

Summer Training Report

On

*De - identification of Named Entities from Electronic Health
Records using One Shot Learning*

Submitted by:

Name: Rangabhatla Sriteja

Roll Number: 20116084

Submitted in partial fulfilment of the requirements for subject

Summer Internship I in V semester

Of

Bachelor of Technology

In

Electronics and Communication Engineering

Submitted to the Department of the Electronics and Communication
Engineering



National Institute of Technology, Raipur - 492010 (C. G.), India, 2022

CONTENTS

1. Declaration	2
2. Certificate	3
3. Abstract	4
4. Acknowledgement	5
5. Introduction	6
6. Chapter 1 - Digital Images	7
7. Chapter 2 - Digital Image Processing	10
8. Chapter 3 - Optical Character Recognition	17
9. Chapter 4 - MATLAB	18
10. Chapter 5 - Python	22
11. Chapter 6 - Libraries	25
12. Chapter 7 - Project	28
13. Chapter 8 - Further Development	40
14. Chapter 9 - References	41

DECLARATION

I certify that

1. The work contained in this report is original and has been done by me under the guidance of my mentor/supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have followed the guidelines provided by the Department in preparing the report.
4. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the reference.

Rangabhatla Sriteja

20116084

Bachelor of Technology

Department of Electronics and Communication Engineering

CERTIFICATE




NITRR/IT/INTERNSHIP/2022/1


NATIONAL INSTITUTE OF TECHNOLOGY RAIPUR
DEPARTMENT OF INFORMATION TECHNOLOGY

Summer Internship-2022

CERTIFICATE

This is certified that Sriteja Rangabhatla has completed one month summer internship on De-identification of Named Entities from Electronic Health Records using One Shot Learning from 14-06-2022 to 17-07-2022 in the Department of Information Technology, National Institute of Technology Raipur.


Dr. Tirath Prasad Sahu
(Supervisor)


Dr. Pavan Kumar Mishra
(Coordinator)

ABSTRACT

In this paper we are going to discuss the major operations that are operated on a digital image when it is given as an input to a Optical Character Recognition software. The text in the image is omitted out as an output. It is simply called pre-processing of the image. The main aim of the pre-processing is to increase the accuracy of the output of the Optical Character recognition software. This preprocessing acts as a bridging stone for the next steps. As the image contains skewing, noise, scaling errors, contrast problems etc. It will be hard for the Optical Character Recognition software to identify the text accurately.

For implementing the above operations there are multiple platforms. In this paper, I have mainly focused on Python and MATLAB softwares for coding. The library contains built-in modules that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardised solutions for many problems that occur in everyday programming. OpenCV, NumPy, Pytesseract libraries are used in this paper.

Keywords: *Pixel, Image processing, OCR, Intensity, Grayscale.*

ACKNOWLEDGEMENT

My internship with Dr. Tirath Prasad Sahu, Assistant Professor in the Department of Information Technology at the National Institute of Technology Raipur, provided me with a fantastic opportunity for learning and career advancement. I appreciate the guidance and support I received from my colleagues and mentors during the training and course. The internship was really well organised to lay the groundwork for the thoughts to come. I would like to use this opportunity to show my appreciation and special thanks to the Department of Information Technology for providing resources and facilities over the time period, allowing me to complete my project at this prestigious institution, and offering guidance. I am confident that this internship will advance my career and provide me with a variety of opportunities. In order to achieve my intended career goals, I will endeavour to utilise the acquired skills and information as effectively and practically as I can, and I will continue to work on their improvement. By the end of the internship period, I felt secure enough to create tiny programmes on my own and comprehend upcoming topics.

I also want to express my gratitude to the Department of Electronics and Communication Engineering of the National Institute of Technology, Raipur for approving and letting me enrol in an internship programme. I owe a great deal of gratitude to my family and friends who provided insightful advice and support for the completion of my project. I found this cooperation and constructive criticism to be beneficial.

Rangabhatla Sriteja

20116084

Bachelor of Technology

Department of Electronics and Communication Engineering

Introduction:

Introduction about the course:

The course is well structured starting with an introduction to digital image processing followed by learning various image processing techniques along with the requisites of MATLAB and python to implement the algorithms. Implementation of various image processing techniques were done in both the softwares in the following. Next up, regarding the Optical Character Recognition, various available techniques and softwares. The coursework encouraged theoretical and practical knowledge. The code is written in Python using various libraries of python such as numpy and pytesseract to run the matrix calculations and optical character recognition.

Organization of the Report:

The chapter of the report are organised as follows:

Chapter - 1 gives introduction to digital images, its notations. History and development of digital images along with the development of the sensors used to capture them.

Chapter - 2 gives insights about digital image processing. Development of various digital image processing techniques. Applications of digital image processing in various fields.

Chapter - 3 introduces us to the Optical character recognition and describes the factors that affect the accuracy of the text obtained.

Chapter - 4 and 5 describes the softwares used to code the processing techniques on image. They are MATLAB, python and their uses in digital image processing.

Chapter - 6 introduces the different libraries used in python like OpenCV, NumPy, Pytesseract

Chapter - 7 discusses the algorithms used to pre process the digital image to give it as an input to the optical character recognition software.

Chapter - 8 explains the further developments can be made and also can be implemented in other relatable scenarios.

Chapter - 1:

Digital Images:

A digital image is made up of picture elements, also referred to as pixels, pels, each of which has a finite, discrete quantity of numerical representation for its intensity or gray level as an output from its two-dimensional functions fed as input by its spatial coordinates denoted with the letters x , y on the x -axis and the y -axis, respectively. That is amplitude f at any coordinate (x, y) is known as gray level or intensity of the image at that point.

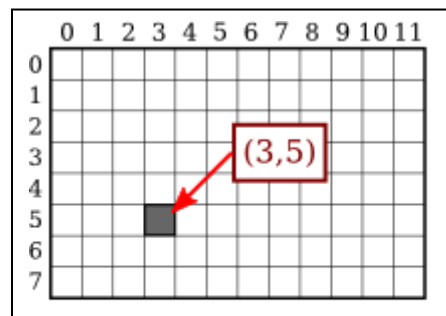


Figure 1 : digital image with coordinate system indicating location of the pixel.

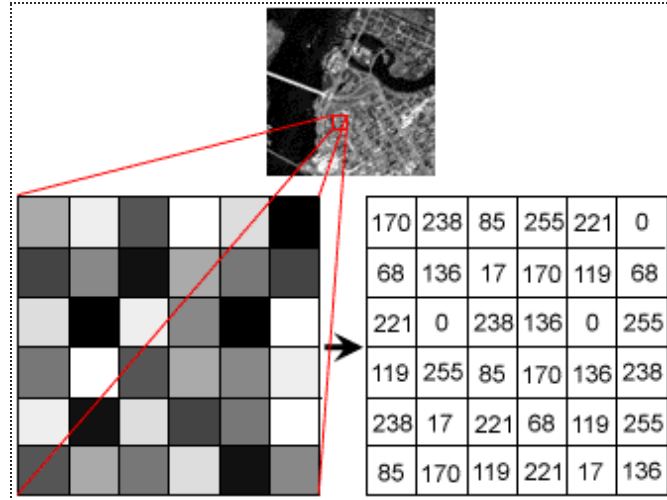


Figure 2 : a grayscale image on the left side and their intensity values on the right side.

History and developments of Digital Images:

In the early 1960s, digital imaging technology continued to evolve along with the advancement of the space programme and medical research. Digital images were employed in research projects at the Jet Propulsion Laboratory, MIT, Bell Labs, the University of Maryland, and other institutions to expand videophone technology, character recognition, medical imaging, satellite imagery, and other fields.

With the emergence of MOS integrated circuits in the 1960s and microprocessors in the early 1970s, together with advancements in related computer memory storage, display technologies, and data compression methods, rapid advancements in digital imaging started.

