System"(UHGRS) created has been instrumental in increasing the accuracy of the decisions, provide flawless replays to help see what the naked eye misses. The technology does not only benefit the players but is equally instrumental in providing a better viewing experience to the audience. Often while watching a cricket match live at a stadium, it is difficult to discern the umpire decisions due to poor visibility. The score-keepers have to be constantly alert as to when there is a fall of a wicket or a wide ball, dead ball, no ball etc. "To err is human". As humans are prone to error this technology come to their rescue. If the scorekeeper accidentally misses to display the correct signals on the flat screen, the audience will definitely disapprove it. Thus, the "UHGRS" is a clever way to use technology and minimize the errors associated with displaying the correct umpire decisions.

Advantages:

- Capable of recognizing a group of six umpire gestures from the game of cricket

Disadvantages:

- No performance of segmenting gestures.

2.5 Title: Automatic Labeling of Sports Video Using Umpire Gesture Recognition [5]

Author: Graeme S. Chambers, Svetha Venkatesh, and Geoff A.W. West.

Annotating sports videos Data from accelerometers is used to augment sports video. Umpires

in the game wear wrist bands. A hierarchical hidden Markov model to solve the problem of automatic segmentation and robust gesture classification. The algorithm proceeds as follows: for each period of movement ahead in time (up to 10sec) of the start of a candidate gesture, calculate the likelihood of each model for that region. Gestures can be considered to exist at multiple levels in a hierarchy, where simple movements are grouped into more complex movements and complex movements are grouped into ordered sequences. The advantage of hierarchical modelling is this temporal decomposition of gestures. The classification stage becomes more manageable for increasingly complex gestures as the dynamics of the gesture are explicitly encoded. Not only does grouping allow for segmenting a gesture into its subparts, hierarchical modelling allows new gestures to be learnt on-line by reusing subparts from already known gestures. Sports officials perform many gestures which are indicative of what is going on in the game. Their gestures can provide something meaningful about a player, a team, or the entire game. If the gestures of these officials are able to be recognized, meaningful information can be derived. Authors refer to a gesture as an intentional action where by part of the body is moved in a predefined way to indicate a specific event. Detecting these events enables automatic generation of highlights and more importantly, rich, contextual labelling of video. To solve this problem Authors addressed the issues of segmenting continuous gesture data and performing robust gesture classification. Potentially more than one model will exceed the threshold for the filler model, however in this work, they simply took the maximum. The difference between the most likely model and the second most likely model is not taken into account. A Gaussian distribution modelling the magnitude of gravity is used to detect periods of movement over a sliding window. The likelihood of the Gaussian is used to make a binary decision on whether the window contains movement. In some cases, when there is little acceleration, spurious responses to the Gaussian distribution can occur. For example, the sequence 1,1,0,1 (where 1 is movement and 0 is no movement), may result, however this is not consistent labelling since there is a large degree of overlap (48/144). Thus, this sequence is replaced by the sequence 1,1,1,1. Similarly, if the sequence 0,0,1,0 occurs, it is replaced with the sequence 0,0,0,0. These two filters are ensuring that contiguous regions of data have consistent labelling over a sliding window. Once the regions of movement indicating candidate gestures are detected, the regions must be classified. Since many sport umpire gestures contain pauses such as the cricket, there needs to be a method for grouping adjacent regions of identified movement including the pause periods. The problem is that a pause can be either a valid sub-gesture (as part of a gesture) or a pause between gestures. Authors approach to overcoming this is by using a conservative estimate for the maximum length of a gesture (10sec) and grouping all detected gestures and pauses in the window. This window is then iteratively reduced in

length removing the last region of movement at each step. For example, the first gesture starting after time 50 would have four candidate regions for grouping (times 69 to 106, 69 to 86, 69 to 79, and 69 to 75). The algorithm proceeds as follows: for each period of movement ahead in time (up to 10sec) of the start of a candidate gesture, calculate the likelihood of each model for that region. After all candidate regions have been identified and their corresponding model likelihoods calculated, they are normalized by the filler ratio. The region and model corresponding to the maximum of the calculated ratios is considered the gesture for that region. Using a filler model to compare different length observations for accurately finding segmentation endpoints is novel and has worked well on data and example domain. Without a filler model, different length observations cannot be compared across models as the HMM likelihood function is non-linear.

Advantages:

- He system performs well overall with the exception of handling unknown movements which have similarities to known movements.

Disadvantages:

- Filler ratio requires further investigation for deciding when a known gesture occurs.

2.6 Title: Vision-Based Sign Language Translation Device [6]

Author: Yellapu Madhuri, Anitha.G, Anburajan.M

This report presents a mobile Vision-Based Sign Language Translation Device for automatic translation of Indian sign language into speech in English to assist the hearing and/or speech impaired people to communicate with hearing people. Sign language is recognized using Lab View Software. The experienced lag time between the sign language and the translation is little because of parallel processing. This allows for almost instantaneous recognition from finger and hand movements to translation. This is able to recognize one handed sign representations of alphabets (A-Z) and numbers (0-9). The results are found to be highly consistent, reproducible, with fairly high precision and accuracy. The gestures of sign language are captured by the inbuilt camera to detect the movement of the hand. Capturing thirty frames per second (fps) is found to be sufficient. Higher fps will only lead to higher computation time of the computer as more input data to be processed. As the acquisition process runs at real time, this part of the process has to be efficient. Thus, previous frame that has been processed will be automatically deleted to free the limited memory space in the buffer Image acquisition process is subjected to many environmental concerns such as the position of the camera, lighting sensitivity and background condition. The camera is placed to focus on an area that can capture the maximum possible movement of the hand and take into account the difference

in height of individual signers. Sufficient lighting is required to ensure that it is bright enough to be seen and analysed. In this work, a vision-based sign language recognition system using LABVIEW for automatic sign language translation has been presented. This approach uses the feature vectors which include whole image frames containing all the aspects of the sign. This project has investigated the different issues of this new approach to sign language recognition to recognize on the hand sign language alphabets and numbers using appearance-based features which are extracted directly from a video stream recorded with a conventional camera making recognition system more practical. Although sign language contains many different aspects from manual and non-manual cues, the position, the orientation and the configuration or shape of the dominant hand of the signer conveys a large portion of the information of the signs. Therefore, the geometric features which are extracted from the signers' dominant hand, improve the accuracy of the system to a great degree. Facial expressions are not focused, although it is well known that facial expressions convey important part of sign-languages. A wearable IOS phone system provides the greatest utility for automatic Sign Language to spoken English translator. It can be worn by the signer whenever communication with a non-signer might be necessary, such as for business or on vacation. Providing the signer with a self- contained and unobtrusive first-person view translation system is more feasible than trying to provide second-person translation systems for everyone whom the signer might encounter during the day. To increase the performance and accuracy of the Automatic Sign Language Translator (ASLT), the quality of the training database used should be enhanced to ensure that the ASLT picks up correct and significant characteristics in each individual sign and further improve the performance more efficiently. A larger dataset will also allow experimenting further on performance in different environments. Such a comparison will allow to tangibly measuring the robustness of the system in changing environments and provide training examples for a wider variety of situations. Adaptive colour models and improved tracking could boost performance of the vision system. Current collaboration with Assistive Technology researchers and members of the Deaf community for continued design work is under progress. The gesture recognition technology is only one component of a larger system that one day be an active tool for the Deaf community. This project did not focus on facial expressions although it is well known that facial expressions convey important part of sign-languages. The facial expressions can e.g. be extracted by tracking the signers' face. Then, the most discriminative features can be selected by employing a dimensionality reduction method and this cue could also be fused into the recognition system. This system can be implemented in many application areas examples include accessing government offices for filling out forms whereby no interpreter may be present to help.

Advantages:

- Translator between deaf and people who do not understand sign language.

Disadvantages:

- Doesn't focus on facial expressions.

2.7 Title: RGB-H-CbCr Skin Color Model for Human Face Detection [7]

Author: Nusirwan Anwar bin Abdul Rahman, Kit Chong Wei and John See

This paper presents a novel skin color model, RGB-H-CbCr for the detection of human faces. Skin regions are extracted using a set of founding rules based on the skin color distribution obtained from a training set. The proposed scheme was also compared with the well-known AdaBoost face detector/classifier by Viola and Jones. The proposed scheme is able to reach comparable standards to that achieve by the AdaBoost algorithm (90.17%) on the similar data set. Face detection in colour images has also gained much attention in recent years. Colour is known to be a useful cue to extract skin regions, and it is only available in colour images. This allows easy face localisation of potential facial regions without any consideration of its texture and geometrical properties. In this paper, Authors have presented a novel skin colour model, RGB-H-CbCr to detect human faces. Skin region segmentation was performed using a combination of RGB, H and CbCr subspaces, which demonstrated evident discrimination between skin and non-skin regions. The experimental results showed that new approach in modelling skin colour was able to achieve a good detection success rate. On a similar test data set, the performance of their approach was comparable to that of the AdaBoost face classifier. The RGB-H-CbCr skin color model is able to deal with various brightness and illumination conditions, but it remains susceptible to detection of non-skin objects that possess similar chrominance levels as skin colour. The eccentricity property measures the ratio of the minor axis to major axis of a bounding ellipse. Eccentricity values of between 0.3 and 0.9 are estimated to be of good range for classifying face regions. Though this property works in a similar way as box ratio, it is more sensitive to the region shape and is able to consider various face rotations and poses. The proposed scheme was also compared with the well-known AdaBoost face detector/classifier by Viola and Jones and results showed that the proposed scheme (with the right configuration of morphological operators) is able to reach comparable standards to that achieve by the AdaBoost algorithm (90.17%) on the similar data set. To evaluate the effectiveness of the RGB-H-CbCr skin colour model, the face detection system was tested with various combination of colour models, each represented by its own set of bounding rules. The combination of all 3 subspaces resulted in the best DSR and lowest FDR values. The proposed method sometimes failed to detect a face correctly, as seen from the high

FDR of 28.29%. This could be attributed to the usage of morphological operators. Though these operators are used parallelly to improve the likelihood of detecting faces, it may sometimes cause “over-detection” of faces. In this paper, Authors have presented a novel skin colour model, RGB-H-CbCr to detect human faces. Skin region segmentation was performed using a combination of RGB, H and CbCr subspaces, which demonstrated evident discrimination between skin and non-skin regions. The experimental results showed that approach in modelling skin colour was able to achieve a good detection success rate. On a similar test data set, the performance of the approach was comparable to that of the AdaBoost face classifier. Authors intend to refine the use of morphological operations in the post-processing of the extracted skin regions. An adaptive training (incremental learning) of the skin colour model can be used to improve the overall classification of skin regions. Primarily, the elimination of false detections and false dismissals is crucial to the success of a robust face detector. The nextstep of the face detection system involves the use of morphological operations to refine the skin regions extracted from the segmentation step.

Firstly, fragmented sub-regions can be easily grouped together by applying simple dilation on the large regions. Hole and gaps within each region can also be closed by a flood fill operation. The problem of occlusion often occurs in the detection of faces in large groups of people. Even faces of close proximity may result in the detection of one single region due to the nature of pixel-based methods. Hence, they used a morphological opening to “open up” or pull apart narrow, connected regions. Additional measures are also introduced to determine the likelihood of a skin region being a face region. Two region properties – box ratio and eccentricity are used to examine and classify the shape of each skin region. The box ratio property is simply defined as the width to height ratio of the region bounding box. By trial and error, the good range of values lie between 1.0 and 0.4. Ratio values above 1.0 would not suggest a face since human faces are oriented vertically with a longer height than width. Meanwhile, ratio values below 0.4 are found to misclassify arms, legs or other elongated objects as faces.

Advantages:

- Brightness and illumination conditions can be effectively dealt.

Disadvantages:

- Low success of a robust face detector.

2.8 Title: A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition [8]

Author: Ali Moin, Andy Zhou, Abbas Rahimi, Alisha Menon.

Wearable devices that monitor muscle activity based on surface electromyography could be of use in the development of hand gesture recognition applications. Most devices with local processing cannot offer training and updating of the machine-learning model during use, resulting in suboptimal performance under practical conditions. The system can classify 13 hand gestures with 97.12% accuracy. A high accuracy (92.87%) is preserved on expanding to 21 gestures. A HD computing algorithm³¹ for training and inference of hand gestures is used to provide a real-time analysis of physiological signals, wearable biosensors can implement machine-learning models for signal processing. Local (in-sensor) processing of signals from biosensors has advantages over wirelessly streaming raw data to an external computational device, including reduced communication link bandwidth and radio power requirements. To provide a real-time analysis of physiological signals, wearable biosensors can implement machine-learning models for signal processing. Local (in-sensor) processing of signals from biosensors has advantages over wirelessly streaming raw data to an external computational device, including reduced communication link bandwidth and radio power requirements. Processing the signals locally can also offer improved latency and security. Machine learning models for in-sensor processing are, however, typically trained offline before they are implemented in low-power embedded processors. The 1,000-dimensional item memory elements were generated sequentially using a cellular automaton with a hardcoded seed for a smaller memory footprint. Subsequent processing steps were exactly as described in the previous section on HD classification architecture, replacing algebraic operations on bipolar hyper vectors with Boolean operations on binary vectors⁵². A shift register consisting of 21 hyper vectors was used as the AM to store trained prototype hyper vectors and search for the closest class during inference. Training examples for creating or updating a model or as queries for inference using a trained model. A prototype hyper vector for each class is formed by computing the class centroid. For binary and bipolar hyper vectors, this amounts to finding the majority of each element across all training examples. These prototype hyper vectors are then stored in an associative memory (AM), an entirely feedforward operation with a single pass over training data. This is in contrast to other neuro-inspired approaches in which training often employs sophisticated, iterative frameworks and is much more computationally demanding than classification (for example, gradient descent with backpropagation⁴⁶). Adding new classes to the model simply involves adding new prototype hyper vectors to the AM, again differentiating HD computing from other algorithms that may require full retraining or modifications to the architecture. Once all prototypes have been computed and stored, classification involves finding the nearest-neighbouring prototype to a query hyper vector. Authors tuned model hyper parameters and validated the HD algorithm using the offline dataset before optimizing it for efficient implementation using hardware description language (HDL)

for synthesis on the device's FPGA. Raw 15-bit analogue-to-digital converter codes were used as the input for feature extraction, with incremental calculations of the 50-sample MAV performed with each new sample. The implemented arithmetic operations consisted only of addition, two's complement inversion, and arithmetic right shift for division. Features were quantized and saturated to 6-bit integers based on analysis of the offline dataset, optimizing for the dynamic range (range divided by step size) given the arithmetic requirements. The 1,000-dimensional item memory elements were generated sequentially using a cellular automaton with a hardcoded seed for a smaller memory footprint. Subsequent processing steps were exactly as described in the previous section on HD classification architecture, replacing algebraic operations on bipolar hyper vectors with Boolean operations on binary vectors⁵². A shift register consisting of 21 hyper vectors was used as the AM to store trained prototype hyper vectors and search for the closest class during inference. A single contextual update with 50% weighting was enabled for each AM entry by merging a predetermined set of 500 bits from a newly trained prototype into the stored one. The frequency contents of the different recordings were qualitatively similar. Spot SNR was calculated by comparing the power spectrum during a gesture performance to the power spectrum during rest. Overall SNR (calculated as the integral of spot SNR) varied for different channels and different gestures. The Cometa and CapgMyo systems exhibited better peak SNR for the best channels and associated best gestures, probably because those systems consist of differential, bipolar recording configurations with improved common-mode noise rejection.

Advantages:

- Low-cost and low-complexity.

Disadvantages:

- Classification accuracy is less.

2.9 Title: A Dataset and Preliminary Results for Umpire Pose Detection Using SVM Classification of Deep Features [9]

Author: Aravind Ravi, Harshwin Venugopal, Sruthy Paul, Hamid R. Tizhoosh.

The umpire in cricket has the authority to make critical decisions about on-field events. The umpire communicates important events through distinct hand signals and gestures. They Determine four such events for classification: SIX, NO BALL, OUT and WIDE based on detecting the umpire's pose from the video frames from a cricket game. CNN algorithm is used, the early layers learn more generic features such as shapes, edges, and colour blobs, the deeper layers learn features more specific to the original dataset. Automatic video summarization has gained increased attention in the recent past. Sports highlights generation, movie trailer

generation, automatic headlines generation for news are some examples of video summarization. Convolutional neural networks (CNNs) have outperformed most of the traditional computer vision algorithms for tasks such as image classification and object detection. A CNN is a combination of a feature extractor and a classifier. The convolutional layers of the CNN are the feature extractors. They learn the representations automatically from the input data. The early layers in the CNN learn more generic features such as shapes, edges, and colour blobs, while the deeper layers learn features more specific to that contained in the original dataset. This work extends on the idea of identifying events in cricket videos based on detecting the pose of the umpire. In addition to this, they propose the SNOW dataset which comprises of five classes of umpire actions each corresponding to four events such as Six, No Ball, Out, Wide (SNOW) and a no action class is included. They have set the benchmark evaluation for image classification based on a linear SVM classifier trained on features extracted from pertained networks. Automatic video summarization has gained increased attention in the recent past. Sports highlights generation, movie trailer generation, automatic headlines generation for news are some examples of video summarization. The focus of the present work is sports video summarization in the form of highlights. The highlights of a game provide the summary of important events of that game such as a goal in soccer or a wicket in cricket. It is a challenging task to summarize the highlights from sports videos as these videos are unscripted in nature. An efficient approach can be based on identifying key events from the sports video and use them to automatically generate the highlights. Among sports, cricket is the most popular game in the world after soccer and has the highest viewership rating. A detailed explanation of the game of cricket can be found. In the game of cricket, the umpire is the person with the authority to make important decisions about events on the field. The umpire signals these events using hand signals, poses and gestures. This innate characteristic of the cricket video can be leveraged as one approach for solving the problem of cricket highlight generation. Therefore, a system can be developed to detect the unique signals and poses shown by the umpire to automatically generate cricket highlights. A method for umpire pose detection for generating cricket highlights based on transfer learning is proposed in this work. Authors explore the use of features extracted from the pre-trained networks such as Inception V3 and VGG19 networks pre-trained on ImageNet dataset. A linear support vector machine (SVM) classifier is trained on the extracted features for detecting the pose of the umpire. A new dataset, SNOW, is introduced in this work and all experiments are performed on this dataset. The system built using this dataset is evaluated on cricket videos for highlights generation. The paper is organized as follows: In Section II, the back ground work covering existing techniques will be discussed. Section III introduces the proposed dataset. Section IV outlines the overall system design and methodology for evaluating the proposed dataset as a benchmark for umpire pose detection and highlight generation. The

results of the experiments and their analysis are discussed in Section V. Section VI concludes the paper by providing directions for future work. A benchmark database such as TREC Video Retrieval Evaluation (TRECVID) for general video indexing, summarization and retrieval has been used in many studies. In the domain of sports video annotation and summarization, several works. Automatic summarization of soccer videos has been proposed. Similar studies for sports such as basketball, baseball, and tennis have also been reported. More recent work, in the domain of sports video summarization, has been reported in for the game of cricket. Prior studies have used event detection in cricket videos as the basis for highlight generation. Hari et al. have proposed a method based on intensity projection profile of umpire gestures for detecting events. A technique based on Bayesian belief networks for indexing broadcast sports video has been proposed. The use of sequential pattern mining to segment cricket videos into shots and identify the visual content is presented.

Advantages:

- The proposed system is an effective solution for the application of cricket highlights generation.

Disadvantages:

- Time complexity is high.

2.10 Title: An Approach to Automate the Scorecard in Cricket with Computer Vision and Machine Learning [10]

Author: Md. Asif Shahjalal, Zubaer Ahmad, Rushrukh Rayan, Lamia Alam.

