



University of
Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation:

Master's in Computer Science (Specialization
in Data Science)

Spring, 2021

Open

Author: Bukhari, Syed Irtza Akhtar

Programme coordinator: Sheryl Josdal

Supervisor(s): Øyvind Meinich-Bache & Karl Skretting

Title of master's thesis:

Object Character Recognition from patient monitor screen

Credits: 30

Keywords: Image processing, Optical
Character Recognition, easyOCR, Tesseract.

Number of pages: 51

+ supplemental material/other: 8

Stavanger, 14/2021



Universitetet
i Stavanger

Object Character Recognition From
Patient Monitor Screen

By

Syed Irtza Akhtar Bukhari

*A dissertation submitted in partial fulfilment of the requirements of the award of
Master of Science in Computer Science at the University of Stavanger, Norway*

Abstract

The new paradigm shift with the expansion of data requires us to plan better ways to manage it. This data can be in the form of text or digital images and can be found in different domains. This data when managed well, can be of much importance to us. While handling of text data is considered to be an easy process because it has a specific structure. But data in digital images is much larger and requires considerable effort to handle. Handling of such a large data (video format) to smaller amounts (text and numbers in different data structures) while preserving the important information, comes with a lot of benefits such as sorting, easy transmission and searching.

In the medical field, the data from Patient monitor screens are available in the form of video recordings. SimCapture uses these recordings for their medical training purposes. It is believed that handling such data can be very helpful to serve soulful purposes in this domain. The manual handling of such data can be a very time-consuming task for which automated solutions are required.

For this purpose, a model is proposed that has been developed using different OCR techniques to extract important information from patient monitor screens. This proposed algorithm follows three main steps where at the first step, the input (Stream of videos from SimCapture) is pre-processed and the frames are captured based on a certain threshold for the video feeds. The next step is to apply OCR techniques to identify bounding regions of information. There are different OCRs available with their own computational complexities and limitations on different kinds of data. Lastly, the information that has been extracted is stored into different data-frames for further use. Such solutions can be computationally very exhausting but the project also aims at providing the low-complexity solution for companies having resource limitations by reducing the information size for each video from MegaBytes to Kilobytes of text.

In order to achieve the aims set for this project, we are using two different OCRs to test with our data set i.e. tesseract and easyOCR. While tesseract is considered a good solution for other problems, it has presented a very low accuracy when used for SimCapture's dataset but the algorithm is computationally very cheap. On the other hand, EasyOCR solves the problem with much better accuracy but is computationally expensive. Thus, if the resource is not a limitation, EasyOCR is the better model to extract the information from a patient monitor screen and present it in the form of data frames to be fed to a training model.

Keywords: Image processing, Optical Character Recognition, easyOCR, Tesseract, Python

Acknowledgment

First of all, I would like to thank my advisor Øyvind Meinich-Bache and co-supervisor Karl Skretting, at University of Stavanger for their guidance and support throughout the completion of this project and providing me a way forward whenever i am stuck. I would also show my gratitude to Laerdal Medical AS for their trust and support in me and provided me an opportunity to write my Master thesis in collaboration with them.

Finally, I would like to thank my family and friends for support and motivation for doing this project.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Thesis Description | 2 |
| 1.2 | Thesis structure and content | 3 |
| 1.3 | Research objectives and relevance | 3 |
| 2 | Background and Literature Review | 4 |
| 2.1 | Background: Patient Monitor Screens | 4 |
| 2.2 | Literature Review | 5 |
| 2.2.1 | Computer Vision and Image Processing | 5 |
| 2.2.2 | OpenCV | 5 |
| 2.2.3 | Minimum area bounding rectangle (MBR) | 7 |
| 2.2.4 | Image Masking | 9 |
| 2.3 | Optical Character Recognition | 10 |
| 2.3.1 | Tesseract | 12 |
| 2.3.2 | EasyOCR | 18 |
| 2.4 | Object Detection | 20 |
| 2.4.1 | YOLO for Object Detection | 21 |
| 2.5 | Commercial Engines | 23 |
| 3 | Algorithm and Implementation of Model | 24 |
| 3.1 | System Design and Algorithm Overview | 24 |
| 3.1.1 | Initialization | 26 |
| 3.1.2 | Function calls | 27 |
| 3.1.3 | Image Processing | 27 |
| 3.1.4 | Post processing | 28 |
| 3.2 | Implementation of Algorithm | 29 |
| 3.2.1 | Prompter and Framing function | 29 |
| 3.2.2 | Image Pre-processing | 30 |
| 3.2.3 | Image Masking | 30 |
| 3.2.4 | OCR Recognition | 32 |
| 3.2.5 | OCR using Tesseract | 32 |
| 3.2.6 | OCR using EasyOCR | 33 |
| 3.2.7 | OCR_header() | 34 |

| | | |
|----------|--|-----------|
| 3.2.8 | OCR_values() | 34 |
| 3.2.9 | Bounding Area Rectangle | 35 |
| 3.2.10 | Post Processing | 36 |
| 4 | Experimental Results | 37 |
| 4.1 | Image gathering and preparation | 37 |
| 4.2 | Recognition in Perfect Conditions | 38 |
| 4.3 | Recognition in Varying Conditions | 39 |
| 4.4 | Recognition in Low Resolution Conditions | 42 |
| 4.5 | Digit Recognition | 42 |
| 4.6 | Timing Efficiency | 43 |
| 5 | Evaluation | 45 |
| 5.1 | Analysis of the Results | 45 |
| 5.2 | Discussion | 45 |
| 5.3 | Limitations and Future work | 46 |
| 6 | Conclusion | 47 |
| | Appendix A | 51 |
| A.1 | Terminologies | 51 |

List of Figures

| | |
|---|----|
| 1.0.1 Patient Monitor Screens by Laerdal Medical | 1 |
| 2.2.1 Digital Image Processing phases | 6 |
| 2.2.2 OpenCV architecture design | 7 |
| 2.2.3 Bounding Box - Rotated Rectangle | 8 |
| 2.3.1 Optical Character Recognition Example | 11 |
| 2.3.2 OCR processes | 12 |
| 2.3.3 OCR process flow with tesseract | 13 |
| 2.3.4 Component Architecture of Tesseract | 14 |
| 2.3.5 Word Segmenting | 15 |
| 2.3.6 Chop-points in a figure text | 16 |
| 2.3.7 Hard example: disjoint words | 17 |
| 2.3.8 Word Classification process | 18 |
| 2.3.9 EasyOCR framework | 20 |
| 2.4.1 Objects detected using a YOLOv3 model trained on COCO dataset | 21 |
| 2.4.2 Architecture of CNN | 22 |
| 3.1.1 Flowchart of proposed Algorithm | 25 |
| 3.1.2 Initialization process flow | 26 |
| 3.1.3 Function calls | 27 |
| 3.1.4 Image Processing process | 28 |
| 3.1.5 Post-processing | 29 |
| 3.2.1 Prompter: opening file | 30 |
| 3.2.2 Output: from function inRange() | 31 |
| 3.2.3 Output: Bitwise masking | 32 |
| 3.2.4 Parameters format with corresponding values | 34 |
| 3.2.5 Result of bounding rectangle | 35 |
| 3.2.6 Dataframe of results | 36 |
| 4.1.1 Parameters to be recognized by OCR Model | 38 |
| 4.3.1 Correlation of Results | 40 |
| 4.4.1 Results based on Resolution | 42 |
| 4.5.1 Correlation between confidence interval and image resolution | 43 |
| 4.6.1 Processing time comparison of both engines | 44 |
| 4.6.2 Comparison of both engines w.r.t accuracy and computation power | 44 |

List of Tables

| | |
|--|----|
| 4.2.1 Recognition and classification of various parameters in Perfect Conditions . | 39 |
| 4.3.1 Recognition and classification of various parameters in varying Conditions . | 41 |

Chapter 1

Introduction

Technical advancements have made our daily life seem very easy and working is more fun than before. Making use of these advancements into our work life can also increase productivity and accuracy of any work. This gives us liberty to think out of the box solutions in different walks of life to come up with informed decisions in various fields like business, education, travel, leisure, medical or other. In the medical field specifically, patient diagnosis is a very critical process that requires a lot of precision and care. For this purpose, different patient monitoring screens in intensive care units are used to capture the patient's conditional data like ECG, heart rate and blood pressure in digital form (One of such screens can be seen in the Figure 1.0.1 [21]).

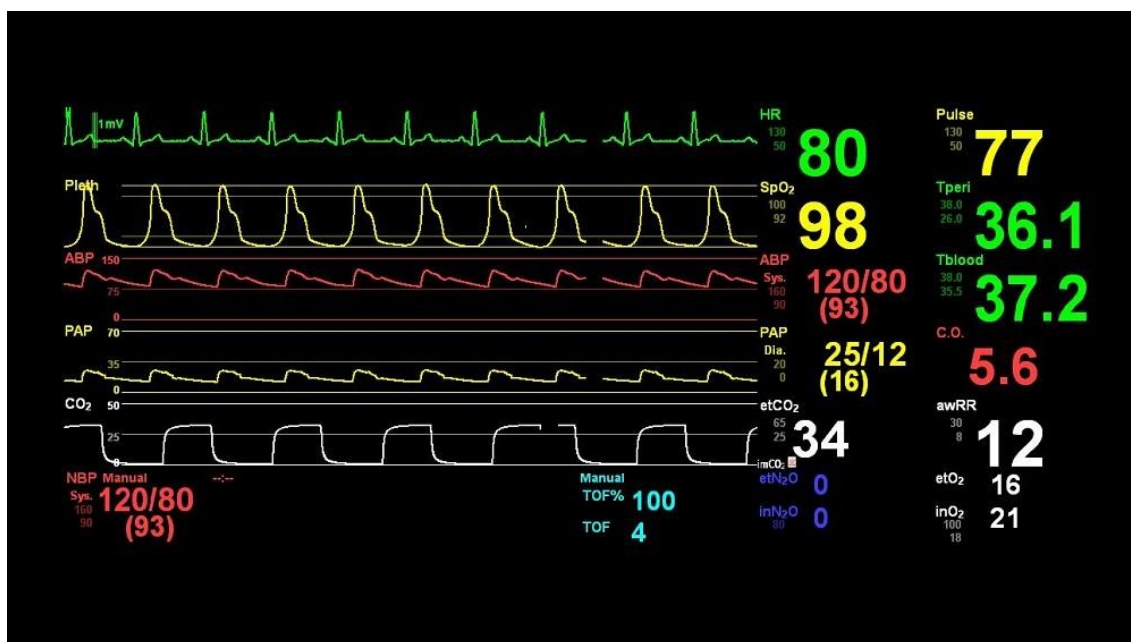


Figure 1.0.1: Patient Monitor Screens by Laerdal Medical

These screens help practitioners to understand the patient's situation completely to come up with the correct decisions in their diagnosis. To extract such data for analysis and diagnosis, hospitals are often bound to monitor the patient monitor recordings manually and lead the whole process to slow down. To overcome this problem, different Optical

character recognition techniques can be applied to help convert image data into characters and store them in files. The data from patient monitor screens and digital simulators is recorded, and managed by SimCapture as part of a single web-based interface. SimCapture uses this data to provide solutions to medical schools, nursing schools and hospitals to deliver high-quality training, education and quality improvement programs. SimCapture uses the patient monitor screens, showing the condition of the patient, as part of their training sessions that adds great value of information to these sessions and patient treatments. Knowing the importance and value of this information, the need is established to store this information. By extracting the information from the video and audio recordings of the patient monitor screens, we can create timeline of events and patient's vital signs. These can further be used to evaluate the treatment automatically. Traditionally this task has been done in small capacity, using the image processing techniques but due to the noisy data, blurry images, low resolution of the images, and/or the quality of the processes, a lot of important information is lost in the processing. This is where our proposed method comes in to help extract and store the information with better accuracy.

1.1 Thesis Description

This thesis is written in collaboration with Laerdal Medical AS. They develop advanced patient care products and simulation tools to help gather patient records. This thesis aims to deliver a solution based on optical character recognition that will be able to record the vital signs from these patient monitoring streams. These vital signs can be placed on different locations on the monitor, depending upon the manufacturer, in the form of keys (text like HR, SPO2) and value (numbers) pairs of the same colour. Optical character recognition can be applied in two different ways depending upon the input stream

- Pattern recognition where OCR models are given the input as text with various fonts and formats and it recognises the characters in an image by comparing the formats.
- Feature detection where OCR models try to recognize the letters and characters based on their formation like lines, curves, and angles.

The input will then be provided to our proposed model that will use the Neural Networks to train the model to identify the characters better so, that we can use this trained model in real time scenario on unseen data to achieve high accuracy. Apart from Convolutional Neural Networks, there are other types of neural networks are also available and it will be interesting to see the results from these to compare them for accuracy. It will also be interesting to see the results achieved from this model and compare the results with the traditional image processing models for the comparison of performance. When dealing with such real-world applications, our target is to achieve low-complexity solutions because there are limitations to the resources owned by the organizations. The objective of this project is to design a low-complexity solution to store the data from patient monitoring screens based on their manufacturer and able to achieve good accuracy from our trained neural network to work with unforeseen data.

1.2 Thesis structure and content

The remainder of this thesis is structured as follows:

Chapter 2 introduces relevant background material and related works with a focus on OCR (Optical Character Recognition), its different uses and how this can be used in extracting the related information from a patient monitoring screen.

Chapter 3 studies the background of the problem and reviews the domain of image processing, computer vision to find a theoretical explanation to the said problem solution along with exploration of different algorithms and techniques used. This chapter also gives a theoretical detail of the image processing techniques used in this project from Tesseract to EasyOCR.

Chapter 4 presents a code survey and implementation details of the algorithm with visual aids. This chapter also gives an overview of our methodology and the overall system architecture.

Chapter 5 provides an experimental evaluation of the algorithm and experimental results of the proposed solutions for real-time problems.

Chapter 6 lists the challenges encountered during the implementation of low-complexity solutions and pinpoints the key factors considered when working with such a model. This chapter also concludes and presents suggestions for further work.

1.3 Research objectives and relevance

The research objectives within this paper are to:

- Draw information from the videos of SimCapture’s patient monitoring screens using optical character recognition.
- Designing a low complexity-solution to store this information
- Identify the best alternative available for optical character recognition.
- To be able to achieve good accuracy from our proposed solution in order for it to be accepted by different manufacturers.

Chapter 2

Background and Literature Review

This chapter will introduce the working of Patient monitor screens and their use in the medical domain. We will also cover some detailed theoretical concepts about the domain including image processing; OpenCV (a computer vision library for image processing), and OCR (Optical Character Recognition); tesseract (an optical character recognition engine) to set a ground to understated concepts better to make comprehension for this report and its results. The chapter will cover the mathematics behind the different algorithms used in this report and will make you able to understand their practical implementation better. This chapter will also cover the advantages and disadvantages of using different image processing and OCR techniques and will establish the ground for our choices.

2.1 Background: Patient Monitor Screens

The Patient Monitor system is an old technique which is used to continuously monitor the measurements of patient parameters such as heart rate, blood pressure, respiratory rate, and oxygen level etc. Such parameters are considered as the common parameters for any patient while monitoring the patient's condition for diagnostics.

This all began in 1965 by Venetian Doctor Santorio, who published his methods to measure body temperature with the help of spirit thermometer and use of pendulum for counting of heart rate (pulse). With increasing efficient computing power and the advent of integrated circuits, more reliable, fast, accurate and less expensive techniques were introduced, and better software modules were developed. In the current times, we call such systems as computer-based patient monitoring systems (Patient monitors).

To make an accurate and on the spot decision for any critical patient, such kind of data is useful and is used to predict the patient's health rapidly. To serve this rapid decision-making purpose, different electronic monitors which we call as patient monitor screens are used to gather and display physiological data. Such data are collected with the help of non-intrusive sensors from different areas such as hospital's emergency units, delivery units, blood donation units, nursing homes, or patients at home. The variety of data detected from less seriously ill patients helps in predicting life threatening conditions

and record data resourcefully. Patient monitor systems are also used to assist doctors and researchers in collection, storage, and prediction including interpretation of clinical data and various alarming conditions. Patient monitor systems were primarily used in Intensive Care Units (ICUs) which were used to monitor any serious condition of the patient in real time. The techniques that were just a few years ago used only in the ICU are now used in other caring units in hospitals and by patients at home as well. This is an opportunity for many researchers and industry to use this data from Patient monitor screens and use it for different decision making processes with help of different classification and prediction methods for the text on them. One of such efficient methods is by using optical character recognition for text. Some other good methods and techniques can also be found in the literature of this domain.

