CONTENTS

CHAPTER	PAGE NO.
ABSTRACT	
1. INTRODUCTION	**1**
1.1. ABOUT THE PROJECT	1
1.2. PROJECT DESCRIPTION	1
1.3. MODULES	4
2. SYSTEM ANALYSIS	**6**
2.1. DOMAIN ANALYSIS	6
2.2. REQUIREMENT ANALYSIS	6
2.2.1. Functional Requirements	6
2.2.2. Non-Functional Requirements	7
2.2.3. User Interfaces	7
2.2.4. Software Interfaces	7
2.2.5. Manpower Requirements	7
2.3. EXISTING SYSTEM	8
2.3.1. Disadvantages	8
2.4. PROPOSED SYSTEM	9
2.4.1. Advantages	9
3. FEASIBILITY STUDY	**11**
3.1. TECHNICAL FEASIBILITY	11
3.2. ECONOMICAL FEASIBILITY	11
3.3. OPERATIONAL FEASIBILITY	12
4. SYSTEM REQUIREMENTS	**13**
4.1. SOFTWARE REQUIREMENTS	13
4.2. HARDWARE REQUIREMENTS	13

CHAPTER	PAGE NO.
5. SYSTEM DESIGN	**14**
5.1. UML DIAGRAM'S INTRODUCTION	14
5.2. SYSTEM DESIGN ASPECTS	14
5.2.1. Design of Code	15
5.2.2. Design of Input	15
5.2.3. Design of Output	15
5.2.4. Design of Control	15
5.3. UML DIAGRAMS	15
5.3.1. Use Case Diagram	24
5.3.2. Class Diagram	25
5.3.3. Sequence Diagram	26
5.3.4. Activity Diagram	27
6. OVERVIEW OF TECHNOLOGIES	**28**
6.1. PYTHON TECHNOLOGY	28
6.2. HISTORY OF PYTHON	30
7. OVERVIEW OF OCR IN CNN	**34**
8. IMPLEMENTATION	**36**

CHAPTER	PAGE NO.
9. SOURCE CODE	**38**
10. TESTING	**43**
10.1. SOFTWARE TESTING TECHNIQUES	43
10.1.1. Testing Objectives	43
10.1.2. Test Case Design	43
10.2. SOFTWARE TESTING STRATEGIES	44
10.2.1. Unit Testing	44
10.2.2. Integration Testing	44
10.2.3. Validation Testing	45
10.2.4. System Testing	47
10.2.5. Security Testing	47
10.2.6. Performance Testing	47
11. OUTPUT SCREENS	**48**
12. CONCLUSION	**51**
13. FUTURE SCOPE	**52**
REFERENCES	

FIG NO.	NAME OF THE FIGURE	PAGE NO.

Fig 5.1.	Class Diagram	5
Fig 5.2.	Use Case Diagram	5
Fig 5.3.	Sequence Diagram	6
Fig 5.4.	Activity Diagram	13
Fig 5.5.	Collaboration Diagram	14
Fig 5.6	Deployment Diagram	14
Fig 5.7.	State Chart Diagram	15
Fig 5.8.	Component Diagram	15
Fig 5.9	Data Flow Diagram	16
Fig 5.10.	Use case Diagram for Overall Project	17
Fig 5.11.	Class Diagram for Overall Project	17
Fig 5.12.	Sequence Diagram for Overall Project	17
Fig 5.13.	Activity Diagram for Overall Project	17

LIST OF SCREENS

SCREEN NO.	SCREEN NAME	PAGE NO.
11.1	Actual Tire	48
11.2	Predicted Text from the Tire	48
11.3	Actual Tire	49
11.4	Predicted Text from the Tire	49
11.5	Accuracy for the first text	50
11.6	Accuracy for the second text	50

\

LIST OF ABBREVIATIONS

ABBREVIATION	FULL FORM
OCR	Optical Character Recognition
OpenCV	Open-Source Computer Vision Library
CNN	Convolution Neural Networks
Numpy	Numerical Python

ABSTRACT

Reading or Recognizing Text from Images is a challenging Task in the field of Computer Vision. This is mainly because Text in Images exhibit diversity and variability. Backgrounds of Images are virtually unpredictable as some images may have dark background, some may have light background and there are even cases where image fonts and background have same color with a small outline differentiating Text from the Background. There might be patterns of Text appearing in the image with different fonts, orientations, and varying lengths. The Text in Images can also be blurred or distorted

Tire text detection refers to the process of identifying and extracting text from images of tires. This can be useful for a variety of purposes, such as automatically reading the size and type of a tire, or extracting other information such as the manufacturer and model number. There are various techniques for tire text detection using computer vision algorithms and machine learning techniques, which are trained on a dataset of tire images with annotated text. In this project, we will consider Convolution Neural Network algorithm for identification of text in tire images. The extracted text can then be used for tire identification, tracking, and other applications. The accuracy of the CNN will be measured using various metrics.

1. INTRODUCTION

1.1. ABOUT THE PROJECT

The Tire Text Detection and Analysis project aims to develop a computer vision system that can accurately identify and extract text from images of tires. The project utilizes advanced image processing techniques, optical character recognition (OCR), and machine learning algorithms to automate the process of reading and analyzing tire text. By extracting important information such as tire size, brand, model, and other details, this project seeks to enhance efficiency, accuracy, and automation in the tire industry.

1.2. PROJECT DESCRIPTION

The Tire Text Detection and Analysis project involves building a robust system that can detect and extract text from tire images. The project will utilize state-of-the-art computer vision algorithms to identify and locate text regions on tire sidewalls. Various preprocessing techniques will be applied to enhance the quality and clarity of the images, ensuring accurate text extraction.

Next, the project will employ OCR techniques to recognize and interpret the extracted text. This may involve the development and training of machine learning models, such as convolutional neural networks (CNNs), to accurately recognize alphanumeric characters and symbols specific to tire text. The project will also involve handling challenges such as variations in font styles, sizes, and orientations commonly found on tire sidewalls.

Furthermore, the project will focus on extracting and analyzing the relevant information from the recognized text. This includes determining the tire brand, model, size, type, construction, wheel size, load rating, speed rating, manufacturing date, and other important parameters. The extracted information will be organized and made available for further analysis, tire identification, inventory management, and other potential applications.

What does Tire Text say?

At first glance, all those numbers and letters on tire sidewalls seem impossible to comprehend. However, each section conveys important information about the tire's type, size, and more. Here's a look at what all that tire text means.

Brand and model

The biggest type on a tire's sidewall will indicate its brand and model. This information is especially useful when buying a used car — we should make sure that all four tires match.

Tire type

Figuring out tires' brand and model was easy, but the next text we are likely to encounter is something that looks like a jumble of letters and numbers — for example, "P215/65 R15." Well, let's start breaking it down. The letter "P" tells us that the tire is designed for a passenger vehicle. There's also LT for light trucks and T for temporary (spare) tires.

Tire width

Continuing with our "P215/65 R15" example, the next thing we see is the number 215. This tells us the width of our tire in millimeters when it's measured from sidewall to sidewall.

Sidewall height

After the forward slash, the next number on our example tire is 65. This is the tire's aspect ratio, or its percentage of width to sidewall height. In this case, 65 means that the tire's height is 65% of its width. Multiply 215 by 0.65, and we get 139.75 millimeters. Tires with a lower aspect ratio are generally built for higher performance, while tires with a higher aspect ratio generally provide a more comfortable ride.

Tire construction

The next part of our "P215/65 R15" example is the letter R. This means that the tires are built with radial construction. Older tires sometimes have the letter D for "diagonal bias" or B for "bias belted."

