Number Plate Recognition System

A COURSE PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Aadhya Mathur [RA2111026010367]
Kanupriya Johari [RA2111026010373]
Sharad Asawa [RA2111026010365]

Under the guidance of

Dr Karpagam M

Assistant Professor , Department of Computational Intelligence

In partial satisfaction of the requirement for

the course of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

with specialization in Artificial Intelligence and Machine Learning

S.R.M. Nagar, Kattankulathur, Chengalpattu District - 603203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that course project report titled **" Number Plate Recognition System"** is the bonafide work of **AADHYA MATHUR (RA2111026010367), SHARAD ASAWA (RA2111026010365), KANUPRIYA JOHARI (RA2111026010373)** of III Year/VI Sem B.tech(CSE-AIML) who carried out the course project work under my supervision for the course 18CSC305J-Artificial Intelligence in SRM Institute of Science and Technology during the academic year 2023-2024(Even sem).

SIGNATURE

Dr Karpagam M
Assistant Professor
Department of Computational Intelligence
SRM INSTITUE OF SCIENCE AND TECHNOLOGY

SIGNATURE

Dr.R.ANNIE UTHRA
Professor & Head
Department of Computational Intelligence
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

ABSTRACT

Number plate recognition (NPR) plays a vital role in modern vehicle management systems, enabling various applications such as traffic monitoring, parking management, and law enforcement. This project proposes an innovative approach to enhance number plate recognition using artificial intelligence (AI) techniques. By leveraging the capabilities of AI, this system aims to achieve accurate and efficient identification of vehicle number plates in diverse real-world scenarios.

The project begins by collecting a large dataset of annotated images of number plates, encompassing various environmental conditions, license plate designs, and vehicle types. This dataset is then utilized to train a deep learning model, specifically a convolutional neural network (CNN), to recognize and extract number plates from images.

By leveraging deep learning and OCR techniques, the proposed system offers a practical solution for accurate and efficient identification of vehicle number plates, paving the way for advanced applications in transportation and law enforcement domains.

TABLE OF CONTENTS

LIST OF FIGURES

Abbreviations

AI - Artificial Intelligence

ANPR - Automatic Number Plate Recognition

IP - Image Processing

I/P - Input

O/P - Output

Chapter 1
Introduction

1.1 Identification of Need

In this project, a Digital Image Processing-based prototype is developed. Actions such as Image Acquisition, enhancement that is pre-processing, Segmentation of the license plate and then application of OCR (Optical Character Recognition) is applied to store the number on text form. The plate number is displayed as text on the terminal using the principle of OCR with the help of pytesseract and Tesseract engine.

It is seen that the security forces and authorities face problems whenever security forces chase a vehicle or they can't catch a vehicle which broke traffic rules. Authorities find it very hectic on a busy day to log the vehicle numbers manually in a parking lot. So, in order to make the entire process autonomous, we can install this system so as to automatically detect the vehicle which breaks the traffic rules, take a picture of it and store the number in the database so as to find the respective owner afterwards. The system can be used in parking so as to take the picture of the vehicle and log the vehicle number in the database (or the cloud, if connected to the internet). This technology reduces the unnecessary hectic manual work required on any busy day, saves the labor cost and is far more efficient than humans. The number of any vehicle once obtained as text, can be displayed, saved in the database or can be searched through the entire database for the details. This project is so versatile that it can be used as an entire application once converted to a software or can be used as a part of any big project.

1.2 Preliminary Investigation

Automatic Number Plate Recognition systems are a proven solution for various security forces and administrative authorities around the world. The automatic number plate recognition (ANPR) system market in 2016 was valued at USD 1.78 Billion and is expected to reach USD 3.57 Billion by 2023, at a CAGR (Compound Annual Growth Rate) of 9.74% between 2017 and 2023. The base year considered for the study is 2016 and the forecast period is between 2017 and 2023. The research methodology used to estimate and forecast

the ANPR system market begins with capturing data on key vendor revenue through secondary research. The vendor offerings are considered to determine the market segmentation. This report provides a detailed analysis of the ANPR system market based on type, component, application, and geography. After arriving at the overall market size, the total market has been split into several segments and subsegments, which have been verified through the primary research by conducting extensive interviews of people holding key positions such as CEOs (Chief Executive Officers), VPs (Vice Presidents), directors, and executives. Market breakdown and data triangulation procedures have been employed to complete the overall market engineering process and arrive at the exact statistics for all segments and subsegments.

Note: Other designations include sales, marketing, and product managers. The 3 tiers of companies were defined on the basis of their total revenue as of 2020; Tier 3 ≤ USD 500 million, Tier 2 = USD 500 million to USD 5 billion, and Tier 1 ≥ USD 5 billion

Fig No: 1 : The breakdown of profiles of primaries is depicted

The traffic management application accounted for the largest market share in 2016. The increasing demand for ANPR systems in urban areas due to high traffic congestion is driving the growth of the ANPR system market for traffic management applications. The market for the electronic toll collection application is expected to grow at the highest rate between 2017 and 2023. The increasing adoption of vehicles and stringent government regulations by various countries for implementing electronic toll collection systems is driving the growth of the ANPR system market.

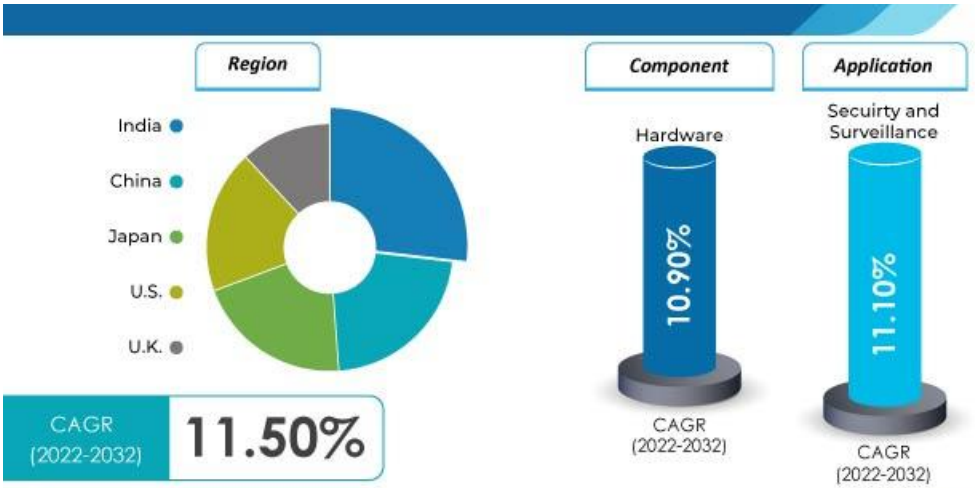

Fig No: 2 : ANPR System Market analysis

India held the largest share of the ANPR system market in 2016, and it is expected to grow at a moderate CAGR between 2017 and 2023. The market in APAC (Asia-Pacific) is estimated to grow at the highest rate during the forecast period. The demand for ANPR systems in APAC is expected to be driven by increasing public infrastructure and highways.

Inconsistency in number plate designs is the restraint for the ANPR market. Number plates differ in terms of their size or fonts in every part of the globe; due to which, it becomes difficult to construct an algorithm that would read all fonts without any discrepancy.

The automatic number plate recognition market is expected to exhibit high growth in near future across the globe, some of the major driving factors contributing to the growth are rising acceptance of smart parking concept in developed as well as developing countries and infrastructure growth in emerging countries. However, development of a shared platform for exchange of data from distinct sources and technological advancements are some of the future trends of the automatic number plate recognition market during the forecast period.

The emergence of the Automatic Number Plate Recognition concept was mainly focused upon monitoring the vehicles. Various countries of the world face crimes of several types and vehicular based crimes are on the rise due to ease of escape through vehicles. Curbing of such crimes had become necessary and hence the use of ANPR systems integrated into the security systems have proved to be of great use.

Chapter 2

Literature Survey

2.1 Existing System

Many developments in Digital Image Processing have been utilized in various fields with advances in Optical Character Recognition Technology as well. Various techniques of employing digital image processing have been developed in recent years. In the 2000s, OCR was made available online as a service (WebOCR), in a cloud computing environment, and in mobile applications like real-time translation of foreign-language signs on a smartphone. The best application of this technology would be to create a reading machine for the blind, which would allow blind people to have a computer read text to them out loud. Various commercial and open source OCR systems are available for most common writing systems, including Latin, Cyrillic, Arabic, Hebrew, Indic, Bengali (Bangla), and Devanagari, Tamil, Chinese, Japanese, and Korean characters.

The OCR engine used here is Tesseract OCR. Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the Apache License.Originally developed by Hewlett-Packard as proprietary software in the 1980s, it was released as open source in 2005 and development has been sponsored by Google since 2006.