2.2 Literature Review

In this section, we will explore some of the tools and techniques in the domain that are helpful and are used for the fulfilment of this project. We will also review the work done by different researchers using these tools in the literature.

2.2.1 Computer Vision and Image Processing

Computer vision is an area of science that makes use of the images for regeneration, prevention and learning. There are three main tasks of computer vision that are Pattern Recognition, Photogrammetry, and Image processing. Where, Image processing is a method in computer vision that is used to perform some operations on digital image, in order to get an enhance it or to extract some useful information from it. The operations in the image processing range from image enhancement, feature extraction, better visualization, or pattern recognition for learning. A good depiction of the different phases in image processing through a flow chart is given by Rose Marry in “Introduction to image processing” published by EngineersGarage on 1st April 2011 [24] and can be seen in Figure 2.2.1.

Image processing helps extract key features from an image to perform these operations on those features specifically. Now, with the advancement in deep neural networks in the field of image processing, efforts have been put up to come up with the solutions to use an image to image architecture without any major pre-processing techniques but still pre-processing on the images before passing them as an input for our model is an important and crucial step. Isola et al [19] described image-image translation as translating one possible representation of some pixel values into another by giving sufficient training data.

2.2.2 OpenCV

In this report, for the purpose of model accuracy, we are making use of different image processing techniques and methods for feature extraction, image enhancement etc. For this purpose, we are using OpenCV, a library for image processing that is available to use with python. OpenCV is a very popular solution commonly used for many projects and

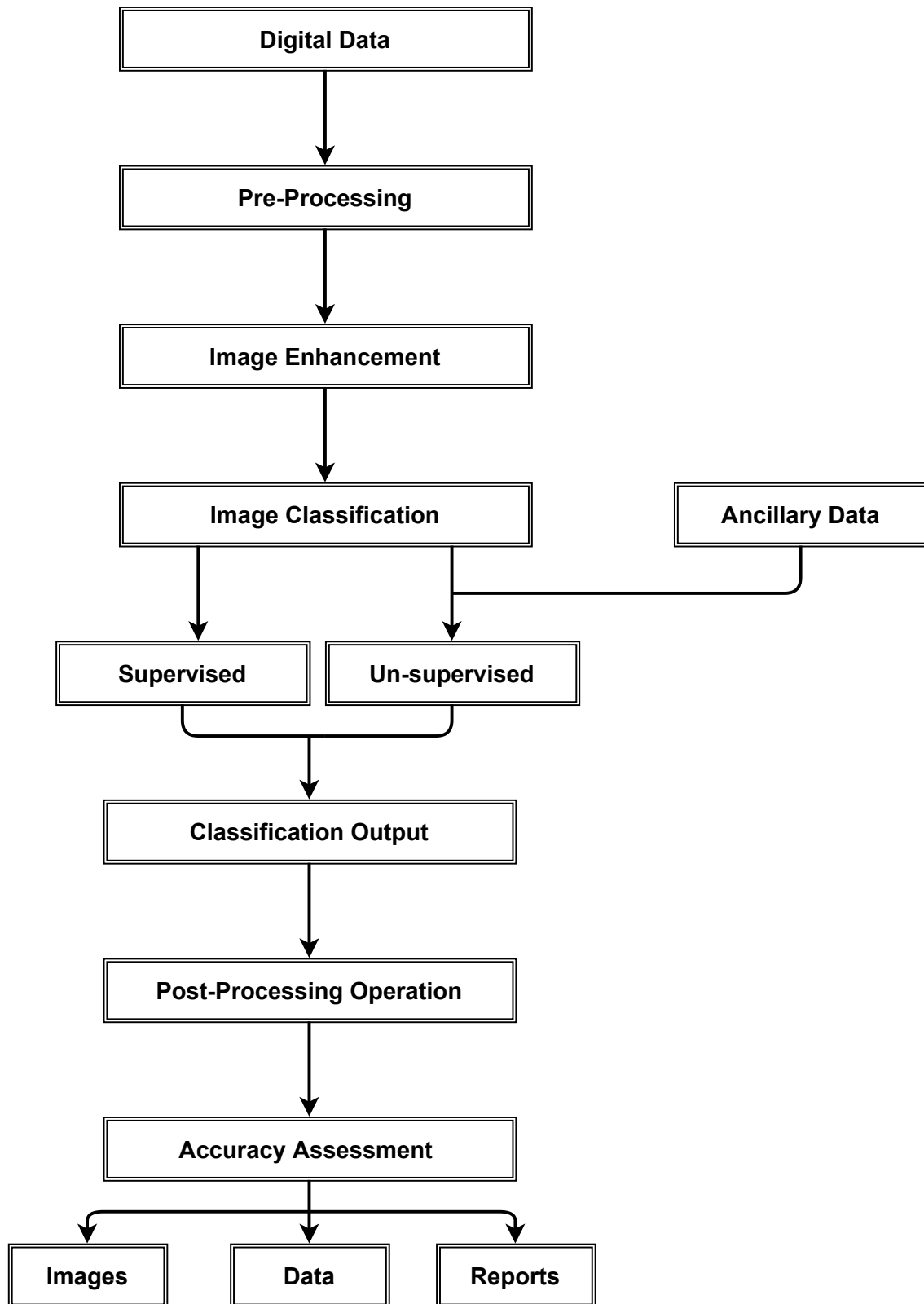


Figure 2.2.1: Digital Image Processing phases

problems related to image processing and computer vision these days. OpenCV stands for open-source computer vision originally generated to support a common infrastructure for computer vision operations including image recognition, face detection, analysing tasks in

videos, objects identification, tracking object movements, and image combining to create a high-resolution image for the scenery making. It has interfaces for C++, C, Python, and Java and it supports multiple platforms like Windows, Linux, Mac, and Android as well. According to the OpenCV documentation, the library has over 2500 of such functions for computer vision and machine learning algorithms. It provides some efficient functions for colour conversions, colour correction, image masking, and different filters etc for image processing that we have used in this project. The architecture design of the OpenCV as presented by Zhiqiang Qiang Chen, in his work Mobile Imaging and Computing for Intelligent Structural Damage Inspection [17] is presented in the Figure 2.2.2.

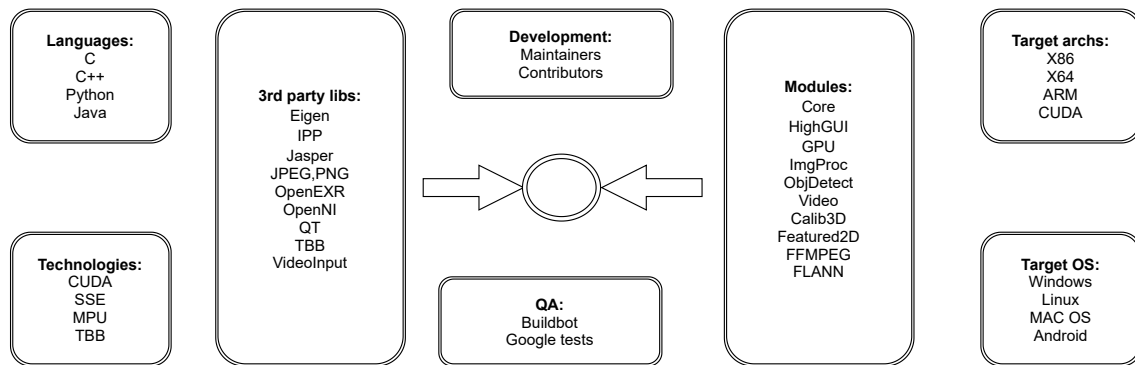


Figure 2.2.2: OpenCV architecture design

In this architecture, components like HighGUI allow users to interact with different operating systems and file systems and other imaging hardware. Where the file system focuses on loading and saving images, HighGUI also allows a simple way to query external hardware components. There may be other offerings available in matlab or so but for image processing, the OpenCV library is one of the best available solutions for such problems because Matlab provides you very generic solutions but in OpenCV you will get a detailed toolbox for image processing. Matlab is also very slow in interpreting the lines of code and overall processing of images as compared to OpenCV, it also takes more system resources than OpenCV. So, because of these reasons, we have decided to use OpenCV for image processing problems in our project. A very important part of image processing is removing the noise from the images, and/or edge detection. OpenCV uses very efficient but simple programs for this purpose. A simple example with good read about different functions of image processing is provided by Muhammad Junaid Khalid in his article “Introduction to Image Processing in Python with OpenCV” [20].

2.2.3 Minimum area bounding rectangle (MBR)

The minimum bounding rectangle (MBR) is also known as the bounding box, or envelope. The idea is to encapsulate components from the image into an envelope or bounding box. It is an expression of maximum extents of an object in some shape i.e. point, line, polygon etc or set of objects within the coordinate system of the 2D (x,y) i.e. min(x), max(x), min(y), max(y) [34]. If the bounding box contains an angle then the image contains a

skew.

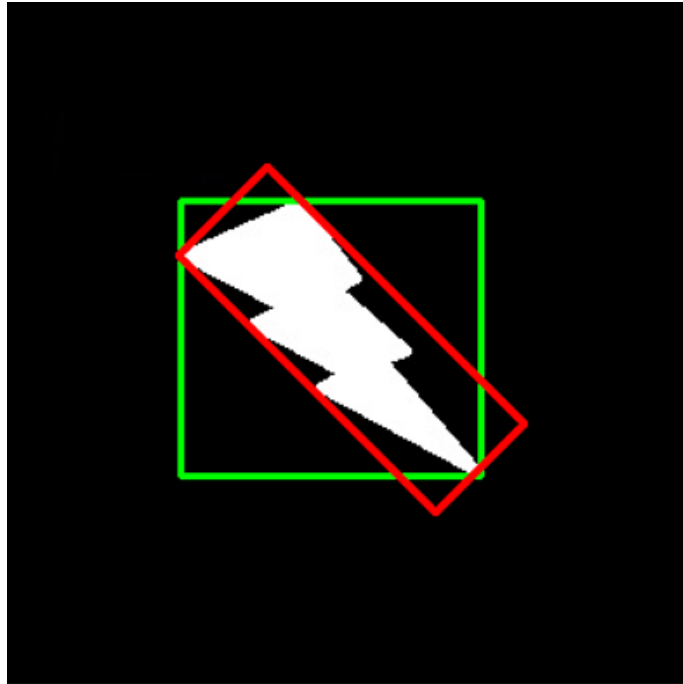


Figure 2.2.3: Bounding Box - Rotated Rectangle

There are two types of bounding rectangles.

- **Straight Bounding Rectangle:** It is a straight rectangle and doesn't consider the rotation of the object. So the area of such a bounding rectangle will not be minimum and can be found in OpenCV by simply using a function `cv.boundingRect()`.
- **Rotated Rectangle:** The bounding rectangles in a rotated one are drawn with minimum area, so it can consider the rotation as well. For this purpose we use the function `cv.minAreaRect()` which returns a box of 2D structure containing the following details in particular i.e. center (x,y), width, height, and angle of rotation.

A clear picture of such bounding boxes can be seen in the Figure 2.2.3 [18]. The MBR algorithm, as defined by Reza Safabakhsh and Shahram Khadivi in their work on Document skew detection using minimumarea bounding rectangle [29] is presented in Algorithm 1:

In the algorithm 1, α is the skew angle, A is the area of the bounding rectangle h_1 and h_2 are the maximum and minimum intercepts from origin having a slope of “ $\tan\alpha$ ” and passing through boundary pixels of the connected component. h_3 and h_4 are the maximum and minimum intercepts from the origin of the lines having a slope of “ $1/\tan\alpha$ ” and passing through boundary pixels of the connected component.

To calculate the area A of the bounding rectangle in the direction with, the following formula is used

$$A = (h_1 - h_2) * (h_3 - h_4) * |\cos \theta \cdot \sin \theta| \quad (2.1)$$

Algorithm 1 Minimum Bounding Rectangle

```
 $\theta = 0$   
Amin = current area of the bounding rectangle of the connected component at angle  $\theta$   
set  $\alpha_{min}, \alpha_{max}, \Delta_\alpha = \text{values}(\text{considering the origin at the center of rectangle})$   
for different values of  $\alpha$ , from  $\alpha_{min}$  to  $\alpha_{max}$  with  $\Delta_\theta$  and  $\alpha \neq 0$  do  
  Compute  $A \Rightarrow \alpha$  (Equation 1)  
  if  $A < A_{min}$  then  
    Set  $A_{min} = A$   
     $\alpha = \theta$   
  end if  
  set rotation angle =  $\alpha$   
end for
```

2.2.4 Image Masking

To improve the accuracy of the results by improving the input itself, we use image masking. Image masking is a step that separates an image from its background based on colour intensities or through any other measures. This way, helps the image to stand out from the background. In masking we set some pixels of an image to null values because some parts of an image are not been examined so we point it as zeros or call it as mask i.e. number plate recognition system for vehicles, masks the surrounding area of the plate before it was run through any optical character recognition system [13]. So image masking can actually hide some portions of an image and reveal some portions. This tool is pretty handy when we have a fixed color gradient and we are looking for a certain point, shape or texture to stand out from it.

Masking can be hard or soft while, in hard masking the pixels affected by masking have their specific intensities set to the background value but in soft masking the resulting intensity change depends on how much of the pixel is inside the image set. We set the background value based on that information then. If a large portion is inside the image then the intensity will be unaffected by it largely. In practice, we apply masking in 3 different ways:

- Layer masking
- Clipping mask
- Alpha channel masking

Layer Masking: In this process, we can hide or reveal portions of an image. We can also use an image to change its opacity at certain points with the help of image masking. This way the visibility of the portions can be changed and that affects the whole image also.

Clipping Masks: With the help of a clipping mask, we use one layer to determine the visibility/transparency of another layer. In this process, we place a layer at the bottom of the layer we want to see the visibility of and from the bottom layer; we control the visibility of the above layer.

Alpha channel masking: Compared to the other two types of masking, alpha masking is considered as a bit complex technique and is used when we need to mask the rough areas from the image. To mask those areas with layer and clipping technique i.e. with simple brush strokes and points, it becomes very difficult and time consuming to select those areas so in conditions where the object and the background has a sufficient amount of contrast, we can use alpha channel masking technique. To work with this method, we use the channel palate to see which channel gives us the most contrast and make a copy of the channels. We then apply different levels on those channels to increase the contrast sufficiently. Then we apply the inverse colour approach to inverse the colours in our image and then paste the channels on to the layer tab to make a new layer [3]. So, Image masking, allows us to focus on those regions of the image that interest us this way helps us extract regions for machine learning, image classification, and object detection.

2.3 Optical Character Recognition

A large amount of data or information is saved in visual form including images. These images can contain text as machine printed or handwritten format. To classify and predict the text in an image, Optical Character Recognition (OCR)[25] is used. Optical Character Recognition is the process of identifying characters visually from images and converting them to the best-guess equivalent computer code that can be used afterwards[22]. In 1914, a machine was developed by Emanuel Goldberg that reads alphabets and numbers to convert them into standard codes. At the same time, another scientist named Edmund Fournier developed the machine called optophone which was a handheld scanner when moved across a page; it produced specific letters and character tones[35].

These established the base of now the well known and widely used technology that converts different kinds of strings in an image to a machine editable text format called OCR. There are varieties of applications of OCR including data entry, passport recognition in airports, assisting CAPTCHA systems, automatic number plate recognition systems etc. In simple words, wherever text in any image comes the OCR, implemented, converts it into text form. An example of OCR can be seen in Figure 2.3.1 [23].