Medical diagnostics greatly benefited from the development of computerised axial tomography (CAT scanning), which uses x-rays to create a digital image of a "slice" through a three-dimensional object. Along with the creation of digital photographs, the digitisation of analogue photos enabled the enhancement and restoration of ancient artefacts. These images were then employed in a variety of sectors, including nuclear medicine, astronomy, law enforcement, the military, and business.

Towards the close of the 20th century, developments in microprocessor technology made it possible to produce and market charge-coupled devices (CCDs) for use in a variety of image capture devices, progressively replacing analogue film and tape in photography and videography. Computer-generated digital images can now approach photorealism thanks to the processing power required to handle digital image capture.

Imaging Sensors:

a. Charged Couple Diode:

The CCD, created in 1969 at Bell Labs by Willard S. Boyle and George E. Smith, was the first semiconductor image sensor. They discovered, while studying MOS technology, that an electric charge could be held on a tiny MOS capacitor and was akin to the magnetic bubble. They connected a sufficient voltage to them in order to step the charge from one to the next because it was relatively simple to construct a row of MOS capacitors. The first digital video cameras for television transmission used a semiconductor circuit called a CCD.

b. Pinned Photodiode:

Early CCD sensors had shutter lag issues. The development of the pinned photodiode greatly eliminated this (PPD). It was created at NEC in 1980 by Nobukazu Teranishi, Hiromitsu Shiraki, and Yasuo Ishihara. It was a photodetector structure with little dark current, little latency, little noise, and excellent quantum efficiency. The PPD started to be included into the majority of CCD devices in 1987, and by 1990, it was a standard feature in consumer electronic video cameras and later digital still cameras. Since that time, almost all CCD sensors and later CMOS sensors have employed the PPD.

c. Active - Pixel Sensor:

Olympus developed the NMOS active-pixel sensor (APS) in Japan in the middle of the 1980s. This was made possible by developments in the manufacturing of MOS semiconductor devices, with MOSFET scaling to micron and subsequently sub-micron levels. Tsutomu Nakamura's team of Olympus created the NMOS APS in 1985. Later, in

1993, the NASA Jet Propulsion Laboratory team led by Eric Fossum created the CMOS active-pixel sensor (CMOS sensor). Sales of CMOS sensors overtook those of CCD sensors by 2007.

Chapter - 2:

Digital Image Processing:

Digital image processing is the use of a digital computer to process digital images using an algorithm. Digital image processing has significant benefits over analog image processing as a branch or specialisation of digital signal processing. It permits the application of a considerably wider variety of algorithms to the input data and can prevent issues like the accumulating noise and distortion during processing. Digital image processing can be described as a multidimensional system because images can exist in at least two dimensions and possibly more. Three main factors, including the development of computers, mathematics, particularly discrete mathematics theory, and the rise in demand for a wide range of applications in the environment, agriculture, military, industry, and medical science, have a significant impact on the generation and development of digital image processing. The usage of digital image processing in some of the key industries is mentioned. Medical field, remote sensing, transmission and encoding, machine/robot vision, colour processing, pattern recognition, video processing, microscopic imaging, and others are just a few of the industries that use image sharpening and restoration.

History of Digital Image processing:

In the 1960s, Bell Laboratories, the Jet Propulsion Laboratory, Massachusetts Institute of Technology, the University of Maryland, and a few other research institutions developed many of the techniques used in digital image processing. These techniques have applications in satellite imagery, the conversion of wire-photo standards, medical imaging, videophone, character recognition, and picture enhancement. Early image processing was done to enhance the image's quality. In image processing, a low-quality image is used as the input, and a higher-quality

image is produced as the result. Image enhancement, restoration, encoding, and compression are all common types of image processing. The American Jet Propulsion Laboratory, or JPL, produced the first successful application. On the countless lunar photographs returned by the Space Detector Ranger 7 in 1964, they applied image processing techniques including geometry correction, gradation transformation, noise reduction, etc. while accounting for the sun's position and the moon's surroundings. The computer's successful mapping of the moon's surface has had a significant positive influence. Later, more intricate image processing was applied to the over 100,000 photographs that the spacecraft brought back, yielding the topographic map, colour map, and panoramic mosaic of the moon, which produced astonishing results and set the stage for a successful landing of astronauts on the moon. However, the computing hardware at the time made processing quite expensive. This changed in the 1970s when more affordable computers and specialised equipment became accessible and digital image processing spread. Due to this, images were eventually handled in real-time for some specific issues like converting television standards. General-purpose computers began to replace specific hardware for all but the most specialised and computationally intensive tasks as they became faster. Digital image processing is now the most widely used type of image processing because of the quick computers and signal processors that became available in the 2000s. This is because it is not only the most flexible way, but also the least expensive.

Applications of Digital Image Processing:

a. Image sharpening and restoration:

Here, the terms "image sharpening" and "restoration" refer to the actions used to improve or edit camera-captured photos in order to obtain desired results. It refers to the standard functions of Photoshop. Zooming, blurring, sharpening, converting from grayscale to colour, detecting edges and vice versa, image retrieval, and image recognition are included.

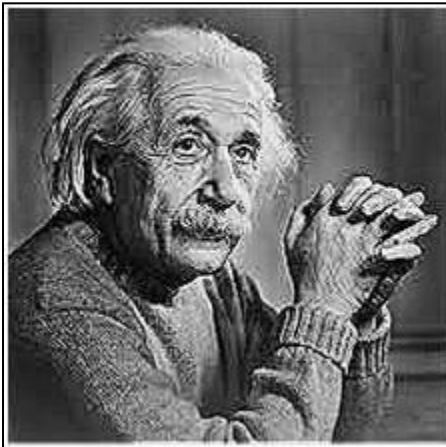
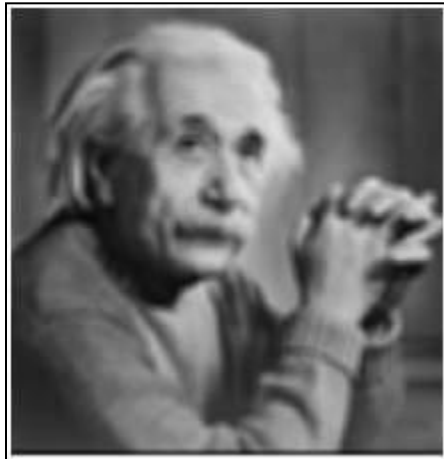
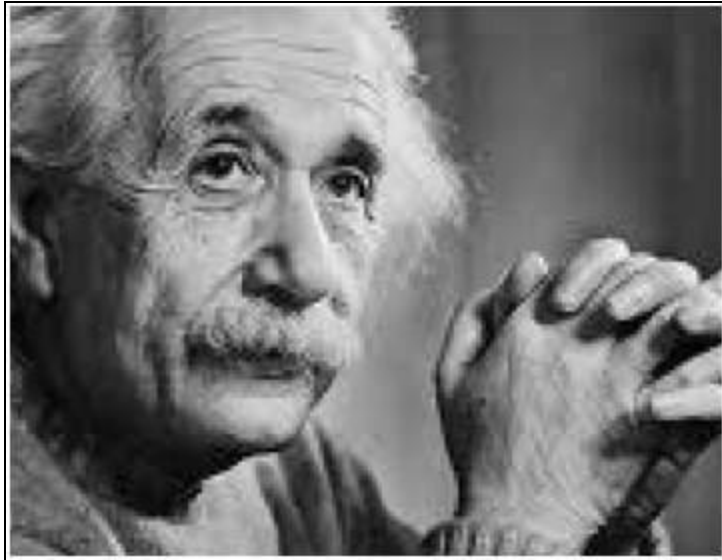
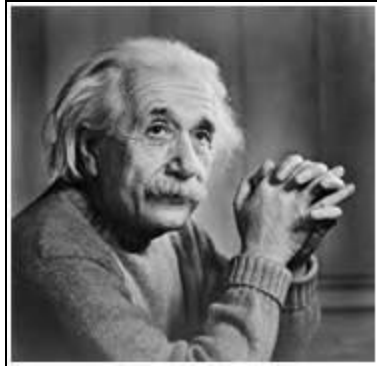


Figure 3 (row-1, column-1) : an image of Einstein

Figure 4 (row-1, column-2) : zoomed image of Einstein

Figure 5 (row-2, column-1) : blurred image of Einstein

Figure 6 (row-2, column-2) : sharpened image of Einstein

Figure 8 (row-3, column-1) : edges of the image of Einstein

b. Medical Field:

Image processing is applied in the medical industry for many different purposes, including cancer cell image processing, PET scanning, X-ray imaging, medical CT scanning, and much more. The diagnostics process has been significantly enhanced by the introduction of image processing to the realm of medical technology.