Wheel size

The final part of "P215/65 R15" is the number 15. This indicates that the tire is designed to fit a 15-inch rim.

Maximum load and speed ratings

The next text we see is a number-letter combination. The number is the load rating, or the amount of weight that it can handle. The letter is the speed rating, or the maximum speed the tire is built for. As an example, a tire rated 90R could safely carry 1,323 pounds and travel at 106 mph. Consult a load and speed index to find out what our tire's specific rating means.

Tire date

Our tire will also have a combination of letters and numbers called the DOT Code. The most important part of this combination is four digits that indicate when our tire was manufactured. For example, "2118" would mean that the tire was made during the 21st week of 2018.

Maximum inflation rating

This rating is generally expressed as both pounds per square inch (psi) and kilopascals (KPA).

OBJECTIVE

The main objective of the Tire Text Detection and Analysis project is to automate the process of extracting valuable information from tire images. By accurately detecting and analyzing tire text, the project aims to achieve the following objectives:

Automation: Develop a system that can automatically identify, locate, and extract text from tire images, eliminating the need for manual intervention and reducing human error.

Efficiency: Improve the efficiency of processes in the tire industry by automating tasks such as tire identification, inventory management, and quality control through the extraction of relevant text information.

Accuracy: Enhance the accuracy of tire information extraction by leveraging advanced computer vision techniques and machine learning algorithms, ensuring precise recognition and interpretation of tire text.

Standardization: Facilitate standardization in the tire industry by providing a consistent and reliable method for reading and analyzing tire text, promoting uniformity in data interpretation and decision-making.

Integration: Enable seamless integration of tire text detection and analysis technology into existing systems and workflows in the automotive, retail, logistics, and other relevant industries, optimizing operations and enhancing customer service.

1.3. MODULES

This project consists of the following modules

1 Preparing the Data set

2 Pre-processing

3 Building OCR (optical character recognition system) using CNN:

4 Measure the accuracy of model

The description for the various modules is as follows

1.3.1 Preparing the Data set

During the research process, we are collected few tire images with different quality, shooting angles and backgrounds. By including images with different qualities, such as varying resolutions or levels of noise, we aimed to simulate real-world scenarios and account for potential challenges in text detection.

1.3.2 Pre-processing

In order to successfully recognize the parameters on the tire, it is necessary to correct the contrast of the image so that the desired parameters stand out more clearly. For this purpose, 3 possible methods of changing the contrast were researched - Linear calculation, Histogram Equalization (HE) and Contrast Limited AHE.

We settled on the CLAHE (Contrast Limited Adaptive Histogram Equalization) method because it prevents the problem of noise amplification and performs well in improving the local contrast and enhancing the definition of edges.

1.3.3 Building OCR (optical character recognition system) using CNN:
Many OCRs are built using convolution neural networks and deep learning techniques. The process includes extraction of image features (like color and background of text) as encoded vectors followed by a recurrent network that uses these encoded features to predict the text embedded in the image.

- Any CNN based architecture requires data generation. So, in this process we generate different font images of a single character (both number and English alphabet) as a dataset. It helps to increase accuracy and get text correct.
- Firstly, the image is pre-processed using different filters. The filter is helpful to differentiate text from background.
- Next find the location of each character in the image individually.
- Now, each character can be taken as input for a neural network as encoded vectors (usually image is gridded into rectangles of some small sizes and taken as input in CNN). Now, the data generated and input undergo analysis to get text.
- Finally, we get each character from the image and can be combined based on their location in the image to get the final text.

1.3.4 Measure the accuracy of model

Compute the output text accuracy obtained for different images then print the text.

2. SYSTEM ANALYSIS

Analysis is a logical process. The objective of this phase is to determine exactly what must be done to solve the problem. Tools such as Class Diagrams, Sequence Diagrams, Data Flow Diagrams and data dictionary are used in developing a logical model of system.

2.1 DOMAIN ANALYSIS

Domain analysis is the process by which a software engineer learns background information, which helps to understand the problem. The word ‘domain’ in the case means the general field of business or technology in which the customers expect to be using the software. For this project, personal experiences of the team members with competing software were observed to understand the domain.

2.2 REQUIREMENT ANALYSIS

A requirement is a relatively short and concise piece of information, expressed as a fact. It can be written as a sentence or can be expressed using some kind of diagram.

2.2.1 Functional Requirements

Functional requirements describe what the system should do. The functional requirements can be further categorized as follows:

- What inputs the system should accept?
- What outputs the system should produce?
- What data the system must store?
- What are the computations to be done?

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and the steps are necessary to put transaction data in to a usable form for processing that can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The

input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

1. What data should be given as input?
2. How the data should be arranged or coded?
3. The dialog to guide the operating personnel in providing input.
4. Methods for preparing input validations and steps to follow when error occur.

2.2.2 Non-Functional Requirements

Non-functional requirements are the constraints that must be adhered during development. They limit what resources can be used and set bounds on aspects of the software's quality.

2.2.3 User Interfaces

The User Interface is a GUI developed using Python.

2.2.4 Software Interfaces

The main processing is done in Python and console application.

2.2.5 Manpower Requirements

5 members can complete the project in 2 – 4 months if they work fulltime on it.

2.3 EXISTING SYSTEM

Tyrexpert: This is a tire text detection system developed by researchers at the University of Wolverhampton. It uses computer vision algorithms and machine learning techniques to identify and extract text from tire images.

TyreID: This is a tire text detection system developed by researchers at the University of Southampton. It uses a combination of computer vision and machine learning algorithms to detect and extract text from tire images.

TyreVision: This is a tire text detection system developed by researchers at the University of Leeds. It uses a deep learning algorithm to identify and extract text from tire images

tire identification, tracking, and other applications.

2.3.1 Disadvantages

High cost and complexity: The development and implementation of tire text detection algorithms require specialized knowledge and expertise, which can be costly.

Dependence on image quality: The accuracy and reliability of tire text detection algorithms can be affected by the quality of the input images, such as lighting and resolution.

Limited applicability: Tire text detection algorithms are only applicable to detecting written text on tires and cannot be used for other types of object recognition.

Ethical and privacy concerns: The use of tire text detection algorithms raises concerns about privacy and ethical considerations, such as the potential for misuse or discrimination.

2.4 PROPOSED SYSTEM

Our proposed tire text detection system is based on the following approach:
OCR Tire: This is a tire text recognition system developed by researchers at the University of Modena and Reggio Emilia. It uses optical character recognition (OCR) technology to extract text from tire images.

Preprocessing: The tire images are preprocessed to improve the visibility of the text. This includes applying image enhancement techniques such as contrast adjustment and noise reduction.
Text detection: A convolutional neural network (CNN) is trained on a dataset of tire images with annotated text. The CNN is then used to detect and localize the text in the tire images.
Text extraction: The detected text is then extracted and formatted for further processing, such as tire identification and tracking.
Evaluation: The performance of the proposed system is evaluated on a dataset of tire images, and the results are compared to existing tire text detection systems.
The proposed system is expected to achieve high accuracy and robustness, and can be easily adapted to different tire types and brands

2.4.1 Advantages

Improved tire identification and tracking: By detecting and extracting text on tires, it is possible to accurately identify and track tires, which is useful for inventory management and quality control.

Enhanced accuracy and reliability: The use of computer vision and machine learning algorithms allows for accurate and reliable tire text detection, even in challenging conditions such as low lighting or low-resolution images.

Automation and efficiency: Tire text detection can be automated, which saves time and effort compared to manual text extraction. This can improve the efficiency and productivity of tire-related operations.

Flexibility and adaptability: Tire text detection algorithms can be easily adapted to different tire types and brands, making them suitable for a wide range of applications.