This form of sports is widely played in more than 125 countries recognized by the International Cricket Council. One of the most challenging issues that first initiates the discussion on its prosperity is the duration of the game. The on-field umpire has to authorize decisions almost after each delivery. Haar-Cascade classifier algorithm is used to classify a specific type of object. This process would eliminate the manual updating of scorecards and thereby reduce the game duration notably. In addition, it excludes the prerequisite of wearing special gloves involving sensors. The efficiency of the algorithm is then cross-checked with the training and test data. This proved to be a very simple but efficient algorithm for umpire's gesture detection. Involvement of gesture recognition technology in sports will make the gameplay fairer and proficient. The gestures performed by the sports officials which indicate to what is going on in the game. Which also can provide something meaningful about a player, a team, or the entire game. If the gestures of these officials are able to be recognized, meaningful information can be derived. Authors refer to a gesture as an intentional action by a person whereby part of the body is moved in a predefined way to indicate a specific event. Detecting these events enables automatic generation of highlights, contextual labelling of video and more importantly helps in

decision making and automatic score update. In contrast to these, authors intended to design a vision-based system using logistic regression machine learning technique which can detect both static and dynamic hand gestures of a cricket umpire. Main goal of authors is trying to recognize the gestures an umpire performs in a match and update the scoreboard for the corresponding gesture accurately. They trained a haar-cascade classifier to detect human wrists from the video stream from a static camera. Then the selected region of interest is continuously checked through the multiclass logistic regression model if a gesture is matched. Then accordingly the score is updated. Sensor-based hand gesture recognition started with the invention of glove-based system, which was further divided into two distinct categories- active data glove and passive data glove over the years. The first glove-based systems were designed as a part of a camera-based LED system to track body and limb position for real-time computer graphics animation in the 1970s, and since then, a number of different designs have been proposed. A low-cost version named the Power Glove, was commercialized by Mattel Intellivision as a control device for the Nintendo video game console in 1989 and became well known among video games players. Chambers et. al proposed a probabilistic hierarchical framework to extract gestures and significant events of Kung Fu martial art movements acted out by an instructor in a simulated training video using accelerometers and the Hidden Markov Model. In another work, Chambers et. al proposed the use of the Hierarchical Hidden Markov Model (HHMM) in conjunction with a filler model for segmenting and classifying gestures at differing levels of detail. In this work, sports video was augmented with accelerometer data from wrist band worn by umpires in the game. The gestures they recognized are: Dead Ball, Four, Last Hour, Leg-Bye, No Ball, One Short, Out, Penalty Runs, TV Replay and Wide. Another popular way to recognize hand gesture is to use Kinect. Gesture detection was successfully recognized using a feature vector and a real-time histogram based algorithm by Gomez et. al. Wang et. al recently developed a simple and inexpensive for 3D articulated user-input using the hands. Their approach uses a single camera to track a hand, wearing an ordinary cloth glove that is imprinted with a custom pattern. Bhansali et. al proposed a method of Gesture recognition to Make Umpire Decisions by using subtraction method and gradient method. They tested the subsequent six gestures particularly OUT, SIX, NEWBALL, NO-BALL, DEADBALL and FOUR. Google is recently developing a system called project soli where they use special radar sensor to detect the fingers movement. Soli sensor technology works by emitting electromagnetic waves in a broad beam. Objects within the beam scatter this energy, reflecting some portion back towards the radar antenna. Properties of the reflected signal, such as energy, time delay, and frequency shift capture rich information about the objects characteristics and dynamics, including size, shape, orientation, material, distance, and velocity. In project soli there are some sensors are used but those are not attached with the gesture performers hand or

body. So these can be called as a hybrid system where sensor and radar vision both are used simultaneously. Sensor-based systems had limited accuracy and were tethered to computers using cumbersome wiring. They were meant for very specific applications and were never commercialized.

Advantages:

- This proved to be a very simple but efficient algorithm for umpire's gesture detection.

Disadvantages:

- Multiple classifiers were need to be trained in order to make it work.

2.11 Comparative Analysis

Table 2.1 Comparative Analysis

Reference	Algorithm/ Technique	Platform used	Performance Metrics	Advantage	Drawback
[1]	Convex Hull Algorithm	Linux	Hand outline, Hand contour, SIFT, SURF	low cost, affordable method	Only looks at the hand posture not hand gesture
[2]	CNN Algorithm	Windows, Linux	Precision, Recall, F1 score	Much simpler and lower computational cost.	Cases gestures present in static images.
[3]	KNN,SVM and ANN Algorithm	Linux	Precision, Recall, F1 score	Achieve good accuracy	More complicate hand gestures.
[4]	Edge detection algorithms	Windows	Gesture Recognition	Recognizing a group of six umpire gestures.	No performance segmenting gestures.
[5]	Segmentation algorithm	Linux	Known, Unknown, Recall	System performs well.	Ratio requires further investigation
[6]	Lab View	Windows,	Geometric	Translator between deaf and	Doesn't focus on

	Software.	Linux,IOS ,Android.	Features of Hand	people who don't understand sign language.	facial expressions.
[7]	RGB-H-CbCr skin colour model, AdaBoost Algorithm	Windows	FDR, DSR	Brightness & illumination can be effectively dealt.	Low success of a robust face detector.
[8]	Neuro-Inspired Hyperdimensional compute Algorithm	Linux	SNR	Low complexity	Mapping are not sparse.
[9]	CNN algorithm	Windows	Accuracy of Inception V3, VGG19-FC1	High detection accuracy.	Complex dataset.
[10]	Haar-Cascade classifier algorithm	Windows	GD,GR,SR	Efficiency enhanced to greater.	Devices expensive, cumbersome experience.

A comparative analysis is made for the literature survey based on the algorithm used, performance, platform used advantages and disadvantages which is shown in Table 2.11.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Existing system

The existing system of the project is to detect the umpire gesture and update it onto the final score board. in real time ICC matches the umpire has a counter device which contains buttons on it such as wicket's, balls, overs, through which umpire sends the decision to the score board updater where he analyzes and then display the output onto the board. everything happening in real time is manual in consideration with the existing project's the updation in this is that it does not consider the crowd and unwanted noise in the stadium it focuses only on the umpire's signal, its feature. whereas existing projects contains detection of both umpire and players signal.

3.1.1 Drawbacks

The system relies heavily on the quality of the input data, including the video feed and the training dataset. If the video feed is of poor quality or the training dataset does not cover a diverse range of gestures and actions, the system's accuracy will be compromised. Additionally, the system is not foolproof and can make errors in recognizing gestures, especially if the umpire's gestures are not distinct or if there is interference from other factors, such as noise or movement in the background. Another limitation of the system is that it may not be able to recognize new gestures that are not part of the training dataset, which could limit its overall effectiveness. Finally, the system requires specialized equipment and software, which could make it expensive and challenging to implement in all cricket stadiums. Therefore, while the gesture recognition system has the potential to improve the accuracy and efficiency of updating the scoreboard, it is important to consider its limitations and drawbacks before implementing it in real-world scenarios.

3.2 Proposed system

In this project, an umpire action and non-action detection and classification is developed based on Histogram of Oriented Gradients (HOG), Non-Linear SVM classifier and CNN. The general methodology of the proposed technique includes division, highlight extraction, and umpire activity and non-activity outlines arrangement.

At first, the Umpire video frames are extracted from the Umpire Frames Segmented database and the 80% of Umpire Action and Non-action frames are selected manually and feature extraction is performed based on HOG and trained to CNN. After that, the remaining 20% of the frames, feature extraction is performed using HOG and tested using trained CNN model, which categories the Action and Non-Action Umpire Frames.

3.3 Feasibility Study

The feasibility of the project is analyzed in this phase and 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. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in the feasibility analysis.

3.3.1 Technical Feasibility

The project requires the use of advanced computer vision and deep learning techniques, such as convolutional neural networks (CNNs), to detect umpire's gestures accurately. These techniques are well established and have been used in similar projects, indicating technical feasibility. The project also requires web cameras, computer hardware, and software, which are readily available and affordable

3.3.2 Operational Feasibility

The project requires skilled personnel to manage and maintain the system. The personnel will need to be trained in computer vision and machine learning techniques, as well as the use of the hardware and software. The system will also need to be tested and validated before deployment to ensure its reliability and accuracy.

3.3.3 Economic Feasibility

The project requires an initial investment in hardware, software, and personnel. However, the cost of these resources is relatively low, and the project can be implemented within a reasonable budget. The long-term cost of maintaining the system will depend on the frequency of use and the number of matches where the system is used.

CHAPTER 4

SYSTEM SPECIFICATION

4.1 Hardware Requirements

- Ram : 8. GB(minimum)
- System : Intel i5
- Hard Disk : 160 GB.
- Speed : 2.4 GHz

4.2 Software Requirements

- Operating System : Windows 10
- Coding Language : Python.
- Tools : Jupyter Notebook

4.3 Functional Requirements

- A Collection of Dataset consisting of various gestures and sign of umpire: To train and test a gesture recognition system, a dataset consisting of various gestures and signs of umpires is needed. This dataset will contain images or videos of umpires performing different gestures, and these gestures will be labelled and used as the training and testing data for the system.
- A layer that can Down sample the input along its spatial dimensions and flattens the input: To process the input images or videos of umpire gestures, a deep learning model is needed. The model will contain a layer that can down sample the input images or videos along their spatial dimensions to reduce the computational complexity of the model. This layer will also flatten the input data to feed it into the next layers of the model.
- A deep learning framework for processing and extracting the features of the dataset: The deep learning framework will be used to process the dataset and extract its features.

A deep learning architecture for training and saving our model: A deep learning architecture will be designed to train and save the gesture recognition model. This architecture will contain different layers, including convolutional layers, pooling layers, and fully connected layers. It will use backpropagation and stochastic gradient descent algorithms to optimize the model's parameters.

- A method for region proposal: To correctly detect the input gesture of the umpire, a method for region proposal is required. This method will identify regions of interest in the input image or video, which may contain the gesture of the umpire. This will help to reduce the computational complexity of the model by focusing only on the relevant regions.
- A deep learning architecture for correctly detecting the input gesture of umpire: The deep learning architecture will be designed to detect the input gesture of the umpire correctly. This architecture will contain a series of convolutional and pooling layers that will process the input data and extract features. It will also contain fully connected layers and output layers that will classify the input gesture into different categories.
- A display board for correctly displaying the recognized gestures: To display the recognized gestures, a display board is required. This board will receive the output of the gesture recognition system and display it in real-time. It may also have additional features like sound effects, animations, or other visual indicators to make it more engaging for the audience.

4.4 Non-Functional Requirements

- **Usability**

Usability here means the degree to which something is able or fit to be used. This Project is useful in Detecting the Umpire's Gesture and correctly guess the output of the gesture.

- **Reliability**

Reliability here means the probability of performing a specified function without failure under given conditions for a specified period of time. So the project should be reliable as to not cause error which can lead to false outcome.

- **Performance**

Performance here should be high i.e. it should not take more time to compute the results of umpire's gestures.

If the performance is low, it may lead to loss of interest and excitement of the audience.

- **Portability**

This Project should be portable so that it can be used in various places without worrying about the shifting cost. Portability is a great functionality to have because in cricket the matches can be organized at different places.

CHAPTER 5

PROJECT DESCRIPTION

5.1 Problem Definition

In a noisy stadium with thousands of spectators, the score board updater may miss umpire signals, leading to inaccuracies in the score board. Human error is also a possibility due to the fast-paced nature of the game.

5.2 Overview of the Project

The project "AI Cricket Score" aims to implement machine learning and artificial intelligence techniques in the field of cricket to detect the umpire's gestures for updating the score board. The project involves capturing images of the umpire's hand gestures using a web camera and saving them in the respective directories for pre-processing and feature extraction. Deep learning algorithms will be implemented on the saved model for prediction, and the dataset size will be increased to more than 5000 RGB images for better accuracy. The real-time prediction of umpire's gestures using image frames from a web camera with rates of 50 to 100 Hz will also be implemented. The project will initially focus on detecting umpire's gestures, but future enhancements may include using multiple cameras to detect the umpire in 360 degrees for improved gesture recognition and adding unique gestures for actions that currently do not have gestures. With enough funding, the project aims to detect non-gesture-based actions such as boundary detection for run out, four, and six runs. Overall, the project aims to improve the accuracy and efficiency of updating the score board in cricket matches using machine learning and artificial intelligence techniques.

5.3 System Architecture

In Figure 5.1, A dataset is there which has umpire's gestures. Open CV is used to save the umpire signs in a particular directory. The captured dataset is now processed for data preprocessing where the noise is removed and passed for features extraction. The extracted features will be the input to the deep leaning model to get trained. Once the model is trained, it is saved for future prediction. When the umpire shows the sign, captured image will be passed to the train model for prediction. Based on the predicted signs, the scoreboard will be updated.

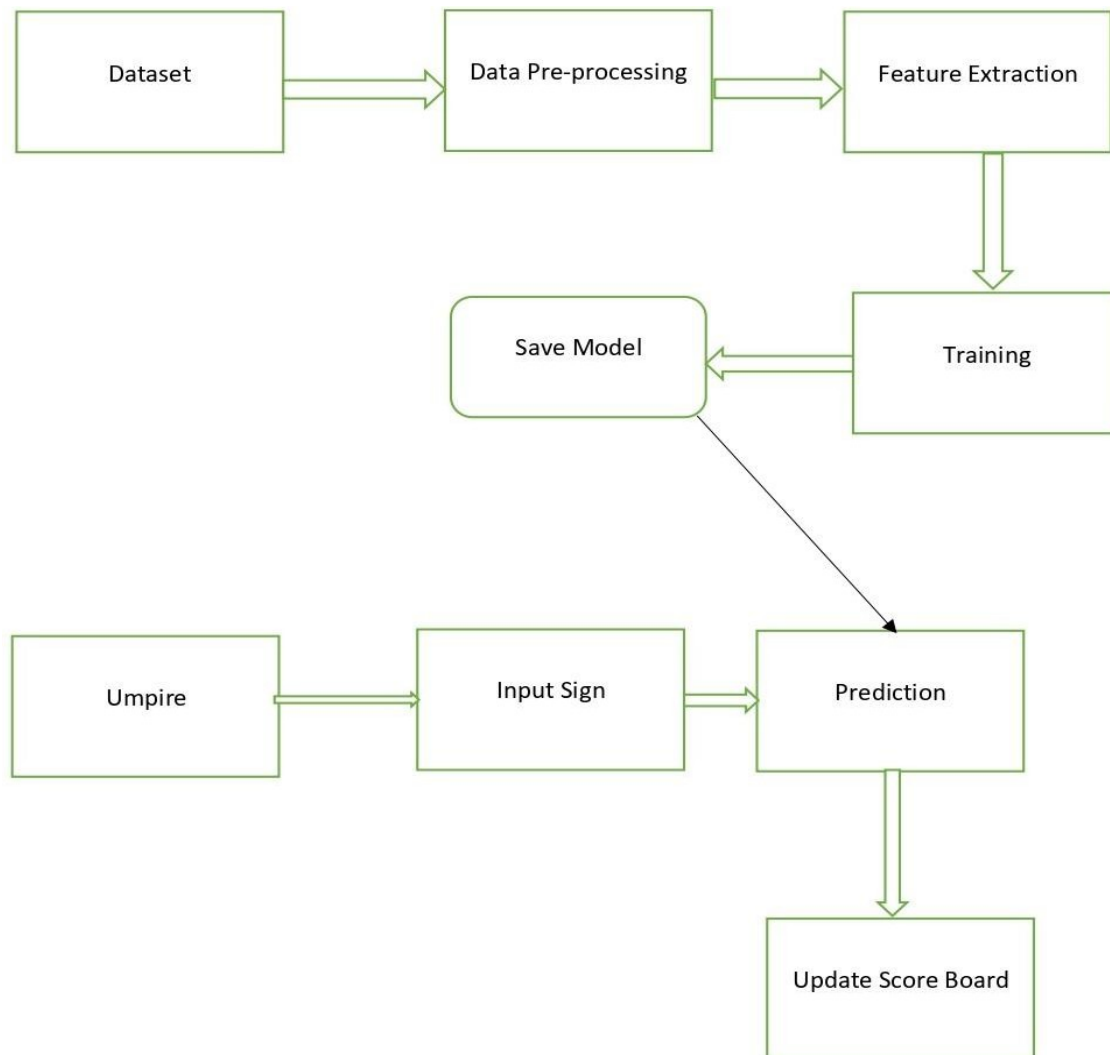


Fig.5.1 System Architecture

5.4 Module Description

A module description provides detailed information about the modules.

- Create Dataset of Umpire and Gestures
- Preprocessing Gesture dataset
- Train and test gesture dataset
- Preprocessing Umpire dataset
- Train and test umpire dataset
- Capturing and detecting Umpire
- Detecting Hand Gesture of Umpire

5.4.1 Create Dataset of Umpire and Gestures:

This Python module is designed to continuously capture images from a camera until a specific number of images is reached. It utilizes the OpenCV (cv2) module for capturing and displaying images from the camera and the numpy module for manipulating image data. The captured images are saved as JPEG files in a designated folder path.

The program starts by importing the necessary modules, including os, time, cv2, and numpy. Then, it creates a Video Capture object that captures images from the default camera (device index 0). It initializes some variables, such as pic_no (the captured image number), total_pic (the total number of images to capture), flag capturing (a boolean variable indicating whether to capture images), and path (the folder path to save the images).

The program enters a while loop that reads frames from the camera using the Video Capture object's read () method. It flips the frames horizontally using the cv2.flip() function, adds a green rectangle to the frame using the cv2.rectangle() function, and displays the frame using the cv2.imshow() function. The while loop also checks for a keypress using the cv2.waitKey() function. If the key 'c' is pressed, the program increments the pic_no variable, crops the frame using NumPy array.

5.4.2 Preprocessing Gesture dataset

This process involves applying various techniques to a collection of hand gesture images to enhance their quality and prepare them for further analysis or use. First, it imports the necessary modules such as os, time, cv2, numpy, and matplotlib.pyplot. Then, it defines the folder path where the dataset is located and the folder path where the processed images will be saved. Next, the script reads the list of gestures from the dataset folder using the os.listdir() function and prints them. It then enters a for loop that iterates over each gesture folder in the dataset, creates a new folder with the same name in the processed image folder using the os.mkdir() function, reads the list of images in the gesture folder, and enters another for loop that iterates over each image in the folder. For each image, the script reads the image using the cv2.imread() function, converts it to grayscale using cv2.cvtColor() function, applies Otsu's thresholding method using cv2.threshold() function, resizes the image to 50x50 pixels using cv2.resize() function, and saves the processed image to the new folder using cv2.imwrite() function. Finally, the script summarizes the preprocessing techniques applied on the dataset images by printing the gesture name and image name for each processed image.

5.4.3 Train and test gesture dataset

This module performs image classification using the DenseNet121 deep learning model. It starts by preparing the dataset and its subsets, splitting them into training, validation, and test sets. The module generates batches of augmented and preprocessed images for training and validation. It then constructs a modified DenseNet121 model by freezing the base layers and adding additional layers for classification. The model is compiled with appropriate optimizer, loss function, and evaluation metrics. Class weights are computed to address class imbalance, and the model is trained using the training and validation data. The module provides functions to visualize the training history, evaluate the model's performance on the test set, and generate a confusion matrix. It also includes a function for making predictions on new images using the model.

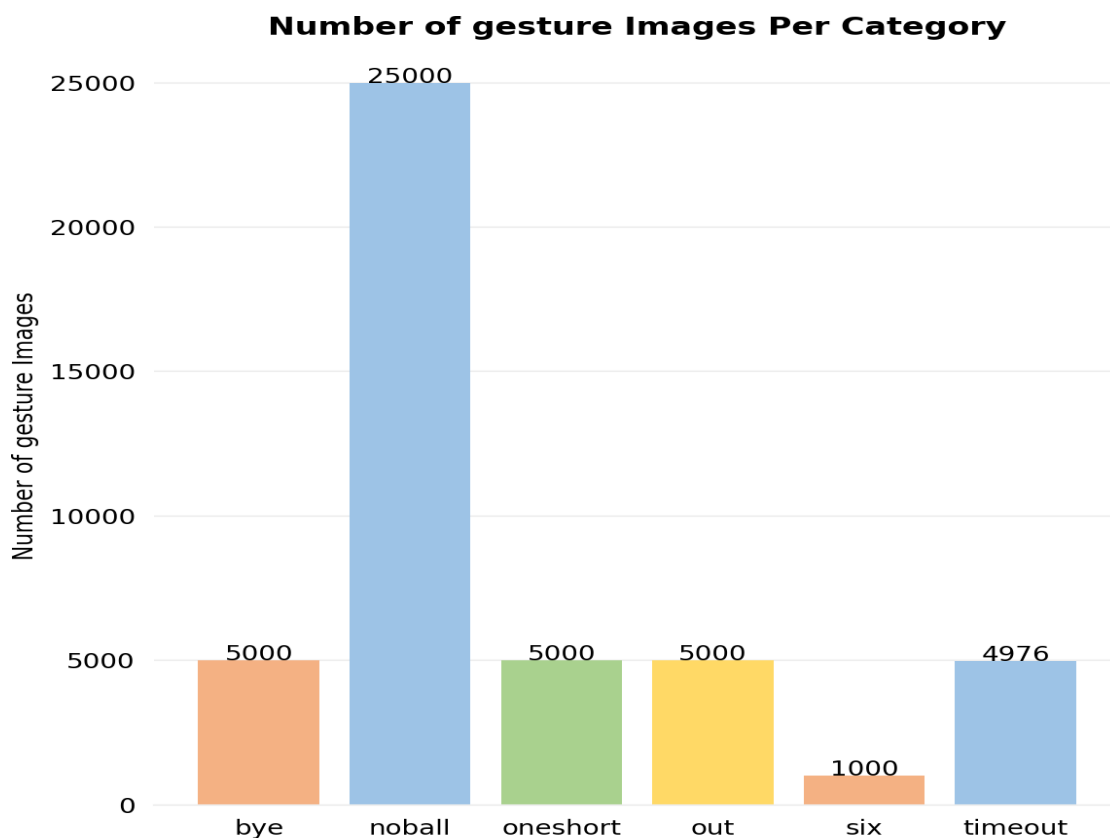


Fig 5.2: Number of images per category

A bar chart is generated to visualize the distribution of images among subdirectories. It begins by setting up the necessary variables, including x-coordinates and colors for the bars. The code then creates a figure and axes with specific formatting options.