In 2006, Tesseract was considered one of the most accurate open-source OCR engines then available. The Tesseract engine was originally developed as proprietary software at Hewlett Packard labs in Bristol, England and Greeley, Colorado between 1985 and 1994, with some more changes made in 1996 to port to Windows, and some migration from C to C++ in 1998. A lot of the code was written in C, and then some more was written in C++. Since then all the code has been converted to at least compile with a C++ compiler. Very little work was done in the following decade. It was then released as open source in 2005 by Hewlett Packard and the University of Nevada, Las Vegas (UNLV).

Tesseract development has been sponsored by Google since 2006. Tesseract was in the top three OCR engines in terms of character accuracy in 1995. It is available for Linux, Windows

and Mac OS X. However, due to limited resources it is only rigorously tested by developers under Windows and Ubuntu.

2.2 Existing System Flaws

Few Inconveniences of the Current Framework are

1. Constant human mediation.
2. High cost.
3. More Manpower is required.

2.3 Proposed System

The proposed system overcomes the above Disadvantages and apart from them has the beneath specified benefits.

1. Automated framework requiring less labor.
2. Number is displayed and with some modification can be stored in a database or be searched or processed.
3. The featured number plate is automatically cropped and displayed separately.

In one of the papers by Anisha Goyal and Rekha Bhatia, from Department of CSE ,Punjabi University Regional Centre for Information Technology and Management, Mohali, Punjab, India, they proposed that till now, all the LPR systems have been created using neural networks. They proposed to execute the system using Gabor filter, OCR and Vision Assistant to make the system quicker and more proficient. Different recognition strategies have been produced and number plate recognition systems are today used in different movement and security applications, for example, parking, access and border control, or tracking of stolen autos.

In another paper by Amr Badr, Mohamed M. Abdelwahab, Ahmed M. Thabet, and Ahmed M. Abdelsadek, proposed that Automatic recognition of car license plate number became a very important in our daily life because of the unlimited increase of cars and transportation systems which make it impossible to be fully managed and monitored by humans, examples are so many like traffic monitoring, tracking stolen cars, managing parking toll, red-light violation enforcement, border and customs checkpoints. This paper mainly introduces an

Automatic Number Plate Recognition System (ANPR) using Morphological operations, Histogram manipulation and Edge detection Techniques for plate localization and character segmentation. Artificial Neural Networks are used for character classification and recognition.

In the third paper by Hamed Sanghaei, an automatic and mechanized license and number plate recognition system is proposed that can extract license plate numbers of the vehicle passing through a given location using image processing algorithms. The resulting data is applied to compare with the records on a database. Experimental results reveal that the presented system successfully detects and recognizes the vehicle number plate on real images. This system can also be used for security and traffic control.

2.4 History

Digital Image Processing means processing digital image by means of a digital computer. We can also say that it is a use of computer algorithms, in order to get enhanced image either to extract some useful information. One of the first applications of digital image was in the newspaper industry, when pictures were first sent by submarine cable between London and New York. Introduction of the Bart lane cable picture transmission system in the early 1920s reduced the time required to transport a picture across the Atlantic from more than a week to less than three hours. Specialized printing equipment coded pictures for cable transmission and then reconstructed them at the receiving end. Some of the initial problems in improving the visual quality of these early digital pictures were related to the selection of printing producers and the distribution of intensity levels. In fact ,digital images require so much storage and computation power that progress in the field of digital image processing has been dependent on the development of digital computers and of supporting technologies that include data storage, display and transmission. The digital image is composed of a finite number of elements, each of which has a location and values. These elements are referred to as picture elements, image elements, pixels or pels. Pixels used to denote the element of a digital image. The process of acquiring an image of the area containing the text, pre-processing that image, extracting the individual characters , describing the character in the form suitable for computer processing & recognizing those individual characters are Digital Image Processing. Digital image processing techniques began in the late 1960s and early 1970s to be used in medical imaging, remote Earth resource observations and

astronomy. The invention in the early 1970s of computerized axial tomography (CAT) also called computerized tomography (CT) is one of the most important events of image processing in medical diagnosis. Computerized axial tomography is a process in which a ring of detectors encircles a patient and an X-Ray source, concentric with a detector ring, rotates about the patient. The X-ray passes through the object and is collected at the opposite end by the corresponding detectors in the ring. As the source rotates, this procedure is repeated. Image processing methods have successfully restored blurred pictures that were the only available records of rare artifacts lost or damaged after being photographed. Image processing methods have successfully restored blurred pictures that were the only available records of rare artifacts lost or damaged after being photographed.

Early optical character recognition may be traced to technologies involving telegraphy and creating reading devices for the blind. In 1914, Emanuel Goldberg developed a machine that read characters and converted them into standard telegraph code. Concurrently, Edmund Fournier d'Albe developed the Optophone, a handheld scanner that when moved across a printed page, produced tones that corresponded to specific letters or characters. In the late 920s and into the 1930s Emanuel Goldberg developed what he called a "Statistical Machine" for searching microfilm archives using an optical code recognition system.

In 1931 he was granted US Patent number 1,838,389 for the invention. The patent was acquired by IBM. With the advent of smart-phones and smart glasses, OCR can be used in internet connected mobile device applications that extract text captured using the device's camera. These devices that do not have OCR functionality built into the operating system will typically use an OCR API to extract the text from the image file captured and provided by the device. The OCR API returns the extracted text, along with information about the location of the detected text in the original image back to the device app for further processing (such as text-to-speech) or display. In 1951, a young Department of Defence engineer named David Shepard developed a scanning device in his home that he nicknamed ‘Gismo.’ This device could read twenty-three letters of the alphabet, interpret Morse Code, and read aloud letter by letter. While crude by today’s standards, it nevertheless captured the imagination of scientists and businessmen alike with its potential as a practical business tool in the data entry field. ‘Gismo’ garnered quite a bit of publicity and consequently spurred on the further development of OCR tools.

Chapter 3

Proposed Methodology

3.1 Problem Formulation

It is seen that the security forces and authorities face problems whenever security forces chase a vehicle or they can't catch a vehicle which broke traffic rules. Authorities find it very hectic on a busy day to log the vehicle numbers manually in a parking lot. So, in order to make the entire process autonomous, we can install this system so as to automatically detect the vehicle which breaks the traffic rules, take a picture of it and store the number in the database so as to find the respective owner afterwards. The system can be used in parking so as to take the picture of the vehicle and log the vehicle number in the database (or the cloud, if connected to the internet). This technology reduces the unnecessary hectic manual work required on any busy day, saves the labor cost and is far more efficient than humans. The number of any vehicle once obtained as text, can be displayed, saved in the database or can be searched through the entire database for the details. This project is so versatile that it can be used as an entire application once converted to a software or can be used as a part of any big project.

3.2 Project Objective

1. Image Acquisition using the computer's primary camera.
2. Image Enhancement and pre-processing to improve the quality of the image and convert the image to binary scale so as to use it in contour extraction.
3. Apply optical character recognition to display the license plate number from the picture as text.

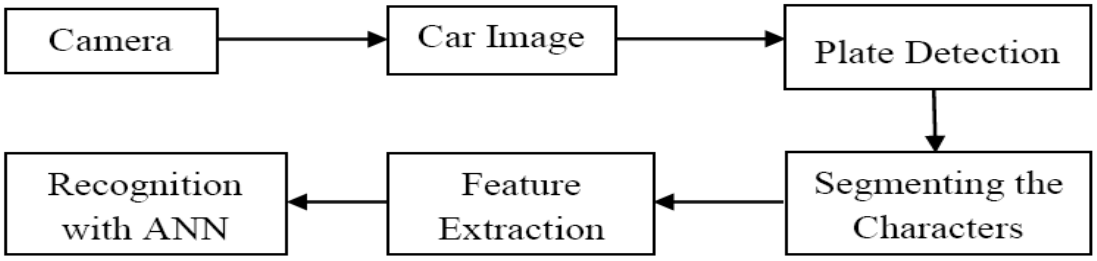

Fig No: 3 : Block Diagram of Input Unit

In the input unit, a preconfigured camera is arranged such that it is interfaced with a display device like a TFT monitor and a switch or a key is attached. Whenever the subject (the car) is in the frame of the picture, the key is pressed in order to capture the picture.