The original image is the one on the left. The processed image is seen on the right. As you can see, the improved image can be used for more accurate diagnosis.

c. Transmission and encoding:

An underwater cable carried the first image ever to be broadcast over the wire, travelling from London to New York. Below is a picture that was sent.



Three hours were needed for the picture to travel from one location to another.

Imagine for a moment that we can now view live video feeds or live cctv footage with only a few seconds of delay from one continent to another. It implies that significant effort has also been made in this area. This field also emphasises encoding in addition to transmission. To encode images for streaming over the internet or other high or low bandwidth networks, a wide variety of formats have been developed.

d. Remote Sensing:

The area of the earth is scanned by a satellite or from a very high ground in the field of remote sensing, and after that, it is processed to get data about it. One application of digital image processing in remote sensing is to detect infrastructure damage caused by an earthquake.

Even when major damages are the target, it takes longer for damage to be understood. Because the area affected by an earthquake can occasionally be so large, it is impossible to visually inspect it to determine the extent of the damage. Even if it is, the process is incredibly stressful and time-consuming. In light of this, digital image processing offers a remedy. To identify the different kinds of damage caused by the earthquake, a photograph of the affected area is taken from above and examined.

e. Machine/ Robot Vision:

Computer vision is one of the most fascinating and practical uses of image processing. To enable a computer to view, recognise objects, and interpret the entire environment, computer vision is utilised. Self-driving cars, drones, and other devices are major applications for computer vision. Obstacle detection, path recognition, and environmental comprehension are all made easier by CV. One of the main issues facing robots is their vision. Make the robot capable of seeing things, recognising them, recognising obstacles, etc. This field has made significant contributions, and a completely new branch of computer vision has been established to focus on it. Numerous robotic devices are

employed in digital image processing. Robots navigate using image processing techniques, such as line follower robots and obstacle detection.

Robot vision is the technique of enabling robots to process visual data from the outside world by combining camera hardware and computer algorithms. Robots navigate using image processing techniques. Robots are essentially blind without digital image processing. Robots employ vision to perform complex tasks in a constantly changing environment. Digital cameras can transfer high-resolution pixel arrays to the robot's computer thanks to their very advanced technology. Digital image processing algorithms enhance and interpret these images.

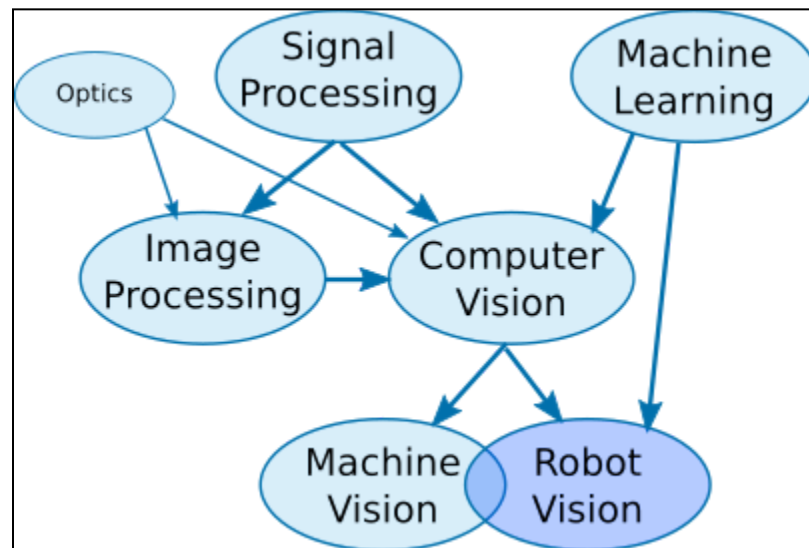


Figure 9 : Computer Vision and Robot Vision Tree

f. Colour Processing:

Every pixel in a digital colour image contains colour information. Three values are calculated for the brightness and chrominance of each pixel in a colour image. The brightness information for each shadowing band is preserved in the digital image data. Colour processing includes processing of colored images and different colour spaces that are used. For instance, the RGB, YCbCr, and HSV colour models. Additionally, it requires how these colour images are transmitted, stored, and encoded.

g. Pattern Recognition:

Pattern recognition involves study from image processing and from various other fields that includes machine learning (a branch of artificial intelligence). Image processing is used in pattern recognition to identify the objects in an image, and machine learning is then used to train the system to recognise changes in patterns. In order to extract important elements from picture or video samples, pattern recognition is useful. Additionally, it is utilised in computer vision for a variety of purposes, including biological and biomedical imaging. when paired with artificial intelligence such that applications like image identification, handwriting recognition, and computer-aided diagnosis are made simple.

h. Video processing:

Video processing is a special instance of signal processing, specifically image processing, where the input and output signals are video files or video streams. A group of frames or images are positioned together to create a fast-moving sequence of images. The quantity of frames or photos per minute and the calibre of each frame employed determine the video's quality. In televisions, VCRs, DVD players, video codecs, video players, video scalers, and other devices, video processing techniques are frequently used. Frame rate conversion, motion detection, aspect ratio conversion, picture stabilisation, detail improvement, noise reduction, and colour space conversion are just a few of the techniques used in video processing.

Chapter - 3:

Optical Character Recognition:

Optical character recognition, also known as OCR or an optical character reader, is the electronic or mechanical conversion of images of typed, handwritten, or printed text into machine-encoded text from scanned documents, photos of documents, scene photos, or text that has been superimposed on an image. It is a common practice to digitise printed texts so they can be electronically edited, searched, stored more compactly, displayed online, and used in machine processes like cognitive computing, machine translation, (extracted) text-to-speech. This method of digitising printed texts is used to enter data from printed paper data records such as passports, invoices, bank statements, computerised receipts, business cards, mail, printouts of static-data, or any suitable documentation.

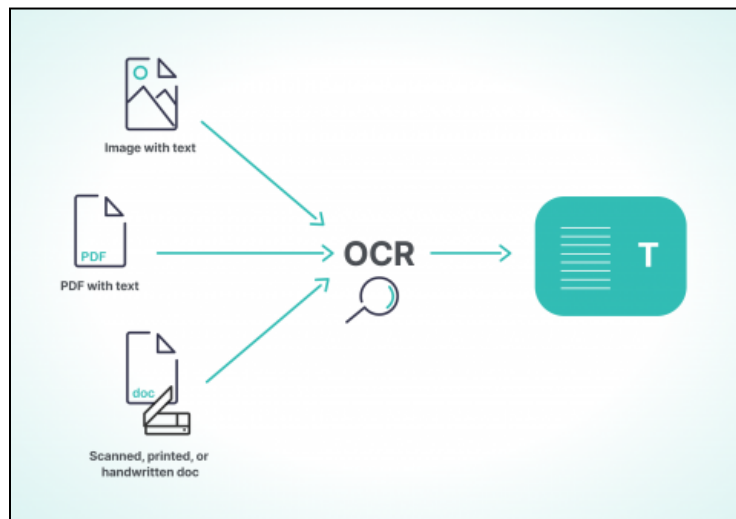


Figure 10 : an example for OCR

Pattern recognition, artificial intelligence, and computer vision are all areas of study in OCR. Early editions worked on one font at a time and required training with photos of each character. Today, sophisticated systems with support for a range of digital image file formats inputs and a high degree of recognition accuracy for the majority of fonts are the norm. Some systems have

the ability to reproduce structured output including elements like columns, graphics, and other non-textual content that closely resembles the original page.