A complete OCR process may also contain several sub-processes to perform accurately. These processes are depicted in the Figure 2.3.2 and can be categorized into the following categories:

- Pre-processing of the image
- Optical Character Recognition
 - Text localization
 - Character segmentation
 - Character Recognition
- Post-processing

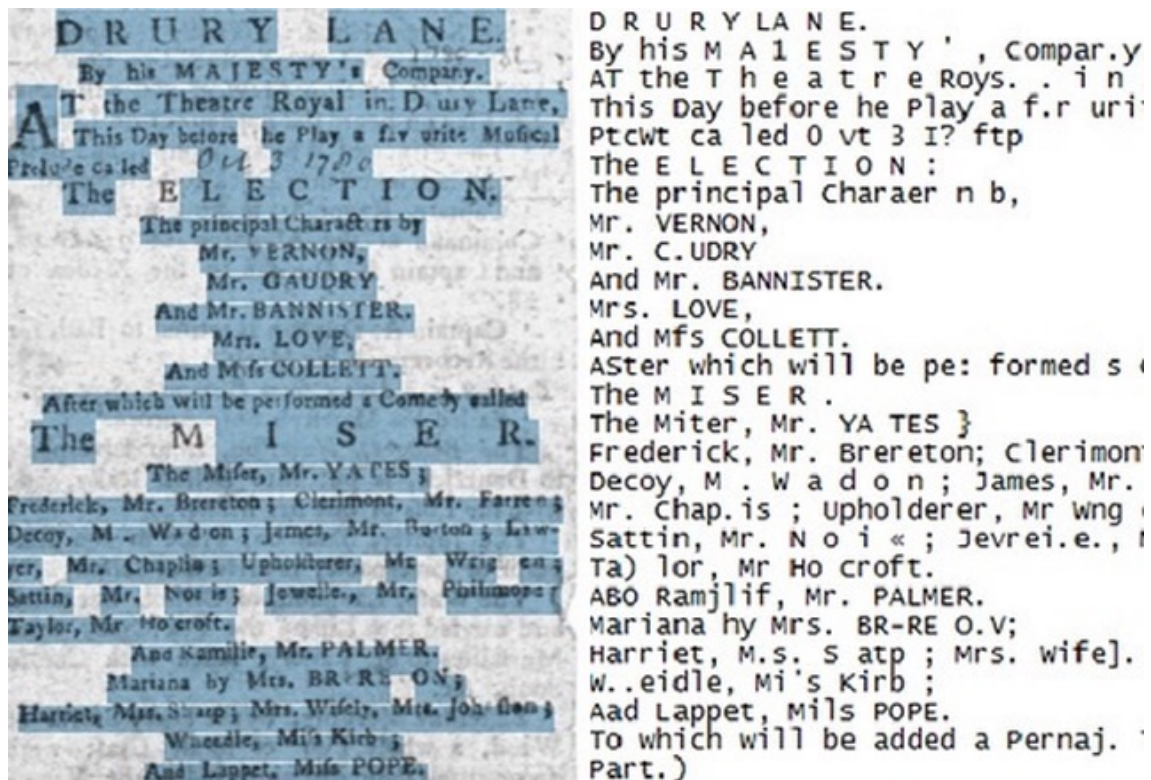


Figure 2.3.1: Optical Character Recognition Example

Where each stage has its own significance and adds to the recognized output. Pre-processing of an image is done to improve the accuracy of the results. Usually in OCR engines, there is no as such hard and fast pre-processing tasks been implemented beforehand which is quite understandable to some extent because too much pre-processing already implemented in OCR engine can make the results more over-fit in some scenarios and it might not come out as the user expected.

Generally, some of the image pre-processing tasks for any image are listed as:

- Converting image into Gray-scale format
- Image resize
- Noise removal
- Segmentation
- Filtering

These general techniques can be applied in order to pre-process the image before feeding it to OCR to acquire better results. Normally, most of the OCR engines perform better with gray-scale images so converting the image into gray-scale is kind of a necessary part of pre-processing. On the other hand, Image thresholding can also be implemented to make an image binarize. Thresholding is also a useful technique if we want to analyse the image based on the colour values. It works by setting a lower and upper limit in a function to make the pixel values convert into 0s and 1s respectively. After that, these

pre-processed results are then sent to Optical character recognition system that applies text localization, character segmentation and character recognition techniques to extract the features (text/information) out of the image. At the end, post-processing is applied on the fetched results to translate into a certain format for further decision making / processing on it.

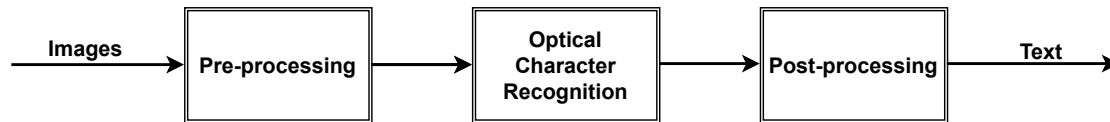


Figure 2.3.2: OCR processes

OCR has already trained libraries available to store data in the database. Whenever, there is any image that needs to be converted into text form, it can be passed through the OCR system which matches and compares the characters stored in the database and displays the results. To serve this purpose, several kinds of both open source and licensed versions of OCR are available in the market which serve differently according to the capability and performance in different scenarios. Like Tesseract that is licensed under the Apache and is available for free on Windows, Linux and OSX. The source code for this library can also be found on GitHub. GOCR is also a free and open-source character recognition software developed under General Public License and is available to use with support for Linux, Windows, and OS2 operating systems. CuneiForm, Kraken, OpenCV, and others are also available for use [30].

OCR can be used to detect the number plates of vehicles on the road. The paper by Agbemenu, and A. S. et al titled “An Automatic Number Plate Recognition System using OpenCV and Tesseract OCR Engine” focusses on the work of automatic number plate recognition system for vehicles by typically tuned the environment for available datasets. Tesseract is the OCR engine that is used in the work along with pre-processing techniques such as edge detection and feature detection combined with mathematical morphology for identifying the number plates [13]. Getting competitive results for Optical Character Recognition from a blurred document, a detailed analysis of Long Short-Term Memory (LSTM) network has been proposed. In this method, OCR using LSTM is implemented mainly on the blurred and Gray-scale document images to avoid error-prone binarization of blurred documents [14]. Again Tesseract performed with much better accuracy in terms of identifying characters considering the image quality had poor resolution.

2.3.1 Tesseract

The most used OCR engine is tesseract for detection of text from images, which is an open source text recognition OCR engine developed under the license of Apache 2.0 and originally developed by HP in the 1980s. Tesseract is an open source Optical Character Recognition engine for various operating systems. Its code is also available on Github. It uses UTF 8 and can support more than 120 languages so far in the latest versions. The most remarkable feature of the Tesseract OCR engine is that it is trainable, meaning a new language and font which is not available in version, we can train the OCR engine with

our own datasets to get more accurate results [36]. Tesseract can be either used directly through a command line interface or you can also use an API (baseapi.h) to extract printed text from images [9]. Like OpenCV, tesseract also supports a number of languages and it does not have its own built-in GUI but external tools, wrappers, and training projects under tesseract are available as Add-ons with 3rd party pages support. The benefit of tesseract is that we can use it directly on our input for text, feature extraction, or can use an API to extract text from image inputs). A common OCR process flow with a tesseract engine is given in Figure 2.3.3 [10]. Where the tesseract engine is fed on the input of images after pre-processing and leads to post-processing and our expected output in the shape of proper text. Trained data set is passed for the training of the model along with Leptonia (A library containing all the text syntaxes and fonts etc.) A latest version of tesseract was

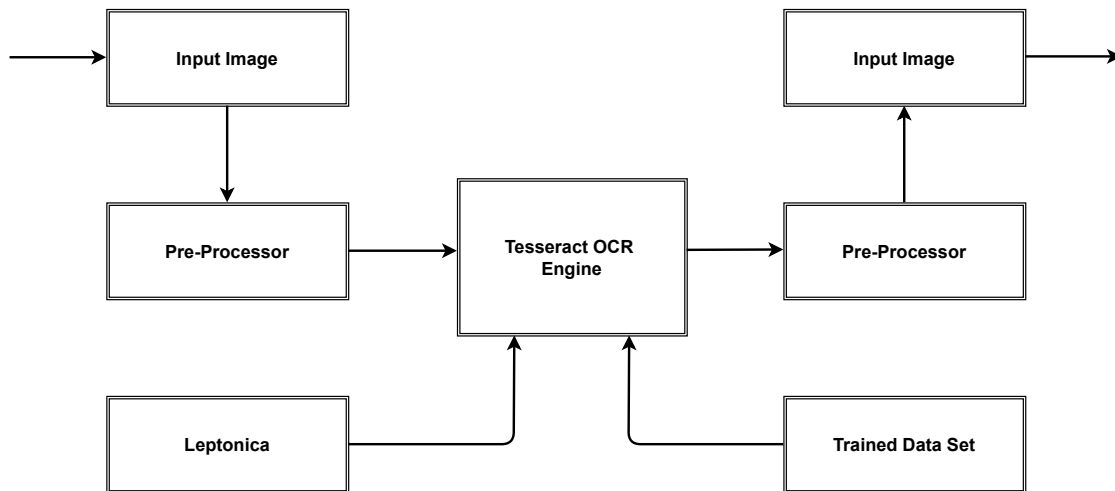


Figure 2.3.3: OCR process flow with tesseract

released and made available in November 2016 and known as Tesseract 4.0. In this version a new OCR engine based on LSTM (a type of neural network) was added along with new scripts, languages and various output format support. The new version also focuses greatly on line recognition while still supporting the legacy Tesseract OCR engine that includes character patterns and supports Unicode (UTF-8) as well. Usually, to extract features (characters) from an image, OCR can be designed to use a convolution neural network (CNN) which is another popular deep neural network. And the CNN classifiers are capable of learning the important features from a 2D image using a soft-max layer but the character length in the image varies the accuracy of CNNs to solve problems. That is why Recurrent Neural Network, one of which is the LSTM, is used to solve such issues arising in feature extraction (i.e. text in the images) but neural networks require significantly more training data and train slower than the legacy tesseract. Installing tesseract on windows is not a big hassle as it is very easily installed with the pre-compiled binaries. It is very important to also edit the path environment variable with the tesseract path. These environment variables are:

- TESSDATA_PREFIX
- OMP_THREAD_LIMIT

Once you are done with the installation, the tesseract engine can be called easily on the image by execution the following commands:

```
$ tesseract image_path stdout
$ tesseract image_path text.csv
```

The results can be written to any file that is following the input image path in the executed command. Another important argument for an OCR engine is its mode which is the `-oem` option and supports four different modes [10]:

- Legacy engine only
- LSTM engine only
- Legacy + LSTM engine
- Default

You can find the details about the tesseract setup and documentation from the tesseract resources available in Github [12]. The component architecture of the tesseract, detailed by Sravan Ch in Optical Character Recognition on Handheld devices 2015 [16], is given in Figure 2.3.4. According to this figure, the input image after pre-processing is passed for connected component analysis where the outlines of components are stored. These outlines are nested to form a Blob. This component also recognizes the lines, and character outlines for character and word segmentation and baseline fitting with use of definite spaces and fuzzy spaces. Then every word is passed to the adaptive classifier for training and testing of our classifier working.

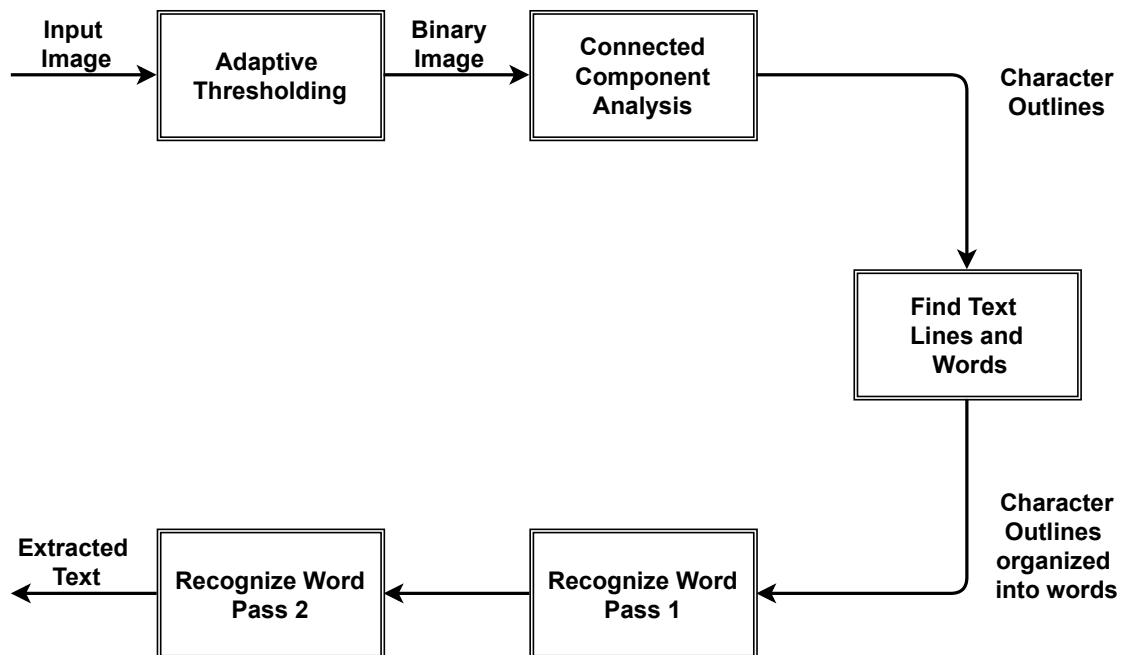


Figure 2.3.4: Component Architecture of Tesseract

Now let's chop these steps into different parts and see its working:

Finding the baselines:

To find a baseline, we need to have knowledge about blob filtering and line construction. A blob, as discussed already, is a work or a symbol that is not connected to the image itself. This means that the dimensions of the blob are approximated which helps the engine to filter and remove all those blobs of text/characters that are too small or considered noise. These blobs are then assigned a position in the text. To keep track of whether these blobs are correctly positioned and assigned to correct text, these are assigned a coordinate called x-coordinate. Once every blob is assigned to a text or a line, baselines are approximated using least median squares [31].

Base fitting:

Once the blobs are found, baselines are fitted to them by careful consideration in the tesseract using a quadratic spline (four parallel lines). This feature helps it to handle curved words.

Word Segmentation:

The recognition process for an OCR also involves recognition of the characters to see how the word should be segmented. Words, where the characters are of the same width (fixed pitch) are treated as a special case where the words are sliced equally based on the pitch and marked. However, sometimes the characters in words may have different width i.e. different pitches and are separately treated [31]. Figure 2.3.5 [8] shows this clearly, where some words are of same width i.e. o,w,a but i is treated differently here and it finds spaces that are too close to the threshold, it makes the space as a fuzzy space. First the initial segmentation output is classified and then next steps of the classification for word recognition are applied to the characters that do not share the same width. Mostly, words segmentation or separating words in a line is not the hard part as there is always space between these words (which means that words include punctuation symbols as separators between them). But separating character can be challenging as it includes isolating one word from another and then separating the various letters of the word.



Figure 2.3.5: Word Segmenting

Character Segmentation:

Character segmentation is to segment the characters from the words. Creating an algorithm that returns a good accuracy as the confidence for segmenting a character is quite difficult and impossible in some cases with noise presence and with different styles of fonts and character formations but is achievable. A good algorithm for this can be where each image character that is segmented is converted into a character code for recognition. If the algorithm is not able to get some character it will produce multiple codes for the same character and then can choose the final one from them. So character segmentation in Tesseract is to segment the characters by chopping the blob that was assigned the worst confidence by the classifier. The chop points can be found with help of polygonal approximation of the outlines of a character like presented in the Figure 2.3.6. This picture is from R.W. Smith's work on *The Extraction and Recognition of Text from Multimedia Document Images* [33]. Andrew Selasi, in his work on *An Automatic Number Plate Recognition System using OpenCV and Tesseract OCR Engine* [13] has also used this technique on the Vehicle number plate recognition and found some good results from it. To check if the curve is convex or not, we can use a built-in function to calculate this. This figure



Figure 2.3.6: Chop-points in a figure text

illustrates the set of candidate chop points presented with arrows and lines across the outline where different characters touch the following ones. In practice it may take up to 3 pairs of chop points to separate such joined characters from the ASCII set. The confidence of all these chopping points is calculated then and the results of such chop points that are dropped that do not show improvement. The broken characters that are associated with each other are passed to the associator. If all the chops have been tried and tested and the word is still not complete then the associator will try a different set of candidate chops from the queue and again evaluate it. R.W. Smith, in his work on *The Extraction and Recognition of Text from Multimedia Document Images*, in November 1987 [33] stated that the advantage of this approach is that the vital pieces of fragments in word missing can be recognized with it but the approach is at worst liable to miss important chops. An example, from the same work, of a hard word where pieces are disjoint and the associator approach was able to recognize it with good accuracy is presented in Figure 2.3.7.



Figure 2.3.7: Hard example: disjoint words

Character Classification:

After we are done with the segmentation part of it, the next step is to classify the characters. The character classification aims to extract the model based on the features to come to a decision about character recognition. Character classification is divided into two different steps where the first step is to pass the characters through a static classifier which acts as a class pruner and creates a list of characters to match with unknown characters. In this step, each feature is fetched from a quantized 3 dimensional lookup table, a matching bit vector class, and a bit-vector. The bit-vectors are summed over all the features and the highest counts become the shortlisted features for the next step. The next step is to pass through an adaptive classifier.