3. FEASIBILITY STUDY

All projects are feasible when provided with unlimited resources and infinite time. Unfortunately, the development of computer-based system or product is more likely plagued by a scarcity of resources and difficult delivery dates. It is both necessary and prudent to evaluate the feasibility of a project at the earliest possible time. Months or years of effort, thousands or millions of dollars, and untold professional embarrassment can be averted if an ill-conceived system is recognized early in the definition phase.

Feasibility and risk analysis are related in many ways. If project risk is great the feasibility of producing quality software is reduced. During product engineering, however, we concentrate our attention on four primary areas of interest.

3.1. TECHNICAL FEASIBILITY

The technical feasibility of a tire text detection using OCR project involves assessing whether the required technology and resources are available to implement the system effectively. In this case, it includes evaluating the capabilities of OCR algorithms, image processing techniques, and machine learning models to accurately detect and extract text from tire images. Additionally, the project requires access to a dataset of tire images with labeled text for training and evaluation purposes. It is essential to consider the computational requirements, such as processing power and memory, to handle the image analysis and OCR tasks efficiently. The availability of suitable software libraries and frameworks, such as OpenCV, TensorFlow, or PyTorch, that support OCR and image processing functionalities is also crucial. Overall, assessing the technical feasibility ensures that the necessary tools and technologies are available to develop a robust and accurate tire text detection system.

3.2. ECONOMICAL FEASIBILITY

The economical feasibility of a tire text detection using OCR project involves evaluating the cost-effectiveness and potential economic benefits of implementing such a system. It requires considering both the initial investment and the long-term operational costs. The initial investment includes expenses related to acquiring necessary hardware, software, and datasets, as well as the cost of developing or acquiring pre-trained OCR models. Additionally, the project may require hiring or training personnel with expertise in OCR, image processing, and machine learning.

Furthermore, assessing the potential economic benefits is crucial. This could include improved efficiency and accuracy in tire text detection, leading to reduced manual effort and associated labor costs in tasks such as tire identification, quality control, or inventory management. By automating these processes, the system can potentially increase productivity and reduce errors, thereby saving time and resources. Additionally, the project's economic feasibility should consider potential revenue generation opportunities if the system is intended for commercial use, such as licensing

the technology or offering it as a service to other businesses.

It is also important to evaluate the return on investment (ROI) and conduct a cost-benefit analysis to determine the project's financial viability. Factors such as the anticipated lifespan of the system, maintenance and upgrade costs, and potential scalability should be considered to ensure the project's economical feasibility. A comprehensive assessment of the economic aspects ensures that the tire text detection using OCR project aligns with the available budget and offers potential economic benefits to justify its implementation.

3.3. OPERATIONAL FEASIBILITY

Operational feasibility focuses on evaluating whether the tire text detection using OCR project can be successfully integrated into the operational environment and meet the operational requirements. It involves considering factors such as the usability and accessibility of the system. The project should have a user-friendly interface that allows users to easily input tire images and retrieve the recognized text. The system's performance and reliability are critical, as it should be able to process images in real-time or within an acceptable time frame. Integration with existing systems or processes, such as tire manufacturing or quality control workflows, should be feasible without disrupting the overall operational flow. Additionally, considering factors like cost-effectiveness, scalability, and support for multiple languages or tire types enhances the operational feasibility of the project. A comprehensive assessment of the operational aspects ensures that the tire text detection system can be practically implemented and seamlessly integrated into the operational environment.

4. SYSTEM REQUIREMENTS

4.1. SOFTWARE REQUIREMENTS

Operating system	: Windows 10 Ultimate.
Coding Language	: Python.
IDE	: Google Collaboratory.

4.2. HARDWARE REQUIREMENTS

Processor	:	Intel Core I5
Speed	:	2.1 GHZ
RAM	:	8 GB
Hard Disk	:	1 TB

5. SYSTEM DESIGN

5.1. UML DIAGRAMS INTRODUCTION

The unified modeling language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntax, semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective.

UML is specifically constructed through two different domains they are:

- UML Analysis modeling, this focuses on the user model and structural model views of the system.
- UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

5.2. SYSTEM DESIGN ASPECTS

Once the analysis stage is completed, the next stage is to determine in broad outline form how the problem might be solved. During system design, we are beginning to move from the logical to physical level.

System design involves architectural and detailed design of the system. Architectural design involves identifying software components, decomposing them into processing modules and conceptual data structures, and specifying the interconnections among components.

Detailed design is concerned with how to package processing modules and how to implement the processing algorithms, data structures and interconnections of standard algorithms, invention of new algorithms, and design of data representations and packaging of software products. Two kinds of approaches are available:

- Top-down approach
- Bottom-up approach

5.2.1. Design of Code

Since information systems projects are designed with space, time and cost saving in mind, coding methods in which conditions, words, ideas or control errors and speed the entire process. The purpose of the code is to facilitate the identification and retrieval of the information. A code is an ordered collection of symbols designed to provide unique identification of an entity or an attribute.

5.2.2. Design of Input

Design of input involves the following decisions

- Input data
- Input medium
- The way data should be arranged or coded
- Validation needed to detect every step to follow when error occurs

The input controls provide ways to ensure that only authorized users access the system guarantee the valid transactions, validate the data for accuracy and determine whether any necessary data has been omitted. The primary input medium chosen is display. Screens have been developed for input of data using HTML. The validations for all important inputs are taken care of through various events using JSP control.

5.2.3. Design of Output

Design of output involves the following decisions

- Information to present
- Output medium
- Output layout

Output of this system is given in easily understandable, user-friendly manner, Layout of the output is decided through the discussions with the different users.

5.2.4 Design of Control

The system should offer the means of detecting and handling errors.

Input controls provides ways per

- Valid transactions are only acceptable
- Validates the accuracy of data

- Ensures that all mandatory data have been captured

All entities to the system will be validated. And updating of tables is allowed for only valid entries. Means have been provided to correct, if any by change incorrect entries have been entered into the system they can be edited.

5.3. UML DIAGRAMS

Why We Use UML in projects?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. Simply, Systems design refers to the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements which can be done easily through UML diagrams.

In the project four basic UML diagrams have been explained among the following list:

- Class Diagram
- Use Case Diagram
- Sequence Diagram
- Activity Diagram
- Collaboration Diagram
- Deployment Diagram
- State Chart Diagram
- Component Diagram

Class Diagram

A Class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes.

This is one of the most important of the diagrams in development. The diagram breaks the class into three layers. One has the name, the second describes its attributes and the third its methods. A padlock to left of the name represents the private attributes. The relationships are drawn between the classes. Developers use the Class Diagram to develop the classes. Analyses use it to show the details of the system.

Architects look at class diagrams to see if any class has too many functions and see if they are required to be split.