Chapter - 4:

MATLAB:

MATLAB is a programming platform designed specifically for engineers and scientists to analyse and design systems and products that transform our world. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics. Millions of engineers and scientists worldwide use MATLAB for a range of applications, in industry and academia, including deep learning and machine learning, signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. It is an easy language to learn, like python or something. But it also allows computationally intensive code to be optimised via what we called “vectorization”: put your data into vectors and do operations on the vectors. So the inner loop of the code could go very fast as it is automatically optimised by the interpreter.

The following figure, **Figure 11** : Depicts Capabilities of MATLAB (first image), **Figure 12** : Applications of MATLAB (middle image), **Figure 13** : Few more applications of MATLAB (final image).

MATLAB Capabilities



Data Analysis

Explore, model, and analyze data



Graphics

Visualize and explore data



Programming

Create scripts, functions, and classes



App Building

Create desktop and web apps



External Language Interfaces

Use MATLAB with Python, C/C++, Fortran, Java, and other languages



Hardware

Connect MATLAB to hardware



Parallel Computing

Perform large-scale computations and parallelize simulations using multicore desktops, GPUs, clusters, and clouds



Web and Desktop Deployment

Share your MATLAB programs



MATLAB in the Cloud

Run in cloud environments from MathWorks Cloud to public clouds including AWS and Azure

Applications:



Automated Driving Systems

Design, simulate, and test automated driving systems



Computational Biology

Analyze, visualize, and model biological data and systems



Control Systems

Design, test, and implement control systems



Data Science

Explore data; build machine learning models; do predictive analytics



Deep Learning

Data preparation, design, simulation, and deployment for deep neural networks



Embedded Systems

Design, code, and verify embedded systems



Enterprise and IT Systems

Use MATLAB with your IT systems



FPGA, ASIC, and SoC Development

Automate your workflow — from algorithm development to hardware design and verification



Image Processing and Computer Vision

Acquire, process, and analyze images and video for algorithm development and system design



Internet of Things

Connect embedded devices to the Internet and gain insight from your data



Machine Learning

Train models, tune parameters, and deploy to production or the edge



Mechatronics

Design, optimize, and verify mechatronic systems



Mixed-Signal Systems

Analyze, design, and verify analog and mixed-signal systems



Power Electronics Control Design

Design and implement digital control for motors, power converters, and battery systems



Power Systems Analysis and Design

Design and simulate electric grids and transportation systems



Predictive Maintenance

Develop and deploy condition monitoring and predictive maintenance software



Robotics

Design, simulate, and verify robotics and autonomous systems



Signal Processing

Analyze signals and time-series data. Model, design, and simulate signal processing systems



Test and Measurement

Acquire, analyze, and explore data and automate tests



Wireless Communications

Create, design, test, and verify wireless communications systems



Aerospace and Defense

Design, simulate, test, and deploy safety and mission critical systems

- Space Systems
- Radar Systems
- Autonomous Underwater Vehicles



Energy Production

Develop and implement models, analyze big data, and automate processes

- Utilities and Energy
- Chemicals and Petrochemicals
- Oil and Gas



Quantitative Finance and Risk Management

Import data, develop algorithms, debug code, scale up processing power, and more

- Machine Learning in Finance
- Model Risk Management
- Central Banking



Automotive

Design, simulate, and deploy tomorrow's mobility

- Automated Driving
- Virtual Vehicle
- Electric Vehicle



Industrial Automation and Machinery

Develop embedded control and signal processing applications for industrial and energy-related equipment

- Power Generation and Transmission Equipment
- Machine Builders
- Building Automation
- Electric Drives and Automation Components



Railway Systems

Model, simulate, and optimize railway applications



Biotech and Pharmaceutical

Develop algorithms, process data, design devices, and perform modeling and simulation for drug discovery and development



Medical Devices

Design, simulate, and test next-generation medical devices while ensuring regulatory compliance

- Therapeutic Devices
- Medical Imaging
- Patient Monitoring
- Hearing Aids
- Surgical Devices
- In Vitro Diagnostic Devices
- FDA Software Validation



Semiconductors

Design analog, digital, and mixed-signal devices



Software and Internet

Explore and analyze data, develop algorithms, and deploy applications for software and internet systems



Communications

Design and simulate communications systems



Technical Services and Consulting

Analyze and design systems and applications, servicing diverse engineering and scientific programs

- Technical Service Providers
- Government Research and Institutes



Electronics

Develop, simulate, and test electronics systems and devices



Metals, Materials, and Mining

Analyze sensor data, implement control strategies, and create predictive maintenance systems

Use of MATLAB in Digital Image Processing:

As MATLAB is a matrix-based language, we can interpret a digital image as a two dimensional matrix with intensity as values, we use MATLAB extensively for processing Digital Images. It helps to perform varied processes on digital images with and without the help of built in functions.

Chapter - 5:

Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasises readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Advantages of Python is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Applications for Python:

a. Web and Internet Development:

Python offers many choices for web development:

1. Frameworks such as Django and Pyramid.
2. Micro-frameworks such as Flask and Bottle.
3. Advanced content management systems such as Plone and django CMS.

Python's standard library supports many Internet protocols:

1. HTML and XML

2. JSON
3. Email processing
4. Support for FTP, IMAP, and other Internet protocols
5. Easy-to-use socket interface

Package Index has yet more libraries:

1. Requests, a powerful HTTP client library.
2. BeautifulSoup, an HTML parser that can handle all sorts of oddball HTML.
3. Feedparser for parsing RSS/Atom feeds.
4. Paramiko, implementing the SSH2 protocol.
5. Twisted Python, a framework for asynchronous network programming.

b. Scientific and Numeric:

Python is widely used in scientific and numeric computing:

1. SciPy is a collection of packages for mathematics, science, and engineering.
2. Pandas is a data analysis and modelling library.
3. IPython is a powerful interactive shell that features easy editing and recording of a work session, and supports visualisations and parallel computing.
4. The Software Carpentry Course teaches basic skills for scientific computing, running bootcamps and providing open-access teaching materials.

c. Education:

Python is a superb language for teaching programming, both at the introductory level and in more advanced courses.

1. Books such as How to Think Like a Computer Scientist, Python Programming: An Introduction to Computer Science, and Practical Programming.
2. The Education Special Interest Group is a good place to discuss teaching issues.

d. Desktop GUIs:

The Tk GUI library is included with most binary distributions of Python. Some toolkits that are usable on several platforms are available separately:

1. wxWidgets
2. Kivy, for writing multitouch applications
3. Qt via pyqt or pyside

Platform-specific toolkits are also available:

4. GTK+
 5. Microsoft Foundation Classes through the win32 extensions
- e. Software Developments:
- Python is often used as a support language for software developers, for build control and management, testing, and in many other ways.
1. SCons for build control.
 2. Buildbot and Apache Gump for automated continuous compilation and testing.
 3. Roundup or Trac for bug tracking and project management.
- f. Business Applications:
- Python is also used to build ERP and e-commerce systems:
1. Odoo is an all-in-one management software that offers a range of business applications that form a complete suite of enterprise management applications.
 2. Tryton is a three-tier high-level general purpose application platform.

Use of Python in Digital Image Processing:

Python provides lots of libraries for image processing, including –

1. OpenCV – Image processing library mainly focused on real-time computer vision with application in wide-range of areas like 2D and 3D feature toolkits, facial & gesture recognition, Human-computer interaction, Mobile robotics, Object identification and others.
2. Numpy and Scipy libraries – For image manipulation and processing.
3. Scikit – Provides lots of algorithms for image processing.
4. Python Imaging Library (PIL) – To perform basic operations on images like create thumbnails, resize, rotation, convert between different file formats etc.