Since, static classifiers extract the characters to be recognized from the extracted outlines and matched to a reference from the training data. They sometimes lack of specific details directly be verified with a reference engine that makes use of polygonal approximation to match associated broken characters to reference. Since the static classifier has to be good at generalizing to any kind of font, its ability to discriminate between different characters or between characters and non-characters is weakened. So, every matched character by the static classifier is then sent to the adaptive classifier for training. Theory presented in [26] and [27] suggests that OCR engines can benefit from the use of adaptive classifiers which is more font-sensitive and is used to obtain great discrimination within each document. The training of the characters in the adaptive classifiers is very important to recognize the characters with help of the information gathered.

Word Classification

The words gathered from different segmentation may have different numbers of characters in them. This makes it hard to compare directly. Tesseract solves this problem with an algorithm where there are two simple steps involved.

In the first step, tesseract generates two numbers for each character that needs to be classified. First number is called the confidence which is computed by subtracting the normalized distance from the prototype. This is a confidence in the sense that the greater numbers are better, farther from zero means the greater the distance. The second number is called a rating, which is computed by multiplying the normalized distance from the prototype by the total outline length from the unknown character. These ratings for characters within a word can be summed for meaningful results; by far the total outline length for all the characters within a word is always the same.

The next step is to recognize the word which Tesseract supports as well. This is done by separating the characters in a word, and then stitching them back together one at a time. To stitch the characters together the algorithm constantly compares the word to a language model. If the character improves the confidence of the word, then this result is picked. If the highest confidence from results is high enough then we consider it to be a complete word otherwise the results are not used. A complete step by step explanation is available from Ray Smith’s training on Tesseract [32] in Figure 2.3.8.

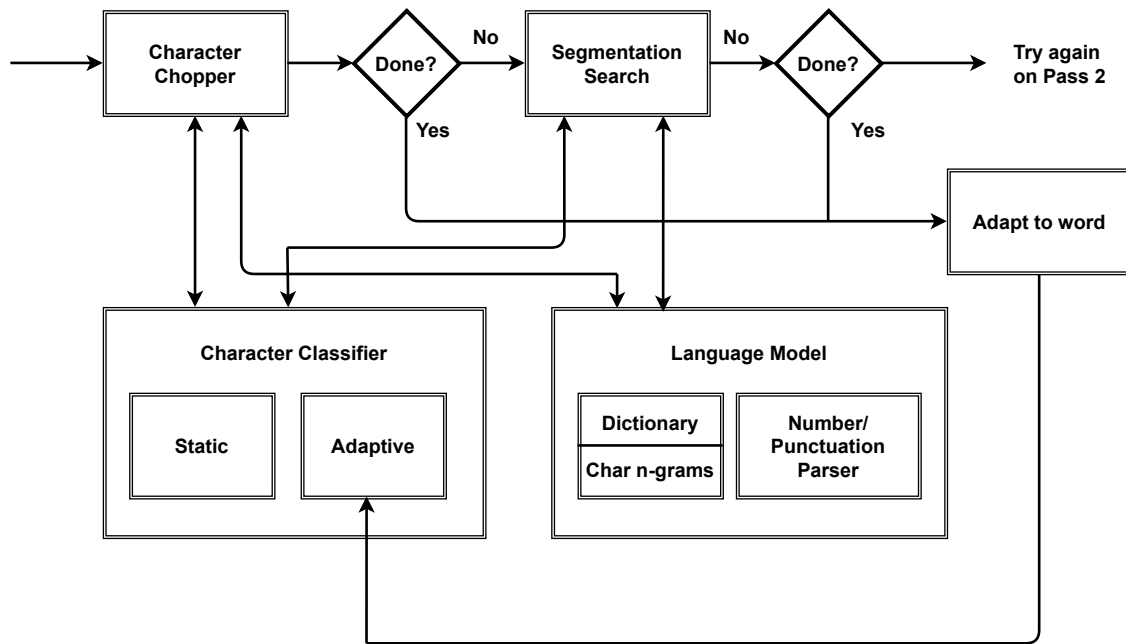


Figure 2.3.8: Word Classification process

At the end we would like to summarize that, although tesseract is a good OCR to be used and it works the best with clean segmented text in the image with distinct background. But, it can be quite challenging to use in practice. The reasons being, the noise in the image can decrease the accuracy of your results, a lot of pre-processing may require to get the best results from the OCR, scaling of the input images is required to improve OCR results, and text must always be horizontally aligned for the model to pick it up as extracted feature. Also, tesseract is not able to recognize handwritten text and may report gibberish as result. Such problems with the OCR results are good reasons to not use the tesseract with our kind of dataset (The accuracy comparison is available in chapter 5).

For these reasons, we will be comparing our results of the tesseract as an OCR with another good OCR named EasyOCR for extracting the features (text) from input images.

2.3.2 EasyOCR

Another Optical Character Recognition (OCR) system, that is available for free to be used with the support of 80 + languages and all popular scripts of languages including Chinese, Arabic, Latin and other. On 20th April 2021, a new version of EasyOCR, was made available named 1.3.1 with added support for PIL images, some new languages and

argument setting updated for command line, better control for merging behaviour when `paragraph = True`. To use EasyOCR, we need to install it first and the installation of EasyOCR with windows is quick but requires you to install torch or torchversion first. The guide to use it with official instructions is available on the Pytorch official page. If you are planning to run on EasyOCR on CPU mode only, select CUDA as none so CUDA is selected as default. To use the EasyOCR with python in your code, you simply need to use the code available in Listing 2.1:

```
import easyocr
reader = easyocr.Reader([ch_sim, en])
result = reader.readtext(file_path.jpg)
```

Listing 2.1: How to use EasyOCR with python

In the code above, the first line is importing the EasyOCR library. Then line 2 needs to run to load the model into memory (This takes some time but you just run it once in your program) and finally the results are output in a list format with each item representing a bounding box, text and confidence level respectively. Sample of one such item is presented below where there is a clear segmentation of a bounding box, text, and its confidence score. For item 1, the coordinates are given by `[[529,173],[569,173],[569,213],[529,213]]`, the text that needs to be identify from language is 'E' and finally its confidence score is 0.8405593.

```
[([[529, 173], [569, 173], [569, 213], [529, 213]], 'E', 0.8405593633651733)]
```

Instead of passing a file path to results, you can also pass openCV image object (numpy array) or image file as bytes as well. For simple output, you can also set the details as 0 in your function as an argument [11].

```
result = reader.readtext(file_path.jpg, detail = 0)
```

An easy to follow and use framework, given by JaidedAI, for EasyOCR is shown Figure 2.3.9 [2] . If you follow this framework, you will see that the image is pre-processed and passed to a text detector model known as CRAFT that detects the text area by exploring each character region and similarity between characters. After mid processing on these characters, these are then passed to LSTM, a neural network, specially designed to learn the patterns and we have already discussed about it in tesseract.

From an implementation point of view, EasyOCR lives up to its name as it is very easy to use for image processing as compare to Tesseract or any hard libraries. Like for example, simple functions are available to achieve all the results, to check if a curve is convex or not is *cv.isContourConvex()*, which returns whether **True** or **False**.

Some other advantages that EasyOCR have on Tesseract are as following:

- OpenCV covers 80-90 percent of the world's languages.
- It has better documentation and API support and better language models for decoding.
- It also has support for handwritten text by using GAN key to generate realistic handwritten dataset.

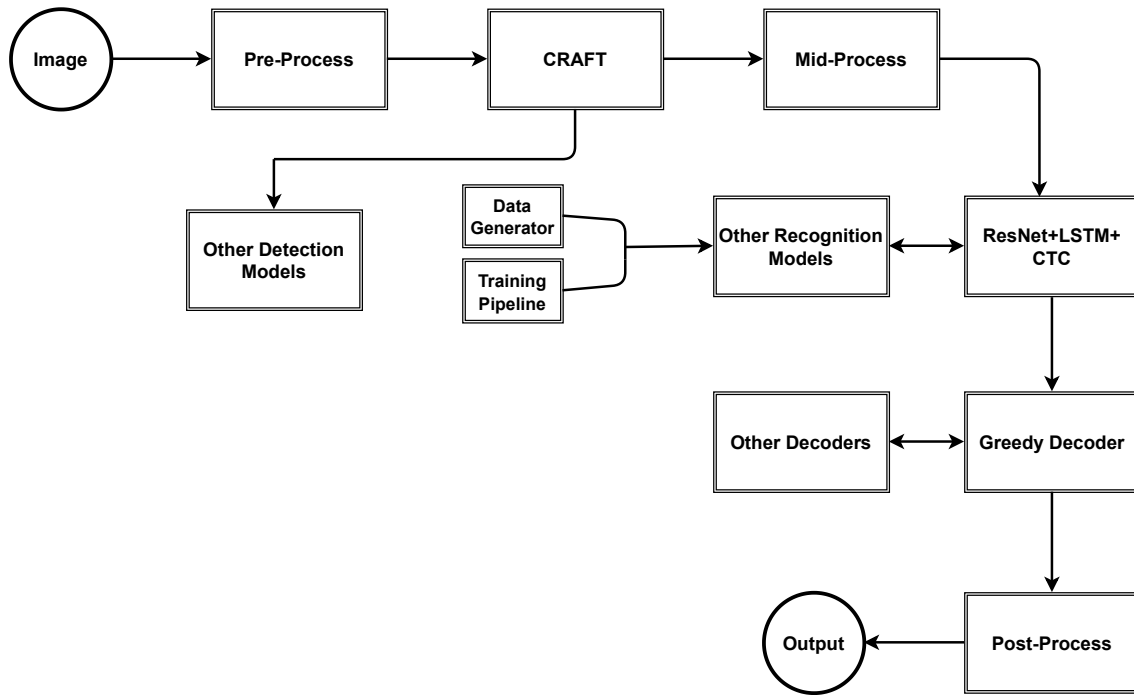


Figure 2.3.9: EasyOCR framework

- It also support model pruning, quantization with faster processing.
- Open dataset and model training pipelines are also available in easyOCR with support for swap-able detection and recognition algorithm with restructured code.

For these reasons, EasyOCR can be a good match for the character recognition.

2.4 Object Detection

Object detection deals with detection of instances of objects from a certain class in digital images and videos. Every class has its own special features that help in classifying the class i.e. human has legs, eyes etc. Object detection method uses these features to detect an instance from a certain class. Usually object detection methods fall into two different approaches: non-neural approach or neural network approach. In non-neural approach we can make use of the following methods i.e. Viola–Jones object detection framework, Scale-invariant feature transform (SIFT), Histogram of oriented gradients (HOG). In neural network approach, we have R-CNN, Single Shot MultiBox Detector (SSD), You Only Look Once (YOLO), Single-Shot Refinement Neural Network for Object Detection (RefineDet), Retina-Net, and Deform-able convolution networks.

Generally, object detection framework takes the following steps to complete the detection of the objects from classes:

- An algorithm is used to model the interest regions with the help of bounding boxes of different shapes.
- Then the features are extracted from each of these bounding boxes based on what

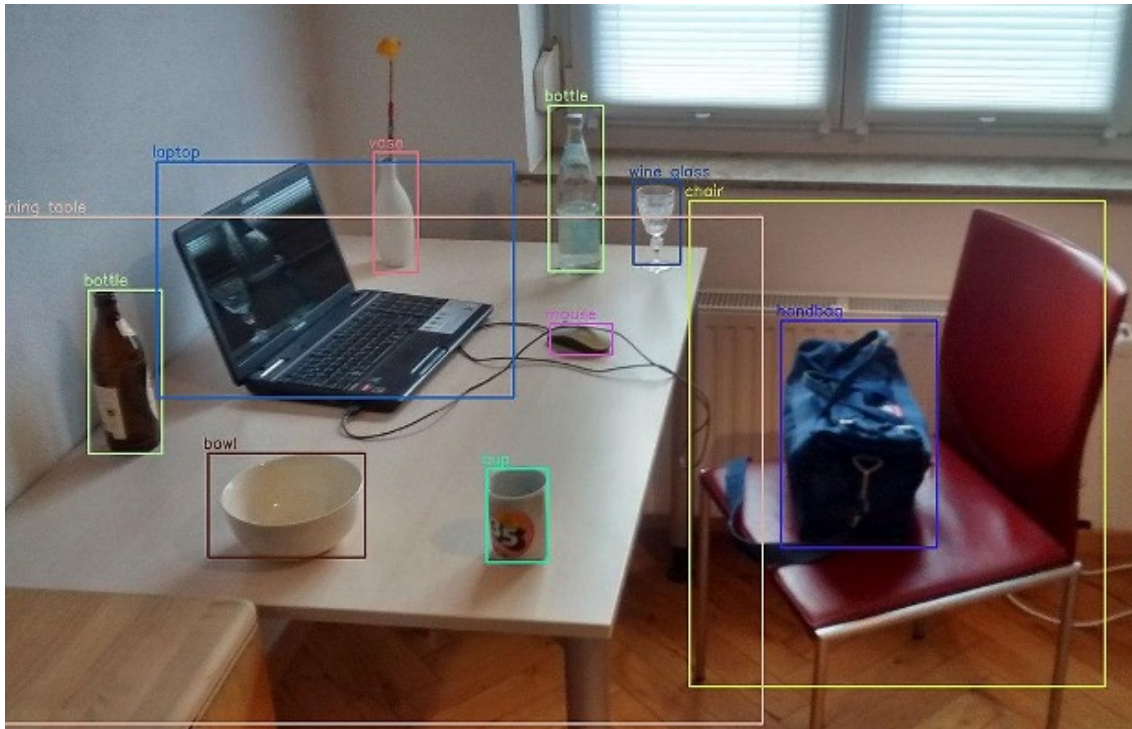


Figure 2.4.1: Objects detected using a YOLOv3 model trained on COCO dataset

you can see (visual features). These features are evaluated and then it is determined whether an object belongs to a class or not based on those visual features.

- At the end, overlapping bounding boxes are combined to form a single bounding box (non-maximum suppression)

2.4.1 YOLO for Object Detection

From different object detection frameworks, YOLO (You only look once) is an object detection algorithm/system targeted for real-time image processing. It is a fast multi-object detection technique which uses CNNs (Convolution Neural Networks) to detect an object. YOLO (You Only Look Once) uses the regression-based model to detect and perform operations in a single run on real-world applications. YOLO came in 2015 as a new real-time object detection system. The earlier version of YOLO was not good in terms of accuracy as compared to the other models like RCNN and SSD but the latest versions have improved accuracy and speed. YOLO is considered as a single-stage object detector this means that it unifies all the components of object detection into the single neural network structure. YOLO divides the image that has been inputted into a grid of $S \times S$ dimensions. Each of these grid cells will predict only one object at max and each grid cell only predicts a fixed number of boundary boxes. For better prediction, YOLO has some limitations as well like how distant each object can be. Each boundary box that has been predicted by the grid cell, a confidence score as well and it detects only one object regardless of the boxes the grid cell predicts.