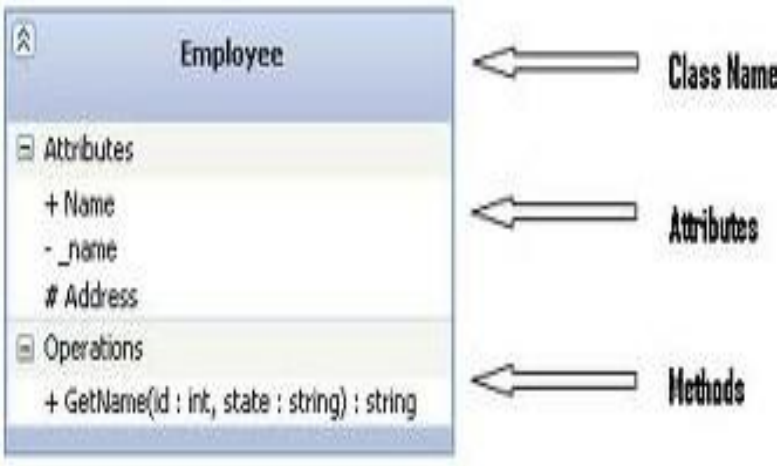


Fig.5.1: Class Diagram

Use Case Diagram

A Use Case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from the

external point of view. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system

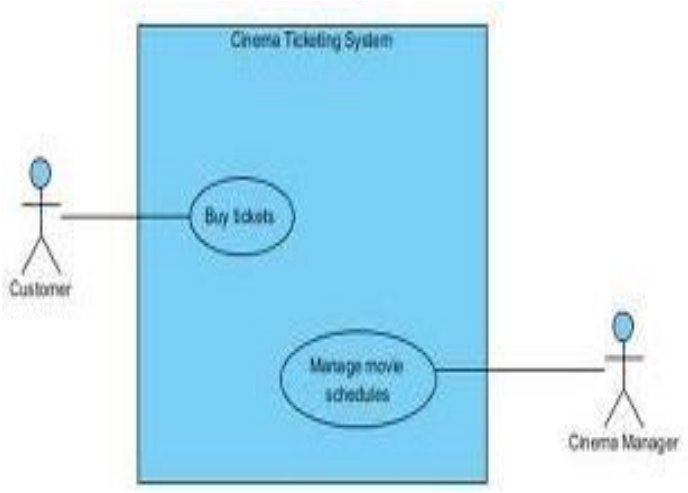


Fig.5.2: Use Case Diagram

Sequence Diagram

A Sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called Event-trace diagrams, event scenarios, and timing diagrams

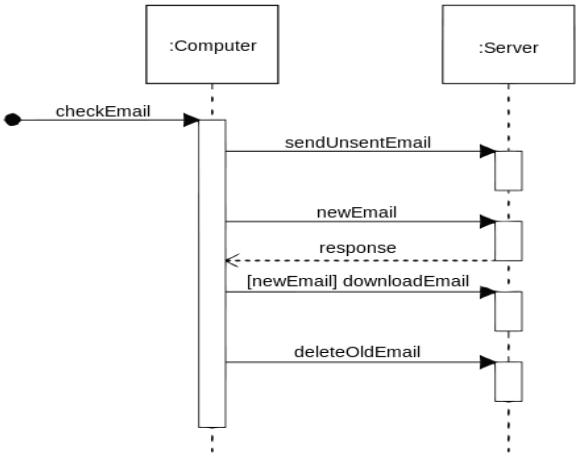


Fig.5.3: Sequence Diagram

Activity Diagram

Activity diagrams are a loosely defined diagram technique for showing workflows of stepwise activities and actions, with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to

describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

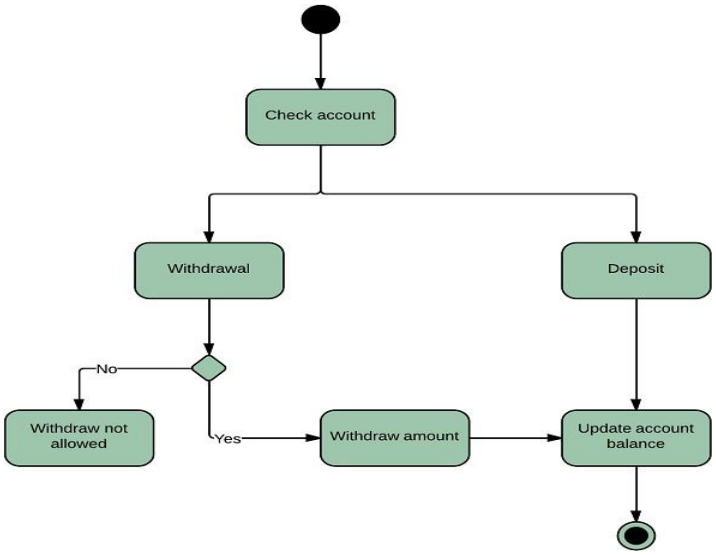


Fig.5.4: Activity Diagram

Collaboration Diagram

A Communication diagram models the interactions between objects or parts in terms of sequenced messages. Communication diagrams represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.

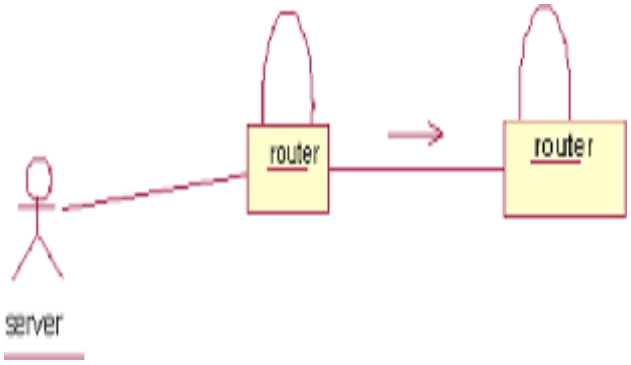


Fig.5.5: Collaboration Diagram

Deployment Diagram

A Deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes. To describe a web site, for example, a

deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected e.g., JDBC, REST

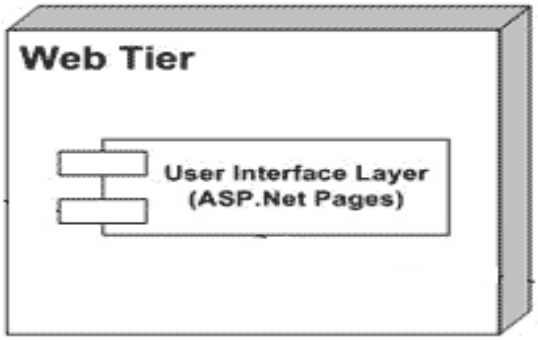


Fig.5.6: Deployment Diagram

State Chart Diagram

A State diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

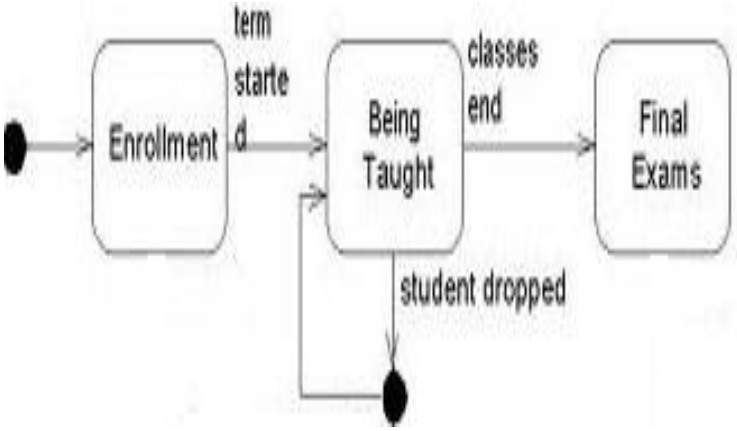


Fig.5.7: State Chart Diagram

Component Diagram

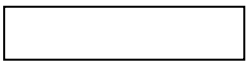
In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

Fig.5.8: Component Diagram

DATA FLOW DIAGRAM

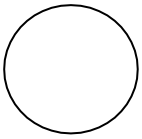
1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

DFD NOTATIONS

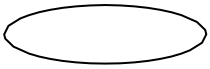


Define source and destination data.

Shows path of the data flow.