Chapter - 6:

Libraries:

a. OpenCV:

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD licence and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimised C/C++ to take advantage of multi-core processing. OpenCV can perform image/video I/O, processing, display (core, imgproc, highgui), Object/feature detection (objdetect, features2d, nonfree), Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab), Computational photography (photo, video, superres), Machine learning & clustering (ml, flann), CUDA acceleration (gpu).

a. Applications of OpenCV:

There are lots of applications which are solved using OpenCV, some of them are listed below:

- a. face recognition
- b. Automated inspection and surveillance
- c. number of people – count (foot traffic in a mall, etc)
- d. Vehicle counting on highways along with their speeds
- e. Interactive art installations
- f. Anomaly (defect) detection in the manufacturing process (the odd defective products)

- g. Street view image stitching
- h. Video/image search and retrieval
- i. Robot and driver-less car navigation and control
- j. object recognition
- k. Medical image analysis
- l. Movies – 3D structure from motion
- m. TV Channels advertisement recognition

Image-Processing:

Image processing is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it. Image processing is basically signal processing in which input is an image and output is image or characteristics according to requirements associated with that image. Image processing basically includes the following three steps: Importing the image Analysing and manipulating the image Output in which result can be altered image or report that is based on image analysis

b. Pytesseract:

Pytesseract or Python-tesseract is an OCR tool for python that also serves as a wrapper for the Tesseract-OCR Engine. It can read and recognize text in images and is commonly used in python OCR image to text use cases. It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, and others.

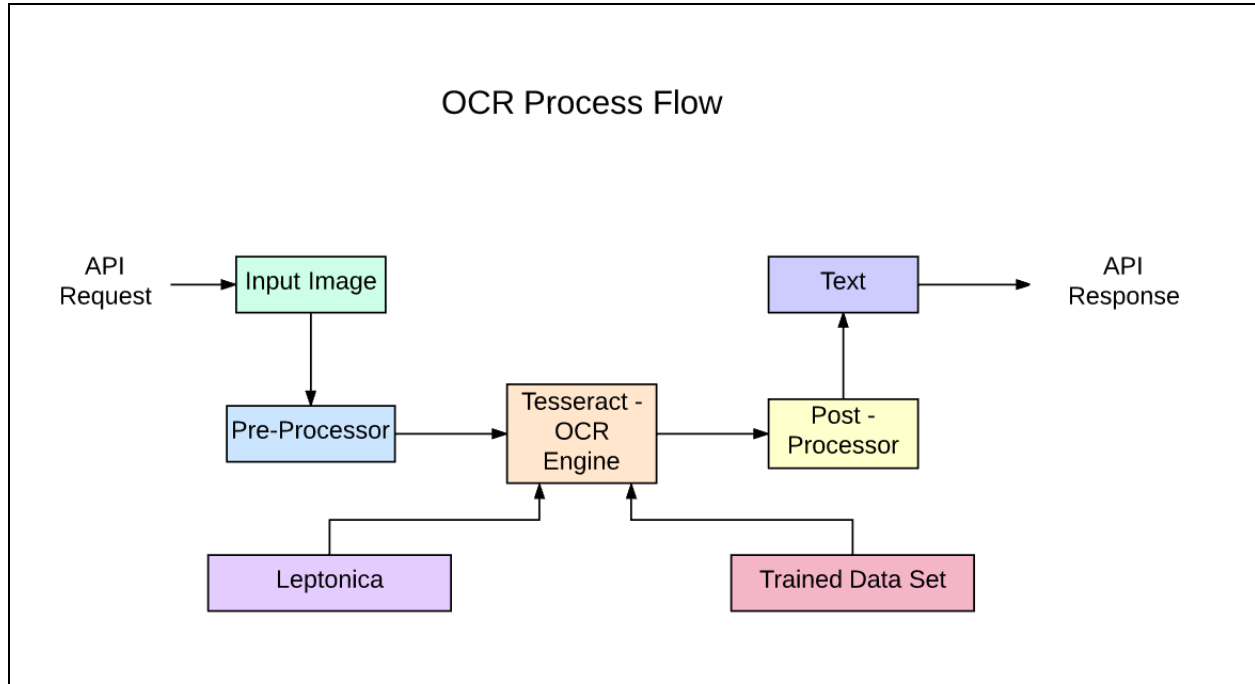


Figure 14 : Process flow chart of Optical Character Recognition

Preprocessing the image:

To avoid all the ways our tesseract output accuracy can drop one need to make sure the image is appropriately pre-processed. It includes rescaling, binarization, noise removal, deskewing etc.

c. NumPy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

- a. A powerful N-dimensional array object
- b. Sophisticated (broadcasting) functions
- c. Tools for integrating C/C++ and Fortran code
- d. Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Chapter - 7:

Basic Idea of the Project:

The topic of the project is “De-identification of Named Entities from Electronic Health Records using One Shot Learning”. In which I have dealt with digital image processing, Optical Character Recognition. I have used python and MATLAB softwares to code. Free open source libraries are available. There are pre-built functions in the libraries. Through which operation can be performed directly. Yet hence I have developed, studied, tested and implemented a few scratch codes and algorithms.

- Algorithm followed is:

Input : low quality digital image

Output : Text

1. Image Processing:

- Grayscale Conversion*
- Gamma Correction*
- Background Text Colour Inversion*
- Binarization of Image*

2. Character Recognition:

Use Two Pass Approach

Initialise $i = 1$

while $i < n$ do

Obtain Phases $A[X_i]$

while $i < m$ do

Get words $B[W_j]$ from phrases

Save coordinates of bounding box $C[a, b]$ of each word

end

end

- a. Software Used:
 - I. MATLAB, MathWorks
 - II. Python, version 3.9.12
- b. Libraries Used:
 - I. opencv-python, version 4.6.0.66
 - II. pytesseract, version 0.3.9
 - III. numpy, version 1.21.6

7.1.i - Grayscale Conversion:

A colour is made up of primary colours Red, Blue, Green. Intensity of a pixel depends on all the three components in case of a colour image. We convert a colour or RGB image into a grayscale image mainly as it simplifies the algorithm and reduces computational requirements. Intensity of a pixel is represented as (R, G, B) or (B, G, R), where R is the intensity of the Red component, G is the intensity of the Green Component, B is the intensity of the Blue Component.

Algorithm:

$$\text{Greyscale} = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

Now intensity at a pixel is (G), where G is between 0 and $2^n - 1$, for a n-bit image.

- a. Python Code:

The OpenCV image is read in BGR format. In python we have a built-in function.
`cvtColor(image, cv2.COLOR_BGR2GRAY).`

And also we can use the formula to extract the grayscale image.

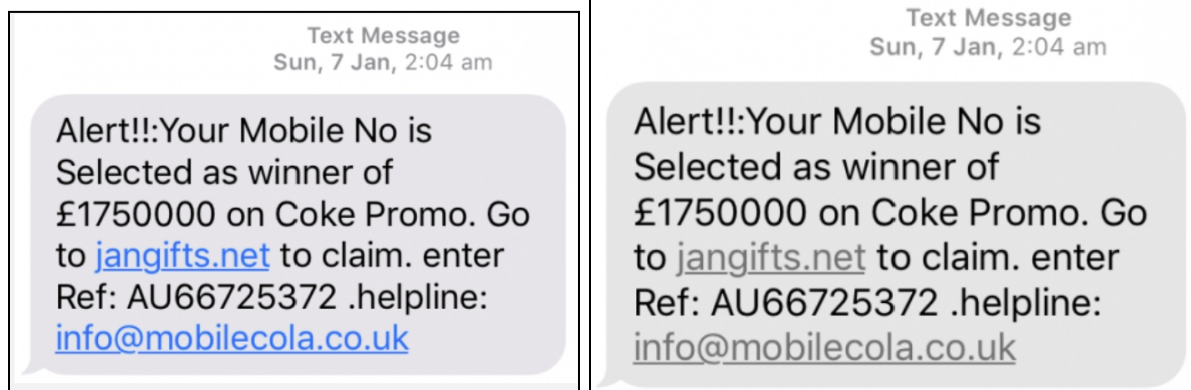


Figure 15 : The left image is the RGB image and the right side image is the Grayscale image.

b. MATLAB Code:

In MATLAB also there is a direct in-built function for conversion of a colour image into grayscale image.