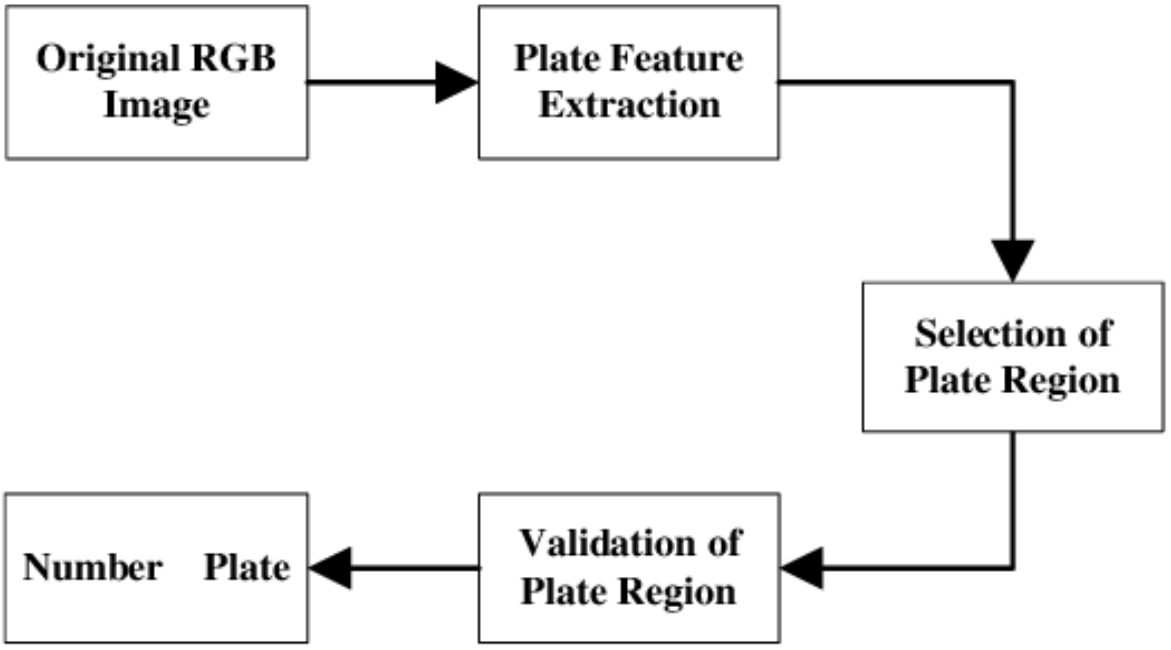

Fig No: 3.1: Block Diagram of output Unit

In the Output Unit, the input image is taken from the input unit and then processed in the processing unit. In the processing unit all sorts of enhancement and extraction is done and then the number on the license plate is extracted using Optical Character Recognition and then it can either be stored in a database or be displayed on a display device or both or can be used to excite an actuator.

3.2.1 Details of Processing

1. Basics of Digital Image Processing: The image of a vehicle whose number plate is to be recognised is taken from a digital camera which is then loaded to a local computer for further processing. OpenCV (Open Source Computer Vision) is a library

of programming functions mainly aimed at real-time computer vision. In simple language it is a library used for Image Processing. It is mainly used to do all the operations related to Images. Python, being a versatile language, is used here as a programming language. Python and its modules like Numpy, Scipy, Matplotlib and other special modules provide the optimal functionality to be able to cope with the flood of pictures. To enhance the number plate recognition further, we use a median filter to eliminate noises but it not only eliminates noise. It concentrates on high frequency also. So it is more important in edge detection in an image, generally the number plates are in rectangular shape, so we need to detect the edges of the rectangular plate. Image Processing mainly involves the following steps:

a. Image acquisition: This is the first step or process of the fundamental steps of digital image processing. Image Acquisition is the capturing of an image by any physical device (in this case the primary camera of the computer) so as to take the input as a digital image in the computer.
b. Image Enhancement: Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc. In this step the quality or rather the clarity of the input image is enhanced and the image is made clear enough to be processed.
c. Morphological Processing: Morphological operations apply a structuring element to an input image, creating an output image of the same size. The image is converted to a binary image, making it more to apply structural extraction to the image and extract any structure related to a particular mathematical model from it, in this case a license plate.
d. Segmentation: Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward a successful solution of imaging problems that require objects to be identified individually.
e. Representation: Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a

representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

f. Recognition: Recognition is the process that assigns a label, such as, “Plate” to an object based on its descriptors.

2. Optical Character Recognition: Optical character recognition or optical character reader, often abbreviated as OCR, is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast). The Techniques in OCR involve:

    1. Pre-processing: OCR software often "pre-processes" images to improve the chances of successful recognition. Techniques include:

        a. De-skew – If the document was not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counter clockwise in order to make lines of text perfectly horizontal or vertical.

        a. Despeckle – remove positive and negative spots, smoothing edges

        b. Binarization – Convert an image from color or grayscale to black-and-white (called a "binary image" because there are two colors). The task of binarization is performed as a simple way of separating the text (or any other desired image component) from the background. The task of binarization itself is necessary since most commercial recognition algorithms work only on binary images since it proves to be simpler to do so. In addition, the effectiveness of the binarization step influences to a significant extent the quality of the character recognition stage and the careful decisions are made in the choice of the binarization employed for a given input image type; since the quality of the binarization method employed to obtain the binary result depends on the type of the input image (scanned document, scene text image, historical degraded document etc.).

        c. Line removal – Cleans up non-glyph boxes and lines

        d. Layout analysis or "zoning" – Identifies columns, paragraphs, captions, etc. as

distinct blocks. Especially important in multi-column layouts and tables.

e. Line and word detection – Establishes baseline for word and character shapes, separates words if necessary.

f. Script recognition – In multilingual documents, the script may change at the level of the words and hence, identification of the script is necessary, before the right OCR can be invoked to handle the specific script.

g. Character isolation or "segmentation" – For per-character OCR, multiple characters that are connected due to image artifacts must be separated; single characters that are broken into multiple pieces due to artifacts must be connected.

h. Normalise aspect ratio and scale.

2. Character recognition: There are two basic types of core OCR algorithm, which may produce a ranked list of candidate characters. Matrix matching involves comparing an image to a stored glyph on a pixel-by-pixel basis; it is also known as "pattern matching", "pattern recognition", or "image correlation". This relies on the input glyph being correctly isolated from the rest of the image, and on the stored glyph being in a similar font and at the same scale. This technique works best with typewritten text and does not work well when new fonts are encountered. This is the technique the early physical photocell-based OCR implemented, rather directly. Feature extraction decomposes glyphs into "features" like lines, closed loops, line direction, and line intersections. The extraction features reduces the dimensionality of the representation and makes the recognition process computationally efficient. These features are compared with an abstract vector-like representation of a character, which might reduce to one or more glyph prototypes. General techniques of feature detection in computer vision are applicable to this type of OCR, which is commonly seen in "intelligent" handwriting recognition and indeed most modern OCR software.[24] Nearest neighbor classifiers such as the k-nearest neighbors algorithm are used to compare image features with stored glyph features and choose the nearest match.Software such as Cuneiform and Tesseract use a two-pass approach to character recognition. The second pass is known as "adaptive recognition" and uses the letter shapes recognised with high confidence on the first pass to

recognise better the remaining letters on the second pass. This is advantageous for unusual fonts or low-quality scans where the font is distorted (e.g. blurred or faded).[22]The OCR result can be stored in the standardised ALTO format, a dedicated XML schema maintained by the United States Library of Congress.For a list of optical character recognition software see Comparison of optical character recognition software.

3. Post-processing: OCR accuracy can be increased if the output is constrained by a lexicon – a list of words that are allowed to occur in a document. This might be, for example, all the words in the English language, or a more technical lexicon for a specific field. This technique can be problematic if the document contains words not in the lexicon, like proper nouns. Tesseract uses its dictionary to influence the character segmentation step, for improved accuracy. The output stream may be a plain text stream or file of characters, but more sophisticated OCR systems can preserve the original layout of the page and produce, for example, an annotated PDF that includes both the original image of the page and a searchable textual representation. "Near-neighbor analysis" can make use of co-occurrence frequencies to correct errors, by noting that certain words are often seen together. For example, "Washington, D.C." is generally far more common in English than "Washington DOC".

Knowledge of the grammar of the language being scanned can also help determine if a word is likely to be a verb or a noun, for example, allowing greater accuracy. The Levenshtein Distance algorithm has also been used in OCR post-processing to further optimize results from an OCR API.

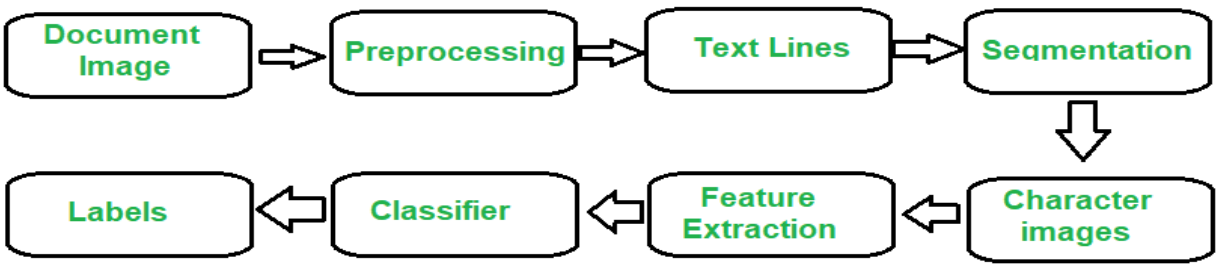

Fig 4. OCR WorkFlow Diagram

3.2.2 Working of the Proposed Methodology

0. In this project, a prototype of Digital Image processing and OCR are executed using different python libraries such as OpenCV 4.2 and pytesseract respectively.
1. The primary camera of the computer is accessed and the image is clicked after any key is pressed, when the vehicle is in the frame.
2. The input image is then fed in the system, for further processing.
3. The Morphed image and the input image and the morphed image is then displayed when any key is pressed. The Morphed image is obtained after morphological transformation.
4. After pressing one more key, the segmented plate is displayed from the morphed image in a new window which is performed using contour extraction on the morphed image.
5. The final step involves performing optical character recognition on the segmented plate using Tesseract engine and a library known as pytesseract in python. The vehicle number is displayed on the terminal and the plate region is highlighted in a new image in a new window, after pressing one more key.