The class confidence score for each of these boxes is computed with the formula:

$$\text{Class_confidence_score} = \text{box_confidence_score} * \text{conditional_class_probability} \quad (2.2)$$

where, the conditional class probability is the probability that the detected object belongs to which class. For example, a $7*7$ image with 1024 output channels, YOLO performs a regression for the final prediction on 2 fully connected layers to make $7*7*2$ boundary boxes, but only keep those with high box confidence scores computed by the equation given above (threshold is greater than 0.25). YOLO is a fast algorithm and it is very good for real-time processing, predictions in YOLO are made from one single network and can be trained end to end to improve accuracy. It is generalized and outperforms other models when used from natural images. Architecture of CNN is described in the Figure 2.4.2 [7]. YOLO uses CNN network in order to reduce the spatial dimensions of the input image to less dimensions of the output:

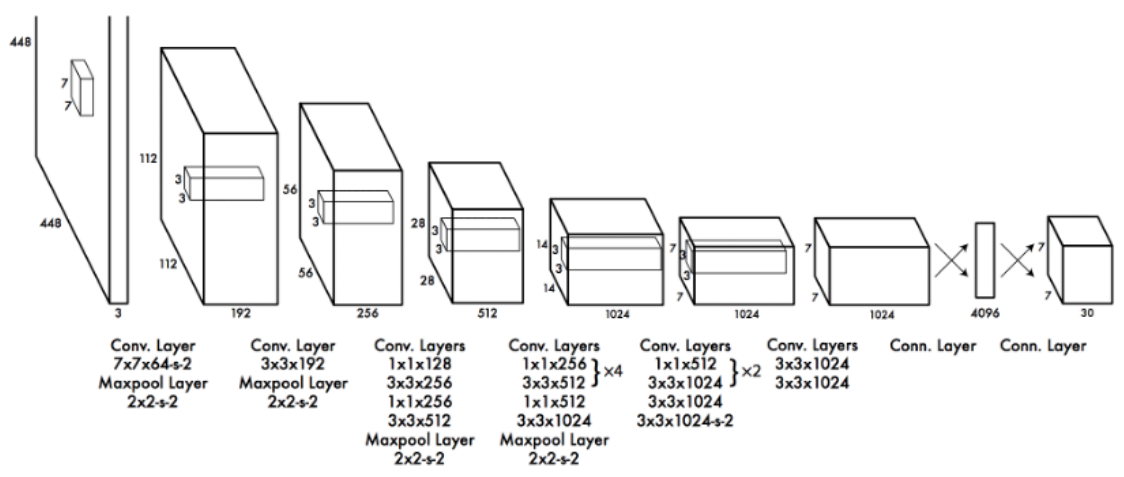


Figure 2.4.2: Architecture of CNN

YOLO has 24 CNN layers followed by two fully connected layers as a completed detection model. Some layers also use $1*1$ reduction layers to reduce the depth of the feature space more. The last output layer shaped ($7*7*1024$) which is then flattened by using two fully connected layers as form of regression and final output is received as ($7*7*30$). The only problem with YOLO is duplicate detection for the same object. To fix this problem YOLO has a method of applying non-maximal suppression to remove duplicates with lower confidence scores. The one possible non-maximal suppression algorithm is as follows [28]:

1. Sort the prediction with respect to the confidence scores for each class.
2. Ignore any current predictions if they are of the same class and IoU value is greater than 0.5, and start with the top scores.
3. Repeat step 2 until all predictions are verified.

2.5 Commercial Engines

Considering the fact that Tesseract needs a lot of pre-processing before feeding it to the model and because of consideration of less training data in the storage, different commercial products already have pre-processing techniques which could perform better when a developer wants to achieve better accuracy in less time. There are a few commercial OCR engines that could perform better with high accuracy and less computational time and cost.

- **Nanonets OCR** A commercial OCR tool that captures data from documents and reduces computational time. It has a variety of OCR products i.e. Invoice OCR, Driver License OCR, Passport OCR etc. You can find more details about Nanonets on their official website [4]
- **ABBYY OCR** A web-based document processing engine that combines AI-based technologies for information extraction and can be easily integrated into several applications via API. You can find more details about ABBYY OCR on their official website [1]

Chapter 3

Algorithm and Implementation of Model

3.1 System Design and Algorithm Overview

The section aims to uncover the hidden information from the Patient Monitors to be able to use this data for patient diagnosis and future use. For this purpose, the whole process is divided into major five different steps:

1. Frame capturing
2. Image Pre-processing
3. OCR and Image processing
4. Post-Processing
5. Create and managing data frames for each recognition

The data from different Patient monitor screens in the form of videos cannot be fed to the OCR as the OCR expects the input data in the form of images. For this, frames need to be captured from these videos with help of a function and stored on the disk. To feed these images as input data to OCR, a little pre-processing of these images is also required. These processed images are then fed to the OCR where characters are recognized and extracted from them and after post processing will be fed to the model as training data.

To keep the implementation results achievable for images, below mentioned assumptions were considered beforehand throughout the implementation:

- The resolution of frames should be constant.
- Background of each frame image should be the same.
- Image capturing should not be done through video recordings using external camera

If we do not follow any of the mentioned assumptions, the results will not be of what we will expect from the model. The complete flowchart of our proposed algorithm is shown in Figure 3.1.1

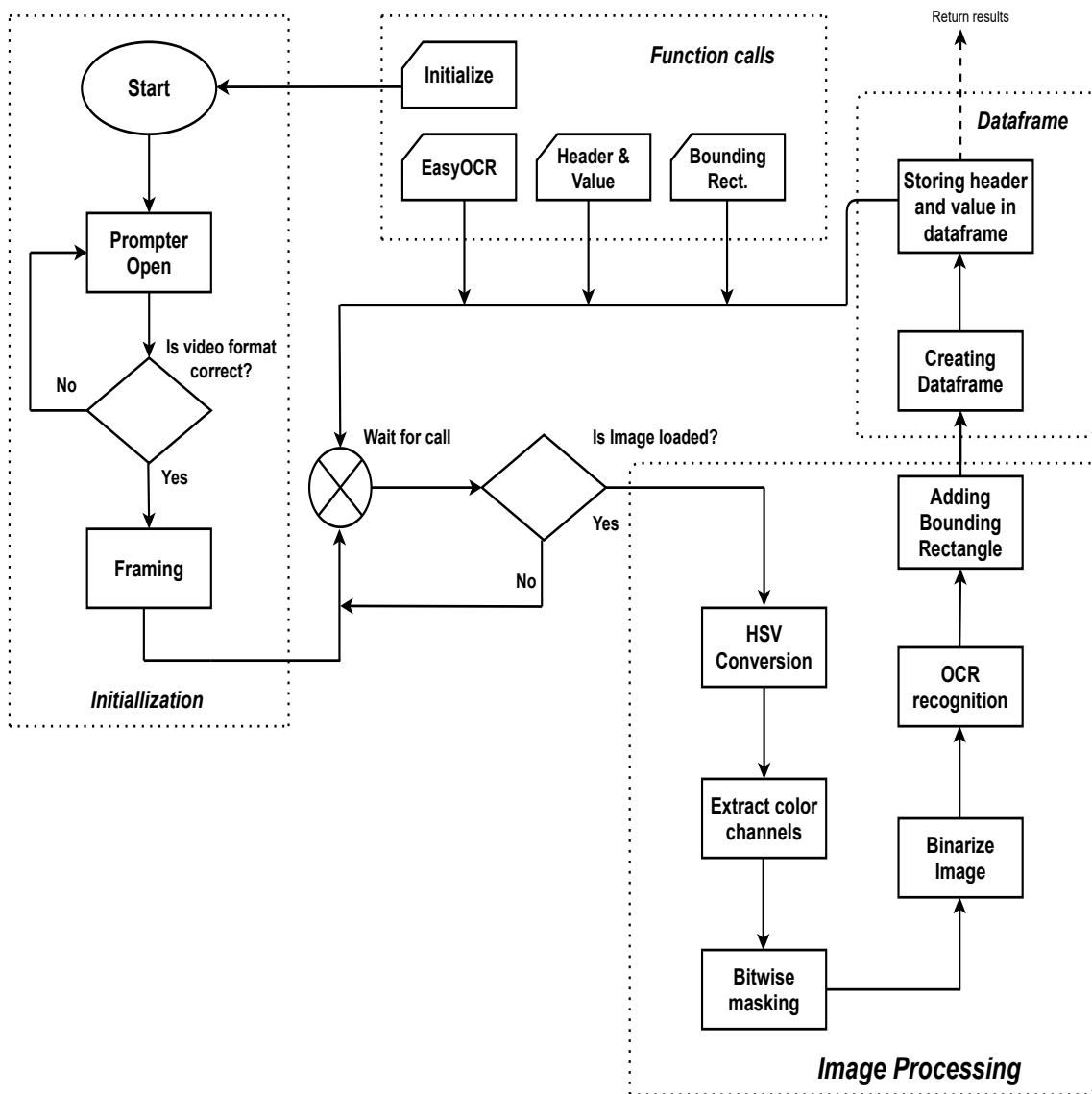


Figure 3.1.1: Flowchart of proposed Algorithm

3.1.1 Initialization

As an initial step, we are processing the videos obtained from the SimCapture server to capture different frames as images. This step helps us extract the required information from these videos. This is a large amount of data that needs to be stored as well and for this purpose we are currently using the disk space to store these images in a folder, especially created for this purpose. For the video to image frames extraction, a function called “getFrame()” is used in python that captures different frames from the video at a given threshold timestamp and returns each extracted frame as an image. The flow chart of this process, in Figure 3.1.2, shows the complete guide, where the process is fed with the video recordings, the function will start the prompter. The prompter checks if the video format, being fed to it, is correct or not e.g. jpeg,png etc. If the format is not correct, the file is discarded and the prompter will take the next input. In case of the correct video format received, the function of framing will be called that will check for the threshold and return the images from the video being cut on the threshold.

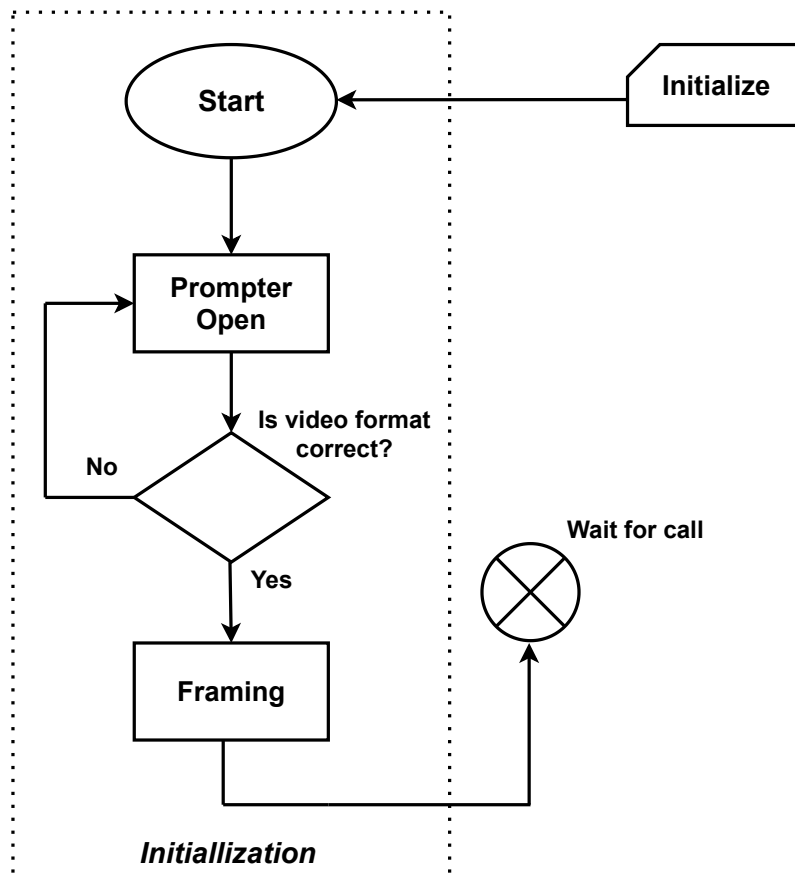


Figure 3.1.2: Initialization process flow

3.1.2 Function calls

There are four function calls which need to start and load into memory before the execution of the program. The first call is *Initialize* which will start and restart on every new video which is required to be performed for execution of the algorithm. The second is the OCR *function* which will be called when framing is done in the initialization process and image is loaded into memory after pre-processing. *Header and Value* are basically two different versions of the same functions which will be used when OCR is performed on the image and the user wants to store the results in a readable format. Another important function in our model is *Bounding Rectangle* which will be loaded when OCR is performed, and we have to draw a minimum bounding area rectangle over the recognized text and values in an image. Through the bounding rectangle, we will be able to decide which area contains the required text and values and store them according to the data frame. Function call process is shown in a flowchart block is shown below in Figure 3.1.3.

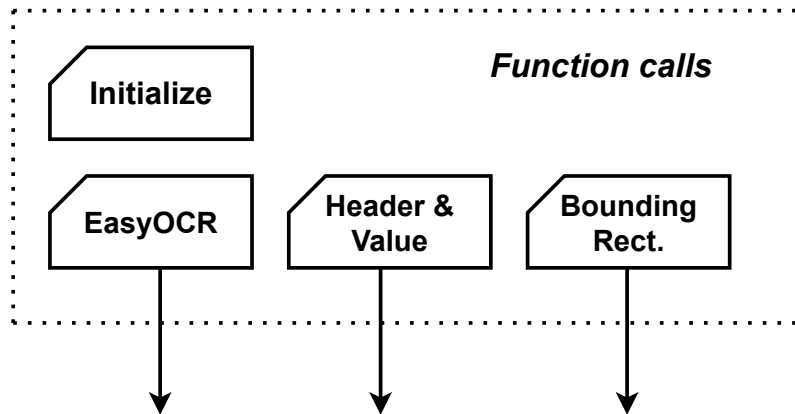


Figure 3.1.3: Function calls

3.1.3 Image Processing

Once the Initialization and function calls process has been done, the program will check if the image is stored and loaded in the memory. If it is true, then it will pass it to the Image processing block. The first function in the process is *HSV conversion* which will convert the image into HSV (Hue Saturation Value) format because to perform pre-processing techniques, we need to have the image in HSV format. After this function, the image is fed into the *Extract color channels* function which will extract different color values from the image and differentiate values based on the lower and higher value of each color. Normally in our dataset, the most occurring parameters have the color range of green, yellow, red, and white so initially we implemented these four different color variants in our program and applied extraction on these colors. After extraction, the extracted color image is fed to *Bitwise Image Masking* function which will apply pixel to pixel AND operation on the original image to mask just the extracted color pixel values on our original image to maintain the image resolution as we have initially for the image. The next step in image processing is *Binarization* which will threshold the values of extracted color channels in 0

and 1 to get the grayscale image. After the discussed pre-processing techniques, the image is loaded into the OCR *recognition engine* and extracted the strings of text and values and in parallel, loaded into the Adding *bounding rectangle* function to draw a minimum area bounding area rectangle on our image to further process the results in the next process. The complete flow-chart for Image processing process is shown in Figure 3.1.4

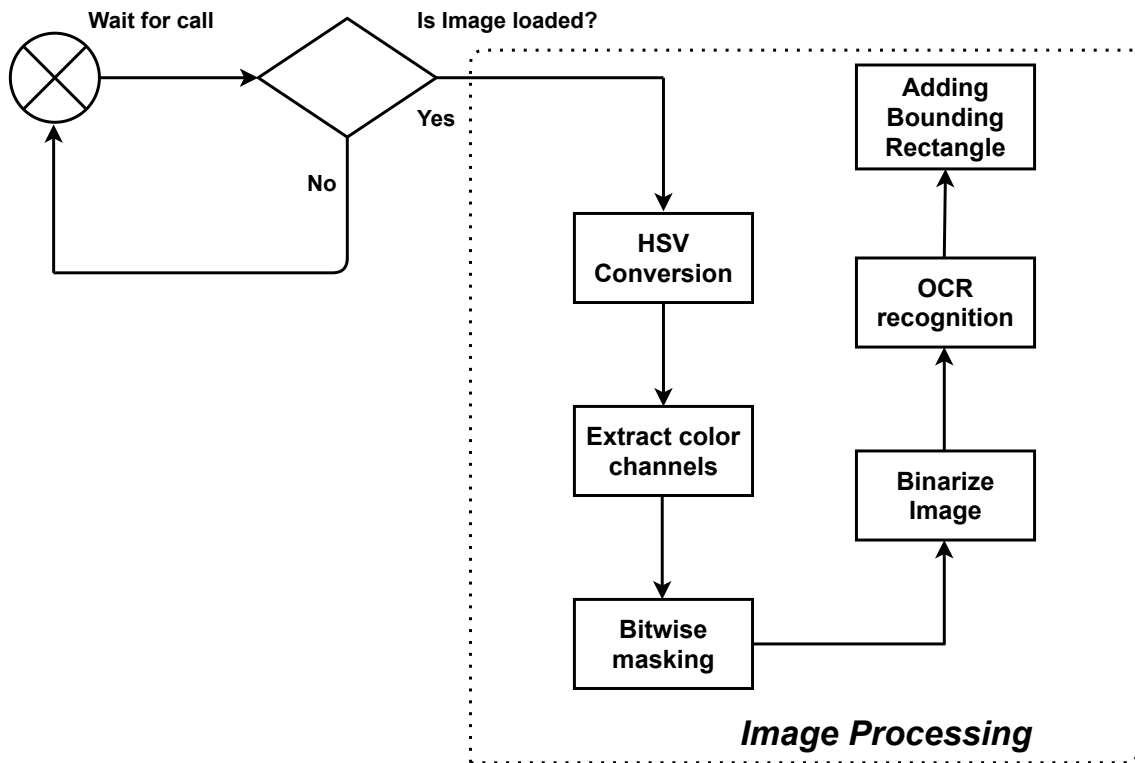


Figure 3.1.4: Image Processing process

3.1.4 Post processing

The last process which we also call post processing is a data frame process which will extract the text and values located in a bounding box to save it in the data frame created manually. The headers which we also call as parameters in our images and values are stored accordingly and this function will return the results to the user. After the result data frame has been returned, the system will return to the waiting state and wait for the next image to be loaded in the memory and restart the OCR engine to start the next process accordingly. Note that the framing function in the initialization process will run only one in the start but image processing and data frame processes will run in a loop till we come up to the processing of all the frames extracted from one video. Data frame module flowchart is shown as Figure 3.1.5

