PROJECT REPORT ON

SIMPLIFYING THE INTRICATE CALCULATIONS THAT RESULT FROM DEEP LEARNING IN ORDER TO CREATE CARTOOON FROM IMAGES

Submitted in partial fulfilment of the Requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Project Associates	**Regd. Nos**
Bandaru Venkata Muralidhara	20091A04N6
Vullebeeti Bhavana	20091A0417
Yanadi Chetan	20091A0424
Shaik Irfan Basha	21095A0408

Under the Esteemed Guidance of
Mr. D. Yovan Snanagan Ponselvan, M.E., (Ph. D)
Assistant Professor

(ESTD-1995)

(ESTD-1995)

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

RAJEEV GANDHI MEMORIAL COLLEGE
OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
Affiliated to JNTUA - Anantapuramu, Approved by AICTE - New Delhi,
Accredited by NBA - New Delhi, Accredited by NAAC with A+ Grade – New Delhi
NANDYAL – 518501 Dist. A.P.
YEAR: 2020 - 2024

RAJEEV GANDHI MEMORIAL COLLEGE OF ENGINEERING & TECHNOLOGY

AUTONOMOUS

(Approved by A.I.C.T.E - New Delhi, Affiliated to JNTUA - Anantapuramu, Accredited by NBA - New Delhi, Accredited by NAAC with 'A+' Grade - New Delhi)

NANDYAL – 518 501, A.P, India

CERTIFICATE

This is to certify that the dissertation entitled "SIMPLIFYING THE INTRICATE CALCULATIONS THAT RESULT FROM DEEP LEARNING IN ORDER TO CREATE CARTOON FROM IMAGES" is being submitted by Bandaru Venkata Muralidhara (20091A04N6), Vullebeeti Bhavana (20091A0417), Yanadi Chetan (20091A0424), Shaik Irfan Basha (21095A0408) under the guidance of Mr. D. Yovan Snanagan Ponselvan , Assistant Professor for Project of the award of B.Tech Degree in Electronics and Communication Engineering, Rajeev Gandhi Memorial College of Engineering and Technology, Nandyal (Autonomous) (Affiliated to JNTUA Anantapuramu) is a record of bonafide work carried out by them under our guidance and supervision.

Dr. Kethepalli Mallikarjuna	Mr. D. Yovan Snanagan Ponselvan
Head of the Department	**Project Guide**

Signature of the External Examiner

Date of Viva-Voce:

CANDIDATES' DECLARATION

We hereby declare that the work done in this project titled **"Simplifying the Intricate Calculations that Result from Deep Learning in order to create Cartoon from Images"** submitted for the completion of the main project in the IV Year II Semester of B. Tech (ECE) at **Rajeev Gandhi Memorial College of Engineering and Technology (Autonomous), Nandyal**, is an authentic record of our original work done under the guidance of **Mr. D. Yovan Snanagan Ponselvan, Assistant Professor**, Dept. of ECE, RGMCET, Nandyal. We have not submitted the material embodied in this main Project for the award of any other degree in any other institution.

Project Associates	**Regd. Nos**	**Signatures**
Bandaru Venkata Muralidhara	20091A04N6	
Vullebeeti Bhavana	20091A0417	
Yanadi Chetan	20091A0424	
Shaik Irfan Basha	21095A0408	

ACKNOWLEDGEMENT

We earnestly take the responsibility to acknowledge the following distinguished personalities who graciously allowed us to carry our project work successfully.

We express deep gratitude to our guide **Mr. D. Yovan Snanagan Ponselvan, Assistant Professor** in ECE Department, RGMCET, Nandyal for his guidance, incessant help and encouragement throughout the course of the project work, which contributed, to the successful completion of this project.

We would like to thank **Dr. Kethepalli Mallikarjuna**, M.Tech., Ph.D, Professor & Head, Department of ECE, RGMCET, Nandyal, for this valuable suggestions and encouragement whenever we encountered difficulties during the project work.

We would like to express my deep sense of gratitude and indebtedness to **Dr. T. Jayachandra Prasad**, Principal, RGMCET, Nandyal for his sustained encouragement and moral support throughout the course.

We are thankful to our honorable Chairman **Dr. M. Santhiramudu**, RGM Group of Institutions and **Sri. M. Sivaram**, Managing Director, RGMCET, Nandyal for providing us with required facilities in the college.