The bar chart is plotted using the x-coordinates and the corresponding number of images. The appearance of the plot is adjusted by removing spines, adding grid lines, and customizing the axes. Subdirectory names are displayed as x-axis labels. Text labels representing the number of images are placed on top of each bar. The plot is then adjusted for a tight layout and saved as an image file. Finally, the plot is displayed.

Percent Distribution of gesture Across Categories

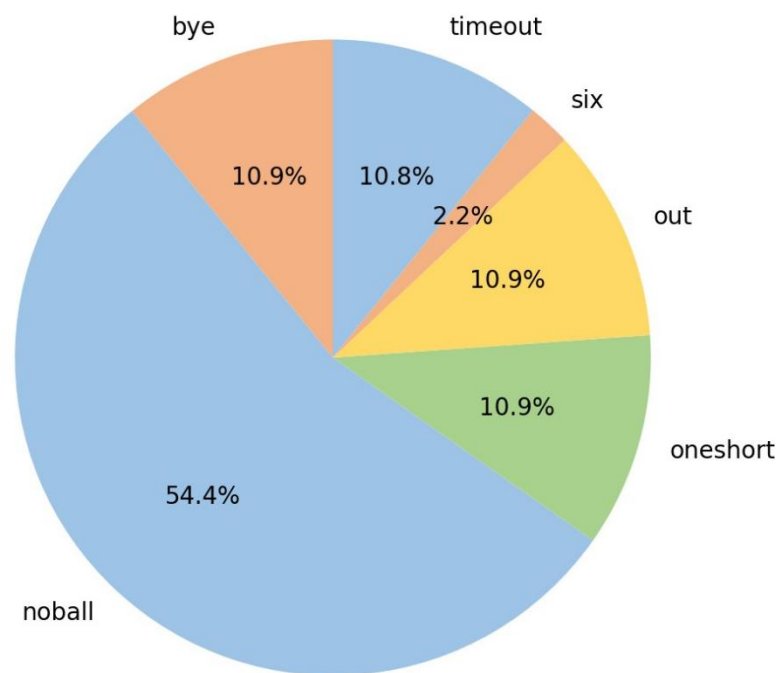


Fig 5.3: Percent Distribution of gesture Across Categories

A pie chart is generated to visualize the percentage distribution of images across subdirectories. It uses `plt.pie()` with `num_of_images` as values, `dir_list` as labels, colors for slice colors, `autopct='%1f%%'` to display percentages, and `startangle=90` to set the initial angle. The chart is given a title using `plt.title()` and saved as "pie_chart.png" with `plt.savefig()`. Finally, `plt.show()` displays the chart. This pie chart is representing the distribution of images across subdirectories by percentage.



Fig 5.4: Random images

The random images are displayed, with one image representing each category. It begins by creating a figure with a white background and dimensions of 12x3. The code then iterates through a loop six times to create subplots for each category. Within each iteration, an axes object is created for the current subplot. Randomly, an image path is selected from the corresponding category using the `random.choice()` function in combination with `glob()`. The selected image is opened using `Image.open()` and displayed in grayscale using `plt.imshow()`. The title of the subplot is set to the category name, and axis ticks and labels are turned off. The resulting figure is saved as an image file named "random_images.png" with a resolution of 200 dpi using `plt.savefig()`. Finally, the figure is displayed using `plt.show()`.

```

Model: "sequential"
-----
Layer (type)                Output Shape         Param #
-----
densenet121 (Functional)    (None, 3, 3, 1024)  7037504
flatten (Flatten)           (None, 9216)         0
batch_normalization (BatchN (None, 9216)         36864
ormalization)
dense (Dense)                (None, 1024)         9438208
dropout (Dropout)           (None, 1024)         0
batch_normalization_1 (Batc (None, 1024)         4096
hNormalization)
dense_1 (Dense)              (None, 512)          524800
dropout_1 (Dropout)          (None, 512)          0
batch_normalization_2 (Batc (None, 512)          2048
hNormalization)
...
Total params: 17,176,390
Trainable params: 10,117,382
Non-trainable params: 7,059,008

```

Fig 5.5: DenseNet121 model

A modified DenseNet121 model is created using a sequential approach. It adds layers to process the input data, including batch normalization and dropout layers for improved performance and prevention of overfitting. Fully connected layers with non-linear activations are used to learn high-level representations and make predictions. The model incorporates a pre-trained base model, DenseNet121, and the final layer outputs class probabilities. Overall, the modified model benefits from the pre-trained base model and employs various techniques to enhance its performance in image classification tasks. The final layer of the model is a dense layer with a softmax activation function. This layer outputs probabilities for each class, allowing the model to make predictions on the input images. The number of units in this layer corresponds to the number of classes in the classification task.

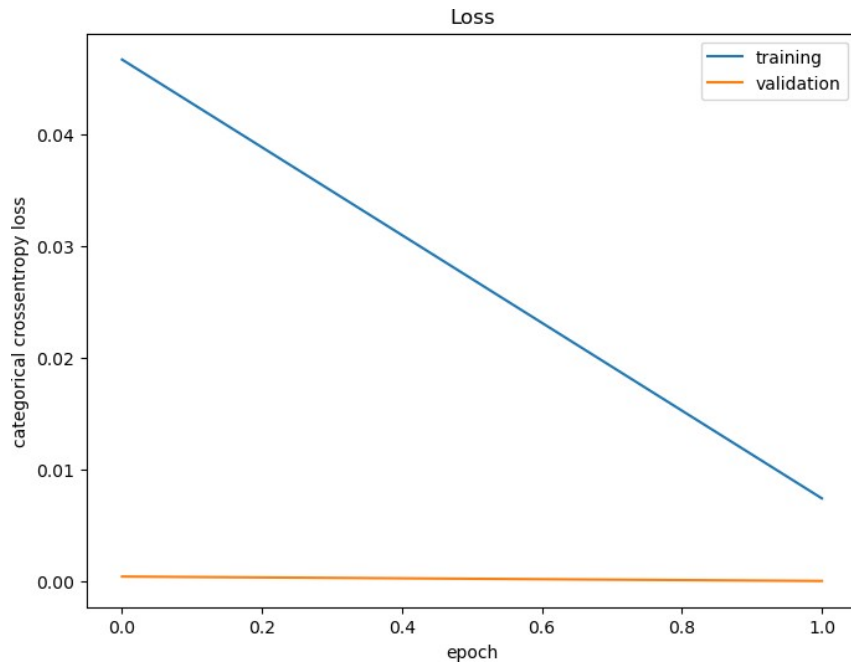


Fig 5.6: Loss history

The graph visualizes the loss history of a model during training. It generates a plot that shows the training loss and validation loss over epochs. The plot helps monitor the model's convergence and generalization performance, providing insights into its learning progress. By comparing the training and validation loss trends, potential issues like overfitting or underfitting can be identified. Overall, the plot serves as a valuable tool for assessing and optimizing the model's training process.

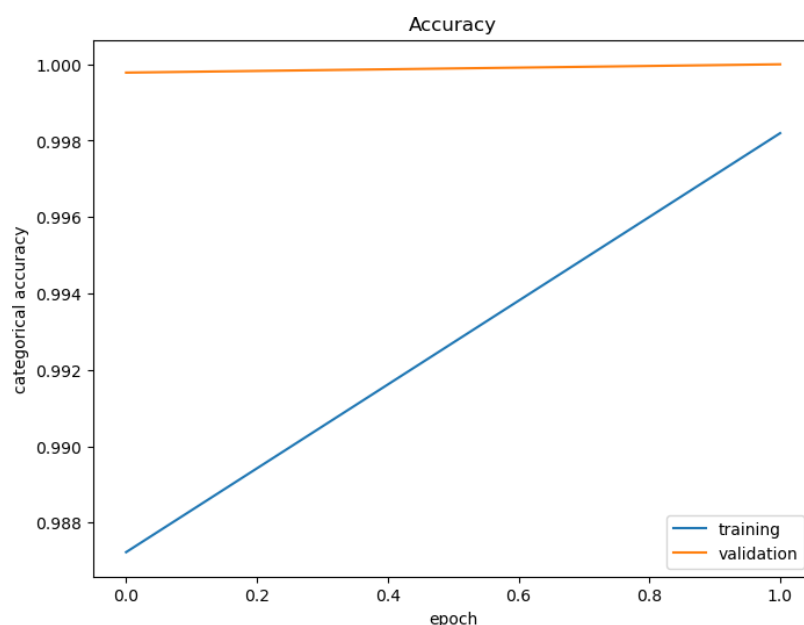


Fig 5.7: Visualizing accuracy history

the graph illustrates the accuracy history of the model during training. This visualization helps

monitor the model's performance in terms of correctly predicting the categorical labels for both the training and validation datasets. By observing the trends in accuracy over epochs, one can gain insights into the model's learning progress and its ability to generalize to unseen data.

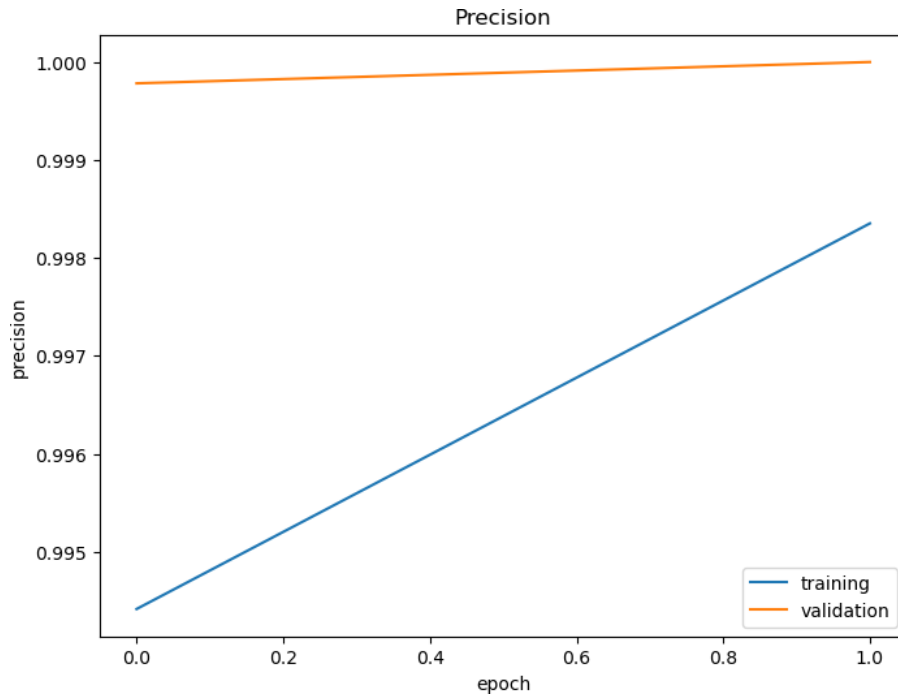


Fig 5.8: Visualizing precision history

The graph visualizes the precision history of a model during training. It plots the training and validation precision over epochs, providing insights into the model's ability to make accurate positive predictions. The plot helps monitor the model's precision performance and assess its overall effectiveness in tasks that require precision.

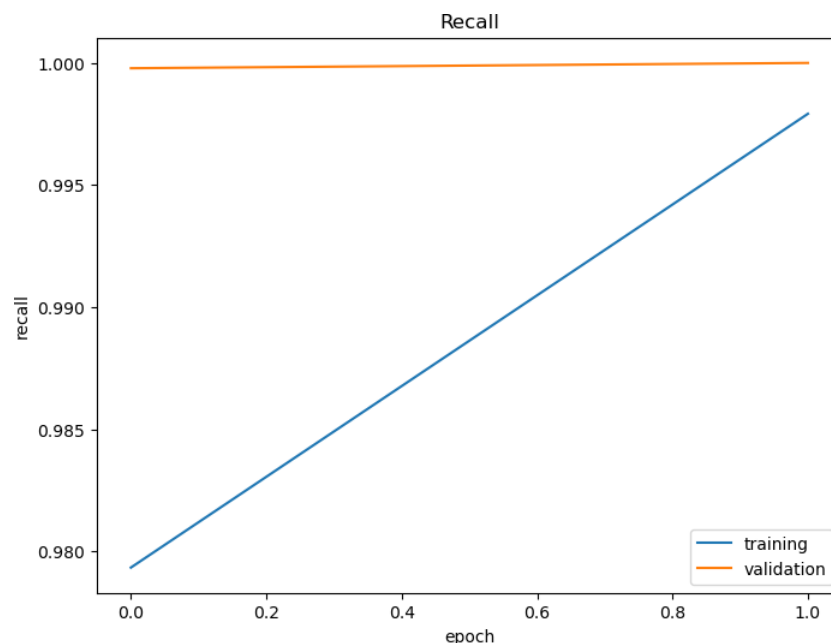


Fig 5.9: Visualizing Recall history

The graph generates a plot visualizing the recall history of a model during training. It shows the training and validation recall over epochs, indicating the model's ability to correctly identify positive samples. The plot helps assess the model's performance in tasks requiring high recall, such as anomaly detection or medical diagnosis.

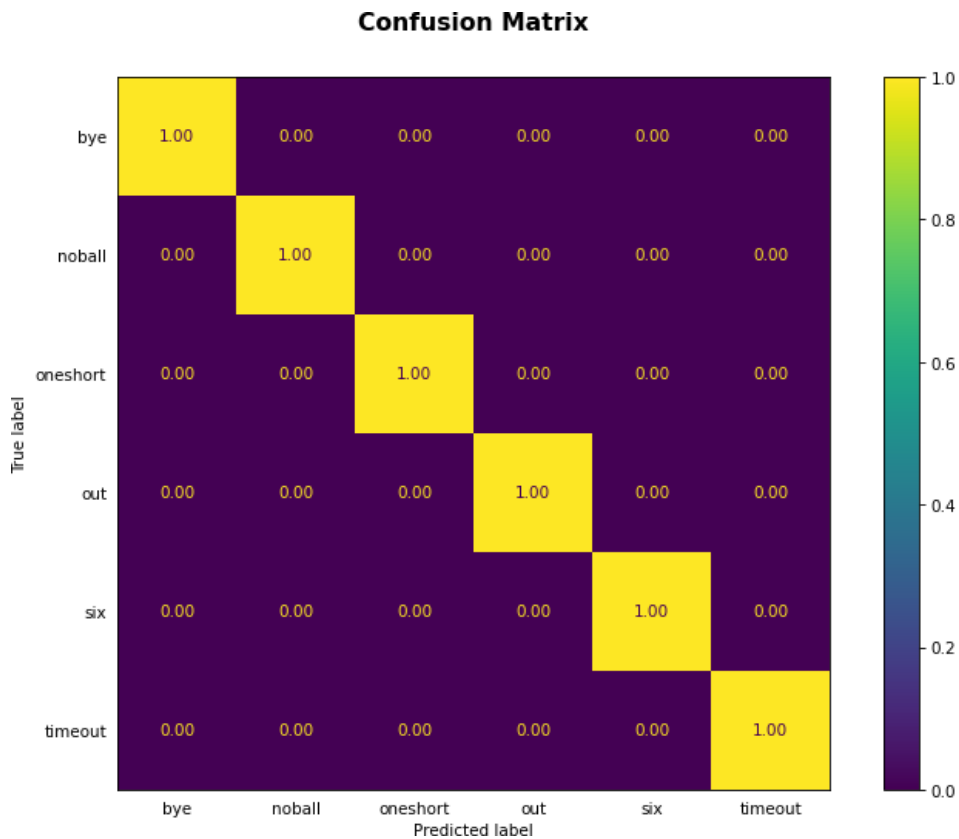


Fig 5.10: Confusion Matrix

The confusion matrix is generated based on true and predicted labels using the ConfusionMatrixDisplay class. The plot displays labels, includes numerical values, and applies normalization. It uses a specified colormap and customizes tick parameters and title. The plot is saved as an image and displayed. The confusion matrix helps evaluate classification model performance by visualizing true vs. predicted labels.

5.4.4 Preprocessing Umpire dataset:

This module provides functionality for preprocessing a dataset of hand gesture images. It includes functions for reading, converting to grayscale, thresholding, resizing, and saving images. It takes the input and output directory paths as arguments. The module retrieves a list of all the subdirectories, each representing a hand gesture class, from the input directory. It then creates a new directory with the same name in the output directory to store the preprocessed images. For each image in the subdirectory, the module applies a binary thresholding technique using the OTSU algorithm to separate the foreground hand gesture from the background. The

resulting binary image is then resized to a 50x50 pixel image. The module saves the resized image to the corresponding subdirectory in the output directory. The output directory will contain the preprocessed binary images that can be used for further analysis or machine learning models. This module is useful for preprocessing images before feeding them into a machine learning model for hand gesture recognition.

5.4.5 Train and test umpire dataset:

This module performs various tasks related to image classification. It imports necessary libraries such as `os`, `random`, `glob`, `matplotlib.pyplot`, `preprocessing Image`. They also define various variables such as `data_dir`, `train_dir`, `valid_dir`, and `test_dir` that store the paths to the directories containing the image data.

The module then uses the `os` module to list all the files and directories in the `data/train/` directory and counts the number of images in each subdirectory. It also creates a bar chart and a pie chart that visualize the distribution of images across the subdirectories. They use the `PIL` library to display a random image from the dataset. It also sets a random seed using the `random.seed()` function to ensure that the results are reproducible. They define a function that creates a sequential model using the `DenseNet121` architecture. It also sets up an image data generator that performs various image preprocessing techniques such as rescaling, horizontal flipping, and rotation. The module then trains the model using the `fit_generator()` method and evaluates it.

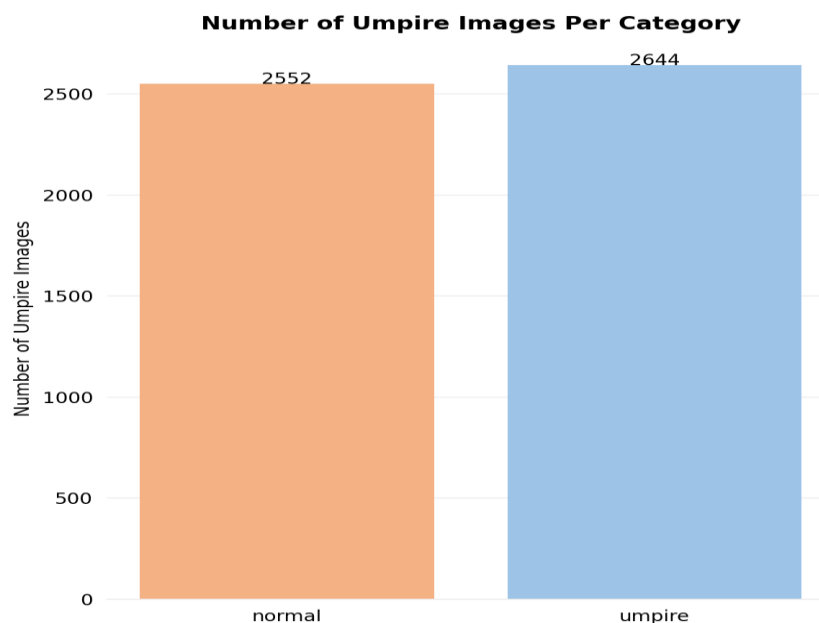


Fig 5.11: Number of umpire images per category

This bar graph represents the distribution of images across two categories: "normal" and "umpire". The category "umpire" is associated with images that depict umpires, while the category "normal" includes all other types of images. The height of each bar in the graph corresponds to the number of images in the respective category.