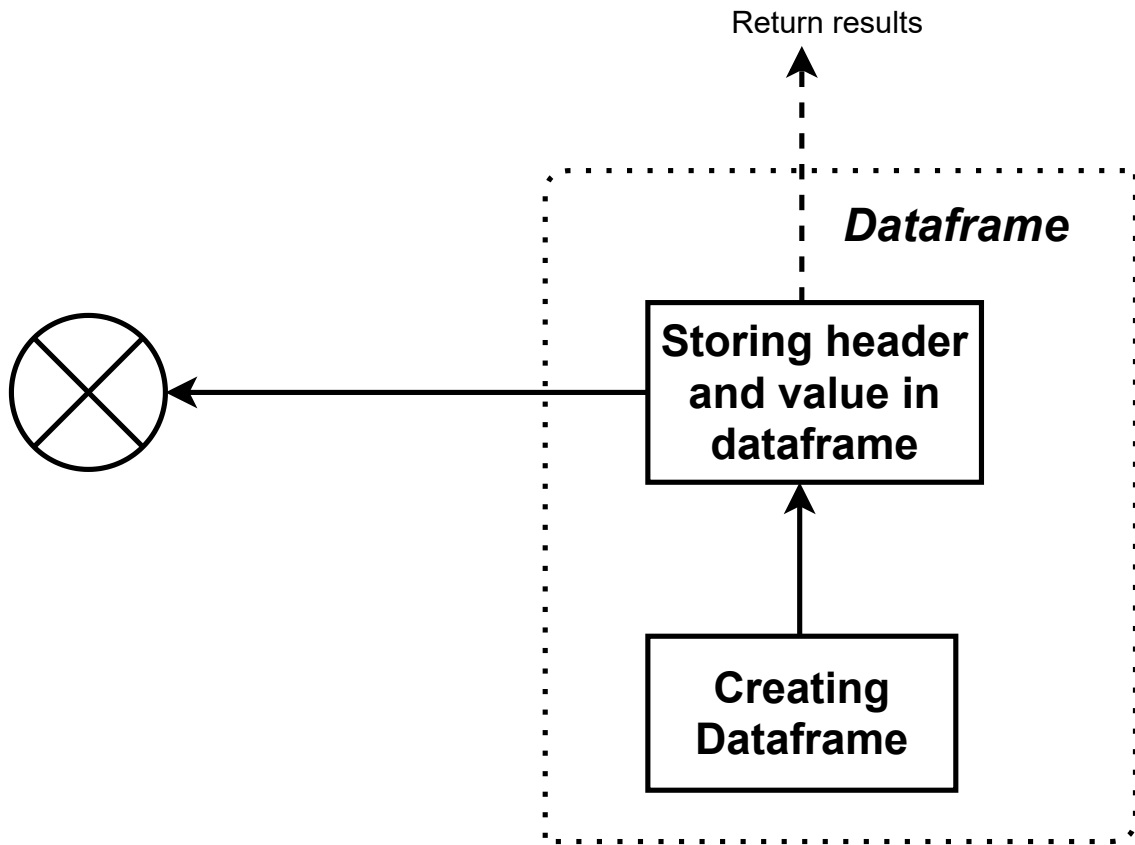


Figure 3.1.5: Post-processing

3.2 Implementation of Algorithm

3.2.1 Prompter and Framing function

When the program starts, the prompter function will be called. Prompter is a python built-in script which is used to ask users to open any file either with an option of all files or of specific extensions. By calling the library it will open a dialogue box as shown in figure 3.2.1

After loading the video in a memory, it will be sent to framing function. This step helps us extract the required information from these videos. In our case, 1 frame per second is enough as the information does not change between the second so if the actual video is of one minute, then 60 frames will be extracted as images. For the video to image frames extraction, a function called *getFrame()* is used in python that captures different frames from the video on a given threshold timestamp and returns the frames as images. In the *getFrame()* function, OpenCV function *cv2.VideoCapture(file_name)* is used to read the frames of video. This function will automatically align the frames of videos in a sequence. After capturing the frames, we used *cv2.imwrite()* function to save the frames on a local disk in a directory.

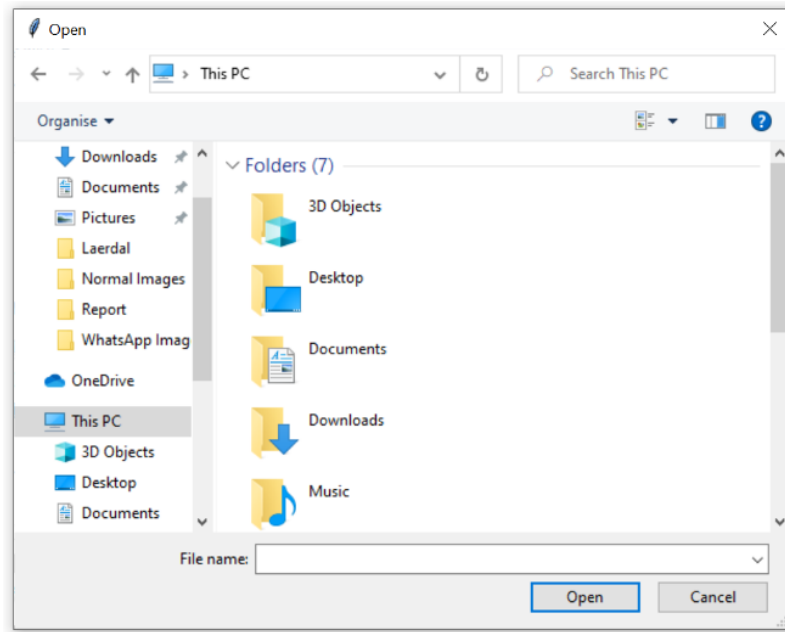


Figure 3.2.1: Prompter: opening file

3.2.2 Image Pre-processing

After extracting the frames from video and storing it on a local disk, the program will return the result as a message “*Framing is done*”. Then images will be accessed using OpenCV’s function `cv2.imread` to load each image into memory to further pre-process the image.

`hsv_conversion()`

Image that is read using OpenCV library is in the default color format in OpenCV which is not RGB but BGR (Blue Green Red). Why it was developed this way is a historical decision that comes from the fact that back then BGR color format was common among camera manufacturers and software providers for Windows. But to further pre-process the images, we want to convert our images in HSV (Hue Saturation Value) format. To serve this purpose, we used `cv2.cvtColor(input_image, flag)` where flag determines the type of conversion. In our case the flag will be `cv2.COLOR_BGR2GRAY` [5].

3.2.3 Image Masking

Next step is to find the image masking in each image for video. Image masking as explained in the previous chapter in detail is used to detect objects or certain parameters using different color channels. By Image masking, it would be easier to extract the different values along with their values. To serve this purpose, initially we implemented four different color channels as Red, Blue, Green and White. For more different color codes, relevant discussion has been presented in future work section in chapter 6. Using the OpenCV library, `cv2.inRange()` is used. Normally for HSV, we have hue [0,179], saturation [0,255] and value [0,255] range. For different color channels, we will set the lower and upper range of color space by adjusting the value range of all the H,S, and V parameters.

`cv2.inRange()` will check if the array of pixel values in an image lies between the set lower and upper values, it will return just the threshold values and everything else other than that range will be set equal to 0.

The function will check the range according to the following formula:

$$dst(I) = lower_limit(I)_o \leq src(I) \leq upper_limit(I)_o$$

The parameters for `cv2.inRange()` conveys the following meanings in this formula:

```
src= Input array
lower_limit= lower boundary of array
upper_limit= upper boundary of array
dst= Output array after excluding the values inside the set limit
```

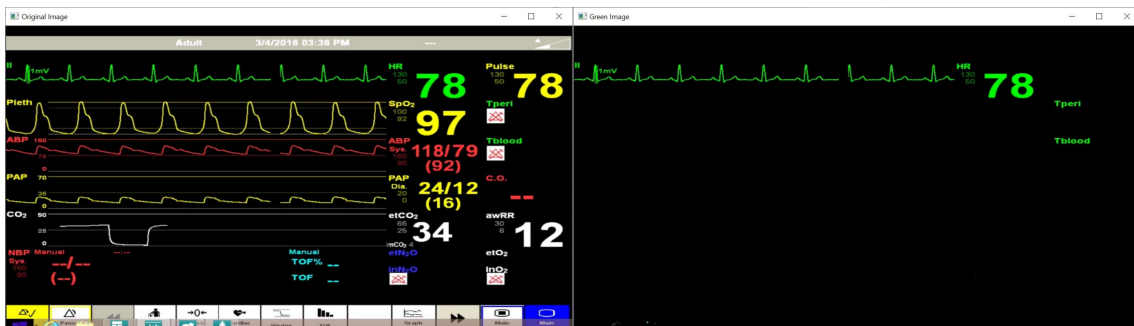


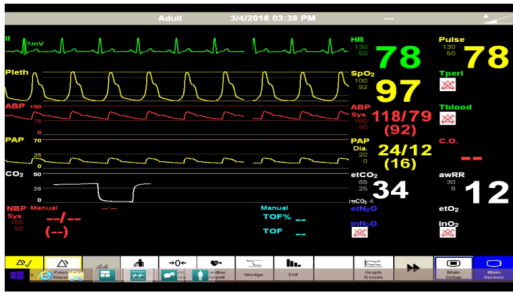
Figure 3.2.2: Output: from function `inRange()`

After getting the output from `cv2.inRange()` as shown in Figure 3.2.2, the next step is to apply the bitwise masking on an original image. The reason behind this logic is that when we apply the `inRange` technique, the output image we get is of low resolution, which in result could give us different results when feeding it to the OCR engine. Bitwise is a conjunction function provided by OpenCV to determine the same number of elements in an array which is also present in some other array as well. It has a variety of operation but we just want to use `cv2.bitwise_and()` [6] operation to mask the original image with the number of pixels with the masked pixel values of certain color channels to get the output image of the same resolution.

The parameters for `cv2.bitwise_and()` are:

```
src1= Input array
src2= Second array
dst= Output array
mask= Operational Mask (optional)
```

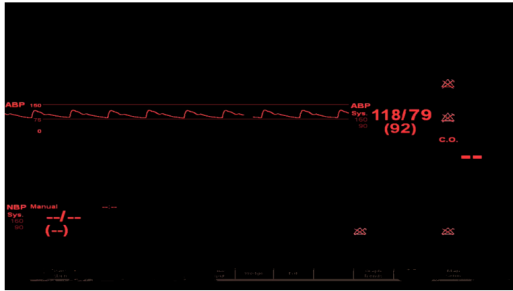
The output results after bitwise masking is shown in Figure 3.2.3



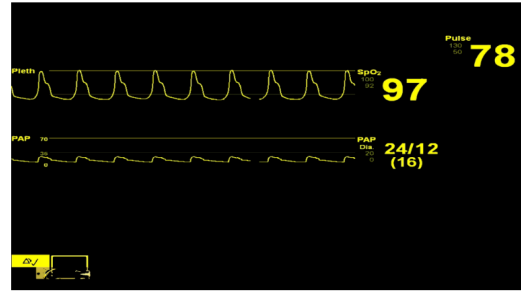
Original Image



Green Masked Image



Red Masked Image



Yellow Masked Image



White Masked Image

Figure 3.2.3: Output: Bitwise masking

3.2.4 OCR Recognition

Now the pre-processing has been done according to the algorithm, the next step to implement is to feed the processed image to OCR. As the theory chapter clearly explained the OCR, the engine will recognise the parameters and values in an image line by line. After image masking, we will have four different images with different parameters of different colours set as Red, Green, Yellow, and White. We store the different images in different variables to process them using OCR according to their colours.

3.2.5 OCR using Tesseract

As discussed previously, Tesseract has a built-in function in the python library called pytesseract. The tesseract will be loaded into memory using following command line in a script:


```
pytesseract.pytesseract.tesseract\_cmd = r'C:\ProgramFiles\Tesseract-OCR\tesseract.exe'
```

After loading the Tesseract into memory, we have to set some parameters prior to feeding the image for processing. There are various parameters such as getting the output in required format. The following parameters have been set for custom configuration. `custom_config = '-oem 3 -psm 6'` where,

`oem` = OCR Engine Mode. Default = 3, based on what is available.

`psm` = Page Segmentation Mode. Default=6 which assumes a single uniform block of text

Since when we applied colour masking on an image with different color channels, the output we received has parameters, values as well as the waveforms in an image as shown in Figures 4.1.2 and 4.1.3. So, when applied tesseract OCR on the image, it will recognise those waveforms also a character or words. The complete detailed results are shown and explained in the results chapter.

3.2.6 OCR using EasyOCR

EasyOCR has also a built-in function available by pytorch. We import the function using `command import EasyOCR`. EasyOCR has a variety of different languages available in a package. So, for our purpose, we need to load the English language in our case using `easyocr.Reader(['en'])`. For OCR, we have defined two different functions named `OCR_headers` and `OCR_values`. To explain this let's consider the Figure 3.2.4. We have different kinds of text and numbers in the sample image but to give the model a way on what exactly to return the parameter and its corresponding value, we have to set some prior parameters to exactly look into the specific values. As carefully observing the values and texts in the images, we can see that to differentiate specific values and text, we can do that by differentiating them based the font area size of each word and since we know that we want our parameter to be called as HR and its corresponding value to be 77 which is the word with the biggest font in the image, we can differentiate and extract the required text and numbers from the image by setting some parameters.



Figure 3.2.4: Parameters format with corresponding values

3.2.7 OCR_header()

When the image is retrieved by *OCR_header()*, the OCR will scan the image and at first point, it will return all the texts and values located in an image. As we have almost the same font size of parameter and the other values so we will differentiate the numbers with the text using the following code listings in Listing 3.1. The results extracted from code listing 2 will return the string of just the headers and it will omit all the integer values in an image and the output will be sent to the bounding area rectangle function as well as create data-frame function, which we will discuss later in this chapter.

```
result = reader.readtext(color,min_size=35,detail=0,text_threshold=0.93)
final_header = []
for item in result:
    if get_all_num(item):
        continue
    elif get_mix_strings(item):
        final_header.append(item)
    elif get_all_alpha(item):
        final_header.append((item))
```

Listing 3.1: code listing: differentiate the numbers from text

3.2.8 OCR_values()

After getting the results for header values and sending them for further post processing, now we have to store the integer values for the corresponding header. For this purpose, *OCR_header()* function is defined in which an image is sent for processing. OCR will be applied on the image again and store all the results in a list. Now to extract only the integer values, this time we will use only the if statement of *get_all_num(list)* from above

mentioned code listing. Note that as we discussed previously in figure 4.1.4, we have several integer values in an image having different sizes. To get the only bigger integer value, we will use the parameter in EasyOCR called *minsize*=' ' to threshold the values less than the set value. In our case, we set the *minsize*=80 to select only the biggest value in an image and store it in a list and pass it on to the data frame function to manage the further post processing. The above discussed logic is implemented in a code listing below in Listing 3.2

```

result = reader.readtext(color,min_size= 80,detail=0,text_threshold=0.93)
value = [x for x in result if any(x1.isdigit() for x1 in x)]

```

Listing 3.2: code listing

3.2.9 Bounding Area Rectangle

The bounding area rectangle is applied after finding the OCR headers and values to safely measure whether the required text and integers have been correctly recognised or not. It is important not only to detect the key parameters but also for the user to manually observe from time to time whether the model is able to recognise the parameters correctly and efficiently. Another advantage of bounded area rectangle could also be to feed the model which exactly text to store in a data-frame based on the area of rectangle i.e. we can also train the model to store the biggest integer value based on the bounded rectangle which has the highest value of area in an image. But in our case to make the model less complex, we used the parameter of *minsize* to train the model in a simpler way and save computational power. The output result for the bounding rectangle is shown in Figure 3.2.5.