% Entering the input

```
prompt = "Enter Image name: ";
```

```
txt = input(prompt, "s");
```

```
i=imread(txt);
```

```
imshow(i);
```

```
img = im2gray(i)
```

```
imshow(img)
```

7.1.ii - Gamma Correction:

Gamma Correction or Power - Law Transformations is given by $s = c \cdot r^\gamma$, where c and γ are positive constants. The below graph shows plots of s as a function of r for various values of γ . Gamma correction can be used to control the overall brightness of an image. It can be used with images that are found to be either bleached out or too dark. As with log transformations, power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels. Note also in the below figure that a family of transformations can be obtained simply by varying γ . Curves generated with values of $\gamma > 1$ have exactly the opposite effect as those generated with values of $\gamma < 1$. When $c = \gamma = 1$ reduces to the identity transformation.

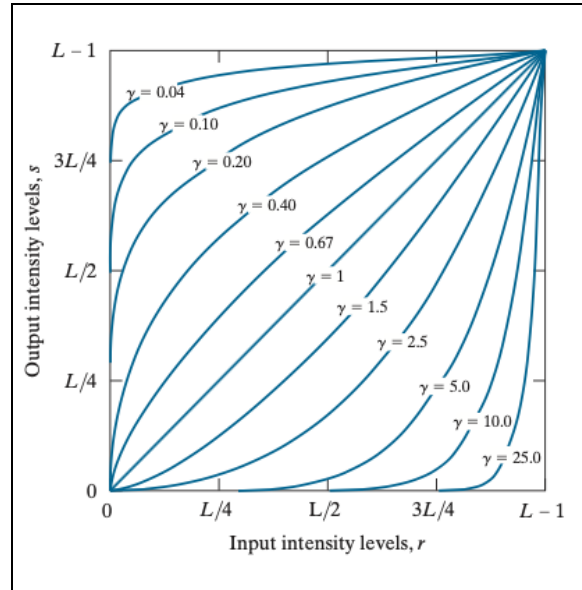


Figure 16: The above figure depicts the plot of the gamma equation.

a. Python Code:

```

1  import cv2
2  import numpy as np
3
4  # open the image
5  img = cv2.imread('0.jpg')
6
7  # convert the image from RGB to Grayscale
8  img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9
10 # Trying 4 gamma values.
11 for gamma in [0.1, 0.5, 1, 1.2, 2]:
12     # Apply gamma correction.
13     gamma_corrected = np.array(255 * (img_gray / 255) ** gamma, dtype='uint8')
14
15     # Save edited images.
16     cv2.imwrite('gamma_transformed' + str(gamma) + '.jpg', gamma_corrected)
17

```

Figure 17 : The above image is the python code used for the gamma correction. I used {0.1, 0.5, 1, 1.2, 2} as gamma values.

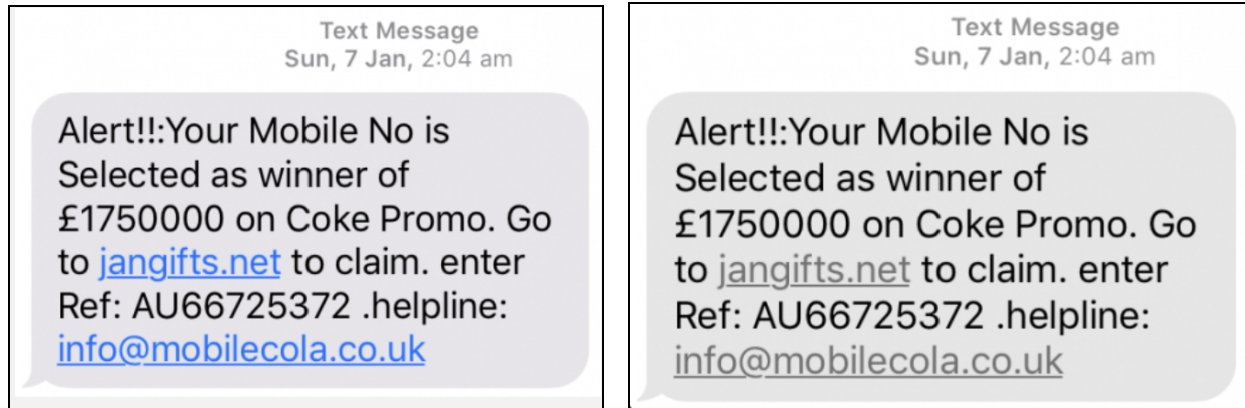
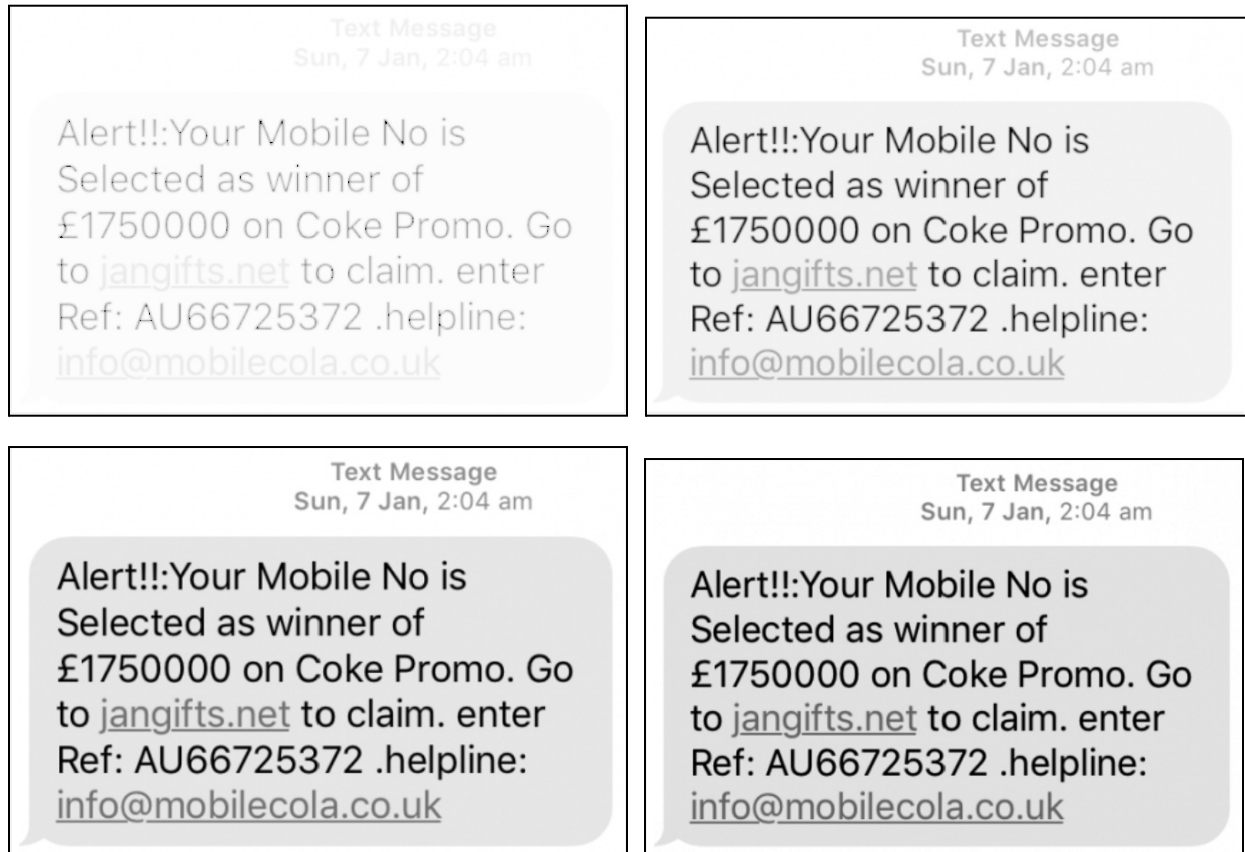


Figure 18 : The left image is the RGB image and the right side image is the Grayscale image.