To represent a process that transforms or modifies the Data



To represent an attribute

Data Store

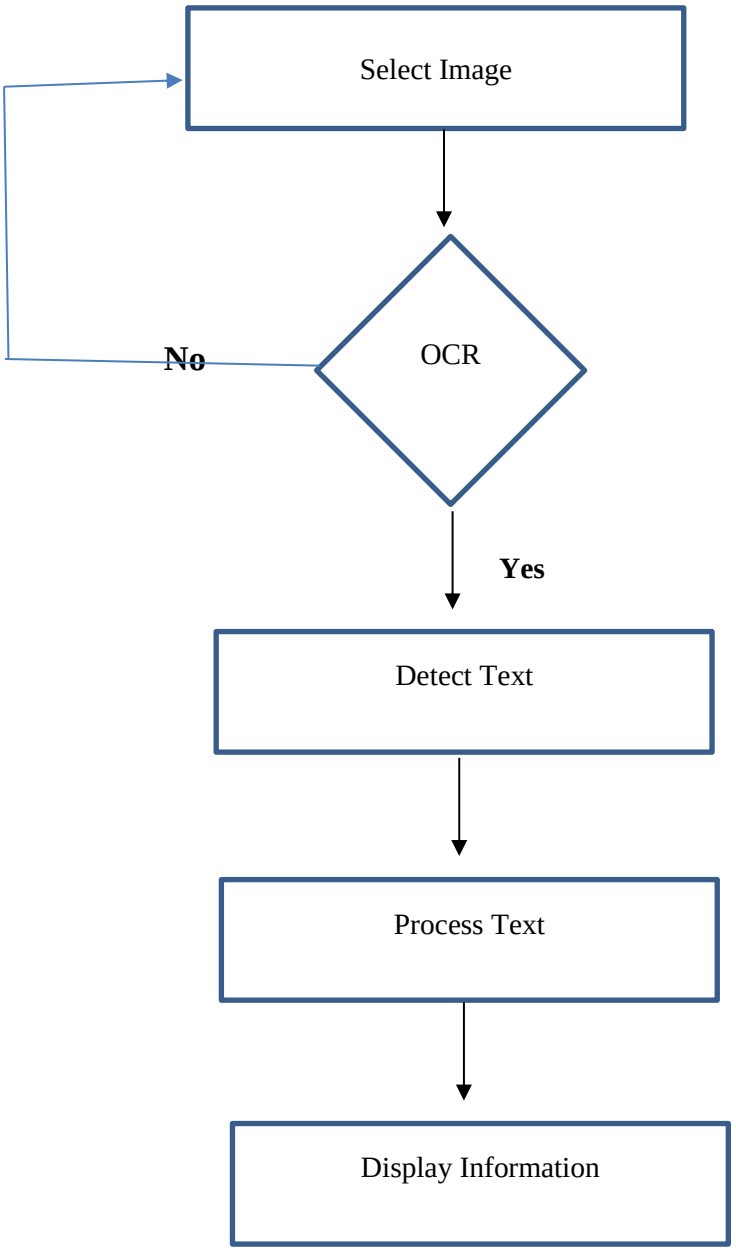


Fig.5.9: Data Flow Diagram

UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

5.3.1. USE CASE DIAGRAM:

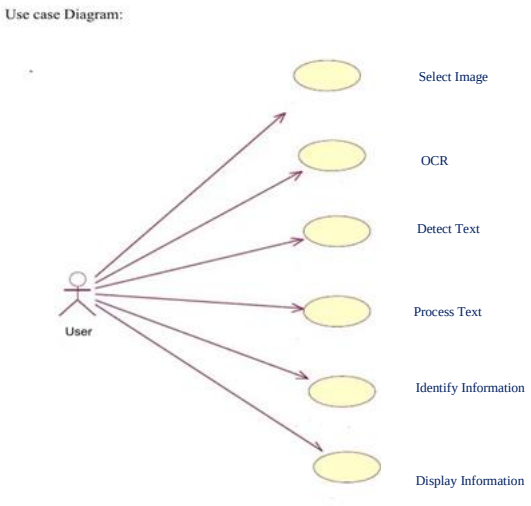


Fig.5.10. Use case Diagram for Overall Project

5.3.2. CLASS DIAGRAM:

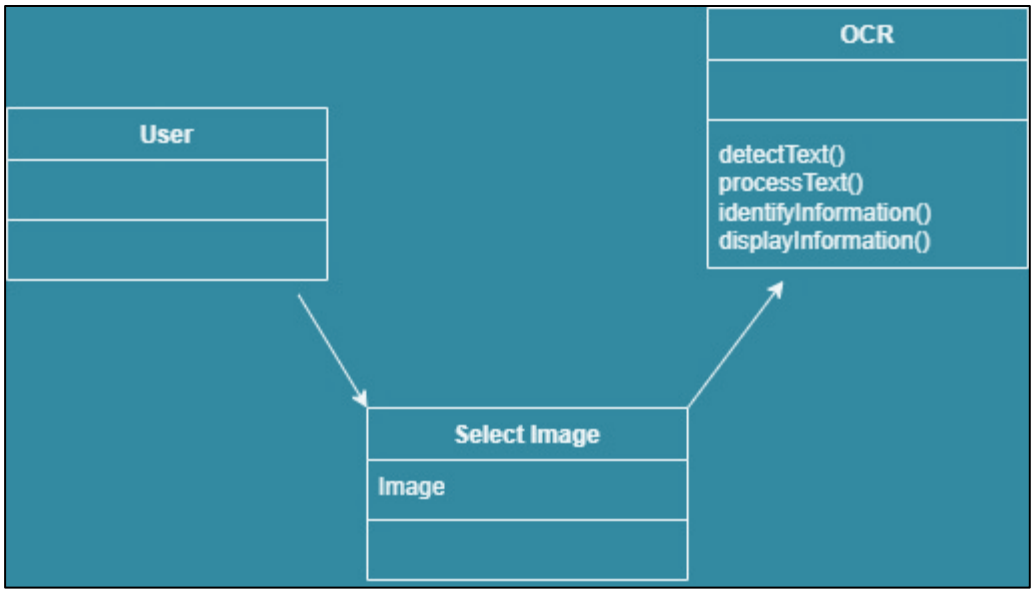


Fig 5.11. Class Diagram for Overall Project

5.3.3. SEQUENCE DIAGRAM:

A Sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

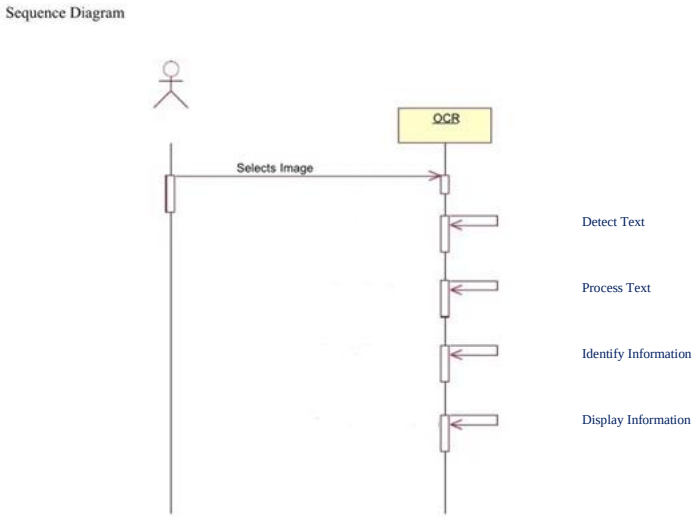


Fig 5.12. Sequence Diagram for Overall Project

5.3.4. ACTIVITY DIAGRAM:

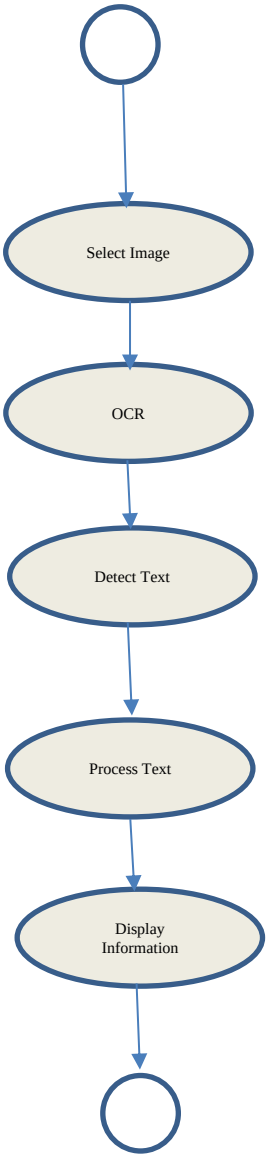


Fig 5.13. Activity Diagram for Overall Project

6. OVERVIEW OF TECHNOLOGIES

6.1. Python

Below are some facts about Python. Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc. The biggest strength of Python is huge collection of standard libraries which can be used for the following –