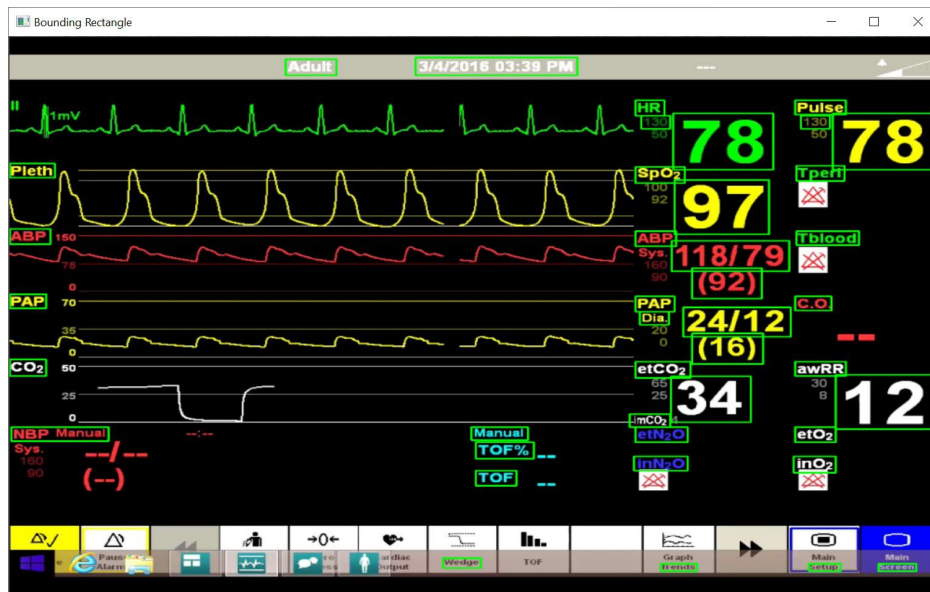


Figure 3.2.5: Result of bounding rectangle

3.2.10 Post Processing

Now that our major implementation of Optical Character Recognition has been done. The final step of our algorithm is to store the parameters and its corresponding values to a data frame with a specific order. To store the values and headers in a data-frame, we first need to create a dictionary or list of headers which we will use to match the results obtained from the OCR and if it matches, then we will store them accordingly as a header in our own created data-frame. After that the values obtained from *OCR_values()* will simply append in a data-frame in a specific column of its parameter detected at an earlier stage. The output result of the data-frame created is as showing in Figure 3.2.6. The complete implementation of algorithm using python is available.[15]

| | HR | ABP Systolic | ABP Diastolic | SpO2 |
|----------|-----------|---------------------|----------------------|-------------|
| 0 | 76 | 125 | 80 | 98.0 |
| 1 | 80 | 41 | 98 | NaN |
| 2 | 76 | 125 | 80 | 98.0 |
| 3 | 127 | 41 | 98 | NaN |
| 4 | 76 | 125 | 41 | 98.0 |
| 5 | 76 | 80 | 41 | 98.0 |

Figure 3.2.6: Dataframe of results

Chapter 4

Experimental Results

This chapter focuses on the experimental procedure demonstrating that the implementation was tested on different sets of data.

4.1 Image gathering and preparation

Training data

Implementing and evaluating the recognition model requires data in the form of images. As discussed earlier in chapter 4, this model will use images generated with the help of initialization process flow. The following training set of images were used to evaluate the results:

- Images captured:
 - Number of Videos: 20
 - Rate of frames captured from each video: 1 frame per second
 - Total number of images: 8000
- Resolution settings:
 - Resolution: 1280 x 1024
 - Color model: RGB

Images annotation

To evaluate the accuracy of the results acquired from the recognition model, the images need to be annotated. For each image, bounding boxes need to be made around each key value and label them as their corresponding parameter. To make the bounding boxes around the values, labelling software was used. To label each image, coordinates of the boxes and the object name were saved in a separate .csv file.

The SimCapture’s provided data and some other patient monitor interfaces over the internet, one thing seemed common in every interface was the name of the parameters of different patient record keys and the values against those key parameters. Although it could be of different sizes and colours but that doesn’t really affect the efficiency of the model. Figure 4.1.1 shows the different key parameters which are used to be trained by both OCR models.



Figure 4.1.1: Parameters to be recognized by OCR Model

4.2 Recognition in Perfect Conditions

The first experimentation was done when the conditions were perfect, meaning the resolution of the videos were as according to mentioned in the training data set, and symbols were the same as trained to the model. This experiment was done to ensure the model’s efficiency whether it could detect the parameters and the values against those parameters accurately under similar conditions or not. The results of these recognised values and areas can help us determine the other results found by different conditions both passed through Tesseract OCR and EasyOCR. To find the accuracy of the detected parameters in a video, first we need to have a ground truth value for specific data in a video. As described in the annotation section of this chapter, we labelled a set of images for each video and compared the results achieved from OCR detected results to find the accuracy results. Table 4.2.1 presents the parameter names both as labelled and detected. Also, the percentage confidence and classification results have been represented as True or False both for Tesseract and EasyOCR. The recognition results presented in this table are for images which have same coordinates for each object i.e. a set of 400 images containing fixed parameters and area but varying values.

| Parameters | Recognition | | Classification by | | Confidence% |
|------------|-------------|---------|-------------------|---------|-------------|
| | Tesseract | EasyOCR | Tesseract | EasyOCR | |
| HR | lala | HR | FALSE | TRUE | 45 |
| Pulse | Pulse | Pulse | TRUE | TRUE | 96 |
| SpO2 | S\$po02 | SpO2 | FALSE | TRUE | 32 |
| Tperi | Tperi | Tperi | TRUE | TRUE | 92 |
| etCO2 | etCO2 | etCO2 | TRUE | TRUE | 94 |
| awRR | awRR | awRR | TRUE | TRUE | 92 |
| inO2 | inQo | inQ2 | FALSE | FALSE | 41 |
| PAP | PAP | PAP | TRUE | TRUE | 96 |
| ABP | ABP | ABP | TRUE | TRUE | 94 |
| c.o. | on OF | C.O | FALSE | TRUE | 23 |

Table 4.2.1: Recognition and classification of various parameters in Perfect Conditions

The clear trend in the table can be read that the parameters with below 83% confidence are not able to recognize correct parameters. However, some of the parameters were quite close to the actual results with low confidence values as well. It is also visible that for SpO2 and inO2, the model was able to detect the first one completely correct. However, for inO2, it detected the parameter a bit misleading however we could make it correct in the post processing by defining a dictionary. It could also lead us to some more pre-processing techniques such as edge detection with different parameters to correctly classify and differentiate two different parameters. It would be considerate to conclude that OCR works to perform character recognition to some extent even with perfect conditions of saturation and pixel sharpness, but the tuning can be performed more efficiently to correctly classify some of the wrong detection in a variety of datasets.

4.3 Recognition in Varying Conditions

To recognise the parameters under different filter techniques of image processing, the objective was to test the robustness of the model under varying conditions. The filters were used to find the probability of recognition of parameters instead of correct classification (as in the case of perfect condition). The filters applied to different images were:

- Image blurring with kernel size = 10
- Adding gaussian noise.
- Image pixilation.

After applying mentioned filters to the images, even though the human can still clearly read the key parameters easily, OCR worked very differently on each parameter. The table depicts the images of some of the parameters that were tested by applying different filter techniques. To justify the table results graphically, the Figure 5.3.1 shows the correlation of results with corresponding percentage confidences of the images under different filtering approaches. The table 4.3.1 and Figure 4.3.1 show the promising results towards both OCR

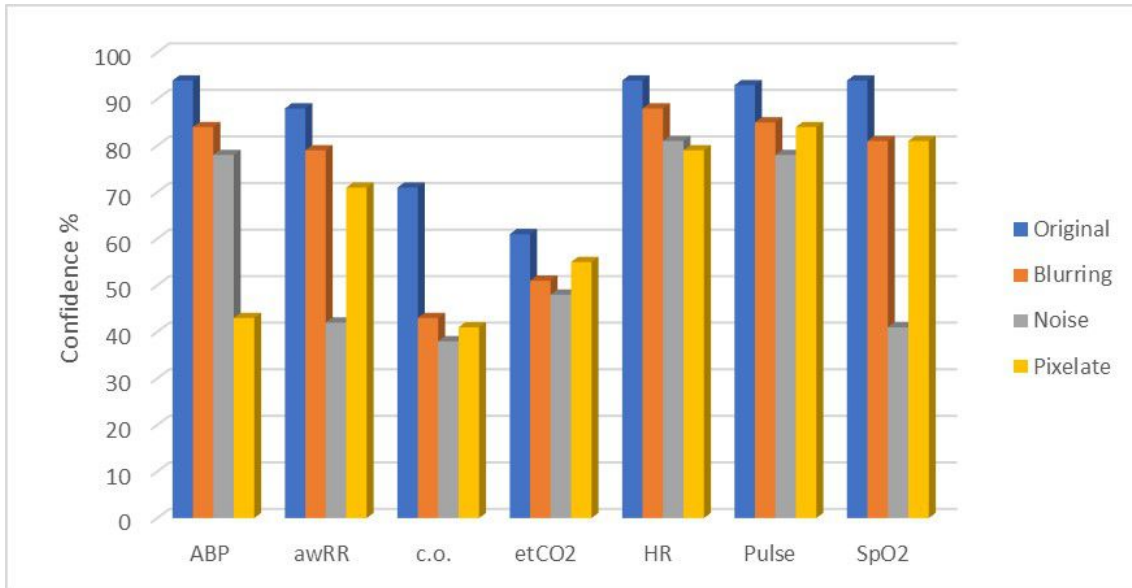


Figure 4.3.1: Correlation of Results

engines under varying conditions. As carefully observing the graph trend of confidence towards various filters, it could be promising to conclude that pixelate and noise filters don't increasingly affect the resulting outputs of various parameters such as awRR, HR, and Pulse. This could be an advantage to make a credible decision that pre-trained reflects an achievable output even if we have noise in certain parameters even though we assumed at initial stage the videos to be of higher resolution and trained images over specific datasets would be enough to acquire good results.





























| Name | Original | Blurred (Kernel=10) | Noise | Pixelate |
|-------------------------|---|---|--|---|
| ABP |  |  |  |  |
| Tesseract | TRUE | TRUE | TRUE | TRUE |
| EasyOCR | TRUE | TRUE | TRUE | FALSE |
| awRR |  |  |  |  |
| Tesseract | TRUE | TRUE | FALSE | TRUE |
| EasyOCR | TRUE | TRUE | FALSE | TRUE |
| c.o. |  |  |  |  |
| Tesseract | FALSE | FALSE | FALSE | TRUE |
| EasyOCR | FALSE | FALSE | FALSE | FALSE |
| etCO₂ |  |  |  |  |
| Tesseract | TRUE | TRUE | TRUE | FALSE |
| EasyOCR | TRUE | TRUE | TRUE | TRUE |
| HR |  |  |  |  |
| Tesseract | FALSE | FALSE | TRUE | TRUE |
| EasyOCR | TRUE | TRUE | TRUE | TRUE |
| Pulse |  |  |  |  |
| Tesseract | TRUE | TRUE | TRUE | TRUE |
| EasyOCR | TRUE | TRUE | TRUE | TRUE |
| SpO₂ |  |  |  |  |
| Tesseract | FALSE | TRUE | FALSE | FALSE |
| EasyOCR | TRUE | TRUE | FALSE | TRUE |

Table 4.3.1: Recognition and classification of various parameters in varying Conditions

Discussing the Tesseract OCR, the most surprising fact is towards getting output of Parameters HR and SpO₂, it has been shown that under normal conditions of pre trained data, the Tesseract wasn't able to correctly recognize initially with normal images. However, it can match the correct output when applied noise filters to it. Normally we don't have such a kind of noise in our dataset but to check the model's efficiency, we did that which leads to the conclusion that to train the model with varying conditions, different filtered images should be considered.

4.4 Recognition in Low Resolution Conditions

By common rule of Image processing, images with low resolution result in inferior results and a lot of more pre-processing is required. As low resolution was the biggest challenge in this project which we will discuss later in this chapter, the experiment was performed and shown graphically in Figure 4.4.1. The experiment conducted by taking the image with highest resolution in the dataset currently we have, the normal images declare to be of highest resolution as 1280 x 1024. On the other hand, data with low resolution as 640 x 480 gives misleading results and eventually decreasing the confidence of recognition of various parameters.

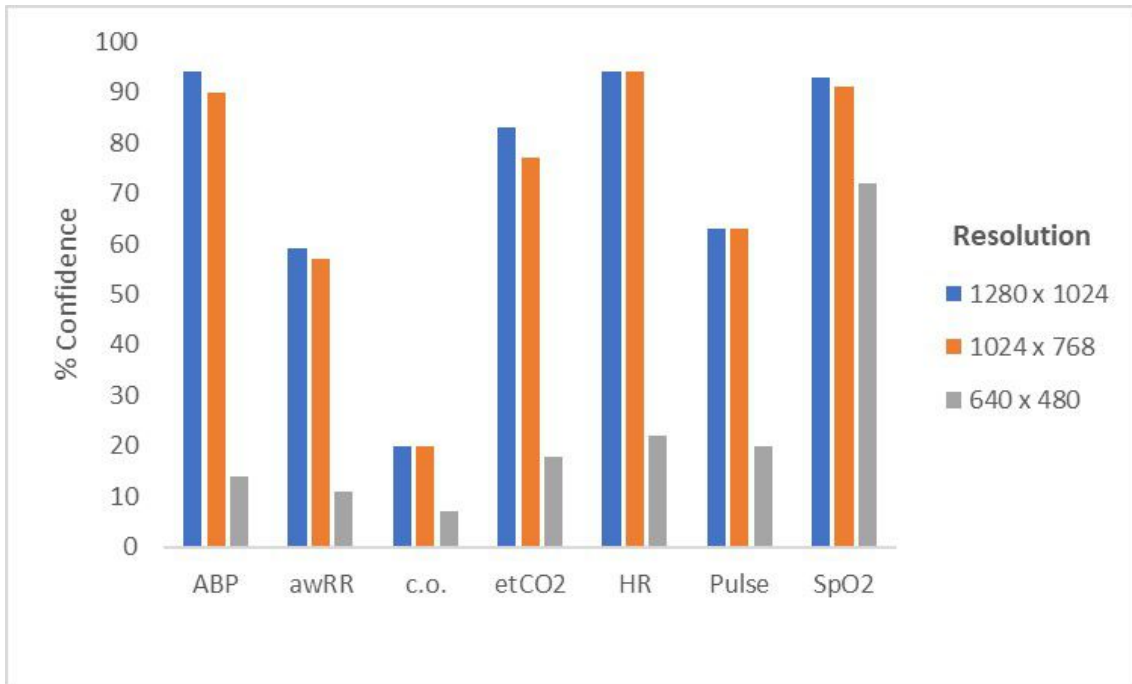


Figure 4.4.1: Results based on Resolution

The results clearly depict that as we try to recognise the parameters from lowest resolution videos in our dataset, there could be much compromise in our output results. Most of the results are not according to the expectation as when we came up to the resolution of 640 x 480. Especially in the case of c.o., it was not recognizable either at any resolution and percentage confidence is quite low.

4.5 Digit Recognition

To be able to evaluate the results from decreasing resolution in our dataset, the same experiment was conducted with digits in the frames captured from videos. As our dataset has a lot of low-resolution videos and some of them are even captured with external cameras hence of low pixel density. Due to that, Tesseract OCR is not able to recognize the parameters and digits correctly in most of the cases, hence we moved on with the implementation of EasyOCR. Even though the results were quite out of the reach but

still with better accuracy and some pre-processing, it was giving achievable results than Tesseract's OCR. Figure 4.5.1 displays the correlation between confidence interval and image resolution now with this time of the digits. This could provide a hint of how well our EasyOCR engine gave results with low-resolution videos.