3.3 Prerequisites of the system

3.3.1 Computer Specification:

0. Disk space: 1 GB
1. Operating systems: Windows 7 or later, MacOS, and Linux
2. Python versions: 3.7.5
3. Included development tools: conda, conda-env, Jupyter Notebook (IPython)
4. Compatible tools: Microsoft Visual Studio, PyCharm
5. Included Python packages: NumPy, SciPy, scikit-learn, pandas, Matplotlib, Numba, Intel Threading Building Blocks, pyDAAL, Jupyter, mpi4py, PIP, and others.
6. Processors: Intel Core i3 processor or later

3.3.2 Required Packages:

Python 3.7: Python is an interpreter, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages Python and CPython.

Syntax and semantics:

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation:

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the

program's semantic structure. This feature is also sometimes termed the off-side rule.

Statements and control flow: Python's statements include (among others):

1. The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing.
2. The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
3. The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
4. The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses.
5. The raise statement, used to raise a specified exception or re-raise a caught exception.
6. The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.

7. The def statement, which defines a function or method.
8. The with statement, from Python 2.5 released on September 2006, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.
9. The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.
10. The assert statement, used during debugging to check for conditions that ought to apply.
11. The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.
12. The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as<alias 2>], ...
13. The print statement was changed to the print() function in Python 3.

Expressions: Some Python expressions are similar to languages such as C and Java, while some are not:

1. Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division and integer division. Python also added the ** operator for exponentiation.
2. From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.
3. In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c.
4. Python uses the words and, or, not for its boolean operators rather than the

symbolic &&, ||, ! used in Java and C.

5. Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.
6. Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
7. Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).
8. Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.
9. Python features sequence unpacking where multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.
10. Python has a "string format" operator %. This functions analogous to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.

11. Python has various kinds of string literals:
12. Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".
13. Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
14. Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.
15. Python has array index and array slicing expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.
16. In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:
17. List comprehensions vs. for-loops
18. Conditional expressions vs. if blocks
19. The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python.

Methods:

Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

Libraries:

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

> Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

1. Graphical user interfaces
2. Web frameworks
3. Multimedia

4. Databases
5. Networking
6. Test frameworks
7. Automation
8. Web scraping[100]
9. Documentation
10. System administration
11. Scientific computing
12. Text processing
13. Image processing

Installation: Python 3.7 for Windows 7 64 bit MSI installer can be downloaded from python.org/downloads/release/ and installed accordingly. The same URL contains the installer for various other operating systems but this project is built on Windows 7.

After installation of python, add the path of python to the environment variables of the system by accessing the advanced System settings from the control panel.

1. Python Install Package (pip): pip is the package installer for Python. You can use pip to install packages from the Python Package Index and other indexes.

   Installing with get-pip.py

   To install pip, securely download get-pip.py:
   https://bootstrap.pypa.io/get-pip.py
   Then run the following:

   python get-pip.py

   One major advantage of pip is the ease of its command-line interface, which makes installing Python software packages as easy as issuing a command:

   pip install some-package-name

Users can also easily remove the package:

pip uninstall some-package-name

Most importantly, pip has a feature to manage full lists of packages and corresponding version numbers, possible through a "requirements" file.[4] This permits the efficient re-creation of an entire group of packages in a separate environment (e.g. another computer) or virtual environment. This can be achieved with a properly formatted requirements.txt file and the following command:

pip install -r requirements.txt

Install some package for a specific version python, where ${version} is replaced for 2, 3, 3.4, etc.:

pip${version} install some-package-name

1. NumPy (version 1.11.1 or higher) : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier. An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric, also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006. This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0. In 2011, PyPy started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy.

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

1. a powerful N-dimensional array object
2. sophisticated (broadcasting) functions
3. tools for integrating C/C++ and Fortran code
4. useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. NumPy is licensed under the BSD license, enabling reuse with few restrictions.

Comparison between Core Python and Numpy:

When we say "Core Python", we mean Python without any special modules, i.e. especially without NumPy.

The advantages of Core Python:

1. high-level number objects: integers, floating point
2. containers: lists with cheap insertion and append methods, dictionaries with fast lookup

Advantages of using Numpy with Python:

1. array oriented computing
2. efficiently implemented multi-dimensional arrays
3. designed for scientific computation

Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

```
import numpy as np

a = np.array([1, 2, 3]) # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2]) # Prints "1 2 3"
a[0] = 5
#Change an element of the array

print(a)

# Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])

#Create a rank 2 array print(b.shape)

#Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])
# Prints "1 2 4"
```

Numpy also provides many functions to create arrays:

```
import numpy as np

a = np.zeros((2,2))  # Create an array of all zeros
print(a)

# Prints "[[ 0.  0.]
#[ 0. 0.]]"

b = np.ones((1,2))    # Create an array of all ones
```

```
print(b) # Prints "[[ 1. 1.]]"
c = np.full((2,2), 7) # Create a constant array
print(c) # Prints "[[ 7. 7.]
#[ 7. 7.]]"

d = np.eye(2) # Create a 2x2 identity matrix
print(d) # Prints "[[ 1.  0.]
#[ 0. 1.]]"

e = np.random.random((2,2)) #Create an array filled with random values
print(e) #Might print "[[ 0.91940167  0.08143941]
#[ 0.68744134 0.87236687]]"
```

You can read about other methods of array creation in the documentation.

Array indexing:

Numpy offers several ways to index into arrays.

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1 2 3 4]

# [ 5 6 7 8]

# [ 9 10 11 12]]

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]

# [6 7]]

b = a[:2, 1:3]
```

```
# A slice of an array is a view into the same data, so modifying
it # will modify the original array.
print(a[0, 1]) # Prints "2"

b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])  # Prints "77"
```

You can also mix integer indexing with slice indexing. However, doing so will yield an array of lower rank than the original array:

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1 2 3 4]
# [ 5 6 7 8]

# [ 9 10 11 12]]

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Two ways of accessing the data in the middle row of the array.

# Mixing integer indexing with slices yields an array of lower
rank, # while using only slices yields an array of the same rank as
the
# original array:

row_r1 = a[1, :] # Rank 1 view of the second row of a
row_r2 = a[1:2, :] # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape) # Prints "[[5 6 7 8]] (1, 4)"

# We can make the same distinction when accessing columns of an array
col_r1 = a[:, 1]

col_r2 = a[:, 1:2]

print(col_r1, col_r1.shape) # Prints "[ 2 6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
#[ 6]
```

#[10]] (3, 1)"

Integer array indexing: When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.

# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]]) # Prints "[1 4 5]"
# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"
# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]]) # Prints "[2 2]"

# Equivalent to the previous integer array indexing
example print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

One useful trick with integer array indexing is selecting or mutating one element from each row of a matrix:

```
import numpy as np

# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a) # prints "array([[ 1, 2, 3],

#[ 4, 5, 6],

#[ 7, 8, 9],

#[10, 11, 12]])"

# Create an array of indices
b = np.array([0, 2, 0, 1])
```

```
#Select one element from each row of a using the indices in b

print(a[np.arange(4), b]) #Prints "[ 1  6  7 11]"
# Mutate one element from each row of a using the indices in b a[np.arange(4), b] += 10
print(a) # prints "array([[11, 2, 3],

#[ 4, 5, 16],

#[17, 8, 9],

#[10, 21, 12]])
```

Boolean array indexing: Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition. Here is an example:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2) # Find the elements of a that are bigger than 2;

# this returns a numpy array of Booleans of the same

# shape as a, where each slot of bool_idx tells
# whether that element of a is > 2.
print(bool_idx)      # Prints "[[False False]
#[ True True]

#[ True True]]"

#We use boolean array indexing to construct a rank 1 array

#consisting of the elements of a corresponding to the True values  #of bool_idx
print(a[bool_idx]) # Prints "[3 4 5 6]"

#We can do all of the above in a single concise statement:

print(a[a > 2])   # Prints "[3 4 5 6]"
```

For brevity we have left out a lot of details about numpy array indexing; if you want to know more you should read the documentation.

Data Types:

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric data types that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. Here is an example:

```
import numpy as np

x = np.array([1, 2]) # Let numpy choose the datatype
print(x.dtype)

# Prints "int64"
x = np.array([1.0, 2.0])
# Let numpy choose the data type print(x.dtype)
# Prints "float64"
x = np.array([1, 2], dtype=np.int64) # Force a particular data type
print(x.dtype)
# Prints "int64"
```

You can read all about numpy data types in the documentation. Array math:

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)

y= np.array([[5,6],[7,8]], dtype=np.float64)

# Element Wise sum; both produce the array

# [[ 6.0 8.0]
# [10.0 12.0]]
```

```
print(x + y) print(np.add(x, y))
# Element Wise difference; both produce the
array
# [[-4.0 -4.0]
# [-4.0 -4.0]]

print(x - y) print(np.subtract(x, y))
# Element Wise product; both produce the array
# [[ 5.0 12.0]
# [21.0 32.0]]

print(x * y) print(np.multiply(x, y))

# Element Wise division; both produce the array
# [[ 0.20.33333333]
# [ 0.42857143 0.5]]

print(x / y) print(np.divide(x, y))
# Element Wise square root; produces the array
# [[ 1.1.41421356]
# [ 1.73205081 2.]]

print(np.sqrt(x))
```

Note that unlike MATLAB, * is element wise multiplication, not matrix multiplication. We instead use the dot function to compute inner products of vectors, to multiply a vector by a matrix, and to multiply matrices. dot is available both as a function in the numpy module and as an instance method of array objects:

```
import numpy as np

x = np.array([[1,2],[3,4]])

y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
```

```
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29
67] print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2
array # [[19 22]
# [43 50]]

print(x.dot(y))

print(np.dot(x, y))
```

Numpy provides many useful functions for performing computations on arrays; one of the most useful is sum:

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x)) # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0)) # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1)) # Compute sum of each row; prints "[3 7]"
```

You can find the full list of mathematical functions provided by numpy in the documentation.

Apart from computing mathematical functions using arrays, we frequently need to reshape or otherwise manipulate data in arrays. The simplest example of this type of operation is transposing a matrix; to transpose a matrix, simply use the T attribute of an array object:

```
import numpy as np

x =np.array([[1,2], [3,4]])
print(x)

# Prints "[[1 2]
```

```
#[3 4]]"

print(x.T)

# Prints "[[1 3]

#[2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:

v = np.array([1,2,3])
print(v)

# Prints "[1 2 3]"
print(v.T) # Prints "[1 2 3]"
```

Numpy provides many more functions for manipulating arrays; you can see the full list in the documentation.

Broadcasting:

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

For example, suppose that we want to add a constant vector to each row of a matrix. We could do it like this:

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

v = np.array([1, 0, 1])

y = np.empty_like(x)

# Create an empty matrix with the same shape as x
```

```
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
y[i, :] = x[i, :] +v
# Now y is the following

# [[ 2  2 4]
#  [ 5  5 7]
#  [ 8 8 10]
# [11 11 13]]
print(y)
```

This works; however when the matrix x is very large, computing an explicit loop in Python could be slow. Note that adding the vector v to each row of the matrix x is equivalent to forming a matrix vv by stacking multiple copies of v vertically, then performing element wise summation of x and vv. We could implement this approach like this:

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

v = np.array([1, 0, 1])

vv = np.tile(v, (4, 1)) # Stack 4 copies of v on top of each other
print(vv)

# Prints "[[1 0 1]
#[1 0 1]

#[1 0 1]

#[1 0 1]]"

y = x + vv # Add x and vv elementwise
```

```
print(y) # Prints "[[ 2 2  4
#[ 5 5 7]

#[ 8 8 10]

#[11 11 13]]"
```

Numpy broadcasting allows us to perform this computation without actually creating multiple copies of v. Consider this version, using broadcasting:

```
import numpy as np

# We will add the vector v to each row of the matrix x, # storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

v = np.array([1, 0, 1])

y = x + v

# Add v to each row of x using broadcasting print(y) #
Prints "[[ 2  2  4]
#[ 5 5 7]

#[ 8 8 10]

#[11 11 13]]"
```

The line y = x + v works even though x has shape (4, 3) and v has shape (3,) due to broadcasting; this line works as if v actually had shape (4, 3), where each row was a copy of v, and the sum was performed element-wise.

Broadcasting two arrays together follows these rules:

If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s

until both shapes have the same length.

The two arrays are said to be compatible in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.

The arrays can be broadcast together if they are compatible in all dimensions.

After broadcasting, each array behaves as if it had a shape equal to the element-wise maximum of the shapes of the two input arrays.

In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension.

If this explanation does not make sense, try reading the explanation from the documentation or this explanation.

Functions that support broadcasting are known as universal functions. You can find the list of all universal functions in the documentation.

Here are some applications of broadcasting:

```
import numpy as np

# Compute outer product of vectors

v = np.array([1,2,3]) # v has shape (3,)
w = np.array([4,5])    # w has shape(2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield

# an output of shape (3, 2), which is the outer product of v and w:

# [[ 4 5]
# [ 8 10]

# [12 15]]

print(np.reshape(v, (3, 1)) * w)

# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])
# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3),
# giving the following matrix:
```

```
# [[2 4 6]

# [5 7 9]]

print(x + v)

# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast

# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column. Gives the following matrix:
# [[ 5 6 7]
# [ 9 10 11]]

print((x.T + w).T)

# Another solution is to reshape w to be a column vector of shape (2,1);
# we can then broadcast it directly against x to produce the same


# output.
print(x + np.reshape(w, (2, 1)))

# Multiply a matrix by a constant:

# x has shape (2, 3). Numpy treats scalars as arrays of shape
(); # these can be broadcast together to shape (2, 3), producing
the # following array:
# [[ 2 4 6]

# [ 8 10 12]]

print(x * 2)
```

Broadcasting typically makes your code more concise and faster, so you should strive to use it where possible.

Installation: Numpy 1.19.1 can be installed by using the pip command:

pip install numpy==1.19.1

After successful installation of numpy, one can open the python terminal in the command prompt and import numpy to test the version of numpy installed as follows:

3.8.1 Code initaiton

1. copy: In Python, Assignment statements do not copy objects, they create bindings between a target and an object. When we use = operator user thinks that this creates a new object; well, it doesn't. It only creates a new variable that shares the reference of the original object. Sometimes a user wants to work with mutable objects, in order to do that the user looks for a way to create "real copies" or "clones" of these objects. Or, sometimes a user wants copies that the user can modify without automatically modifying the original at the same time, in order to do that we create copies of objects.
    a. A copy is sometimes needed so one can change one copy without changing the other. In Python, there are two ways to create copies :
2. Deep copy
3. Shallow copy

In order to make these copies, we use a copy module. We use copy module deep copy operations.

Deep copy is a process in which the copying process occurs recursively. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. In case of deep copy, a copy of an object is copied to another object. It means that any changes made to a copy of an object do not reflect in the original object. In python, this is implemented using the "deepcopy()" function.

```
# importing "copy" for copy operations
import copy
 # initializing list 1
 li1 = [1, 2, [3,5],
 4]
 # using deepcopy to deep
 copy li2 = copy.deepcopy(li1)
```

```
# original elements of list

print ("The original elements before deep
copying") for i in range(0,len(li1)):
    print (li1[i],end="
") print("\r")
# adding and element to new
list li2[2][0] = 7
# Change is reflected in l2

print ("The new list of elements after deep copying ")
for i in range(0,len( li1)):
print (li2[i],end="")
print("\r")
# Change is NOT reflected in original list
# as it is a deep copy
print ("The original elements after deep copying")
for i in range(0,len( li1)):
print (li1[i],end=" ")
```

Installation: copy module in python is normally pre installed but if not, can be installed using pip commands as:

```
python –m pip install copy
```

Note: New python installers come hands on with a pre pre-installed copy module.

1. Pillow: Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7, with Python 3 support to be released "later".

   Development appears to be discontinued with the last commit to the PIL repository coming in 2011. Consequently, a successor project called Pillow has forked the PIL

repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian and Ubuntu (since 13.04).

Pillow offers several standard procedures for image manipulation. These include:

1. per-pixel manipulations,
2. masking and transparency handling,
3. image filtering, such as blurring, contouring, smoothing, or edge finding,
4. image enhancing, such as sharpening, adjusting brightness, contrast or color,
5. adding text to images and much more.

Some of the file formats supported are PPM, PNG, JPEG, GIF, TIFF, and BMP. It is also possible to create new file decoders to expand the library of file formats accessible. This example loads an image from the hard drive and blurs it.