- Machine Learning
- Image processing (like Opencv, Pillow)
- EasyOCR
- Numpy

Advantages of Python: -

Let's see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

2. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

3. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

4. Improved Productivity

The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

5. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

6. Simple and Easy

When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

7. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

8. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

9. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

10. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

11. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

6.2. HISTORY OF PYTHON

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked

as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it."Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

What is Machine Learning : -

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data. Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain.Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

Categories Of Machine Learning :-

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning. Supervised learning

involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. We will see examples of both types of supervised learning in the following section. Unsupervised learning involves modeling the features of a dataset without reference to any label, and is often described as "letting the dataset speak for itself." These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. We will see examples of both types of unsupervised learning in the following section.

Need for Machine Learning :-

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven't surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, "to make decisions, based on data, with efficiency and scale". Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can't do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

Challenges in Machines Learning :-

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of overfitting & underfitting − If the model is overfitting or underfitting, it cannot be represented well for the problem.

Curse of dimensionality − Another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

7. OVERVIEW OF OCR IN CNN

Optical Character Recognition (OCR) is a technology that enables the extraction and interpretation of text from images or scanned documents. It plays a crucial role in various applications, including document digitization, text extraction from images, and automated data entry.

Convolutional Neural Networks (CNNs) have emerged as a powerful tool for OCR due to their ability to effectively capture and analyze visual patterns in images. CNNs are a type of deep learning model specifically designed to process and analyze visual data, making them well-suited for tasks such as image recognition, object detection, and text extraction.

Here is an overview of how OCR works with CNNs:

Preprocessing: The OCR process typically begins with preprocessing the input image to enhance its quality and make it suitable for analysis. This may involve operations such as resizing, normalization, and noise removal to improve the clarity of the image and facilitate better text recognition.

Convolutional Layers: CNNs consist of multiple layers, including convolutional layers, which are responsible for learning and detecting visual patterns within the image. These layers apply filters (also known as kernels) across the input image, scanning it for local features such as edges, corners, or textures. The filters' weights are learned through training, allowing the network to automatically extract relevant features for text recognition.

Pooling Layers: Following the convolutional layers, pooling layers are typically included in CNN architectures. Pooling reduces the spatial dimensions of the input, extracting the most salient features while preserving important information. Common pooling techniques include max pooling and average pooling.

Fully Connected Layers: After the convolutional and pooling layers, the extracted features are flattened and passed through one or more fully connected layers. These layers learn the relationship between the extracted features and the target output (in this case, the recognized text).

Output Layer: The final layer of the CNN is the output layer, which produces the predictions or probabilities for each possible character or symbol. Depending on the OCR task, the output layer may utilize different activation functions such as SoftMax for multi-class classification or sigmoid for binary classification.

Training and Optimization: To train the CNN for OCR, a large dataset of labeled images containing text is required. The network is trained using techniques like backpropagation and gradient descent, adjusting the weights and biases of the network to minimize the difference between predicted and actual text labels. Optimization algorithms such as Adam or stochastic gradient descent (SGD) are commonly used to speed up the training process.

Recognition and Post-processing: Once the CNN is trained, it can be used for text recognition on new images. During this phase, the CNN processes the input image, extracts features, and produces predicted labels for each character or symbol in the text region. Post-processing techniques, such as language models or sequence decoding algorithms, may be employed to refine the results and improve overall accuracy.

By leveraging the power of CNNs, OCR systems can effectively recognize and extract text from images, enabling a wide range of applications across industries, including tire text detection, document processing, automated data entry, and more.

8. IMPLEMENTATION

In the implementation phase software development is concerned with translating design specifications into source code. The primary goal of implementation is to write the source code internal documentation so that conformance of the code to its specification can be easily verified, and so that debugging, testing and modifications are erased. This goal is achieved by making the source code as clear and straightforward as possible. Simplicity, clarity and elegance are the hallmarks of good programs. Obscurity, cleverness and complexity are indications of inadequate design and misdirected thinking.

Source code clarity is enhanced by strutted techniques, by good coding style, by appropriate documents, by go internal comments, and by the features provided in the modern programming languages.

The main aim of structured coding is to adhere to single entry, single exit constructs in the majority of situations since it allows one to understand program behavior by reading the code from beginning to end. Bust strict adherence to this construct may cause problems it raises concerns for the time and space efficiency of the code. In some cases, single entry and single exit programs will require repeated code segments or repeated subroutines calls. In such cases, the usage of this construct would prevent premature loop exits and branching to exception handling code. So, in certain situations we violate this construct to acknowledge the realities of implementation although our intent is not encouraging poor coding style.

In computer programming, coding style is manifest in the patterns used by programmers to express a desired action or outcome good coding style can overcome the deficiencies of primitive programming languages, while poor style can defeat the intent of an excellent language. The goal of good coding style is to provide easily understood straightforward, elegant code.

Every good coding style performs the following Do’s

- Introduce user-defined data types to model entities in the problem domain.
- Use a few standard, agreed-upon control statements.
- Hide data structures behind access functions.
- Use goto's in a disciplined way.

- Isolate machine dependencies in a few routines.
- Use indentation, parenthesis, blank lines and borders around comment blocks to enhance readability.
- Carefully examine the routines having fewer than 5 or more than 25 executable statements.

The following are the Don'ts of good coding style

- Avoid null then statements.
- Don't put nested loops very deeply.
- Carefully examine routines having more than five parameters.
- Don't use an identifier for multiple purposes.

Adherence implementation standards and guidelines by all programmers on a project results in a product of uniform quality. Standards were defined as those that can be checked by an automated tool. While determining adherence to a guideline requires human interpretation. A programming standard might specify items such as:

- The nested depth of the program constructs will not executed five levels.
- The go to statements will not be used.
- Subroutines lengths will not exceed 30 Lines.

Implementation was performed with the following objectives.

- Minimize the memory required.
- Maximize output readability or clarity.
- Maximize source text readability.
- Minimize the number of source statements.
- To ease modification of the program.
- To facilitate formal verification of the program.
- To put the tested system into operation while holding costs, risks and user irritation to minimum.

Supporting documents for the implementation phase include all base-lined work products of the analysis and design phase.