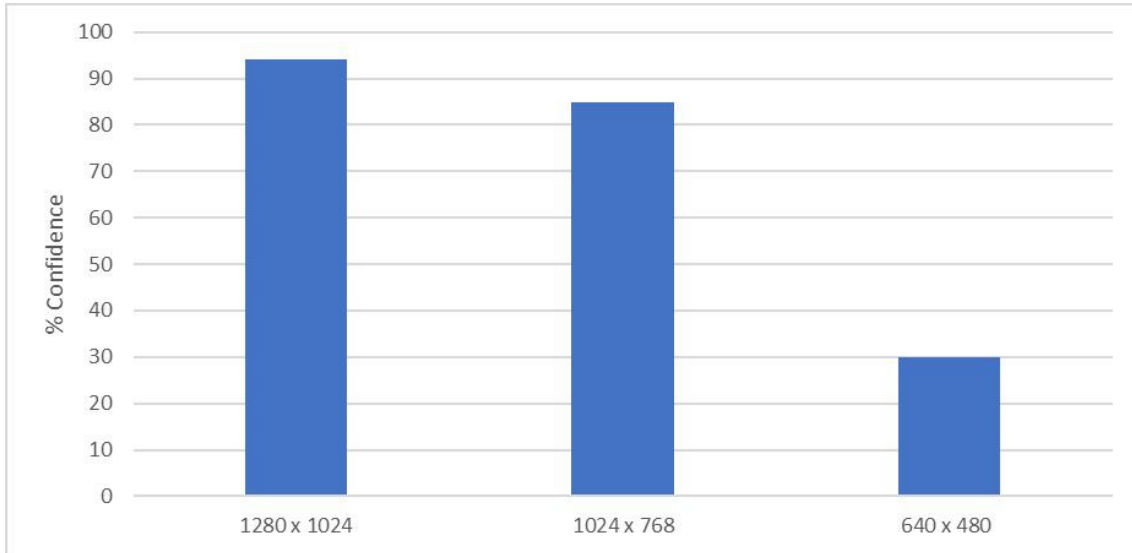


Figure 4.5.1: Correlation between confidence interval and image resolution

4.6 Timing Efficiency

System configuration and environment

- Environment information:
 - OS: Windows 10 version: 20H2
 - Processor: AMD Ryzen 7 3700U
 - Graphic module: Radeon Vega Mobile Gfx 2.30 GHz, 16.0 GB RAM

To test the timing efficiency of both OCR engines, a timer was initiated before a process began and stopped after it was completed. To get the average of the timing, a process was ran 10 times to get the average timing results. Initially we tested the process for less frames in a video i.e. 400 frames to calculate the time efficiency of processes for both Tesseract and EasyOCR. Note that the timing is calculated on the images without any pre-processing steps just to verify the OCR model's processing times on a same kind of dataset. The figure 4.6.1 depicts the timing in seconds for both engines as below:

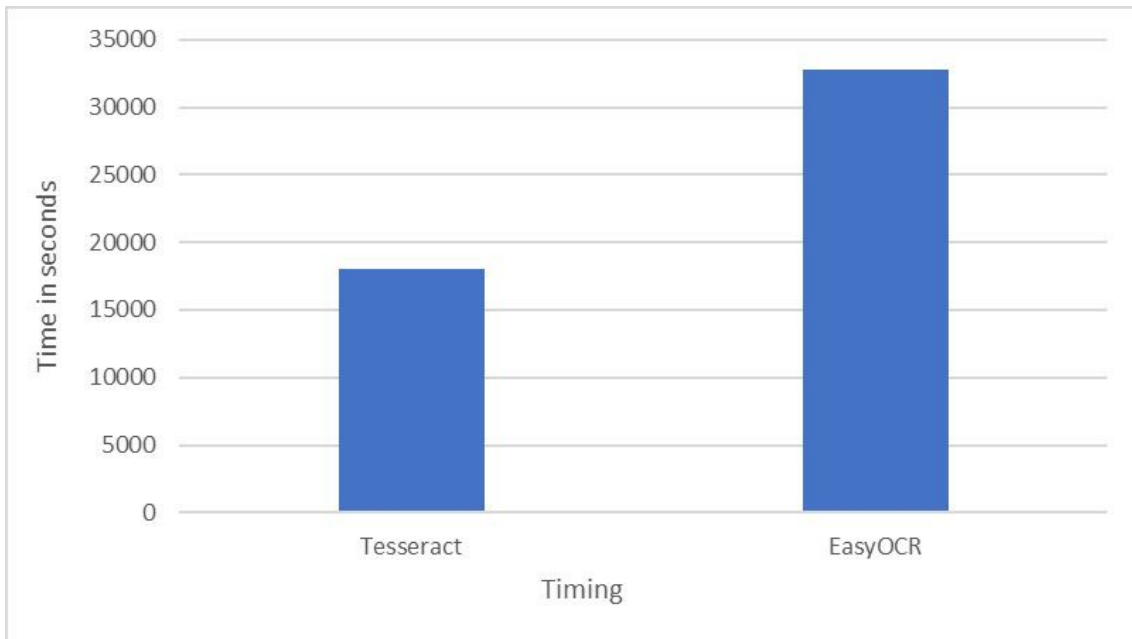


Figure 4.6.1: Processing time comparison of both engines

It is obvious to say that EasyOCR gains a lot of more processing time as compared to Tesseract. Even though it is clearly stated in EasyOCR official documentation that this process runs faster on GPU but to measure the time efficiency of both models, same configuration were applied and tested the compared the values for models.

But on the other hand, there is a clear trade off as discussed above in this chapter, the output results achieved with the help of EasyOCR were quite correct as compared to Tesseract so we could say to summarize that EasyOCR takes almost 40% more computation power as compared to Tesseract including dataset with low resolution and heavy filters applied but gives more accurate results. Below figure 4.6.2 depicts the final comparison between Tesseract and EasyOCR with respect to accuracy and computation power.

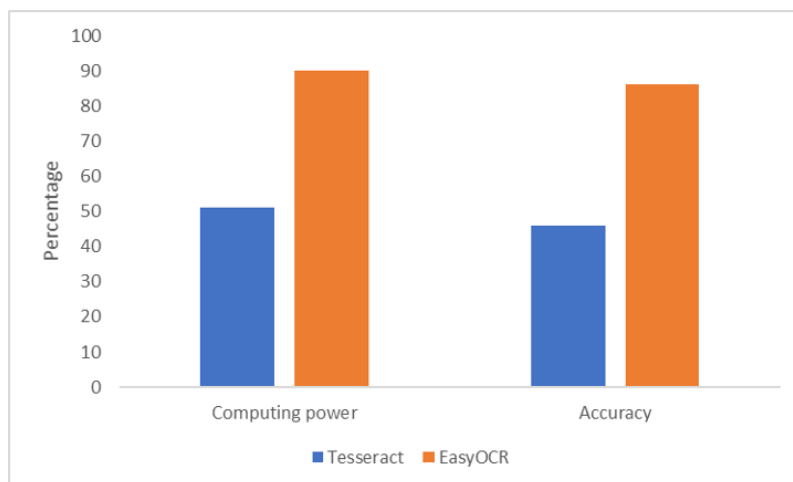


Figure 4.6.2: Comparison of both engines w.r.t accuracy and computation power

Chapter 5

Evaluation

5.1 Analysis of the Results

The results from recognition using Optical Character Recognition states that it is possible to correctly classify the texts and values located anywhere in the image. Even though the OCR engines were tested on different sets of images by applying various kinds of filters, they were somehow able to recognise most of the parameters. This depicts the prominent accuracy of OCR engines that we implemented in our model. But the negative side of it proves that some parameters which we assumed before implementing our algorithm and model that will be recognised easily didn't come up with the correct results even after applying several more pre-processing techniques and fine tuning of the training data. We tried to figure out any solution for the miss-classified characters by retraining the data also but every time, the output results were the same.

Another important key feature of OCR is that it identifies the characters, symbols and other important key features based on the trained features of that parameter. OCR finds features of any trained parameter and investigates trained data for an exact match. If it matches, then it gives us the output. This means that if we have a validation image of the same features but with low resolution, OCR doesn't work very well on the image and we don't get our desired output correctly. This means that OCR still has a lot of potential for improvement and should give the desired results as we get in the case of Image registration.

5.2 Discussion

Optical Character Recognition engines give quite good results based on the logic of Artificial Neural Networks but still there is a lot of more effort needs to be done to make these tools more accurate because lack of generalized models seem to be the problem as they are very accurate on different OCR tasks. The tools surrounding the training of OCR needs a lot of improvement to be able to retrain the model by any external developer's own data set.

The training process of any model is undoubtedly the most important step in any project but due to lack of proper documentation and a proper feedback during imple-

mentation steps is another issue that we faced and make the proper functionality of our model. Another lacking feature of OCR is the box file in which we set all the parameters to train the model with our own symbols and location of each parameter within an image. Right now, we must manually adjust the parameters for each character to be able to recognise them correctly. The training process does not tell if something is missing and provide any additional parameter to process the images efficiently which results into a very time-consuming process.

5.3 Limitations and Future work

The implemented algorithm calculates the parameters more accurately on higher resolution images. It requires more pre-processing steps to make it dynamic to be used in the long term. Even though it provides an overall good result but still lacks some pre-processing for lower resolution values.

In future, it could be a useful approach to detect automatically all the pixel color values of parameters that are not black, and then we could classify all the parameters based on those pixel intensity values to differentiate and feed it to OCR model. Currently we implemented only four colors in our model.

To get good results using deep learning neural network models for patient monitor screens, more convolution layers can be added with an addition to fine tuning that could lead us to get achievable results and to correctly find the region of interest (ROI) from a video to implement OCR technique on it. Using a deep learning neural network model, error rate would be much less as compared to the model implemented using traditional image processing techniques and it consumes low manual inputs.

Chapter 6

Conclusion

Optical character recognition has been applied in models where we try to recognize the numbers and letters from an image based on their formation like lines, curves, and angles, after pre-processing the input images. In this report, the data from SimCapture in the form of videos has been analyzed to identify different important inputs in forms of text and respective numbers to store it into dataframes for further use in medical training programs. For this purpose, a proposed model has been developed using different OCR techniques. The implemented model has been tested on the dataset of several patient monitor screen videos containing different patient records in different screen formations.

Validation of the model required manually annotated videos containing all the key parameters and their corresponding values to implement the model on those specific videos and get the output results. These results were compared with the annotated results to predict the accuracy of the model. The results achieved from this model and the comparison with the traditional image processing models suggested that we can use this model for better performance with such kinds of dataset. Since, there are limitations to the resources owned by the organizations so our target was also to achieve low-complexity solutions when dealing with such real-world applications. After carefully observing the results, there has been a significant trade-off between different OCR engines i.e. Tesseract and EasyOCR. On the dataset, we have used in this project, Tesseract consumed less computational power but gave less accurate results. On the other hand, EasyOCR presented better output results, but it consumed more computational power and was also time consuming. The objective of this project was to design a low-complexity solution to store the data from patient monitoring screens based on their manufacturer and able to achieve good accuracy from the trained model to work with unforeseen data. In our case, EasyOCR was the best choice to get the required output efficiently.

Bibliography

- [1] Abbyy — the digital intelligence company. <https://www.abbyy.com/>. ()
- [2] Github - jaidedai/easyocr: Ready-to-use ocr with 80+ supported languages and all popular writing scripts including latin, chinese, arabic, devanagari, cyrillic and etc. <https://github.com/JaiededAI/EasyOCR>.
- [3] What is image masking? why do you need it? URL <https://www.colorexperthsbd.com/blog/what-is-image-masking/>.
- [4] Intelligent document processing with ai — nanonets. <https://nanonets.com/>. (Nano Net Technologies Inc.).
- [5] Opencv: Changing colorspace. https://docs.opencv.org/master/df/d9d/tutorial_py_colorspaces.html, .
- [6] Opencv: Operations on arrays. https://docs.opencv.org/master/d2/de8/group_core_array.html#ga60b4d04b251ba5eb1392c34425497e14, .
- [7] Real-time object detection with yolo, yolov2 and now yolov3 — by jonathan hui — medium. <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>.
- [8] Segmenting words and characters — how ocr works. <https://how-ocr-works.com/OCR/word-character-segmentation.html>.
- [9] Tesseract user manual — tessdoc. <https://tesseract-ocr.github.io/tessdoc/Home.html>. (User manual).
- [10] [tutorial] ocr in python with tesseract, opencv and pytesseract. <https://nanonets.com/blog/ocr-with-tesseract/>. (Accessed on 06/07/2021).
- [11] End-to-end multi-lingual optical character recognition (ocr) solution. URL <https://pypi.org/project/easyocr/>.
- [12] tesseract-ocr · github. <https://github.com/tesseract-ocr/>. (Accessed on 06/09/2021).
- [13] A. Agbemenu, J. Yankey, and E. O. An automatic number plate recognition system using opencv and tesseract ocr engine. *International Journal of Computer Applications*, 180:1–5, 05 2018. doi: 10.5120/ijca2018917150.

- [14] F. Asad, A. Ul-Hasan, F. Shafait, and A. Dengel. High performance ocr for camera-captured blurred documents with lstm networks. *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 7–12, 2016.
- [15] S. I. A. Bukhari. Github - itti0322/optical-character-recognition-using-patient-monitor-screen. <https://github.com/itti0322/Optical-Character-Recognition-using-Patient-Monitor-Screen>.
- [16] S. Ch, S. Mahna, and N. Kashyap. Optical character recognition on handheld devices. *International Journal of Computer Applications*, 115:10–13, 04 2015. doi: 10.5120/20281-2833.
- [17] Z. Chen and J. Chen. Mobile imaging and computing for intelligent structural damage inspection. *Advances in Civil Engineering*, 2014, 10 2014. doi: 10.1155/2014/483729.
- [18] doxygen. Contour features, Jun 5 2021. URL https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html.
- [19] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks, 2018.
- [20] M. J. Khalid. Introduction to image processing in python with opencv. https://stackabuse.com/introduction-to-image-processing-in-python-with-opencv. (article).
- [21] Laerdal. Patient monitoring screen by laerdal medical, march 2018. URL <https://laerdal.com/us/products/simulation-training/manage-assess-debrief/1leap/>.
- [22] D. Liedle. A brief history of optical character recognition. *filestack*, Nov 2018. URL <https://blog.filestack.com/thoughts-and-knowledge/history-of-ocr/#:~:text=The%20first%20OCR%20tools%20in%20modern%20history%20were,by%20interpreting%20Morse%20Code%20to%20read%20text%20aloud>.
- [23] G. Markus. Ocr example. *EuropeanaTech*, july 2019. URL <https://pro.europeana.eu/page/issue-13-ocr>.
- [24] R. Marry. Introduction to image processing. https://www.engineersgarage.com/article_page/introduction-to-image-processing/. (EngineersGarage).
- [25] S. Mori, H. Nishida, and H. Yamada. *Optical Character Recognition*. John Wiley amp; Sons, Inc., USA, 1999. ISBN 047308196.
- [26] G. Nagy. At the frontiers of ocr. *Proceedings of the IEEE*, 80(7):1093–1100, 1992. doi: 10.1109/5.156472.
- [27] G. Nagy and Y. Xu. Automatic prototype extraction for adaptive ocr. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1997. doi: 10.1109/ICDAR.1997.619856.

- [28] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.
- [29] R. Safabakhsh and S. Khadivi. Document skew detection using minimum-area bounding rectangle. In *Proceedings International Conference on Information Technology: Coding and Computing (Cat. No.PR00540)*, pages 253–258, 2000. doi: 10.1109/ITCC.2000.844226.
- [30] Slant. What are the best ocr libraries? URL <https://www.slant.co/topics/2579/~best-ocr-libraries>.
- [31] R. Smith. An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02, ICDAR '07*, page 629–633, USA, 2007. IEEE Computer Society. ISBN 0769528228.
- [32] R. smith. Training tesseract, 2014. URL <https://www.slideshare.net/temsolin/3-training>.
- [33] R. W. Smith. The extraction and recognition of text from multimedia document images. 1987.
- [34] Wikipedia. Minimum bounding rectangle, . URL https://en.wikipedia.org/wiki/Minimum_bounding_rectangle.
- [35] Wikipedia. Optical character recognition, . URL https://en.wikipedia.org/wiki/Optical_character_recognition.
- [36] Wikipedia. Tesseract(software), . URL [https://en.wikipedia.org/wiki/Tesseract_\(software\)](https://en.wikipedia.org/wiki/Tesseract_(software)).

Appendix A

A.1 Terminologies

- **Digital Image:** A digital image is the computer representation of an actual image with help of small picture elements. The digital image contains a range of colours with different colour values of RGB. In computers, the formats used to store digital images are JPEG (Joint Photographic Experts Group), TIFF (Tagged Image File Format), and PNG (Portable Network Graphics).
- **Binary Image:** A binary image is also a digital image but with only two colours i.e. black and white
- **Pixel:** In a digital image, the pixel is the smallest addressable element which contains the information of colours RGB. These pixels, combined, with different RGB intensities form a complete image, and/or display. Resolution for display is indicated by the number of pixels that forms the display i.e. 1080*1920 display means 1080 pixels in each row and 1920 pixels in each column and there will be a total of 2,073,600 pixels in the grid.