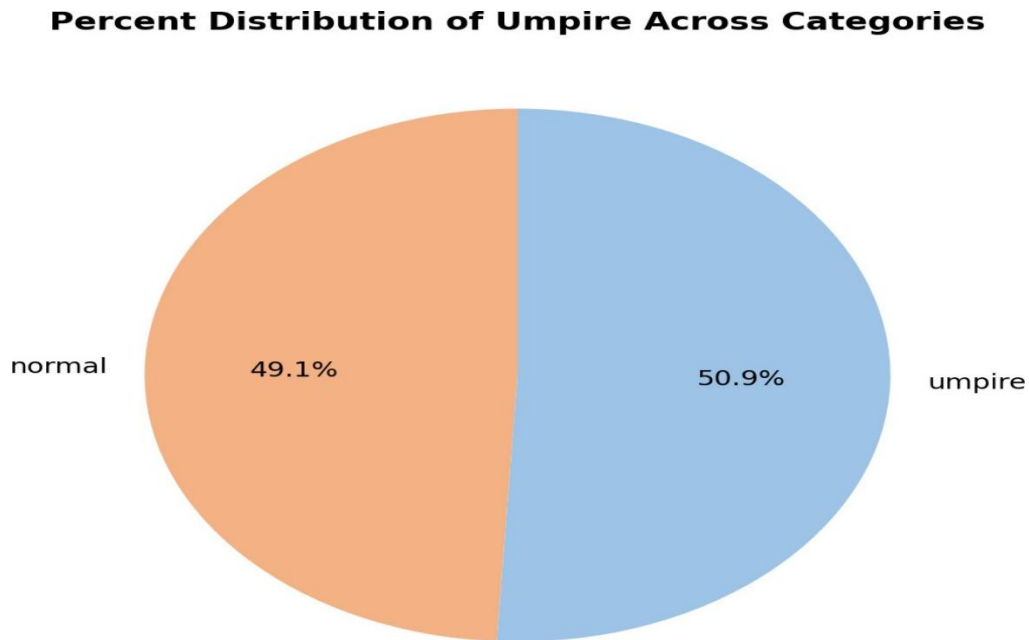


Fig 5.12: Percent Distribution of umpire across categories

The dataset contains two categories of individuals, namely "Normal" and "Umpire". Upon analysis, it was found that 50.9% of the individuals in the dataset were umpires, while the remaining 49.1% were normal individuals such as spectators or coordinators. This distribution can be visually represented through a pie chart, where the umpires' percentage is depicted by one section and the normal individuals' percentage by the other section.



Fig 5.13: Normal and Umpire random image

This figure displays random images from different categories in a given dataset. The figure has a size of 12x3 and a white background. Two directories are selected, and a random image is chosen from each directory using the 'random.choice' and 'glob' functions. Each image is displayed in a subplot with a title that corresponds to the name of the directory, and the axis labels are turned off. The 'imshow' function is used to display the image. The resulting figure provides a sample of images from different categories in the dataset and allows the user to visually inspect them.

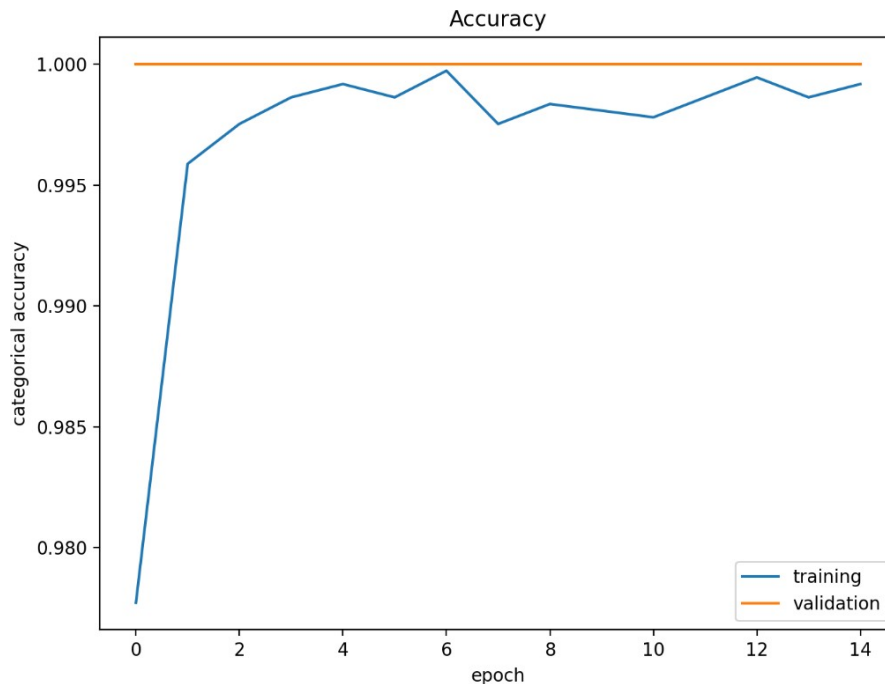


Fig 5.14: Categorical Accuracy

The code snippet generates a line plot that visualizes the accuracy history of a neural network model during training. The plot shows how the training accuracy and validation accuracy change as the number of training epochs increases. The blue line represents the training accuracy, which indicates how well the model performs on the training data. The orange line represents the validation accuracy, which measures the model's performance on a separate validation dataset. By observing the trends and patterns in the plot, one can gain insights into the model's learning progress and its ability to generalize to unseen data. Deviations or discrepancies between the training and validation accuracy can indicate issues such as overfitting (when the training accuracy is significantly higher than the validation accuracy) or underfitting (when both accuracies are low). Analyzing the accuracy history plot can help in making informed decisions about model training, tuning hyperparameters, or adjusting the training strategy to improve the model's overall performance.

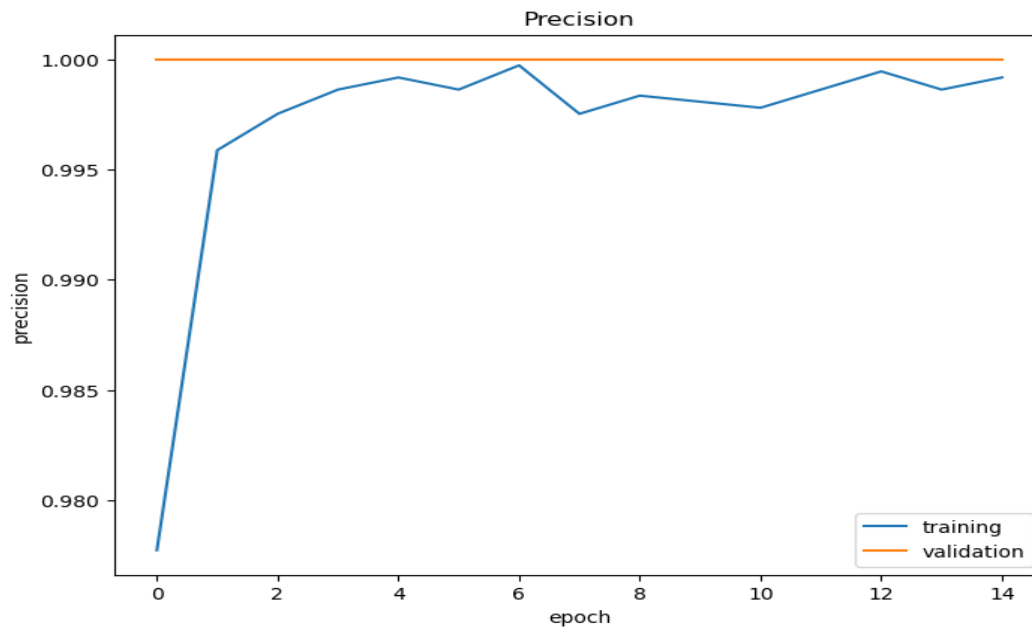


Fig 5.15: Precision

The plotted graph displays the precision history for a neural network model during training. It plots the training precision and validation precision against the number of training epochs. The resulting graph can be used to evaluate the model's performance in terms of precision over time and to compare the performance of different models or hyper parameters.

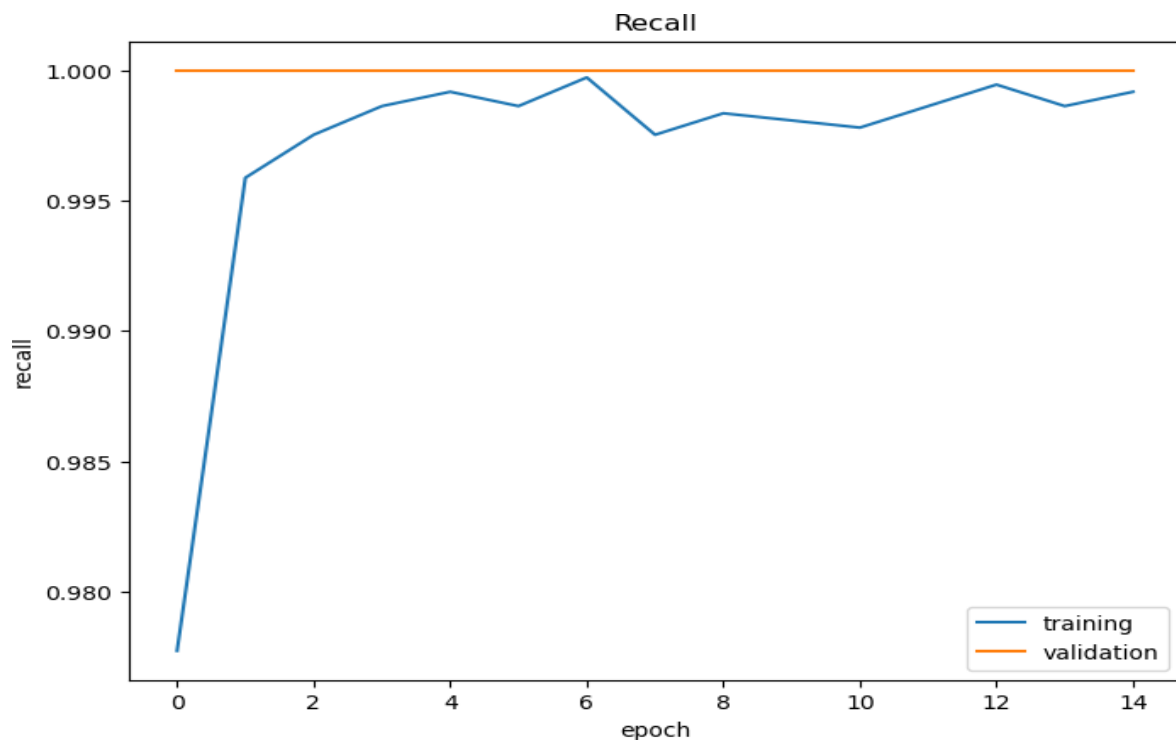


Fig 5.16: Recall

The plotted graph displays the recall history of a neural network model during training. The

resulting plot displays the training and validation recall as a function of the number of training epochs, with the training recall plotted in blue and the validation recall plotted in orange. The plot allows the user to easily visualize the performance of the model over time with respect to recall and identify any issues. This can be useful for comparing the performance of different models or hyper parameters.



Fig 5.17: Confusion Matrix

This generates a confusion matrix plot to evaluate the performance of a classification model using the true and predicted labels. The resulting grid of squares represents the number of samples that fall into each category. The diagonal of the matrix represents the correct predictions, while off-diagonal entries represent misclassifications. The plot allows the user to visualize the performance of the model and identify areas for improvement.

5.4.6 Capturing and detecting Umpire:

The image classification module uses a pre-trained DenseNet121 model from the Keras library to predict the class of an input image. The module provides a function called "predict" that takes an image path as input and returns the predicted class label of the image. The function first loads the input image using the Keras utility function, resizes it to (100, 100) pixels, and then converts it to a NumPy array. The array is then normalized by dividing it by 255. The pre-trained model is used to predict the class label of the image, which is returned by the function.

The module also provides another function called "capture_image" that captures an image using the default camera (usually a USB webcam) and saves it as "img.jpg" in the current directory.

The module then calls the "predict" function on the captured image, and displays the predicted class label along with the image using Matplotlib. The module also writes the predicted class label to a text file called "output.txt". To improve the accuracy of the model, the module uses data augmentation techniques such as horizontal flipping, rotation, and zooming, while training the model. The training data is split into training and validation sets using the SplitFolders library. The model is trained using the Adam optimizer and categorical cross-entropy loss. The module also computes the class weights to handle class imbalance in the training data.

5.4.7 Detecting Hand Gesture of Umpire:

The provided code implements a live camera application that performs gesture recognition for Indian sign language. The code utilizes the OpenCV library to capture frames from the camera and process them for gesture recognition. The application uses a pre-trained model, specifically a DenseNet121 model, which is loaded from the file 'densenet121'.

The main loop of the application continuously captures frames from the camera and performs the following steps for each frame:

- **Preprocessing:** The captured frame is resized and preprocessed to prepare it for prediction. Various image processing techniques such as grayscale conversion and thresholding are applied to extract the hand gesture region.
- **Gesture Prediction:** The preprocessed image of the hand gesture is fed into the pre-trained model for prediction. The model predicts the gesture class based on the input image.
- **Score and Wicket Tracking:** The application keeps track of the score and wicket count based on the predicted gestures. If a particular gesture is recognized, such as 'out', 'bye', 'noball', 'oneshort', 'six', or 'timeout', the corresponding score or wicket count is updated.
- **Display:** The application displays the processed frames with a bounding box indicating the region of interest for gesture recognition. It also shows the current score and wicket count on the screen.

The application allows the user to trigger the gesture recognition by pressing the 'c' key and exit the application by pressing the 'q' key.

5.5 Data Flow Diagram

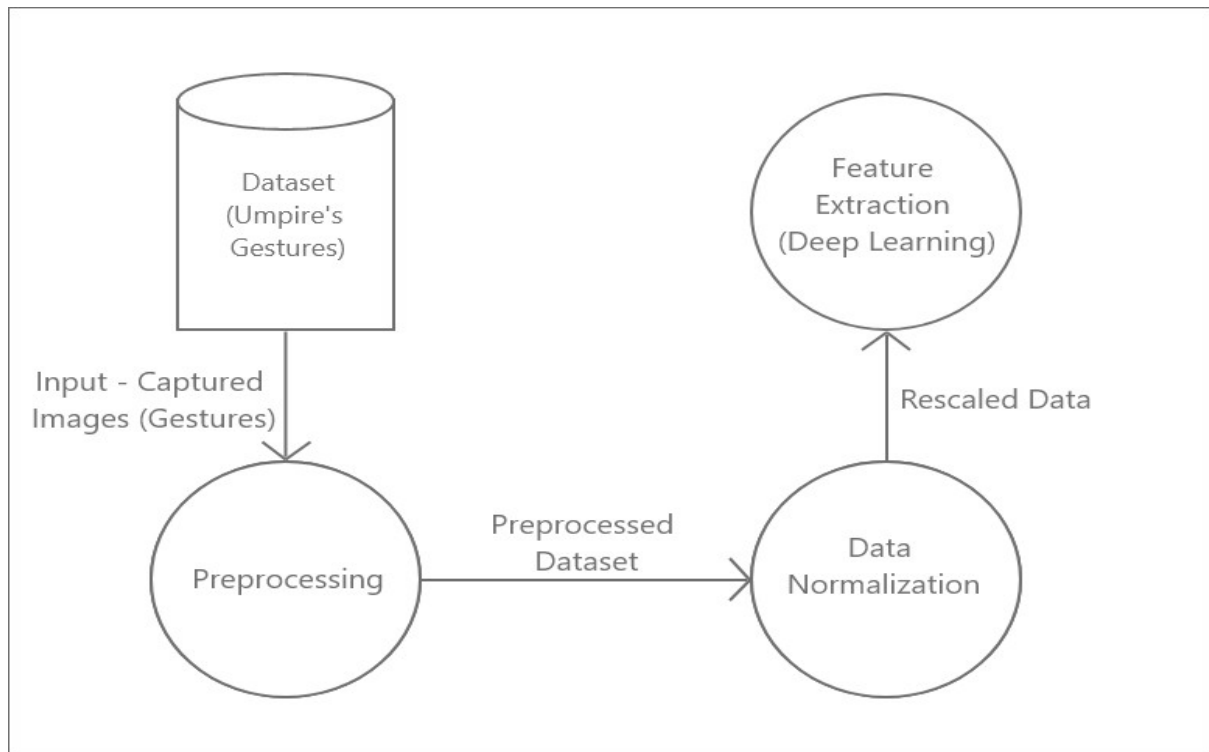


Fig 5.18: Data Flow Diagram (Level 1)

In this fig, A Dataset which has images of the umpire hand signal are Preprocessed using OpenCV and after that data normalization is applied. Once the data is normalized feature extraction technique is used.

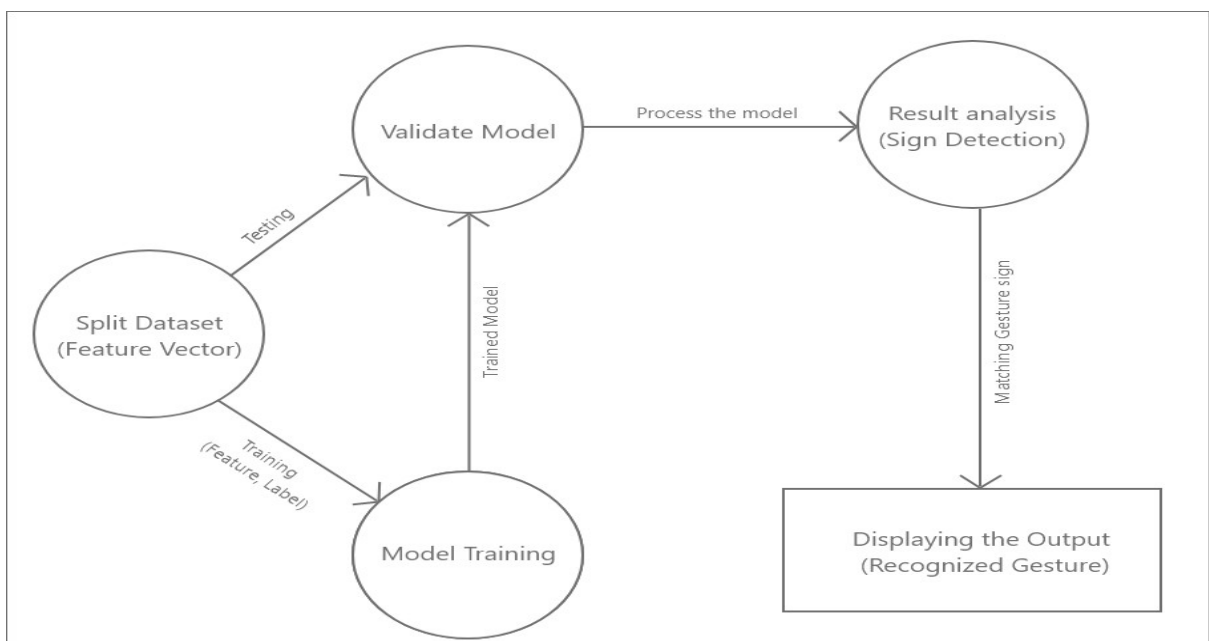


Fig.5.19: Data Flow Diagram (Level 2)

In this fig, The dataset is splitted into training and testing. The training dataset is passed to the deep learning algorithm for training. Then the trained model goes for validation where testing data is also sent. Once the model is Validated, it is processed for Detecting the signs. The matched gesture sign which are recognized is then displayed.