```
from PIL import Image, ImageFilter # imports the library

original = Image.open("file.ppm") # load an image from the hard
drive blurred = original.filter(ImageFilter.BLUR) # blur the image
original.show() # display both images
blurred.show()
```

1. Installation:The project uses Pillow 2.2.1 as the Python image library and uses its functions. The pip statement to install Pillow in the system is :
    a. pip install Pillow==2.2.1
2. 3.8.5 installation cmd
3. pytesseract: Humans can understand the contents of an image simply by looking. We perceive the text on the image as text and can read it.Computers don't work the same way. They need something more concrete, organized in a way they can understand. This is where Optical Character Recognition (OCR) kicks in. Whether it's recognition of car plates from a camera, or hand-written documents that should be converted into a digital copy, this technique is very useful. While it's not always perfect, it's very convenient and makes it a lot easier and faster for some people to do their jobs. Optical Character Recognition involves the detection of text content on images and translation of the images to encoded text that the computer can easily understand. An image

containing text is scanned and analyzed in order to identify the characters in it. Upon identification, the character is converted to machine-encoded text.

The image is first scanned and the text and graphics elements are converted into a bitmap, which is essentially a matrix of black and white dots. The image is then pre-processed where the brightness and contrast are adjusted to enhance the accuracy of the process. The image is now split into zones identifying the areas of interest such as where the images or text are and this helps kickoff the extraction process. The areas containing text can now be broken down further into lines and words and characters and now the software is able to match the characters through comparison and various detection algorithms. The final result is the text in the image that we're given. The process may not be 100% accurate and might need human intervention to correct some elements that were not scanned correctly. Error correction can also be achieved using a dictionary or even Natural Language Processing (NLP).

We will use the Python-Tesseract, or simply PyTesseract, library which is a wrapper for Google's Tesseract-OCR Engine. Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and “read” the text embedded in images. Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine. It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Python Imaging Library, including jpeg, png, gif, bmp, tiff, and others, whereas tesseract-ocr by default only supports tiff and bmp. Additionally, if used as a script, Python-tesseract will print the recognized text instead of writing it to a file.

Support for OpenCV image/NumPy array objects:

```
import cv2

img =cv2.imread(r'/<path_to_image>/digits.png')
print(pytesseract.image_to_string(img))
#OR explicit beforehand converting
print(pytesseract.image_to_string(Image.fromarray(img))
```

Add the following config, if you have a tessdata error like: “Error opening data file…”

```
tessdata_dir_config = r'--tessdata-dir "<replace_with_your_tessdata_dir_path>"'

# Example config: r'--tessdata-dir "C:\Program Files (x86)\Tesseract-OCR\tessdata"'
```

```
# It's important to add double quotes around the dir path.
pytesseract.image_to_string(image, lang='chi_sim', config=tessdata_dir_config)
```

Functions:

get_tesseract_version Returns the Tesseract version installed in the system.

image_to_string Returns the result of a Tesseract OCR run on the image to string

image_to_boxes Returns result containing recognized characters and their box boundaries

image_to_data Returns result containing box boundaries, confidences, and other information. Requires Tesseract 3.05+. For more information, please check the Tesseract TSV documentation

image_to_osd Returns result containing information about orientation and script detection.

Parameters:

1. image_to_data(image,lang=None,config='',nice=0,output_type=Output.STRING)
2. image Object, PIL Image/NumPy array of the image to be processed by Tesseract
3. lang String, Tesseract language code string
4. config String, Any additional configurations as a string, ex: config='--psm 6'
5. nice Integer, modifies the processor priority for the Tesseract run. Not supported on Windows. Nice adjusts the niceness of unix-like processes.
6. output_type Class attribute, specifies the type of the output, defaults to string. For the full list of all supported types, please check the definition of pytesseract.Output class.

INSTALLATION:

Prerequisites:

1. Python-tesseract requires python 2.6+ or python 3.x
2. You will need the Python Imaging Library (PIL) (or the Pillow fork). Under Debian/Ubuntu, this is the package python-imaging or python3-imaging.
3. Install Google Tesseract OCR (additional info how to install the engine on

Linux, Mac OSX and Windows). You must be able to invoke the tesseract command as tesseract. If this isn't the case, for example because tesseract isn't in your PATH, you will have to change the "tesseract_cmd" variable pytesseract.pytesseract.tesseract_cmd. Under Debian/Ubuntu you can use the package tesseract-ocr. For Mac OS users. Please install homebrew package tesseract.

The whole process of Tesseract-OCR is depicted below:

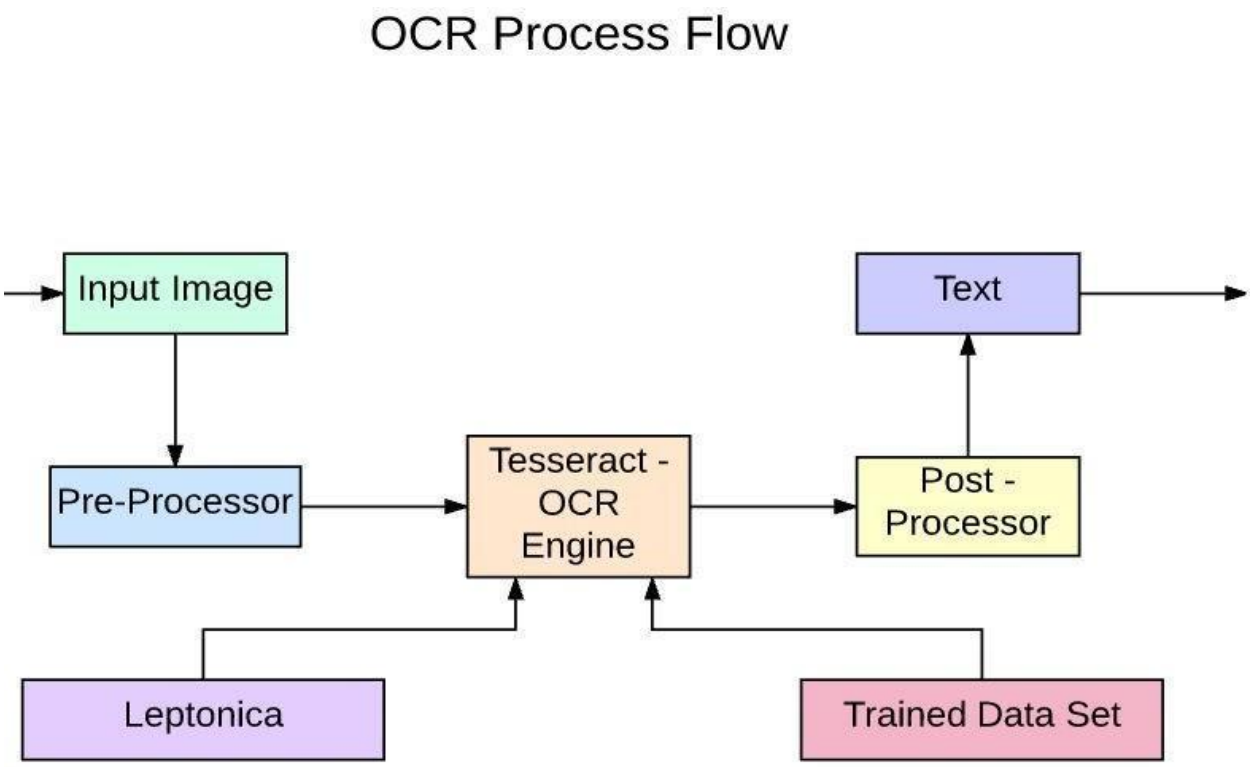

Fig 5. OCR WORKFLOW

1. OpenCV: OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

   The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine

learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people in the user community and an estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

OpenCV's application areas include:

1. 2D and 3D feature toolkits
2. Egomotion estimation
3. Facial recognition system
4. Gesture recognition
5. Human–computer interaction (HCI)
6. Mobile robotics
7. Motion understanding
8. Object identification

9. Segmentation and recognition
10. Stereopsis stereo vision: depth perception from 2 cameras
11. Structure from motion (SFM)

OpenCV runs on the following desktop operating systems: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV runs on the following mobile operating systems: Android, iOS, Maemo,[21] BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub. OpenCV uses CMake.

Installation: The project uses OpenCV (4.4.0) as the computer vision library and requires some prerequisites as :

1. Python-3.7.x.
2. Numpy.
3. Matplotlib (Matplotlib is optional, but recommended since we use it a lot in our tutorials).

Install all packages into their default locations. Python will be installed to C:/Python37/.After installation, open command prompt and write command:

pip install opencv-python

3.4 Project Implementation:

1. In this project Digital Image processing and OCR are executed using different python libraries such as OpenCV 3 and pytesseract respectively.
2. The primary camera of the computer is accessed and the image is clicked after any key.
3. The input image is then fed in the system, for further processing.
4. The Morphed image and the input image and the morphed image is then displayed when any key is pressed. The Morphed image is obtained after morphological transformation.
5. After pressing one more key, the segmented plate is displayed from the morphed image in a new window which is performed using contour extraction on the morphed image.
6. The final step involves performing optical character recognition on the segmented plate

using the Tesseract library known as pytesseract in python. The vehicle number is displayed on the terminal and the plate region is highlighted in a new image in a new window, after pressing one more key.

7. The project is tested on various sets of test data and the system is found to work fine with certain pictures with particular frame measurements.

Code: The whole Python code for the implementation is shown below:

```
import numpy as np
import cv2
from copy import deepcopy
from PIL import Image
import pytesseract as tess

tess.pytesseract.tesseract_cmd = r"C:\Program Files
(x86)\Tesseract-OCR\tesseract.exe"

def preprocess(img):
    cv2.imshow("Input", img)
    imgBlurred = cv2.GaussianBlur(img, (5, 5), 0)
    gray = cv2.cvtColor(imgBlurred, cv2.COLOR_BGR2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_8U, 1, 0, ksize=3)
    ret2, threshold_img = cv2.threshold(sobelx, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    return threshold_img

def cleanPlate(plate):
    print "CLEANING PLATE. . ."
```

```
    gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
    im1, contours, hierarchy = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    if contours:
        areas = [cv2.contourArea(c) for c in contours]
        max_index = np.argmax(areas)
        max_cnt = contours[max_index]
        max_cntArea = areas[max_index]
        x, y, w, h = cv2.boundingRect(max_cnt)
        if not ratioCheck(max_cntArea, w, h):
            return plate, None
        cleaned_final = thresh[y:y+h, x:x+w]
        return cleaned_final, [x, y, w, h]
    else:
        return plate, None


def extract_contours(threshold_img):
    element = cv2.getStructuringElement(shape=cv2.MORPH_RECT,
ksize=(17, 3))
    morph_img_threshold = threshold_img.copy()
    cv2.morphologyEx(src=threshold_img, op=cv2.MORPH_CLOSE,
kernel=element, dst=morph_img_threshold)
    cv2.imshow("Morphed", morph_img_threshold)
    cv2.waitKey(0)
    im2, contours, hierarchy = cv2.findContours(morph_img_threshold,
```

```
mode=cv2.RETR_EXTERNAL,
method=cv2.CHAIN_APPROX_NONE)
    return contours


def ratioCheck(area, width, height):
    ratio = float(width) / float(height)
    if ratio < 1:
        ratio = 1 / ratio
    aspect = 4.7272
    min = 15 * aspect * 15 # minimum area
    max = 125 * aspect * 125 # maximum area
    rmin = 3
    rmax = 6
    if (area < min or area > max) or (ratio < rmin or ratio > rmax):
        return False
    return True


def isMaxWhite(plate):
    avg = np.mean(plate)
    if(avg >= 115):
        return True
    else:
        return False


def validateRotationAndRatio(rect):
```

```
    (x, y), (width, height), rect_angle = rect
    if(width > height):
        angle = -rect_angle
    else:
        angle = 90 + rect_angle
    if angle > 15:
        return False
    if height == 0 or width == 0:
        return False
    area = height * width
    if not ratioCheck(area, width, height):
        return False
    else:
        return True


def cleanAndRead(img, contours):
    for i, cnt in enumerate(contours):
        min_rect = cv2.minAreaRect(cnt)
        if validateRotationAndRatio(min_rect):
            x, y, w, h = cv2.boundingRect(cnt)
            plate_img = img[y:y+h, x:x+w]
            if isMaxWhite(plate_img):
                clean_plate, rect = cleanPlate(plate_img)
                if rect:
                    lang = 'eng'
```

```
x1, y1, w1, h1 = rect

x, y, w, h = x+x1, y+y1, w1, h1

cv2.imshow("Cleaned Plate", clean_plate)

cv2.waitKey(0)

plate_im = Image.fromarray(clean_plate)

text =
tess.pytesseract.image_to_string(plate_im,lang=lang)

print("Detected Text: ", text)

img = cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0),2)

cv2.imshow("Detected Plate", img)