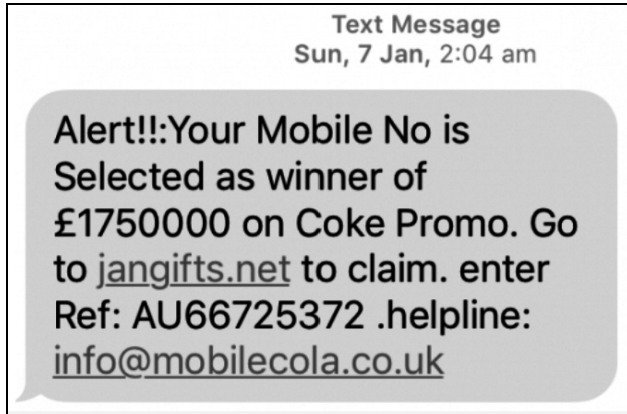


Figure 19 (a), (b), (c), (d) : indicates the images after applying gamma transformation with gamma equals to {0.1, 0.5, 1, 1.2, 2} respectively.

a. MATLAB Code:

In matlab we can use built-in functions. As we used algorithms in python code we will see the built in function in MATLAB.

% Entering the input :

```
prompt = "Enter Image name: ";
```

```
txt = input(prompt, "s");
```

```
img=imread(txt);
```

```
imshow(i);
```

% Initiate Gamma

```
g = 0.1;
```

% Enhancement :

```
img_corrected = imadjust(img,[],[], g)
```

```
figure, imshow(img_corrected)
```

7.1.iii - Background Colour Inversion:

Background colour inversion can be done easily. The technique is to convert all the whites into black and vice versa linearly. That is to subtract the intensity of each pixel from the maximum allowed intensity. For a 8 - bit image it is $255 - f(x,y)$. Sometimes it is also known as complement as it subtract the intensity at that point from the maximum possible value.

a. Python Code:

```
import cv2
import numpy as np

# open the image
img = cv2.imread('0.jpg')
h, w = img.shape[:2]
# convert the image from RGB to Grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# for a gamma values.
gamma = 1.5
# Apply gamma correction.
gamma_corrected = np.array(255 * (img_gray / 255) ** gamma, dtype='uint8')
# Save edited images.
cv2.imshow('gamma_transformed' + str(gamma) + '.jpg', gamma_corrected)
cv2.waitKey(0)

comp_img = 255 - img_gray
cv2.imshow('complementary image', comp_img)
cv2.waitKey(0)
```

Figure 20 : It depicts the python code for inversion of grayscale image.

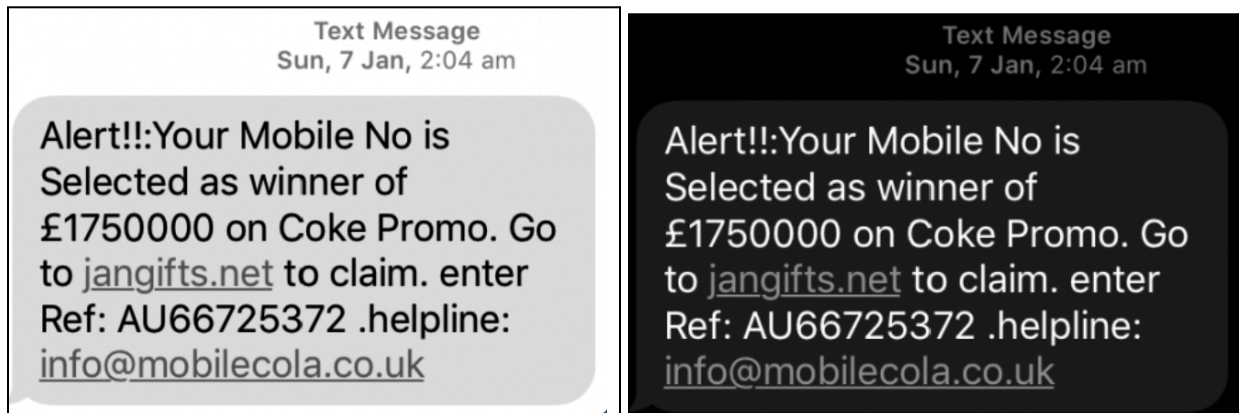


Figure 21 : Left side image shows the gamma corrected image of the grayscale image with gamma equals to 1.5 and the right side image depicts the complement of the gamma corrected image.

b. MATLAB Code:

In built function:

% 1. Entering the input

prompt = "Enter Image name: ";

txt = input(prompt, "s");

i=imread(txt);

imshow(i);

img = im2gray(i)

imshow(img)

% 2. Enhancement

img_gamma = imadjust(img,[],[], 1.5)

figure, imshow(img_gamma)

% 5. Colour Inversion

i1 = imcomplement(img_gamma);

figure, imshow(i1);

7.1.iv - Binarization of Image:

Converting a grayscale image to monochrome is a common image processing task. Otsu's method, named after its inventor Nobuyuki Otsu, is one of many binarization algorithms. In the simplest form, the algorithm returns a single intensity threshold that separates pixels into two classes, foreground and background. This threshold is determined by minimising intra-class intensity variance, or equivalently, by maximising inter-class variance.

Algorithm:

- 1. Compute histogram and probabilities of each intensity level*
- 2. Set up initial $\omega_i(0)$ and $\mu_i(0)$*
- 3. Step through all possible thresholds $t = 1, \dots$ maximum intensity*
 - 1. Update ω_i and μ_i*
 - 2. Compute $\sigma_b^2(t)$*

4. *Desired threshold corresponds to the maximum $\sigma_b^2(t)$*

Otsu Thresholding Explained:

Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels on each side of the threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum. Or it can be said as it exhaustively searches for the threshold that minimises the intra-class variance, defined as a weighted sum of variances of the two classes.

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t , and σ_0^2 and σ_1^2 are variances of these two classes.

The class probability $\omega_{0,1}(t)$ is computed from the L bins of the histogram:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

For 2 classes, minimizing the intra-class variance is equivalent to maximizing inter-class variance.^[2]

$$\begin{aligned} \sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) = \omega_0(t)(\mu_0 - \mu_T)^2 + \omega_1(t)(\mu_1 - \mu_T)^2 \\ &= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \end{aligned}$$

which is expressed in terms of class probabilities ω and class means μ , where the class means $\mu_0(t)$, $\mu_1(t)$ and μ_T are:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