5.6 Use case Diagram:

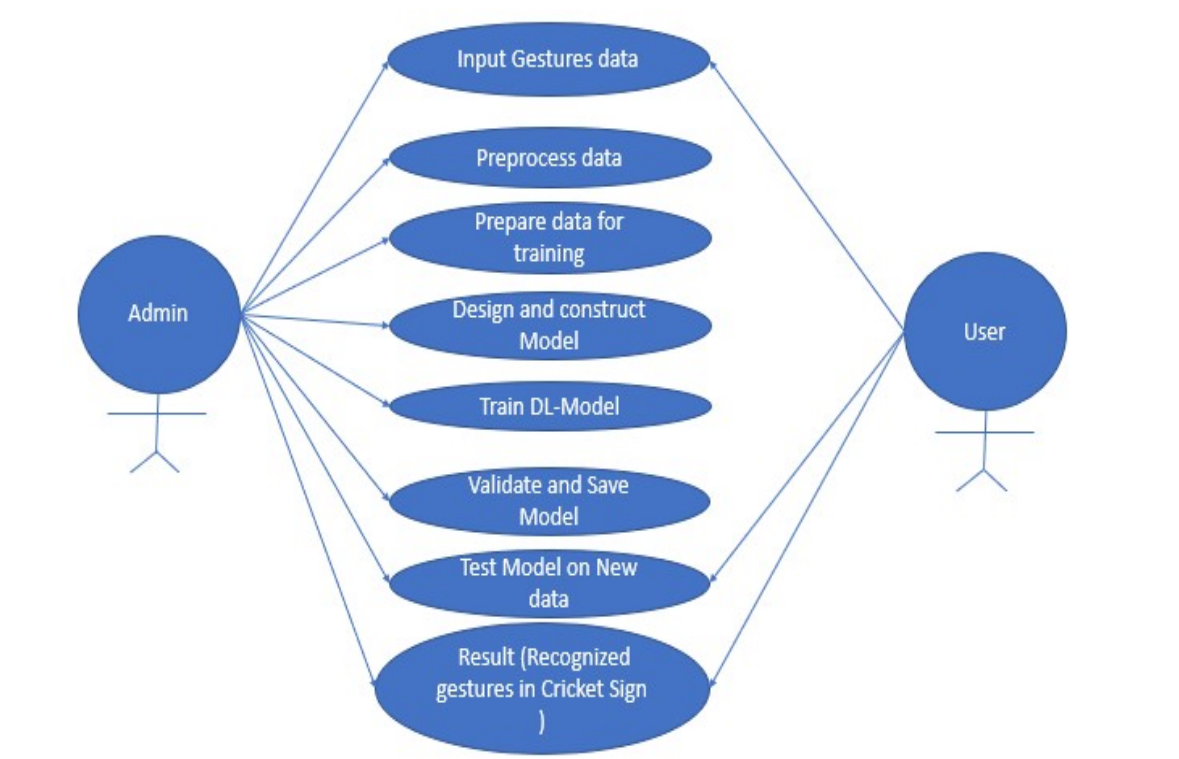


Fig 5.20: Use Case Diagram

This use case diagram illustrates the various tasks that an admin can perform in order to develop and deploy a gesture detection system. The first step is to preprocess the input data using OpenCV and then normalize it for further analysis. The preprocessed data is then subjected to feature extraction techniques in order to obtain relevant information for training the model. Once the data is ready, the admin can design and construct a suitable neural network model and train it using the prepared data. The model is validated using test data to ensure its accuracy and avoid overfitting. If the validation is successful, the admin saves the final model for future use. The system is capable of detecting umpire gestures, and when a user inputs a gesture, the system matches it with the output from the trained model and displays the matched gesture on the screen. This process allows the user to easily identify the detected gesture and take the appropriate action.

5.7 Sequence Diagram

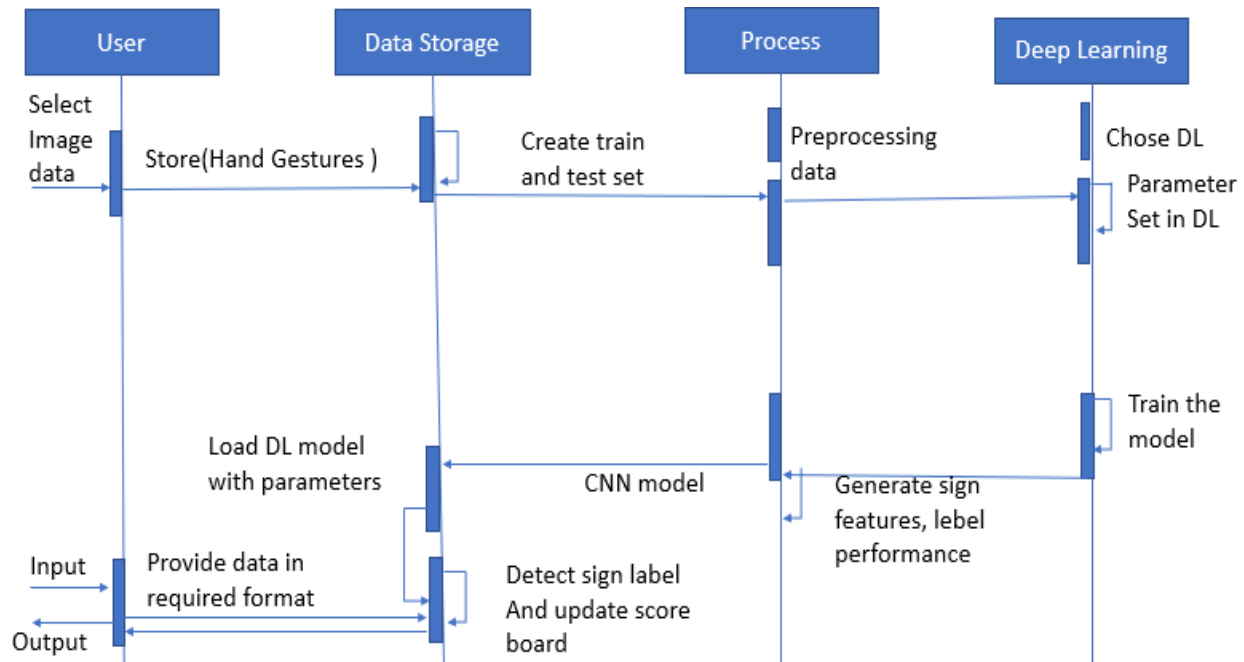


Fig 5.21: Sequence Diagram

Here in this fig, First the user selects the image data for the gesture to be detected. This data is then stored and sent for both training and testing.

Once the data is prepared, preprocessing occurs before deep learning is applied. The admin chooses a deep learning method and sets the required parameters. The model is then trained using the training data. A convolutional neural network (CNN) model is used to detect the sign label, and the score board is updated accordingly.

After training, the deep learning parameters are loaded and the data is provided in the required format. Finally, the system detects the umpire gesture and updates the score board based on the user input.

CHAPTER 6

SYSTEM TESTING

6.1 INTRODUCTION

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration testing. System testing is based on process description and flows, emphasizing pre-driver process and integration points.

6.2 TEST CASES

Table 6.1 Test Cases Specifications of Gestures

TEST CASE ID	INPUT DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	REMARKS
1.	Capturing the Dataset	Capturing the dataset includes creating a list of images of gestures and umpire which is done by capturing each frame of video and saving it as jpg file of 50x50 pixel size.	Same as expected	Successful
2.	Preprocessing the gestures images.	Pre-processing is done by converting the video to a sequence of RGB frames. Each frame having the same dimensions. Skin color segmentation used to extract skin region with HSV.	Same as expected	Successful

TEST CASE ID	INPUT DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	REMARKS
3.	Segmentation	Filtered image as input. Grey to binary image. Image morphological operation.	Same as expected	Successful
4	Feature extraction	We proposed the CNN model extracted temporal features from the frames which was used further to predict gestures based on sequence of frames.	Same as expected	Successful
5	Training	Train deep learning model with feature matrix of gestures and umpire. Saved trained model with 'CNN_MODEL.H5' which is of gestures dataset and 'densenet121.H5' which is of umpire dataset and 'labels.pkl'.	Same as expected	Successful
6	Classification using CNN model	Input RGB images Pass features to trained deep learning model	Same as expected	Successful
7	Sign Recognition And umpire detection	Our process is able to predict in real-time using image frames captured from a web camera and the Softmax layer's predictions. When the live camera feed starts, the process detects and recognizes the umpire first and then	Same as expected	Successful

		signs in real-time, and the results are displayed.		
--	--	--	--	--

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

The goal of testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality. Test cases and results are shown in the Tables.

The Table 6.1. gives a brief overview of the testing process conducted for the Gestures. This was carried out to test and determine whether or not the project can perform under certain constraints. If the project clears the test at each particular level the project is deemed a success. Not all the tests gave the immediate green signals. The test was to be run over and over correcting the error found during each try. If the project never crossed a certain test case that case will have to be repeated and the errors rectified. If errors are too much and are not able to cross that test case that means completing the project is not succeeded and still lots of work is to be done.

System testing is important because of the following reasons:

- System testing is a step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 INTRODUCTION

A new method for identifying and recognizing Umpire Action and Non-Action Gestures in cricket matches has been presented. The method uses a Histogram Oriented Gradient (HOG) feature extraction approach and a Non-Linear Support Vector Machine (NL-SVM) classifier. The proposed method achieves a high level of accuracy, sensitivity, specificity, and precision, indicating its superiority over existing methods. The dataset, focused on tasks such as Non-Umpire Detection, Umpire Detection, Umpire Segmentation and Classification which was manually created and is of over 1 lakh images. By adding additional Umpire Action and Non-Action images to the dataset, the classifier's accuracy was further improved. The proposed method has significant potential in designing intelligent and efficient human-computer interfaces for sports gesture analysis, video summarization, and automatic sports highlights generation in the game of cricket.

7.2 SCREEN SHOTS

```

C:\Windows\System32\cmd.exe - jupyter notebook
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ACER\Downloads\umpire\umpire>jupyter notebook
[I 02:28:22.506 NotebookApp] Serving notebooks from local directory: C:\Users\ACER\Downloads\umpire\umpire
[I 02:28:22.506 NotebookApp] Jupyter Notebook 6.5.1 is running at:
[I 02:28:22.514 NotebookApp] http://localhost:8888/?token=c0c631d9e09fd6c1d1b23fe098df5bf3b078c35f3e5f51fd
[I 02:28:22.514 NotebookApp] or http://127.0.0.1:8888/?token=c0c631d9e09fd6c1d1b23fe098df5bf3b078c35f3e5f51fd
[I 02:28:22.514 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 02:28:22.722 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/ACER/AppData/Roaming/jupyter/runtime/nbserver-12784-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=c0c631d9e09fd6c1d1b23fe098df5bf3b078c35f3e5f51fd
or http://127.0.0.1:8888/?token=c0c631d9e09fd6c1d1b23fe098df5bf3b078c35f3e5f51fd
[E 02:28:25.479 NotebookApp] Could not open static file ''
[W 02:28:26.531 NotebookApp] 404 GET /static/components/jquery-ui/themes/smoothness/jquery-ui.min.css (:::1) 103.990000ms
referer=http://localhost:8888/tree?token=c0c631d9e09fd6c1d1b23fe098df5bf3b078c35f3e5f51fd
[W 02:28:54.287 NotebookApp] 404 GET /static/components/jquery-ui/themes/smoothness/jquery-ui.min.css (:::1) 23.990000ms
referer=http://localhost:8888/notebooks/module2_resize_images.ipynb
[W 02:28:54.519 NotebookApp] 404 GET /static/components/jquery-ui/themes/smoothness/jquery-ui.min.css (:::1) 39.990000ms
referer=http://localhost:8888/notebooks/module3_Training.ipynb
[W 02:28:56.612 NotebookApp] Notebook module3_Training.ipynb is not trusted
[W 02:28:57.242 NotebookApp] Notebook module2_resize_images.ipynb is not trusted
[I 02:28:58.106 NotebookApp] Kernel started: b63599c5-7b18-4f5f-bb3a-78ef30cec836, name: python3
[W 02:29:00.655 NotebookApp] 404 GET /static/components/jquery-ui/themes/smoothness/jquery-ui.min.css (:::1) 8.000000ms
referer=http://localhost:8888/edit/module4_cap.py
[I 02:29:00.767 NotebookApp] Kernel started: 8df98d83-6588-4684-9dab-5601d2945b98, name: python3
[W 02:29:00.831 NotebookApp] 404 GET /static/components/jquery-ui/themes/smoothness/jquery-ui.min.css (:::1) 0.000000ms

```

Fig 7.1 Anaconda Command Prompt

The command to run the code for the detection of Umpire Gesture is given in the command prompt after changing the directory as shown in Fig 7.1.

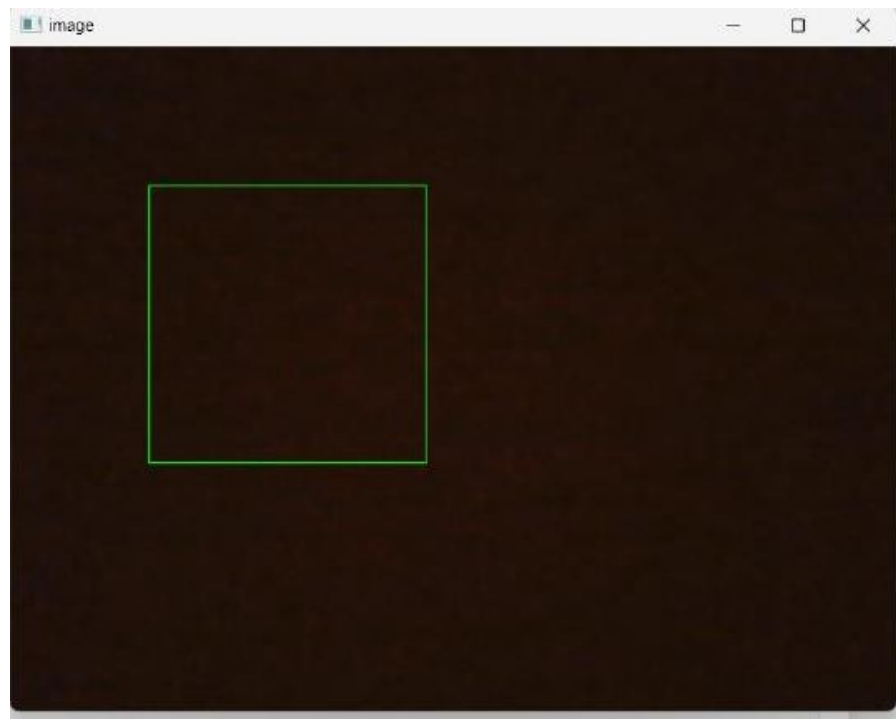


Fig 7.2 Camera feed to capture the dataset

The Camera feed is used to capture the image to create the dataset as shown in Fig 7.2 and it is also used to identify whether it's umpire or non-umpire.

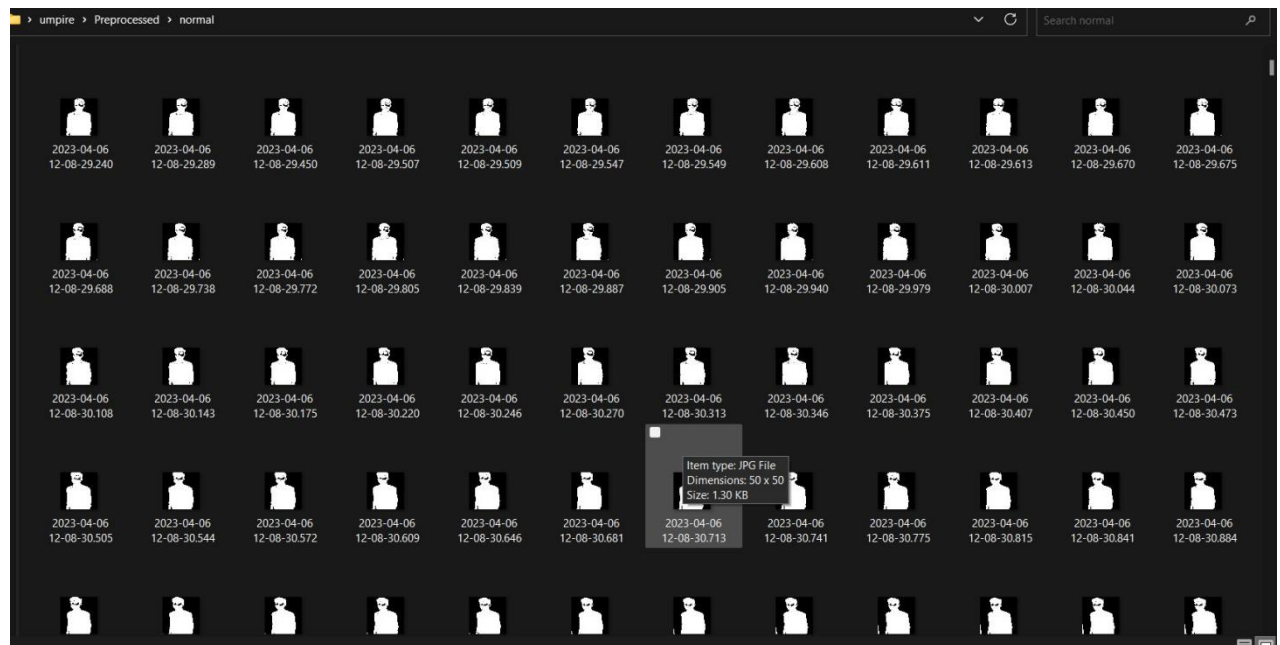


Fig 7.3 Preprocessed Dataset

The above shown Fig 7.3 describe the dataset we have created by camera feed to identify the umpire hand gesture.

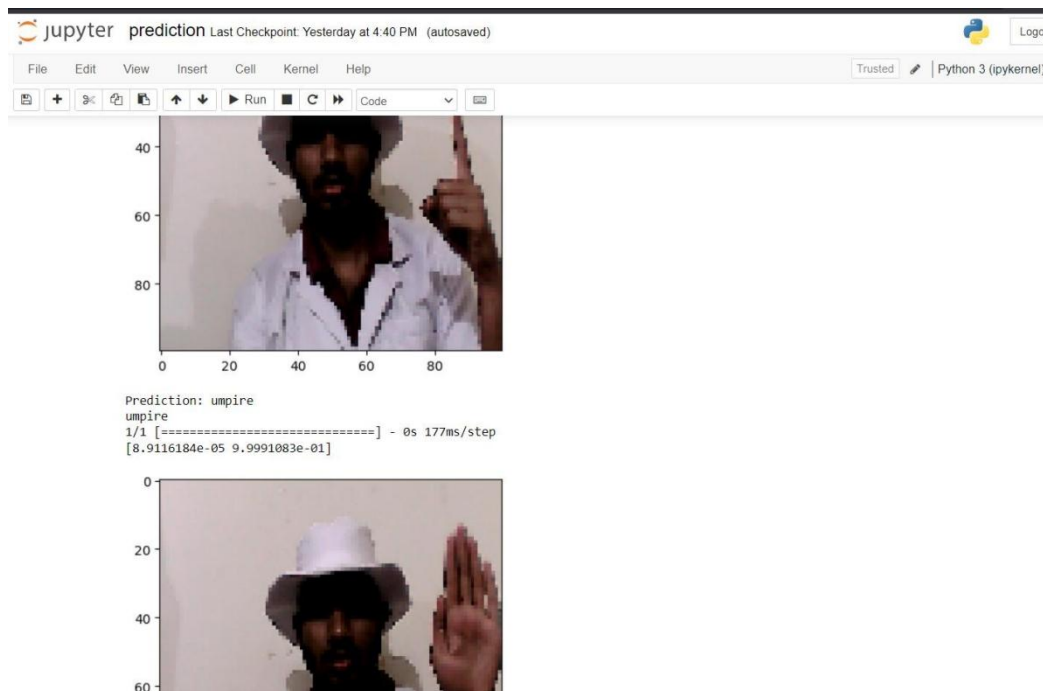


Fig 7.4 Umpire Detection

The above Fig 7.4 tells about the umpire detection and if its umpire or not an umpire and saves it as “img.jpg”.