9. SOURCE CODE

```
#Importing Libraries

import cv2

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

# !pip install easyocr

import easyocr as ocr


#path = "//content//drive//MyDrive//Tyre_text_extraction//dataset"
path ="drive/My Drive/Dataset"

imgpath =  path + "tyre-1.jpg"

#imgpath = "/content/drive/MyDrive/Tyre_text_extraction/dataset/tyre-9.jpg"
imgpath = "drive/My Drive/Dataset/tyre-1.jpg"
img = cv2.imread(imgpath)

# cv2.imshow('Original image',img)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
th = 125
max_val = 255

ret, o1 = cv2.threshold(img, th, max_val, cv2.THRESH_BINARY)
ret, o2 = cv2.threshold(img, th, max_val, cv2.THRESH_BINARY_INV)
ret, o3 = cv2.threshold(img, th, max_val, cv2.THRESH_TOZERO)
ret, o4 = cv2.threshold(img, th, max_val, cv2.THRESH_TOZERO_INV )
ret, o5 = cv2.threshold(img, th, max_val, cv2.THRESH_TRUNC)

output = [img, o1, o2, o3, o4, o5]

titles = ['Original', 'Binary', 'Binary Inv',
          'Zero', 'Zero Inv', 'Trunc']

for i in range(6):
   plt.subplot(2, 3, i+1)
   plt.imshow(output[i], cmap='gray')
   plt.title(titles[i])
   plt.xticks([])
   plt.yticks([])

plt.show()
```

```
th = 0
max_val = 255

ret, o6 = cv2.threshold(img, th, max_val, cv2.THRESH_BINARY + cv2.THRESH_OTSU )
ret, o7 = cv2.threshold(img, th, max_val, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU )
ret, o8 = cv2.threshold(img, th, max_val, cv2.THRESH_TOZERO + cv2.THRESH_OTSU )
ret, o9 = cv2.threshold(img, th, max_val, cv2.THRESH_TOZERO_INV +
cv2.THRESH_OTSU )
ret, o10 = cv2.threshold(img, th, max_val, cv2.THRESH_TRUNC + cv2.THRESH_OTSU )

output = [img, o1, o2, o3, o4, o5]

titles = ['Original', 'Binary', 'Binary Inv',
          'Zero', 'Zero Inv', 'Trunc']
```

```
for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])

plt.show()



block_size = 413
constant = 2
th1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, block_size, constant)
th2 = cv2.adaptiveThreshold (img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, block_size, constant)

output = [img, th1, th2]

titles = ['Original', 'Mean Adaptive', 'Gaussian Adaptive']

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])

plt.show()



block_size = 413
constant = 2
th1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, block_size, constant)
th2 = cv2.adaptiveThreshold (img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, block_size, constant)
```

```
output = [img, th1, th2]

titles = ['Original', 'Mean Adaptive', 'Gaussian Adaptive']

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])

plt.show()


images = [img,o1,o2,o3,o4,o5,o6,o7,o8,o9,o10,th1,th2]
titles = ['original','o1','o2','o3','o4','o5','o6','o7','o8','o9','o10','th1','th2']

plt.imshow(img)


reader = ocr.Reader(['en'],model_storage_directory='.')
# result = reader.readtext(np.array(th1))

for i in range(len(images)):
  result = reader.readtext(np.array(images[i]))
  print(titles[i])
  for j in result:
    print(j[-2],j[-1])
  print("-----------------")


kernal = np.ones((3,3), np.uint8)

erosion = cv2.erode(img, kernal, iterations=1)

guassian = cv2.GaussianBlur(erosion, (5,5),cv2.BORDER_DEFAULT)

_,t = cv2.threshold(guassian, 90, max_val, cv2.THRESH_TRUNC + cv2.THRESH_OTSU )

mean = cv2.adaptiveThreshold(guassian, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, block_size, constant)
plt.imshow(t, cmap='gray')
plt.show()
```

```
plt.imshow(mean, cmap='gray')
result = reader.readtext(np.array(mean))
result
```

10. TESTING

10.1. SOFTWARE TESTING TECHNIQUES

Software Testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding, Testing presents an interesting anomaly for the software engineer.

10.1.1. Testing Objectives

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a probability of finding an as yet
3. undiscovered error.
4. A successful test is one that uncovers an undiscovered error.
5. These above objectives imply a dramatic change in view port.

Testing cannot show the absence of defects, it can only show that software errors are present.

10.1.2. Test Case Design

Any engineering product can be tested in one of two ways:

White Box Testing

This testing is also called as glass box testing. In this testing, by knowing the specified function that a product has been designed to perform test can be conducted that demonstrates each function is fully operation at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis Path Testing

- Flow graph notation
- Cyclomatic Complexity

Deriving test cases Control Structure Testing

- Condition testing
- Data flow testing
- Loop testing

Black Box Testing

In this testing by knowing the internal operation of a product, tests can be conducted to ensure that "all gears mesh", that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods
- Equivalence partitioning
- Boundary value analysis
- Comparison testing
- Graph matrices

10.2. SOFTWARE TESTING STRATEGIES

A Strategy for software testing integrates software test cases into a series of well planned steps that result in the successful construction of software. Software testing is a broader topic for what is referred to as Verification and Validation. Verification refers to the set of activities that ensure that the software correctly implements a specific function. Validation refers he set of activities that ensure that the software that has been built is traceable to customer's requirements.

10.2.1. Unit Testing

Unit testing focuses verification effort on the smallest unit of software design that is the module. Using procedural design description as a guide, important control paths are tested to uncover errors within the boundaries of the module. The unit test is normally white box testing oriented and the step can be conducted in parallel for multiple modules.

10.2.2. Integration Testing

Integration testing is a systematic technique for constructing the program structure, while conducting test to uncover errors associated with the interface. The objective is to take unit tested methods and build a program structure that has been dictated by design.

Top-Down Integration

Top down integrations is an incremental approach for construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control program. Modules subordinate to the main program are incorporated in the structure either in the breath-first or depth-first manner.

Bottom-up Integration

This method as the name suggests, begins construction and testing with atomic modules i.e., modules at the lowest level. Because the modules are integrated in the bottom up manner the processing required for the modules subordinate to a given level is always available and the need for stubs is eliminated.

Regression Testing

In this contest of an integration test strategy, regression testing is the re execution of some subset of test that have already been conducted to ensure that changes have not propagate unintended side effects.

10.2.3 Validation Testing

At the end of integration testing software is completely assembled as a package. Validation testing is the next stage, which can be defined as successful when the software functions in the manner reasonably expected by the customer. Reasonable expectations are those defined in the software requirements specifications. Information contained in those sections form a basis for validation testing approach.

Reasonable expectation is defined in the software requirement specification – a document that describes all user-visible attributes of the software. The specification contains a section titled “Validation Criteria”. Information contained in that section forms the basis for a validation testing approach.

Validation Test Criteria

Software validation is achieved through a series of black-box tests that demonstrate conformity with requirement. A test plan outlines the classes of tests to be conducted, and a test procedure defines specific test cases that will be used in an attempt to uncover errors in conformity with requirements. Both the plan and procedure are designed to ensure that all functional requirements are satisfied, all

performance requirements are achieved, documentation is correct and human-engineered; and other requirements are met.

After each validation test case has been conducted, one of two possible conditions exists: (1) The function or performance characteristics conform to specification and are accepted, or (2) a deviation from specification is uncovered and a deficiency list is created. Deviation or error discovered at this stage in a project can rarely be corrected prior to scheduled completion. It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

Configuration Review

An important element of the validation process is a configuration review. The intent of the review is to ensure that all elements of the software configuration have been properly developed, are catalogued, and have the necessary detail to support the maintenance phase of the software life cycle. The configuration review sometimes called an audit.

Alpha and Beta Testing

It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted. Strange combination of data may be regularly used; and output that seemed clear to the tester may be unintelligible to a user in the field.

When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements. Conducted by the end user rather than the system developer, an acceptance test can range from an informal “test drive” to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.

The beta test is conducted at one or more customer sites by the end user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a “live” application of the software in an environment that cannot be controlled by the developer. The customer records all problems that are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta test, the software developer makes modification and then prepares for release of the software product to the entire customer base.