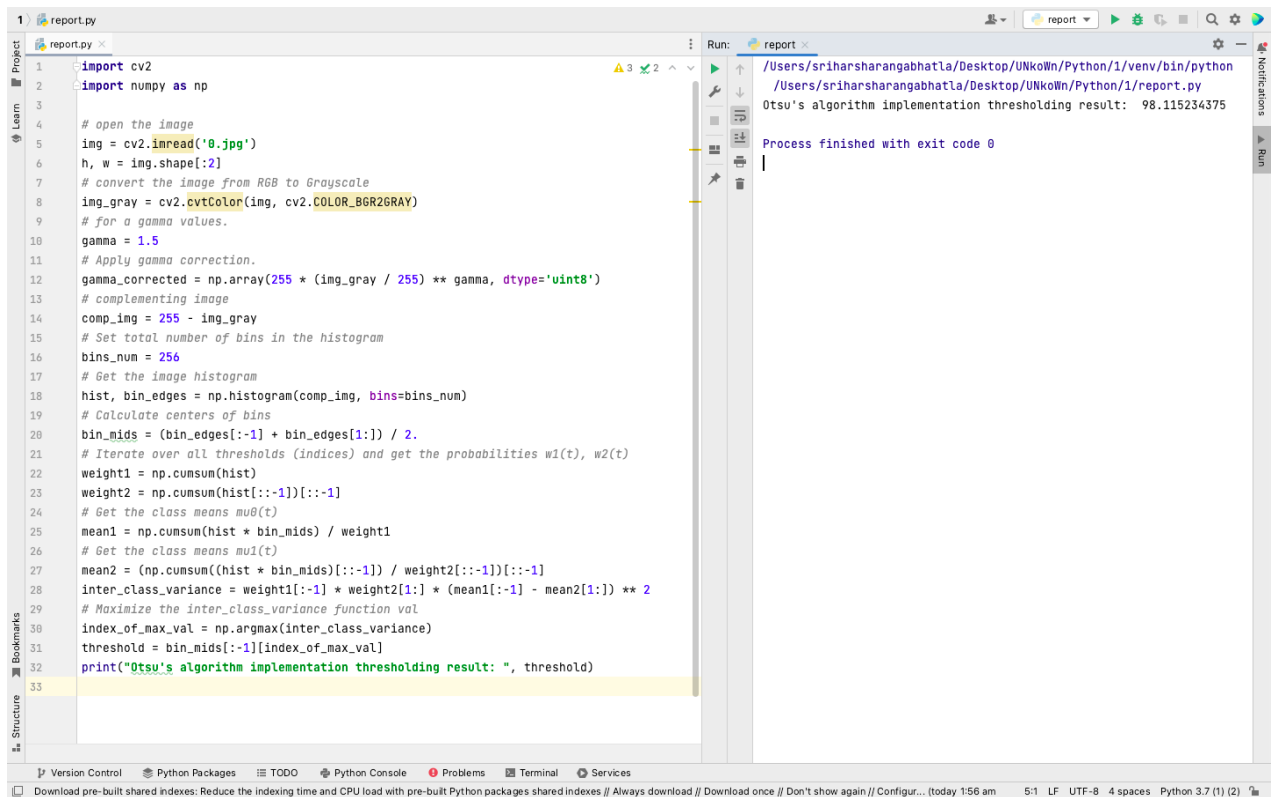
$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

The following relations can be easily verified:

$$\begin{aligned} \omega_0\mu_0 + \omega_1\mu_1 &= \mu_T \\ \omega_0 + \omega_1 &= 1 \end{aligned}$$

The class probabilities and class means can be computed iteratively. This idea yields an effective algorithm.

a. Python Code:



The screenshot shows a Python IDE with a file named 'report.py'. The code implements Otsu's algorithm for image binarization. It starts by importing 'cv2' and 'numpy as np'. The image '0.jpg' is read and its dimensions are determined. It is then converted to grayscale. A gamma correction is applied with a gamma value of 1.5. The image is complemented, and a histogram is calculated with 256 bins. The algorithm then iterates over all possible thresholds to find the one that maximizes the inter-class variance. The resulting threshold is printed as 'Otsu's algorithm implementation thresholding result: 98.115234375'. The IDE interface includes a sidebar with 'Project', 'Learn', 'Bookmarks', and 'Structure' views, and a bottom status bar showing '5:1 LF UTF-8 4 spaces Python 3.7 (1) (2)'.

```
1 import cv2
2 import numpy as np
3
4 # open the image
5 img = cv2.imread('0.jpg')
6 h, w = img.shape[:2]
7 # convert the image from RGB to Grayscale
8 img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9 # for a gamma values.
10 gamma = 1.5
11 # Apply gamma correction.
12 gamma_corrected = np.array(255 * (img_gray / 255) ** gamma, dtype='uint8')
13 # complementing image
14 comp_img = 255 - img_gray
15 # Set total number of bins in the histogram
16 bins_num = 256
17 # Get the image histogram
18 hist, bin_edges = np.histogram(comp_img, bins=bins_num)
19 # Calculate centers of bins
20 bin_mids = (bin_edges[:-1] + bin_edges[1:]) / 2.
21 # Iterate over all thresholds (indices) and get the probabilities w1(t), w2(t)
22 weight1 = np.cumsum(hist)
23 weight2 = np.cumsum(hist[::-1])[::-1]
24 # Get the class means mu0(t)
25 mean1 = np.cumsum(hist * bin_mids) / weight1
26 # Get the class means mu1(t)
27 mean2 = (np.cumsum((hist * bin_mids)[::-1]) / weight2[::-1])[::-1]
28 inter_class_variance = weight1[:-1] * weight2[1:] * (mean1[:-1] - mean2[1:]) ** 2
29 # Maximize the inter_class_variance function val
30 index_of_max_val = np.argmax(inter_class_variance)
31 threshold = bin_mids[:-1][index_of_max_val]
32 print("Otsu's algorithm implementation thresholding result: ", threshold)
33
```

Figure 22 : The above picture depicts the python code for Otsu Binarization.

for i in range (0, h):

for j in range (0, w):

if comp_img[i,j] < 98.115234375 :

comp_img[i, j] = 0

else: comp_img[i, j] = 255

cv2.imshow('Thresholded Image', comp_img)

cv2.waitKey(0)

Adding above code in the end displays a thresholded image. The resulting image may or may not be completely binarized due to some noise in the image. Noise can be cleared using appropriate filters.



Figure 23 : The above is the final binarized image required.

b. MATLAB Code:

In MATLAB a built in function is available. “*otsuthresh*” which takes image and number of bins as a input and omits the thresholding value.

```
% 1. Entering the input
prompt = "Enter Image name: ";
txt = input(prompt, "s");
i=imread(txt);
imshow(i);
img = im2gray(i)
imshow(img)
```

```

% 2. Enhancement
img_gamma = imadjust(img,[],[], 1.5)
figure, imshow(img_gamma)

% 5. Colour Inversion
i1 = imcomplement(img_gamma);
figure, imshow(i1);

%3 Otsu Binarization
[counts, x] = imhist( i1, 256);
stem(x,counts);
T = otsuthresh(counts);
BW = imbinarize(i1, T);
figure
imshow(BW)

```

Chapter - 8:

Further Developments:

As we understand that for an Optical Recognition Software we need to give an input of an image pre-processed, such that it does not affect the accuracy of the output text. Pre- processing includes rescaling, binarization, noise removal, deskewing etc.

We need to detect particular text from the images. With the use of various deep learning techniques we can develop a self learning network to identify that certain text. Recurrent Neural Network is one of the most promising networks as per the internal memory is concerned. Hence it is perfectly suited for machine learning problems that involve sequential data. We then train the neural network with the dataset.

Chapter - 9 :

References:

1. Smith, Ray. "An overview of the Tesseract OCR engine." *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. IEEE, 2007.
2. Abak, Ali T., U. Baris, and Bülent Sankur. "The performance evaluation of thresholding algorithms for optical character recognition." *Proceedings of the fourth international conference on document analysis and recognition*. Vol. 2. IEEE, 1997.
3. Mikolov, Tomáš, et al. "Recurrent neural network based language model." *Eleventh annual conference of the international speech communication association*. 2010.
4. Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modelling." *arXiv preprint arXiv:1412.3555* (2014).
5. Sharp, John V., and Donald R. Thompson. "Method and apparatus for increasing image resolution." U.S. Patent No. 3,573,789. 6 Apr. 1971.
6. Farid, Hany. "Blind inverse gamma correction." *IEEE Transactions on Image Processing* 10.10 (2001): 1428-1433.
7. Weeks, Arthur Robert, Carlos E. Felix, and Harley R. Myler. "Edge detection of color images using the HSL colour space." *Nonlinear Image Processing VI*. Vol. 2424. International Society for Optics and Photonics, 1995.
8. Trier, Eivind Due, and Torfinn Taxt. "Evaluation of binarization methods for document images." *IEEE transactions on pattern analysis and machine intelligence* 17.3 (1995): 312-315.
9. Kurita, Takio, Nobuyuki Otsu, and N. Abdelmalek. "Maximum likelihood thresholding based on population mixture models." *Pattern recognition* 25.10 (1992): 1231-1240.
10. Coates, Adam, et al. "Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning." *ICDAR*. Vol. 11. 2011.
11. Basu, Subhadip, et al. "A two-pass approach to pattern classification." *International Conference on Neural Information Processing*. Springer, Berlin, Heidelberg, 2004.