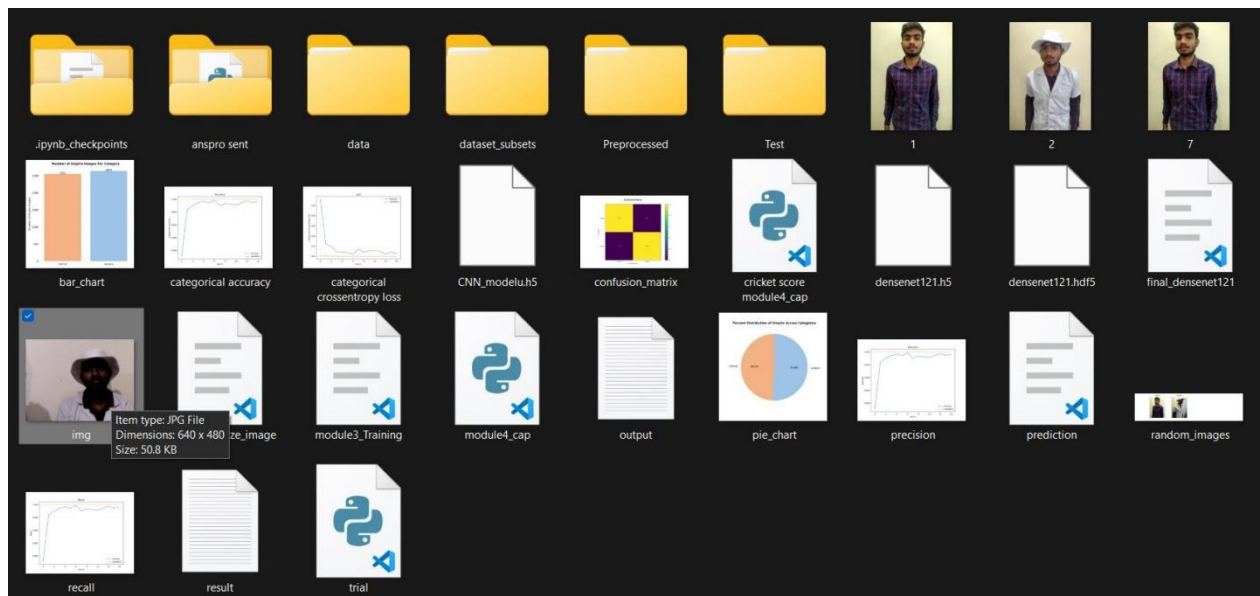
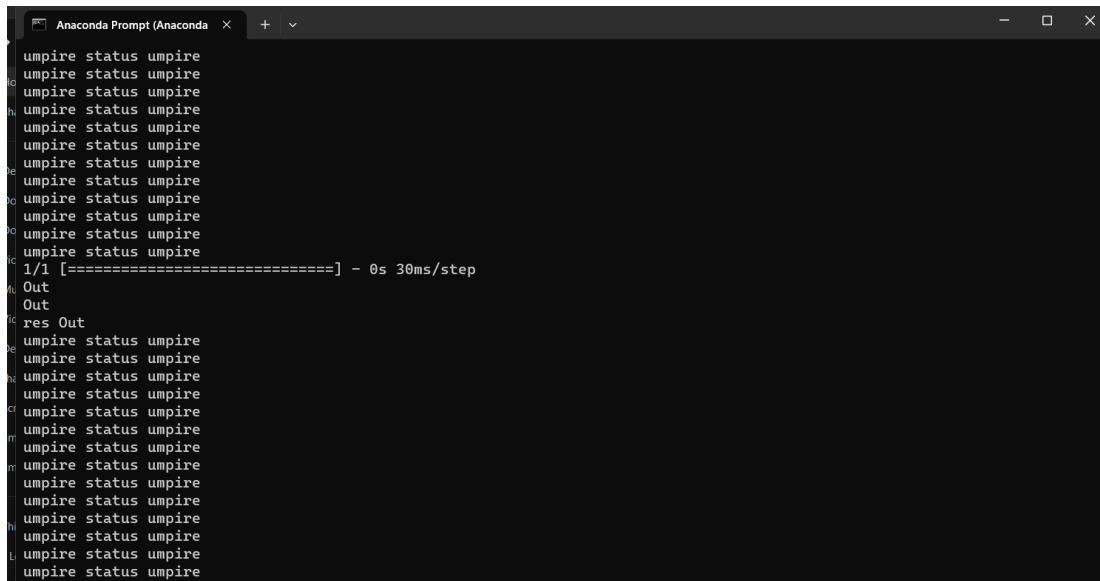


Fig 7.5 Saving the Result of Umpire detection

The above Fig 7.5 is about results of umpire detection that have been saved, the file img is where the umpire detection is saved.



```
Anaconda Prompt (Anaconda) x + -
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
1/1 [=====] - 0s 30ms/step
Out
Out
res Out
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
umpire status umpire
```

Fig 7.6 Checking if umpire is detected or not

In anaconda prompt we can see whether its umpire status is detected or its normal status as shown in Fig 7.6.

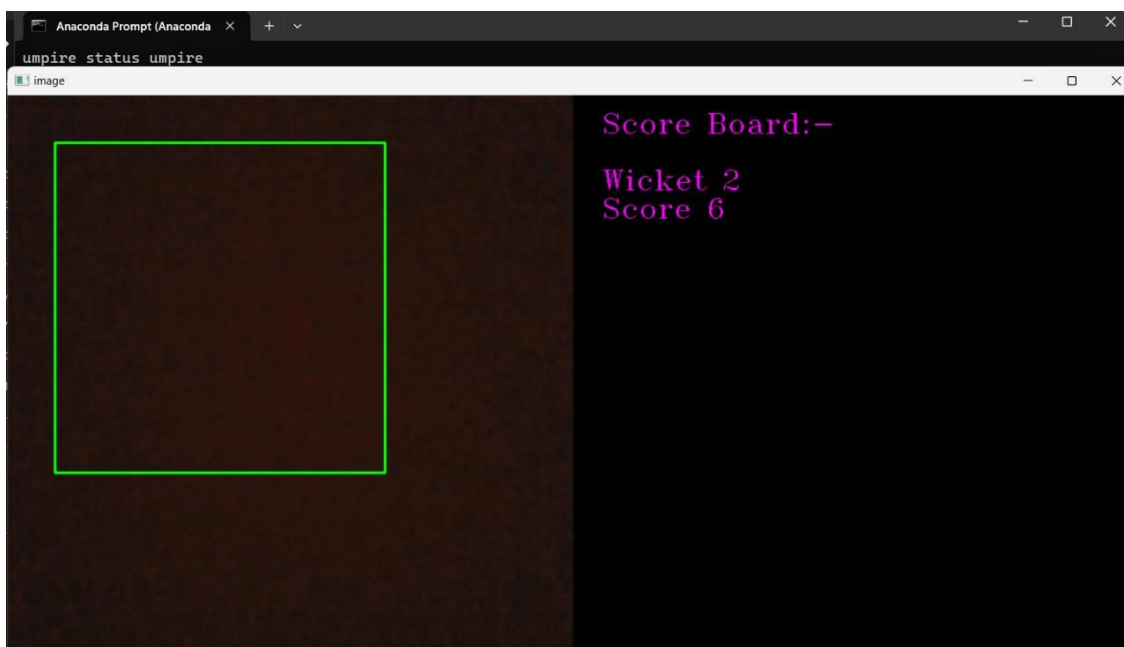


Fig 7.7 If umpire is detected then recognizing the gestures

When umpire is detected then it is recognized the gestures and the score board is updated as shown in Fig 7.7.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

We Started with the searching and finalizing of our project name and domain where we found that Machine learning and Artificial intelligence is a blooming industry and it is not yet implemented in Cricket so we finalized “AI Cricket Score” as our project. We intend to implement the detection of sign concept in the field of cricket. To make it happen we will have to capture the umpire’s signs in the field of cricket. We surveyed total of 10 Research Papers of different authors that have done projects in this field. We found that using web camera, we can capture the images and save in the respective directory of the sign. The images will then undergo for pre-processing techniques so that better quality of the images can be used for feature extraction. We also found about deep learning algorithms that can be implemented on the saved model for prediction. We have researched about modules that can be used in our project for high accuracy and we have started learning about them. Our work will also include data gathering using a web camera to increase the dataset size to more than 5000 RGB images making our prediction more robust. Process of real-time prediction using image frames from a web camera with rates of 50 to 100 Hz can also be used. Currently the aim of this project is limited to detecting Umpire’s gesture. Future Enhancement may include more cameras for detecting the umpire in 360 degrees to improve gesture recognition and unique gestures for actions that do not have gestures can be added to reduce the workload on the review team. With enough funds this project can overtake and detect even the non-gesture based actions like boundary detection for run out, four and six runs.

8.2 FUTURE ENHANCEMENT

Our work aims to enhance the research on the Umpire Gesture Detection, Recognition, and Classification structure by exploring ways to increase accuracy. The existing project can be extended using more training data, using better validation techniques, and incorporating temporal information. To handle occlusions and variations, more robust feature extraction and advanced machine learning models can be used .

REFERENCES

- [1] M. Fernando and J. Wijayanayaka, "Low cost approach for real time sign language recognition," *2013 IEEE 8th International Conference on Industrial and Information Systems*, 2013, pp. 637-642, doi: 10.1109/ICIIInfS.2013.6732059.
- [2] M. Z. Islam, M. S. Hossain, R. ul Islam and K. Andersson, "Static Hand Gesture Recognition using Convolutional Neural Network with Data Augmentation," *2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, 2019, pp. 324-329, doi: 10.1109/ICIEV.2019.8858563.
- [3] E. Kaya and T. Kumbasar, "Hand Gesture Recognition Systems with the Wearable Myo Armband," *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, 2018, pp. 1-6, doi: 10.1109/CEIT.2018.8751927.
- [4] Lesha Bhansali and Meera Narvekar. Gesture Recognition to Make Umpire Decisions. *International Journal of Computer Applications* 148(14):26-29, August 2016.
- [5] Suvarna Nandyal and Suvarna Laxmikant Kattimani 2021 *J. Phys.: Conf. Ser.* 2070 012148
- [6] Y. Madhuri, G. Anitha. and M. Anburajan., "Vision-based sign language translation device," *2013 International Conference on Information Communication and Embedded Systems (ICICES)*, 2013, pp. 565-568, doi: 10.1109/ICICES.2013.6508395.
- [7] Nusirwan Anwar bin Abdul Rahman, Kit Chong Wei and John See Faculty of Information Technology, Multimedia University.
- [8] Moin, A., Zhou, A., Rahimi, A. *et al.* A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nat Electron* 4, 54–63 (2021). <https://doi.org/10.1038/s41928-020-00510-8>
- [9] Ravi, H. Venugopal, S. Paul and H. R. Tizhoosh, "A Dataset and Preliminary Results for Umpire Pose Detection Using SVM Classification of Deep Features," *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2018, pp. 1396-1402, doi: 10.1109/SSCI.2018.8628877.
- [10] M. A. Shahjalal, Z. Ahmad, R. Rayan and L. Alam, "An approach to automate the scorecard in cricket with computer vision and machine learning," *2017 3rd International Conference on Electrical Information and Communication Technology (EICT)*, 2017, pp. 1-6, doi: 10.1109/EICT.2017.8275204.

APPENDIX

A.1 FRONTEND

The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.) Running `python -m tkinter` from the command line should open a window demonstrating a simple Tk interface, letting know that tkinter is properly installed on the system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Tkinter Modules are a number of additional modules are available as well. The Tk interface is located in a binary module named `_tkinter`. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter. In addition to the Tk interface module, tkinter includes a number of Python modules, `tkinter.constants` being one of the most important. Importing tkinter will automatically import `tkinter.constants`, so, usually, to use Tkinter all you need is a simple import statement.

The Tk class is instantiated without arguments. This creates a toplevel widget of Tk which usually is the main window of an application. Each instance has its own associated Tcl interpreter. `Tcl (screenName=None, baseName=None, className='Tk', useTk=0)`.

The `Tcl()` function is a factory function which creates an object much like that created by the Tk class, except that it does not initialize the Tk subsystem. This is most often useful when driving the Tcl interpreter in an environment where one doesn’t want to create extraneous toplevel windows, or where one cannot (such as Unix/Linux systems without an X server). An object created by the `Tcl()` object can have a Top-level window created (and the Tk subsystem initialized) by calling its `loadtk()` method.

Tkinter Life Preserver section is not designed to be an exhaustive tutorial on either Tk or Tkinter. Rather, it is intended as a stop gap, providing some introductory orientation on the system. Tk/Tcl has long been an integral part of Python. It provides a robust and platform independent windowing toolkit, that is available to Python programmers using the tkinter package, and its

The tkinter package is a thin object-oriented layer on top of Tcl/Tk. To use tkinter, the user don't need to write Tcl code, but they will need to consult the Tk documentation, and occasionally the Tcl documentation. tkinter is a set of wrappers that implement the Tk widgets as Python classes. In addition, the internal module `_tkinter` provides a threadsafe mechanism which allows Python and Tcl to interact. tkinter's chief virtues are that it is fast, and that it usually comes bundled with Python. Although its standard documentation is weak, good material is available, which includes: references, tutorials, a book and others. tkinter is also famous for having an outdated look and feel, which has been vastly improved in Tk 8.5.

The packer is one of Tk's geometry-management mechanisms. Geometry managers are used to specify the relative positioning of widgets within their container - their mutual master. In contrast to the more cumbersome placer (which is used less commonly), the packer takes qualitative relationship specification. The size of any master widget is determined by the size of the "slave widgets" inside. The packer is used to control where slave widgets appear inside the master into which they are packed. The user can pack widgets into frames, and frames into other frames, in order to achieve the kind of layout desired. Additionally, the arrangement is dynamically adjusted to accommodate incremental changes to the configuration, once it is packed. Note that widgets do not appear until they have had their geometry specified with a geometry manager.

It's a common early mistake to leave out the geometry specification, and then be surprised when the widget is created but nothing appears. A widget will appear only after it has had, for example, the packer's `pack()` method applied to it. The only kinds of variables for which this works are variables that are sub classed from a class called `Variable`, defined in tkinter.

There are many useful subclasses of `Variable` already defined: `StringVar`, `IntVar`, `DoubleVar`, and `BooleanVar`. To read the current value of such a variable, call the `get()` method on it, and to change its value call the `set()` method. If the user follows this protocol, the widget will always track the value of the variable, with no further intervention.

A.2 BACKEND

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows the user to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. The user can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The simplest way is with Spyder. From the Navigator Home tab, click Spyder, and write and execute your code.

Jupyter Notebooks can also be used the same way. Jupyter Notebooks are an increasingly popular system that combine the code, descriptive text, output, images, and interactive interfaces into a single notebook file that is edited, viewed, and used in a web browser.

Open the Jupyter Notebook application as shown in Fig A.1

- Log in to AEN.
- Select the project to work on or create a new project and open it.
- On the project home page, click the Jupyter Notebook icon:



Fig A.1 Anaconda Navigator

Jupyter Notebook opens in a new browser window as shown in Fig A.2



Fig A.2 Jupyter Notebook Home page

Open any example notebook to experiment and see how it works.

A.3 SOURCE CODE

Module 1- Create Dataset Of Umpire and Gestures

```
import os
```

```
"""The OS module in Python is a part of the standard library of the programming language.
When imported, it lets the user interact with the native OS Python is currently running on.
In simple terms, it provides an easy way for the user to interact with several os functions that
come
```

```
in handy in day to day programming"""
```

```
import time
```

```
"""the Python time module provides many ways of representing time in code, such as objects,
numbers, and strings.
```

```
It also provides functionality other than representing time,
```

```
like waiting during code execution and measuring the efficiency of your code."""
```

```
import cv2
```

```
"""cv2 is the module import name for opencv-python, "Unofficial pre-built CPU-only OpenCV
packages for Python" """
```

```
import numpy as np
```

```
"""The import numpy portion of the code tells Python to bring the NumPy library into your
current environment.
```

```
The as np portion of the code then tells Python to give NumPy the alias of np.
```

This allows you to use NumPy functions by simply typing np"""

```
vc = cv2.VideoCapture(0)
```

#To capture a video, we need to create a VideoCapture object.

#A device index is just the number to specify which camera.

#A device index is just the number to specify which camera, Normally one camera will be connected 0 is passed here.

#Second camera can be selected by passing 1 and so on.

```
pic_no = 0
```

#We initially declare pictures captured as 0.

```
total_pic = 1000
```

#This specifies how many pictures to click, can be as per convenience.

```
flag_capturing = False
```

#Initially we set capturing as false (It will not capture images) because a key is pressed to start the capturing.

```
path = 'Dataset/noball'
```

#Here we need to set a path where the pictures captured will be stored.

#Path need to be manually created.

```
while(vc.isOpened()):
```

```
    #If the Camera is Opened (Streaming)
```

```
    rval, frame = vc.read()
```

```
    #It will read the frames
```

```
    #read() in OpenCV returns 2 things, boolean and data
```

```
    #Here frame will have the image data and rval will have boolean data of the data
```

```
    frame = cv2.flip(frame, 1)
```

```
    #we flip the images to obtain original images as image captured is flipped.
```

```
    #Using a flip code value of 1 indicates that we flipped the image horizontally, around the y-axis.
```

```
    cv2.rectangle(frame, (300,300), (100,100), (0,255,0),0)
```

```
    #cv2.rectangle(frame, (639,479), (0,0), (0,255,0),0) #for full screen green box
```

```
    """Syntax: cv2.rectangle(image, start_point, end_point, color, thickness)"""
```

```
    """Parameters:
```

image: It is the image on which rectangle is to be drawn.

start_point: It is the starting coordinates of rectangle.

The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

end_point: It is the ending coordinates of rectangle.

The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

color: It is the color of border line of rectangle to be drawn.

For BGR, we pass a tuple. eg: (0, 255, 0) for blue color.

thickness: It is the thickness of the rectangle border line in px.

Thickness of -1 px will fill the rectangle shape by the specified color.

Return Value: It returns an image. """

```
cv2.imshow("image", frame)
```

```
"""Syntax: cv2.imshow(window_name, image)
```

Parameters:

window_name: A string representing the name of the window in which image to be displayed.

image: It is the image that is to be displayed.

Return Value: It doesn't returns anything. """

```
crop_img = frame[100:300, 100:300]
```

#Syntax: image[rows,columns] it will crop the image by using slicing function

```
if flag_capturing:
```

```
#this if condition will activate when key 'c' is pressed
```

```
pic_no += 1
```

```
#it will
```

```
save_img = cv2.resize( crop_img, (50,50) )
```

```
#this will resize the image into 50x50 size
```

```
save_img = np.array(save_img)
```

```
#saves the image locally as an array
```

```
cv2.imwrite(path + "/" + str(pic_no) + ".jpg", save_img)
```

```
"""it will save image from "save_image" array to a path. "/" means it will save it inside a
```

path and since "pic_no" is

in integer format we convert it into string and give the format as .jpg. This will save images from 1 to 1000 in

jpg format and the naming will be in numerical format"""

```
keypress = cv2.waitKey(1)
```

#waitkey() function of Python OpenCV allows users to display a window for given milliseconds or until any key is pressed.

#This will capture image continuously with a delay of 1ms until 1000 images are reached

```
if pic_no == total_pic:
```

```
    flag_capturing = False
```

```
    #this will break the image capturing process when 1000 images are captured
```

```
    break
```

```
if keypress == ord('q'):
```

```
    #ord('q') returns the Unicode code point of q
```

```
    #when q is pressed the image capturing process will halt
```

```
    break
```

```
elif keypress == ord('c'):
```

```
    #ord('c') returns the Unicode code point of c
```

```
    #when c is pressed the if condition "flag_capturing" will be initialized
```

```
    flag_capturing = True
```

```
vc.release()
```

to release the videocapture object (camera), if not done it will raise conflict with other softwares using camera

```
cv2.destroyAllWindows()
```

#destroyAllWindows() function allows users to destroy or close all windows at any time after exiting the script.

```
cv2.waitKey(1)
```

Module 2- PreProcess Gesture dataset

```
import os
import time
import cv2
import numpy as np

path = 'Dataset/'
#we specify where the data is to be taken for preprocessing
path2 = 'Preprocessed/Train/'
#This is the directory where the preprocessed picture will be saved
#We need to create this directory

gestures = os.listdir(path)
#os.listdir() method in python is used to get the list of all files and directories in the specified
directory.
#This will take all the folders and files in Dataset

print(gestures)
#This will print all the name of the folder

for ix in gestures:
    print(ix)
    #This will print all the available folder in Dataset folder
    images = os.listdir(path + ix)
    #This will get a list of files inside an ix folder which lies in dataset
    os.mkdir(path2 + ix)
    #this will create ix folders from dataset to path2 which is train folder
    for cx in images:
        print(cx)
        #prints the name of all the images(cx) inside folder ix
        img_path = path + ix + '/' + cx
        #This is the path of an image inside ix folder which is present in dataset folder
        img = cv2.imread(img_path)
        #cv2.imread() method loads an image from the specified file
        grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

#cv2.cvtColor() method is used to convert an image from one color space to another.

"""Syntax: cv2.cvtColor(src, code)"""

Parameters:

src: It is the image whose color space is to be changed.

code: It is the color space conversion code."""

```
thresh = cv2.threshold(grey, 127, 255,  
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)[1]
```

"""Thresholding is the binarization of an image. In general, we seek to convert a grayscale image

to a binary image, where the pixels are either 0 or 255."""

#cv2.THRESH_BINARY_INV is used to create binary image where object is black in color and background is white.

"""Otsu's method assumes that our image contains two classes of pixels: the background and the foreground.

It Compute the threshold value T and Replace image pixels into white in those regions, where saturation is greater than T and into the black in the opposite cases."""

"""In Otsu Thresholding, a value of the threshold isn't chosen but is determined automatically.

A bimodal image (two distinct image values) is considered.

The histogram generated contains two peaks.

So, a generic condition would be to choose a threshold value that lies in the middle of both the histogram

peak values."""