cv2.waitKey(0)
```

Screenshots: The screenshots of the implementation of the project is attached below:

1. When the plate is first detected, the script performs as:

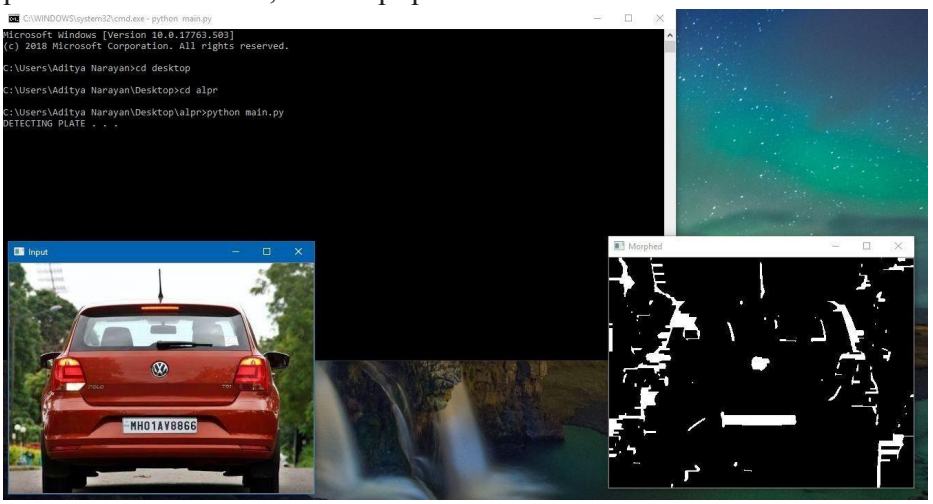

Fig.6.0 : Plate Detection

2. When the plate is extracted in a separate window:

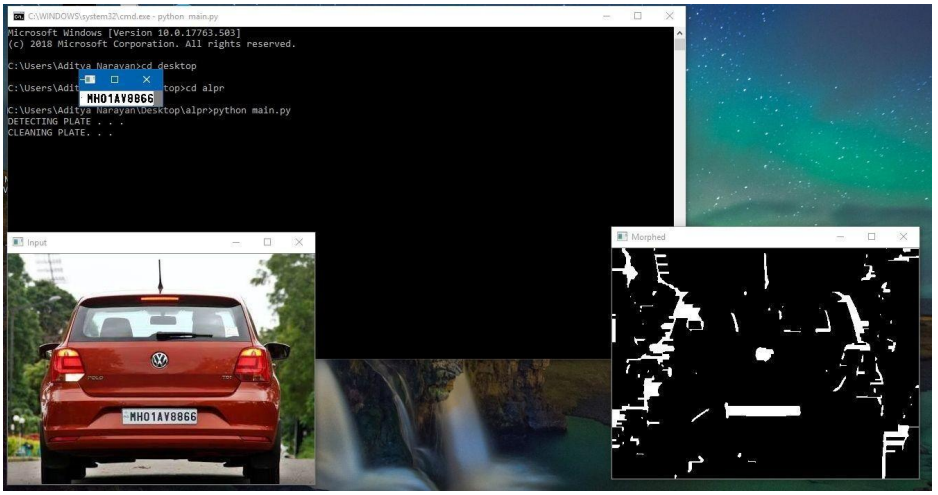

Fig.6.1 : Plate Recognition

3. The final output on the terminal and the featured plate is highlighted as:

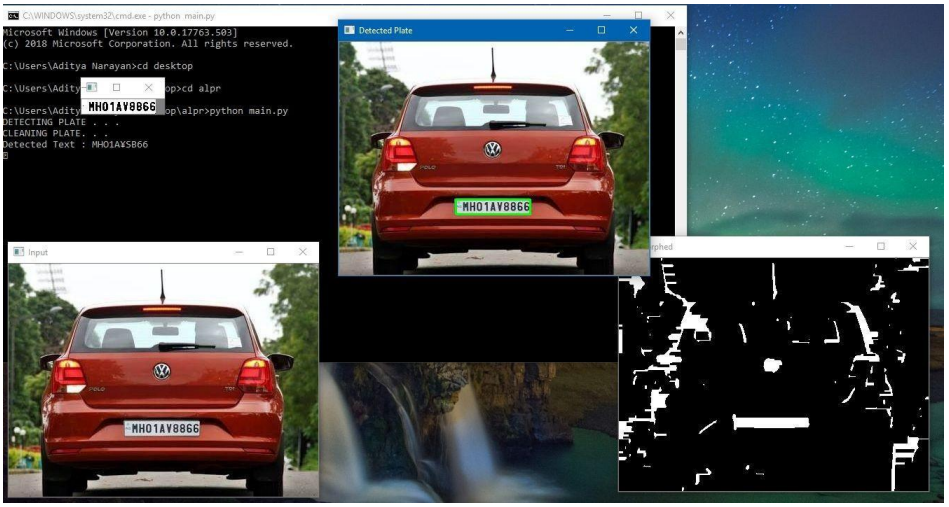

Fig.6.2 : Plate Number Output

Chapter 4
Project Analysis

4.1 Testing

. The project script is tested on various pictures and was successful with certain pictures that were available under certain dimension frames.

i. The image was successfully enhancing the image quality and was converting the image to binary and morphed image.

ii. The project extracts the number plate from the car image and displays it separately.

iii. The project successfully prints half of the License plate number after performing Optical Character Recognition using pytesseract. The accuracy issues are there in the Tesseract engine and can be enhanced after enhancing the configuration of the engine.

4.2 Inference

The final prototype is performing all the functions including the image acquisition, License plate extraction, pre-processing, character segmentation and character recognition along with printing it on the display or terminal. The project can be modified to store the number in a database.

4.3 Application

Automatic Number Plate Recognition has a wide range of applications since the license number is the primary, most widely accepted, human readable, mandatory identifier of motor vehicles.

ANPR provides automated access to the content of the number plate for computer systems managing databases and processing information of vehicle movements.

Below we indicated some of the major applications, without the demand of completeness:

1. Parking: One of the main applications of ANPR is parking automation and parking

security: ticketless parking fee management, parking access automation, vehicle location guidance, car theft prevention, "lost ticket" fraud, fraud by changing tickets, simplified, partially or fully automated payment process, amongst many others.

2. Access Control: Access control in general is a mechanism for limiting access to areas and resources based on users' identities and their membership in various predefined [1]groups. Access to limited zones, however, may also be managed based on the accessing vehicles alone, or together with personal identity. License plate recognition brings automation of vehicle access control management, providing increased security, car pool management for logistics, security guide assistance, event logging, event management, keeping access diary, possibilities for analysis and data mining.
3. Motorway Road Tolling: Road Tolling means that motorists pay directly for the usage of a particular segment of road infrastructures. Tolls are a common way of funding the improvements of highways, motorways, roads and bridges: tolls are fees for services. Efficient road tolling increases the level of related road services by reducing travel time overhead, congestion and improving roadways quality. Also, efficient road tolling reduces fraud related to non-payment, makes charging effective, reduces required manpower to process events of exceptions. License plate recognition is mostly used as a very efficient enforcement tool, while there are road tolling systems based solely on license plate recognition too.
4. Border Control: Border Control is an established state-coordinated effort to achieve operational control of the country's state border with the priority mission of supporting the homeland's security against terrorism, illegal cross border traffic, smuggling and criminal activities. Efficient border control significantly decreases the rate of violent crime and increases the society's security. Automatic number plate recognition adds significant value by event logging, establishing investigate-able databases of border crossings, alarming on suspicious passing, and many more.

4.4 Advantages

1. Automatic number plate recognition cameras are used to measure the average vehicle speed over longer distances
2. Used to identify a motorist when he/she drives away without paying for their fuel

---

1

3. Targeted advertisement
4. Automatic number plate recognition cameras are used for Traffic management systems.

5. Used to Analyze the behavior (route choice, origin-destination etc.) of a motorist for transport planning purposes
6. ANPR camera solutions automatically recognize customers based on their license plate and provide them with the complete information about the items that they ordered the last time they used the service.
7. Automatic license plate recognition camera solutions are used to recognize the guest.

4.5 Disadvantages

1. Privacy Concerns: The project involves capturing and processing visual data, raising concerns about privacy breaches and unauthorized access if the data is not adequately protected.
2. Reliance on High-Quality Cameras: The accuracy of number plate recognition can be impacted by factors such as low lighting conditions, camera angle, and distance. The system's performance may be compromised if cameras vary in quality or placement.
3. Variation in License Plate Designs: The system may encounter challenges in accurately recognizing diverse license plate designs and formats used across different regions and countries. Adaptations and training on diverse datasets may be required to ensure robust recognition.
4. Computational Requirements: Deep learning models used for number plate recognition demand significant computational resources, potentially limiting scalability and posing challenges in deploying the solution in resource-constrained environments or on edge devices.

Chapter 5
Conclusion

This project performs mainly four tasks. The first task is to input an image of the car and this will happen with help of the webcam of the computer for the prototype. When the image is fed the image is enhanced in quality. The enhancement is done in the resolution and the thresholding. The image is constrained to a fixed image frame size.

After the enhancement the image is processed to segment the number plate from the full picture based on the mathematical model of the rectangle. The segmented plate is shown in a new window with all the characters in binary form.

The enhanced segmented plate is then processed for OCR or Optical Character Recognition to segment all the characters in the picture in the form of Text and then it can be stored in a database or can be displayed as in this prototype.

The project is designed so that we can understand the technology used in now-a-days Automatic license plate systems and OCR systems used in most of the developed countries like Germany, France, Singapore, Japan, etc.

It is seen that security forces all over the world face the problem of locating or registering vehicle numbers to track any culprit. It is also seen that technology can greatly help us in this situation by solving it.

Chapter 6

Future Scope

As a future work the developed system would be concentrated upon increasing the accuracy of text localization and graphics removal in caption text images. It can be evaluated using various other available image data bases and using various other classifiers. The proposed methods can be further improvised and applied for automatic mixed mail sorting.

The future and scope for the AI-based number plate recognition project are vast, presenting numerous opportunities for advancements and applications. One potential direction for future development lies in real-time recognition capabilities. By optimizing the system's algorithms, hardware infrastructure, and image processing techniques, the project can achieve near-instantaneous recognition of number plates. This would enable real-time applications such as toll collection systems, where vehicles can be seamlessly identified and charged without causing delays or congestion.

Furthermore, integrating the number plate recognition system with existing traffic management infrastructure holds immense potential. By deploying cameras equipped with AI-powered recognition capabilities at key junctions and intersections, traffic authorities can efficiently monitor traffic flow, identify violators, and manage congestion more effectively. This could lead to enhanced traffic control, reduced accidents, and improved overall road safety.

Expanding the project to include a comprehensive vehicle management system is another valuable future prospect. By integrating the number plate recognition system with a centralized database, authorities can track and manage vehicles more efficiently. This would facilitate functions like automated parking management, monitoring of stolen vehicles, and enforcement of parking regulations. Moreover, the system can be integrated with law enforcement agencies to aid in identifying vehicles involved in criminal activities or traffic violations, thereby enhancing public safety and security.

Another potential avenue for future development lies in improving the system's adaptability and robustness. This can be achieved by expanding the dataset used for training to include a wider range of license plate designs, vehicle types, and environmental conditions. Incorporating advanced machine learning techniques, such as transfer learning, can help the system generalize better to unseen scenarios and improve its accuracy across different regions and countries.

Furthermore, exploring the potential of edge computing can be beneficial. By deploying the number plate recognition system on edge devices, such as cameras or smart traffic lights, the need for constant data transmission to a centralized server can be reduced. This would not only enhance the system's responsiveness but also mitigate concerns related to data privacy and network latency.

In conclusion, the future and scope for the AI-based number plate recognition project are promising. With advancements in real-time recognition, integration with existing traffic management infrastructure, comprehensive vehicle management capabilities, adaptability to diverse scenarios, and exploration of edge computing, the project has the potential to revolutionize vehicle management, traffic control, and law enforcement, leading to safer and more efficient transportation systems.

REFERENCES

[1] P. Li, Y. Zhang, Y. Liu, and Z. He, "Automatic Number Plate Recognition with Deep Learning: A Review," IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 9, pp. 3388-3407, Sep. 2021.

[2] S. Chen, L. Li, and Y. Huang, "An Automatic Number Plate Recognition System Based on Gradient Features and SVM Classifier," IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 1, pp. 155-166, Mar. 2022.

[3] J. Li, Y. Huang, and W. Yang, "Automatic Number Plate Recognition Based on Principal Component Analysis and Support Vector Machines," IEEE Transactions on Intelligent Transportation Systems, vol. 8, no. 2, pp. 377-388, Jun. 2020.

[4] K. Chen, X. Lu, L. Lin, and J. Wang, "License Plate Recognition Using Convolutional Neural Networks and Progressive Search Optimization," IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 7, pp. 2647-2656, Jul. 2020.

[5] M. Saberian, M. Hosseinzadeh, and M. H. Rohban, "Automatic Number Plate Recognition Using Edge Detection and Morphological Operations," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1071-1081, Apr. 2021.

[6] L. Zhao, Y. Wang, and Y. Shang, "An Improved ANPR Algorithm Based on Deep Convolutional Neural Networks," in Proceedings of the IEEE International Conference on Intelligent Transportation Systems, Yokohama, Japan, 2021, pp. 2232-2237.

[7] X. Liu, W. Jiang, and Y. Ma, "A License Plate Recognition System Based on Robust Feature Extraction and SVM," IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 1, pp. 297-305, Mar. 2022.