10.2.4. System Testing

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all system elements have been properly integrated to perform allocated functions.

10.2.5. Security Testing

Attempts to verify the protection mechanisms built into the system.

10.2.6. Performance Testing

This method is designed to test runtime performance of software within the context of an integrated system.

11. OUTPUT SCREENS

Fig 11.1: Actual Tire

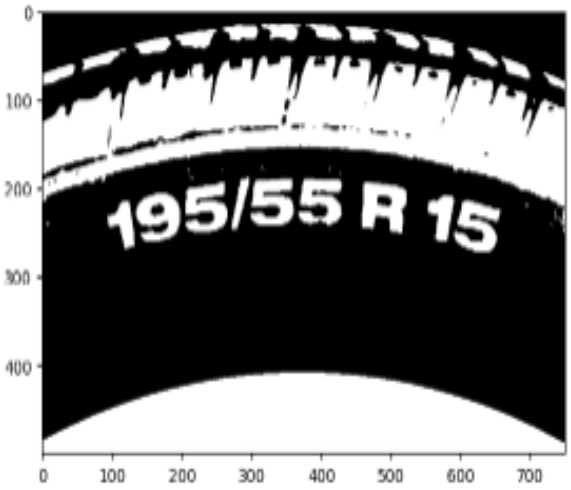


Fig 11.2: Predicted Text from the Tire

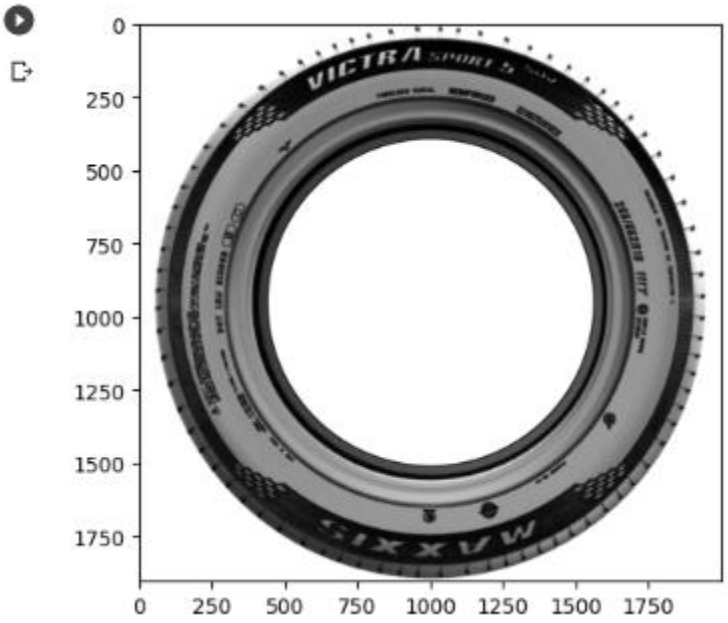


Fig 11.3: Actual Tire

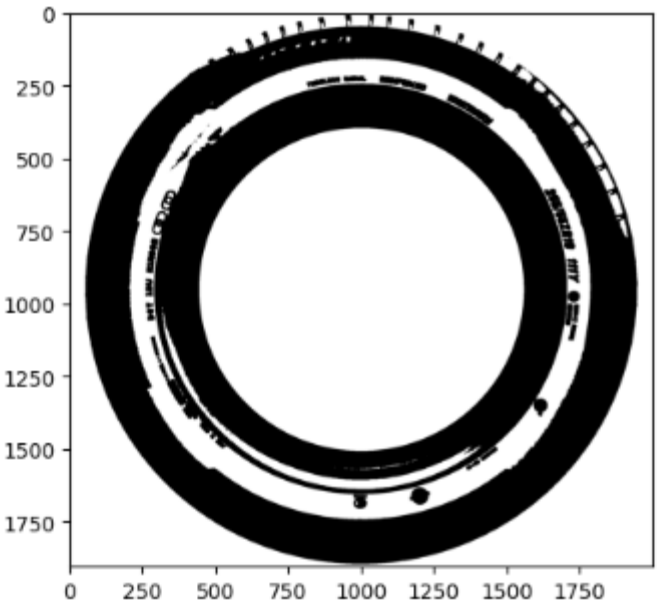


Fig 11.4: Predicted Text from the Tire

```
result
```

```
[([[79, 163], [451, 163], [451, 283], [79, 283]],
  '195/55',
  0.8976390514293622),
 ([[459.0578369393376, 176.04945092370735],
   [668.8775453144893, 213.52040911425178],
   [650.9421630606624, 288.95054907629265],
   [441.1224546855107, 251.47959088574822]],
  'r 15',
  0.9400584966654411)]
```

Fig 11.5: Accuracy for First Text

```
result
```

```
[([[239, 704], [323, 704], [323, 1074], [239, 1074]], '1', 0.921818210546931),
 ([[1691, 843], [1755, 843], [1755, 949], [1691, 949]],
  'R',
  0.04847038952451843),
 ([[1616.5944308127798, 602.0102870117646],
   [1670.344507703036, 584.0074212895619],
   [1742.4055691872202, 814.9897129882354],
   [1688.655492296964, 831.9925787104381]],
  '1',
  0.628641111964928)]
```

Fig 11.6: Accuracy for Second Text

12. CONCLUSION

In this project, we recognized the text on the tire and also can indicate what the text on the tire represents its usage, and its importance. We formulated a procedure to extract text from the tires using convolution neural networks that help to build an optical character recognition system. We also used various kernels to filter images in the pre-processing step. At last, we came up with all these mentioned ideas and formulated some lines of code in Python language which moderately succeeded in detecting the text. Moreover, it can be improved by considering the problems that we come across.

13. FUTURE SCOPE

Future scope for the tire text detection project, particularly in cases where the tire is fully damaged, could involve exploring alternative techniques or complementary approaches to overcome the drawbacks. Additionally, the project could focus on developing advanced image processing algorithms to enhance the system's ability to extract text from heavily damaged or worn-out tires and also it involves extending the tire text detection system to operate in a dynamic motion vehicle detection scenario. This expansion would enable the system to detect and extract text from tire images captured in real-time while vehicles are in motion. By considering these avenues, the project can aim to provide more comprehensive and accurate tire identification solutions, even in challenging scenarios.

REFERENCES

1. Kazmi, W., Nabney, I., Vogiatzis, G., Codd, P., & Codd, A. (2020). Vehicle tyre detection and text recognition using deep learning. IEEE Transactions on Intelligent Transportation Systems. https://doi.org/10.1109/TITS.2020.2967316
2. Wajahat Kazmi1., et al. “Vehicle tire (tyre) detection and text recognition using deep learning”2019 IEEE 15th International Conference on Automation Science and Engineering (CASE).978-1-7281-0356-3/19/$31.00 2019 IEEE.
3. Tasneem Wahdan, et al. “TIRE TYPE RECOGNITION THROUGH TREADS PATTERN RECOGNITION AND DOT CODE OCR” Ubiquitous Computing and Communication Journal. The University of Jordan Amman 11942, Jordan.
4. Zhang, Y., et al. "TyreXpert: A tyre text detection system." International Conference on Machine Learning and Computing. Springer, Singapore, 2018.
5. Lomagno, F., et al. "OCRTyre: A tyre text recognition system." International Conference on Computer Vision Systems. Springer, Berlin, Heidelberg, 2015.
6. Tomasi, A., et al. "TyreID: A tyre text detection system." International Conference on Computer Vision Systems. Springer, Berlin, Heidelberg, 2014.
7. Lu, J., et al. "TyreVision: A deep learning approach for tire text detection." International Conference on Pattern Recognition. Springer, Berlin, Heidelberg, 2020.