```
save_img = cv2.resize(thresh, (50,50))
```

#uses the threshold value and resized it into 50x50 pixel size

```
cv2.imwrite(path2 + ix + '/' + cx, save_img)
```

#saves the image to "train" folder"

Module 3 - Train and test gesture dataset

```
import os
```

```
import cv2
```

```
import time
```

```
import numpy as np
```

```
from keras.layers import Conv2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import MaxPooling2D
from keras.models import Sequential, save_model
from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings(action = 'ignore')
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
path = 'Preprocessed/Train/'
gestures = os.listdir(path)
dict_labels = {
    'out': 1,
    'noball': 2
}
print(list(dict_labels.keys()))
x, y = [], []
for ix in gestures:
    images = os.listdir(path + ix)
    for cx in images:
        img_path = path + ix + '/' + cx
        img = cv2.imread(img_path, 0)
        img = img.reshape((50,50,1))
        img = img/255.0
```

```
x.append(img)
y.append(dict_labels[ix])
x
X = np.array(x)
Y = np.array(y)
print(Y)
Y = np_utils.to_categorical(Y)
print(Y)
print(type(Y),len(Y))
Y.shape
plt.figure(figsize = (18,8))
sns.countplot(x=list(dict_labels.keys()))
Y.shape
categories = Y.shape[1]
X, Y = shuffle(X, Y, random_state=0)
X.shape
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
print(X_train.shape, X_test.shape)
print(Y_train.shape, Y_test.shape)
model = Sequential()
model.add(Conv2D(64, kernel_size=(3,3), activation = 'relu', input_shape=(50,50 ,1) ))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.20))
model.add(Dense(categories, activation = 'softmax'))
```

```
model.summary()
model.compile(optimizer='Adam', metrics=['accuracy'], loss='categorical_crossentropy')
history = model.fit(X_train, Y_train, batch_size=128, epochs=50, validation_data=[X_test,
Y_test])
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Accuracy")
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train','test'])
plt.show()
model.save('CNN_model.h5')
m = load_model('CNN_model.h5')
test_data = os.listdir('Test/')
dict_labels
for ix in test_data:
    print(ix)
x, y = [], []
for ix in test_data:
    images = os.listdir('Test/' + ix)
    for cx in range(1,201):
        img_path = 'Test/' + ix + '/' + str(cx) + '.jpg'
        img = cv2.imread(img_path, 0)
        if img is not None:
            img = img.reshape((50,50,1))
            img = img/255.0
            x.append(img)
            y.append(dict_labels[ix])
        else:
            print("Error: could not load image from", img_path)

X_t = np.array(x)
y_t = np.array(y)
Y_t = np_utils.to_categorical(y_t)
```

```
X_t.shape
y_pred = m.predict(X_t)
acc = accuracy_score(Y_t, y_pred.round())
print('Accuracy:', acc)
print(classification_report(y_pred.round(), Y_t))
```

Module 4 - PreProcess Umpire dataset

```
import os
import time
import cv2
import numpy as np
import matplotlib.pyplot as plt

path = 'data/'
path2 = 'Preprocessed/'

gestures = os.listdir(path)

print(gestures)

for ix in gestures:
    print(ix)
    images = os.listdir(path + ix)
    os.mkdir(path2 + ix)
    for cx in images:
        print(cx)
        img_path = path + ix + '/' + cx
        img = cv2.imread(img_path)
        grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        thresh = cv2.threshold(grey, 127, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)[1]
        save_img = cv2.resize(thresh, (50,50))
        cv2.imwrite(path2 + ix + '/' + cx, save_img)
```


Module 5 - Train and test umpire dataset

```
import os
import random
from glob import glob

import matplotlib.pyplot as plt
import numpy as np
#pip install split-folders
import splitfolders

from IPython.display import display
from PIL import Image
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.applications.densenet import (DenseNet121,
                                                    preprocess_input)
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.layers import (BatchNormalization, Dense,
                                     Dropout, Flatten, Input)
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

data_dir = 'data'
train_dir = data_dir + '/train'
valid_dir = data_dir + '/valid'
test_dir = data_dir + '/test'

DATASET_PATH = 'data/train/'
dir_list = os.listdir(DATASET_PATH)
# This code uses the os module in Python to list all files and directories in the directory specified
# by the DATASET_PATH variable.
# The os.listdir() function returns a list of strings, where each string represents the name of a file
```

or directory in the specified directory.

```
print(dir_list)
```

```
num_of_images = []
```

```
for root, dirs, files in os.walk(DATASET_PATH):
```

```
    # os.walk() is called with DATASET_PATH as the argument, which means it will traverse all
    directories and subdirectories within DATASET_PATH.
```

```
    if root == DATASET_PATH: continue
```

```
    # This line of code is a conditional statement that skips the current iteration of the loop if root
    is equal to DATASET_PATH.
```

```
    # The root variable is a string representing the full path to the current directory being
    processed by the loop.
```

```
    # The continue statement is used to skip the current iteration of a loop and move on to the next
    iteration.
```

```
    print('{}: {}'.format(os.path.basename(root), len(files)))
```

```
    # The os.path.basename() function is used to extract the name of the current directory from
    the full path stored in the root variable.
```

```
    num_of_images.append(len(files))
```

```
    # This line of code appends the number of files in the current directory being processed by the
    loop to the num_of_images list.
```

```
    # The len() function is used to count the number of files in the current directory, which is
    stored in the files variable.
```

```
    # The append() method is used to add the number of files to the end of the num_of_images list
    SEED = 123
```

```
random.seed(SEED)
```

```
#By calling random.seed(SEED), the random number generator is initialized with the SEED
value, which is typically an integer.
```

```
# For example, if SEED is set to 42, calling random.seed(SEED) will initialize the random
number generator with the seed value 42
```

```
rand_img = Image.open(random.choice(glob(DATASET_PATH + '*/*/*')))
```

```
print('path:', rand_img.filename)
print('size:', rand_img.size)
print('mode:', rand_img.mode)
print('format:', rand_img.format)
display(rand_img)
# Plot the number of images per subdirectory
x_coord = np.arange(len(dir_list))
colors = ['#f4b183', '#9dc3e6', '#a9d18e', '#ffd966']
# These colors are a light orange, light blue, light green, and a light yellow, respectively.

fig, ax = plt.subplots(facecolor='white', figsize=(6,6))
# The plt.subplots() function is used to create the figure and axes objects. The facecolor
parameter is set to 'white', which sets the background color of the figure to white.
# The figsize parameter is set to (6,6), which sets the size of the figure to 6 inches by 6 inches.

ax.bar(x_coord, num_of_images, color=colors)
# The ax.bar() method is used to create the bar chart. The first argument, x_coord, is a list of x-
coordinates for the bars.
# The second argument, num_of_images, is a list of heights for the bars, which correspond to the
number of images in each subdirectory.
# The color parameter is set to colors, which is a list of colors for the bars.

ax.spines['top'].set_visible(False)
# The ax.spines attribute is used to access the individual spines of the plot.
# In this case, the 'top' spine is accessed using square brackets, and the .set_visible(False)
method is called to hide it.
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['bottom'].set_color('#EEEEEE')

ax.yaxis.grid(True, color='#EEEEEE')
ax.set_axisbelow(True)

plt.xticks(x_coord, dir_list)
```

```
ax.tick_params(bottom=False, left=False)

plt.title('Number of Umpire Images Per Category', fontweight='bold')
plt.ylabel('Number of Umpire Images')

for i in range(len(x_coord)):
    plt.text(i, num_of_images[i], num_of_images[i], ha='center')

plt.tight_layout()
plt.savefig('bar_chart.png', dpi=200)
plt.show()

# Plot the percent distribution of images across subdirectories
plt.figure(facecolor='white', figsize=(6,6))
plt.pie(num_of_images,
        labels=dir_list,
        colors=colors,
        autopct='%0.1f%%',
        startangle=90)
plt.title('Percent Distribution of Umpire Across Categories', fontweight='bold')
plt.savefig('pie_chart.png', dpi=200)
plt.show()

# Display random images, one from each category
plt.figure(facecolor='white', figsize=(12,3))
for i in range(2):
    ax = plt.subplot(1, 4, i + 1)
    path = random.choice(glob(DATASET_PATH + '/' + dir_list[i] + '/*'))
    plt.imshow(Image.open(path), cmap='gray')
    plt.title(dir_list[i])
    plt.axis('off')
plt.savefig('random_images.png', dpi=200)
plt.show()

# Split the dataset folder into test, train, and val folders
OUTPUT_PATH = 'dataset_subsets'
if not os.path.exists(OUTPUT_PATH):
```

```
splitfolders.ratio(DATASET_PATH, OUTPUT_PATH, SEED, (.7, .1, .2))
IMG_DIMS = 100
TARGET_SIZE = (IMG_DIMS, IMG_DIMS)
COLOR_MODE = 'rgb'
CLASS_MODE = 'categorical'
BATCH_SIZE = 16
# Take the path to the train folder and generate batches of data
TRAIN_PATH = os.path.join(OUTPUT_PATH, 'train')
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=5,
    width_shift_range=0.05,
    height_shift_range=0.05,
    brightness_range=[0.9, 1.1],
    zoom_range=[0.9, 1.1],
    horizontal_flip=True)
train_gen = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=TARGET_SIZE,
    color_mode=COLOR_MODE,
    class_mode=CLASS_MODE,
    batch_size=BATCH_SIZE,
    seed=SEED)
# Take the path to the val folder and generate batches of data
VAL_PATH = os.path.join(OUTPUT_PATH, 'val')
val_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)
val_gen = val_datagen.flow_from_directory(
    VAL_PATH,
    target_size=TARGET_SIZE,
    color_mode=COLOR_MODE,
    class_mode=CLASS_MODE,
    batch_size=BATCH_SIZE,
    seed=SEED)
```

```
# Take the path to the train folder and generate batches of data
TEST_PATH = os.path.join(OUTPUT_PATH, 'test')
test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)
test_gen = test_datagen.flow_from_directory(
    TEST_PATH,
    target_size=TARGET_SIZE,
    color_mode=COLOR_MODE,
    class_mode=CLASS_MODE,
    batch_size=BATCH_SIZE,
    shuffle=False,
    seed=SEED)
print(train_gen.directory)
print(val_gen.directory)
print(test_gen.directory)
print(train_gen.class_indices)
# Print the shapes of the first training batch tuple
images, labels = next(iter(train_gen))
print('images shape:', images.shape) # (batch_size, *target_size, channels)
print('labels shape:', labels.shape) # (batch_size, num_classes)
# Instantiate a Densenet121 architecture without the fully-connected layer
base_model = DenseNet121(
    include_top=False,
    weights='imagenet',
    input_tensor=Input(shape=(IMG_DIMS, IMG_DIMS, 3)),
    input_shape=(IMG_DIMS, IMG_DIMS, 3))
# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False
# Build the modified DenseNet121 model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(BatchNormalization())
```

```
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())
model.add(Dense(256, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()
# Define hyperparameters
EPOCHS = 15
ETA = 0.0001
METRICS = ['categorical_accuracy', Precision(), Recall()]
# Compile the model
model.compile(optimizer=Adam(ETA),
              loss='categorical_crossentropy',
              metrics=METRICS)
# Create a learning rate scheduler callback
def scheduler(epoch):
    if epoch > 19:
        return ETA * 0.1
    else:
        return ETA
callback = LearningRateScheduler(scheduler)
# Estimate class weights for the unbalanced dataset
cls_wt = compute_class_weight(class_weight='balanced',
                              classes=np.unique(train_gen.labels),
                              y=train_gen.labels)
class_weight = {0: cls_wt[0], 1: cls_wt[1]}
print(class_weight)
# Train the model
history = model.fit(train_gen,
                    epochs=EPOCHS,
                    callbacks=callback,
```

```
        validation_data=val_gen,
        class_weight=class_weight)

def plot_metric_history(metric, val_metric, y_label, title):
    """
    Plots a training history metric.

    :param metric: the metric key
    :param val_metric: the validation metric key
    :param y_label: the y-axis label
    :param title: the title
    :return: None
    """
    plt.figure(facecolor='white', figsize=(8, 6))
    plt.plot(history.history[metric])
    plt.plot(history.history[val_metric])
    plt.ylabel(y_label)
    plt.xlabel('epoch')
    plt.title(title)
    plt.legend(['training', 'validation'])
    plt.savefig(y_label + '.png', dpi=200)
    plt.show()

# Visualize loss history
plot_metric_history(metric='loss',
                    val_metric='val_loss',
                    y_label='categorical crossentropy loss',
                    title='Loss')

# Visualize accuracy history
plot_metric_history(metric='categorical_accuracy',
                    val_metric='val_categorical_accuracy',
                    y_label='categorical accuracy',
                    title='Accuracy')

# Visualize precision history
plot_metric_history(metric='precision',
                    val_metric='val_precision',
```



```
        y_label='precision',
        title='Precision')
# Visualize recall history
plot_metric_history(metric='recall',
                    val_metric='val_recall',
                    y_label='recall',
                    title='Recall')

# Print metrics values for the model in test mode
result = model.evaluate(test_gen)
dict(zip(model.metrics_names, result))
predictions = model.predict(test_gen)
# Initialize true and predicted label values
y_true = test_gen.labels
y_pred = np.argmax(predictions, axis=1)
# Build a text report showing the main classification metrics
report = classification_report(y_true, y_pred)
print(report)
# Plot a confusion matrix given true and predicted labels
fig, ax = plt.subplots(facecolor='white', figsize=(12, 8), dpi=75)
fig = ConfusionMatrixDisplay.from_predictions(y_true,
                                             y_pred,
                                             display_labels=test_gen.class_indices,
                                             include_values=True,
                                             normalize='pred',
                                             values_format='.2f',
                                             cmap='viridis',
                                             ax=ax)

plt.tick_params(left=False, bottom=False)
plt.title('Confusion Matrix', fontsize=15, fontweight='bold', pad=30)
plt.savefig('confusion_matrix.png', dpi=200)
plt.show()
model.save_weights("densenet121.h5")
filepath="densenet121.hdf5"
model.save(filepath)
```

```
from tensorflow.keras.models import load_model
classifier = load_model('densenet121.hdf5')
def predict(image_path):
    from skimage import io
    from keras.preprocessing import image
    #path='imbalanced/Scratch/Scratch_400.jpg'
    import tensorflow as tf

    img = tf.keras.utils.load_img(image_path, grayscale=False, target_size=(100, 100))
    show_img=tf.keras.utils.load_img(image_path, grayscale=False, target_size=(100, 100))
    disease_class = ['normal','umpire']
    x = tf.keras.utils.img_to_array(img)
    x = np.expand_dims(x, axis = 0)
    #x = np.array(x, 'float32')
    x /= 255

    custom = classifier.predict(x)
    print(custom[0])

    #plt.gray()
    plt.imshow(show_img)
    plt.show()

    a=custom[0]
    ind=np.argmax(a)

    print('Prediction:',disease_class[ind])

predict('1.jpg')
predict('2.jpg')
predict('7.jpg')
```

Module 6 - Capturing and detecting Umpire

```
import os
import random
from glob import glob

import matplotlib.pyplot as plt
import numpy as np
#pip install split-folders
import splitfolders

from IPython.display import display
from PIL import Image
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.applications.densenet import (DenseNet121,
                                                    preprocess_input)
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.layers import (BatchNormalization, Dense,
                                     Dropout, Flatten, Input)
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
classifier = load_model('densenet121.hdf5')
def predict(image_path):
    from skimage import io
    from keras.preprocessing import image
    #path='imbalanced/Scratch/Scratch_400.jpg'
    import tensorflow as tf

    img = tf.keras.utils.load_img(image_path, grayscale=False, target_size=(100, 100))
    show_img=tf.keras.utils.load_img(image_path, grayscale=False, target_size=(100, 100))
    p_class = ['normal','umpire']
    x = tf.keras.utils.img_to_array(img)
```

```
x = np.expand_dims(x, axis = 0)
#x = np.array(x, 'float32')
x /= 255
```

```
custom = classifier.predict(x)
print(custom[0])
```

```
#x = x.reshape([64, 64]);
```

```
#plt.gray()
plt.imshow(show_img)
plt.show()
```

```
a=custom[0]
ind=np.argmax(a)
```

```
print('Prediction:',p_class[ind])
f=open('output.txt','w')
f.write(str(p_class[ind]))
f.close()
return str(p_class[ind])
```

```
import cv2
import time
def capture_image():

    # access the default camera (usually USB webcam)
    cap = cv2.VideoCapture(1)
    if not cap.isOpened():
        print("Error: Could not open camera.")
        return 0
    else:
        ret, frame = cap.read()
```

```
cv2.imwrite("img.jpg", frame)
cap.release()
```

```
capture_image()
while True:
    capture_image()
    filename = 'img.jpg'
    result=predict(filename)
    print(result)
    time.sleep(5)
```

Module 7 - Detecting Hand Gesture of Umpire

```
import os
import cv2
import time
import numpy as np
from keras.models import load_model

import warnings
warnings.filterwarnings(action = 'ignore')

model = load_model('CNN_modelu.h5')

gestures = {
    1:'Out',
    2:'No_ball',
    3:'Six',
    4:'Two',
    5:'None'
}

def predict(gesture):
    img = cv2.resize(gesture, (50,50))
```

```
img = img.reshape(1,50,50,1)
img = img/255.0
prd = model.predict(img)
index = prd.argmax()
return gestures[index]

vc = cv2.VideoCapture(0)
rval, frame = vc.read()
old_text = ""
pred_text = ""
count_frames = 0
total_str = ""
flag = False
wicket=0
score=0
status='1'
while True:
    #time.sleep(10)

    if frame is not None:

        frame = cv2.flip(frame, 1)
        frame = cv2.resize( frame, (600,600) )

        cv2.rectangle(frame, (400,400), (50,50), (0,255,0), 2)

        crop_img = frame[100:300, 100:300]
        grey = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)

        thresh =
cv2.threshold(grey,210,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)[1]

        blackboard = np.zeros(frame.shape, dtype=np.uint8)
        cv2.putText(blackboard, "Score Board:- ", (30, 40), cv2.FONT_HERSHEY_TRIPLEX, 1,
```

```
(255, 0, 255))

    if count_frames > 20 and pred_text != "":
        total_str += pred_text
        count_frames = 0

    cv2.putText(blackboard, "Wicket "+str(wicket), (30, 100),
cv2.FONT_HERSHEY_TRIPLEX, 1, (255, 0, 255))
    cv2.putText(blackboard, "Score "+str(score), (30, 130), cv2.FONT_HERSHEY_TRIPLEX,
1, (255, 0, 255))

    f = open('C:/Users/Shantanu J/Desktop/umpire/output.txt','r')
    ustatus=f.read()
    f.close()
    print('umpire status',ustatus)
    if flag == True and status=='1' and ustatus=='umpire':

        pred_text = predict(thresh)
        print(pred_text)

        old_text = pred_text
        #f = open('H:/code check/Cricket_Score/output.txt','w')
        #f.write(str(pred_text))
        #f.close()

        if str(pred_text)=='Out':
            res = 'Out'
            print(res)
            wicket=wicket+1
            if wicket==10:
                status='0'
                flag=False
            elif str(pred_text)=='No_ball':
                res = 'No_ball'
                print(res)
                score=score+1
            elif str(pred_text)=='None':
```

```
        res = 'None'
        print(res)
    elif str(pred_text)=='Six':
        res = 'Six'
        print(res)
        score=score+6
    elif str(pred_text)=='Two':
        res = 'Two'
        print(res)
        score=score+2
'''

elif str(pred_text)=='Four':
    res = 'Four'
    print(res)
    score=score+4
'''

print('res',res)

cv2.putText(blackboard, res, (30, 70), cv2.FONT_HERSHEY_TRIPLEX, 1, (255, 0,
255))

cv2.putText(blackboard, "Wicket "+str(wicket), (30, 100),
cv2.FONT_HERSHEY_TRIPLEX, 1, (255, 0, 255))
cv2.putText(blackboard, "Score "+str(score), (30, 130),
cv2.FONT_HERSHEY_TRIPLEX, 1, (255, 0, 255))
f = open('result.txt','w')
f.write(str(res))
f.close()
if old_text == pred_text:
    count_frames += 1
else:
    count_frames = 0
#print('*****',pred_text)
#f = open('H:/code check/Cricket_Score/result.txt','w')
```

```
#f.write(str(res))
#f.close()

#cv2.putText(blackboard, total_str, (30, 80), cv2.FONT_HERSHEY_TRIPLEX, 1, (0,
255, 0))
res = np.hstack((frame, blackboard))

cv2.imshow("image", res)
#cv2.imshow("hand", thresh)
#time.sleep(1)
rval, frame = vc.read()
keypress = cv2.waitKey(1)
flag=False
if keypress == ord('c'):
    flag = True
if keypress == ord('q'):
    break

vc.release()
cv2.destroyAllWindows()
cv2.waitKey(1)
vc.release()
```

A.4 INSTALLATION PROCEDURE

Anaconda is a free and open-source distribution of Python and R programming languages for scientific computing, data science, and machine learning applications. It includes pre-installed packages and tools, as well as a package manager called conda. It simplifies the process of setting up a Python or R environment, and is available for Windows, macOS, and Linux.

Installation of Anaconda Navigator is as follows:

Step 1: Visit the Anaconda website and choose the required installer and start to download.

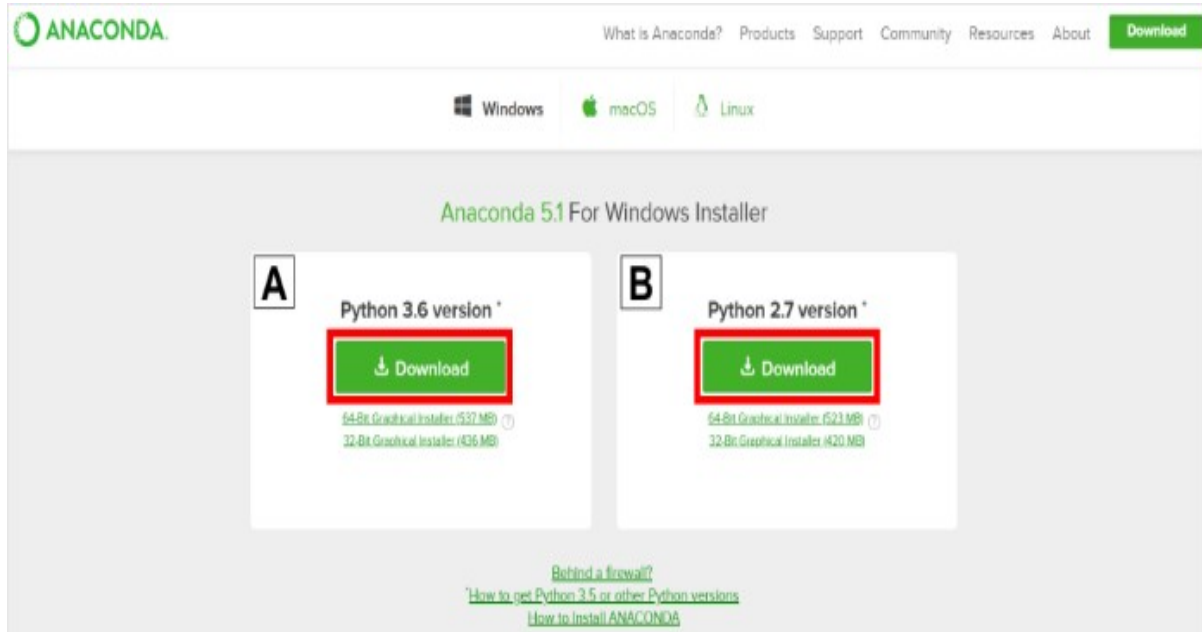


Fig A.3: Anaconda website

Step 2: Locate your download and double click it

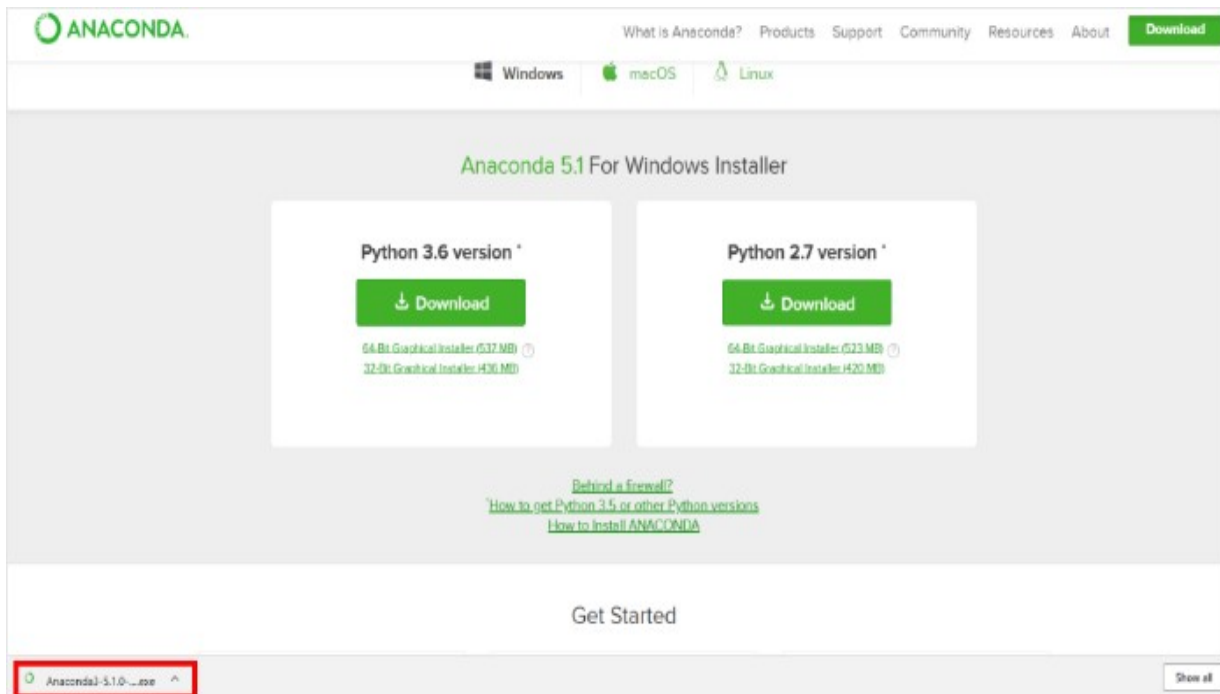


Fig A.4: Locate your download and double click it

Step 3: When the screen below appears, click on Next.



Fig A.5: Click on next

Step 4: Read the license agreement and click on I Agree.

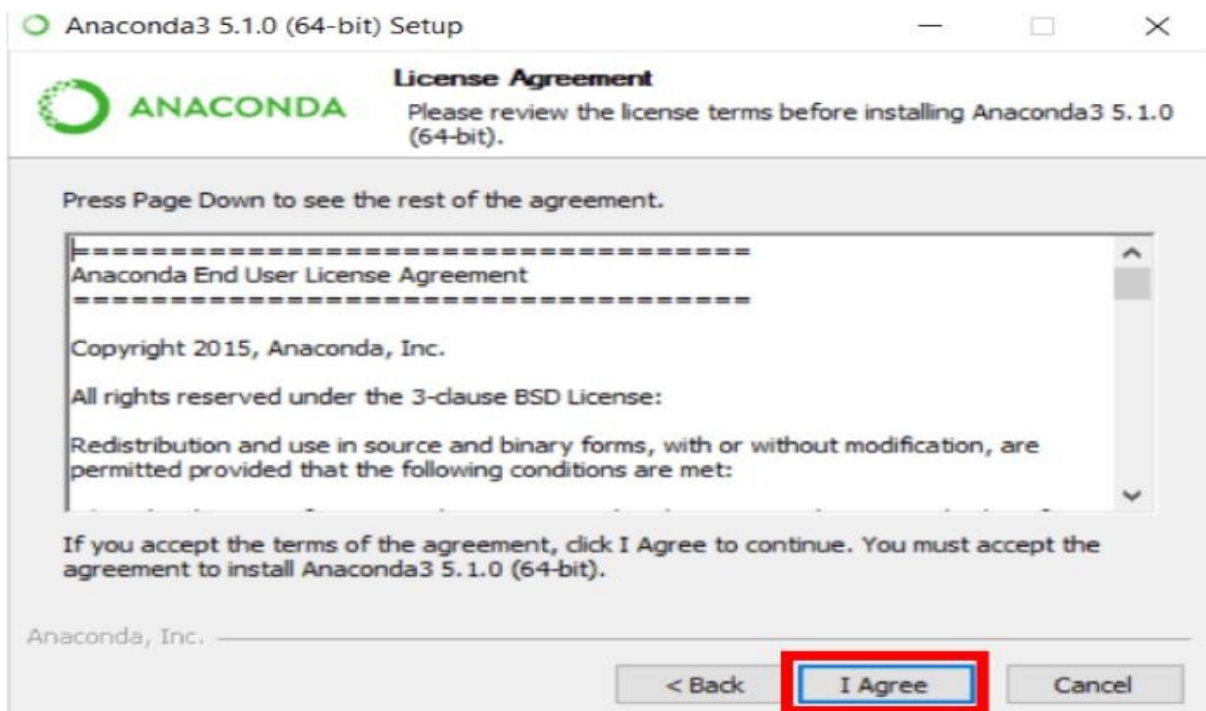


Fig A.6: Anaconda License Agreement

Step 5: Set the preference to “just me”.

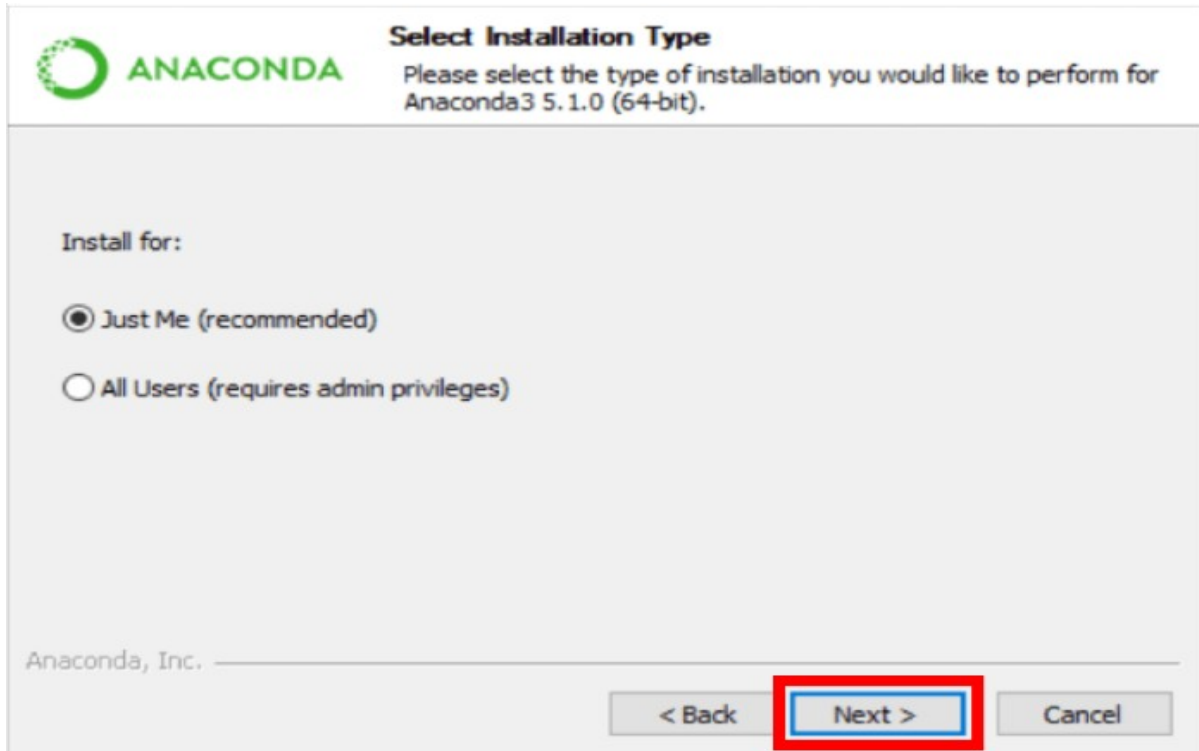


Fig A.7: Anaconda Installation type selection

Step 6: Click next

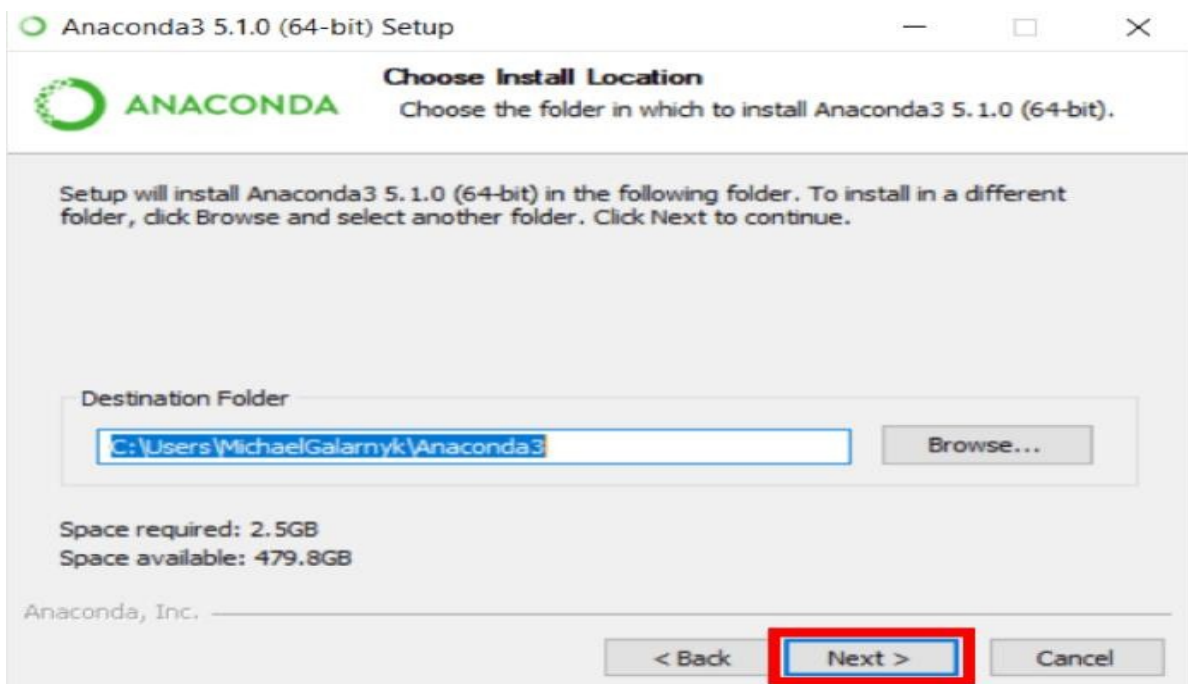


Fig A.8: Choose Installation Location

Step 7: Choose to register and click Install.



Fig A.9: Advanced Installation Options

Step 8: Wait till the installation process is completed.

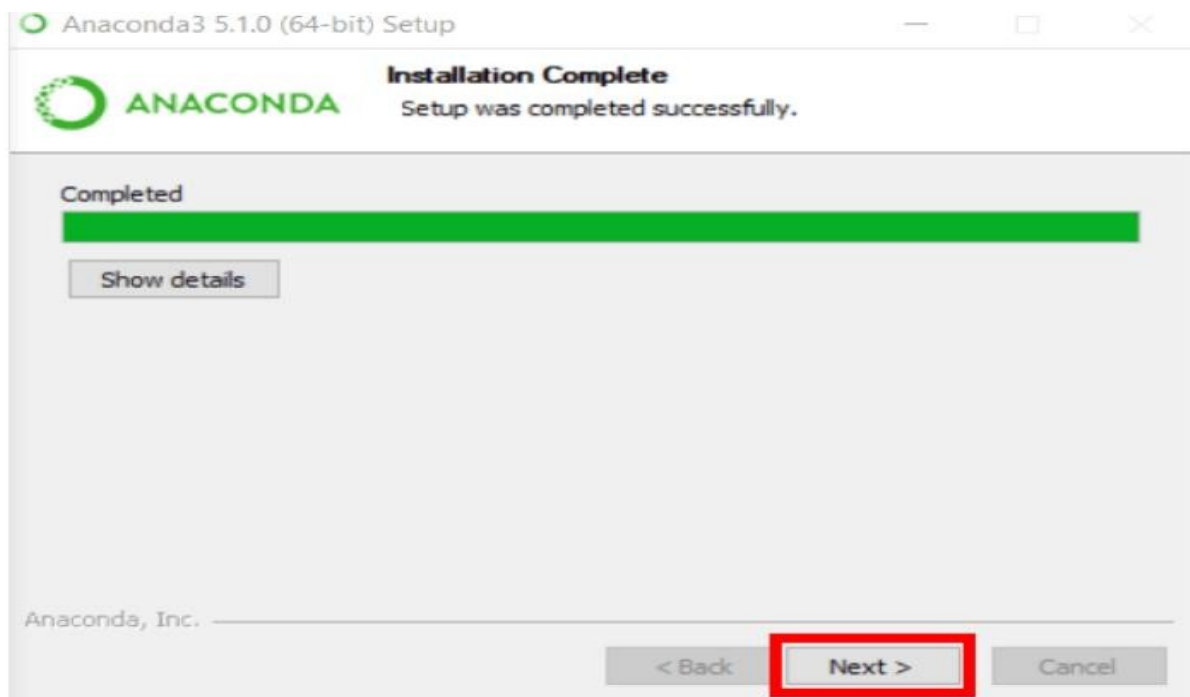


Fig A.10: Click on next

Step 9: Click finish to complete the installation process.

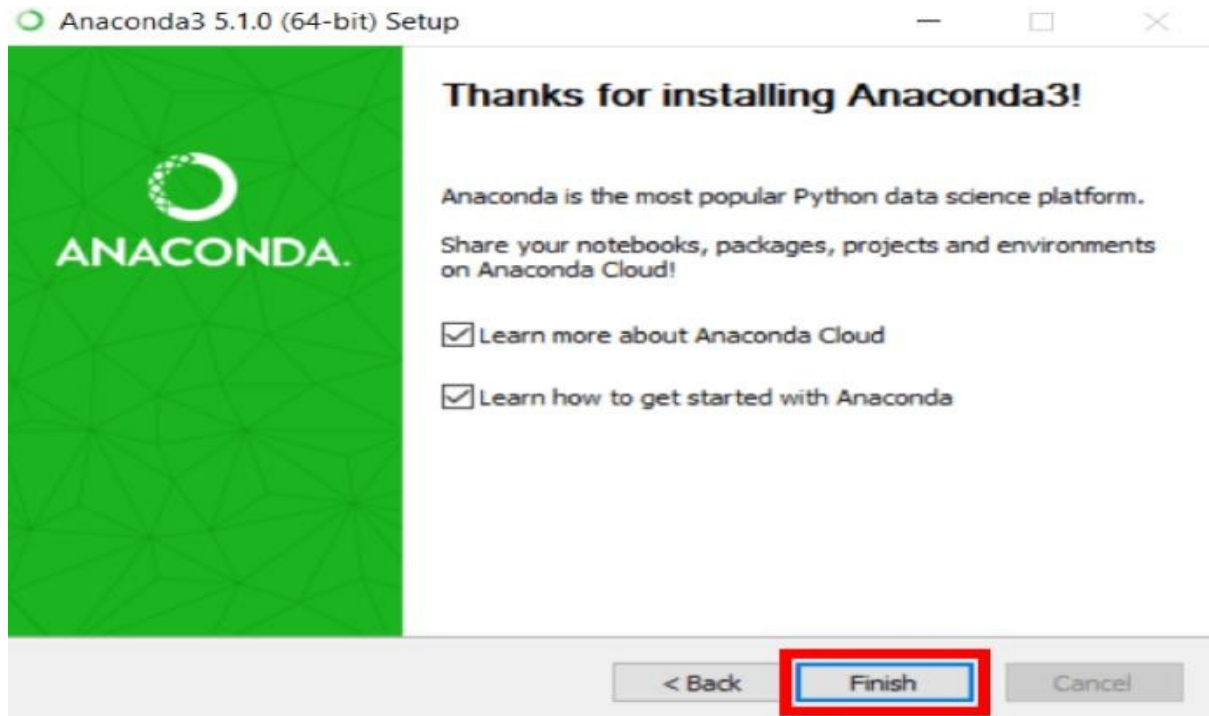


Fig A.11: Installation Complete