We express our thanks to all the faculty and staff members of ECE Department, RGMCET, Nandyal, for their support in various aspects in completing the project.

INDEX PAGE NO

List of Figures

List of Tables

ABSTRACT

The main focus of this project is to use image processing techniques to transform real-life images into cartoon-like images. These cartoon images are characterized by well-defined edges, a reduced number of colors compared to the original image, and smooth areas of colour. Recently, Deep Learning methods have been developed to generate cartoon images from regular images. However, these methods often require extensive computations, large datasets, and considerable time, unlike traditional image processing methods which directly manipulate the input images. In order to achieve the desired cartoon effect, several steps are involved in the process. Firstly, the edges of the image are extracted and expanded to enhance their visibility in the final-colored cartoon image. Additionally, a specific formulation is used to assign colors based on separate colour channels, which is known as colour quantization. Various techniques such as noise removal, edge detection, dilation, averaging filters, and others are applied to eliminate unnecessary elements from the image during the conversion process. The implementation of this project involves utilizing concepts like median filter, edge detection, dilation, average filter, quantization, and image channels in MATLAB.

The objective is accomplished through the introduction of a colour quantization technique that not only balances the histogram, but also improves the image's dynamic range. We aim to demonstrate that this algorithm has the ability to generate visually superior cartoon images of high quality from real-life images. The resulting images showcase improved contrast and colour quantization when compared to existing image processing methods. Moreover, this approach outperforms deep learning methods in terms of computational requirements and processing time, eliminating the necessity for model training.

CHAPTER I

1 INTRODUCTION

1.1 Background

As the digital world is achieving great heights, people are looking for different ways to represent themselves. One of the best ways is creating their cartoon images. People are using wonderful platforms like Avatoon, Photo Lab and many more. Even one of the popular platforms, Snapchat, is providing the cartoon images. Some big and popular applications like Photoshop, Adobe Illustrator, and many others help in achieving the goal of image conversion to cartoon. Working on thousands of images, it will be very difficult to implement the process individually on these platforms. Thus, it is required that we adopt an efficient algorithm that can skip the human involvement and within milliseconds can produce the cartoon image.

The task of converting an image into a cartoon can be achieved with Image Processing as well as Machine Learning. There are many existing algorithms reported in literature for this. The image processing-based techniques use filtering and morphological operations for cartoon image generation. These Machine Learning algorithms are time consuming and require a huge dataset for training purposes. Sometimes, the model is not always the desired one and may lead to underfitting or overfitting. Therefore, with fundamental concepts of Image Processing, the time consumption, complex computations and model creation can be avoided. we have used 5 different types of original images like human face, flowers, buildings, animals, using methods like Robert, Canny, Sobel, Prewitt.

1.2 Problem statement

Normally, Original images require more data to store and transferring. As the quality of images improves (HD), the requirement of data also increases. while sharing original images the data loss can be easily visible to our naked eyes (like blur, pixel loss, edges loss, quality, size) compared to cartoon images. Cartoon images simplifies complex ideas and making information more accessible and engaging. They easily capture attention making content more memorable. Cartoon allows for creative expressions enabling artists to convey messages in unique and imaginative ways. Here We employ Digital image processing for image conversion, eliminating the need to train any model compare to deep learning.

1.3 Objectives

The objective of the paper is to develop an algorithm to convert the image into a cartoon image. The said problem statement is addressed by applying the concept of median filter, edge detection, dilation, average filter, quantization, and image channel in MATLAB. The input image is a 3D matrix that contains rows, columns, and three channels (RGB). All the mentioned procedures are individually applied on these channels and then combined to give the desired output. There may be a chance of the addition of additive noise. Such noise can be overcome by applying median filters and average filters. For detecting the edges, a Canny edge detector is used, which is further dilated. This operation is applied on the image by converting it to grayscale image. For reducing the colour quantity, every pixel value of the image is quantized. After performing both procedures, the detected edge is combined with the output of colour saturation and quantization process. The final image gives the cartoon version of the input image.

1.4 Motivation

In this modern era, we have more advanced technology to do image processing. The main motivation of this project is to bring back the old methods for image processing and show that both the old and new advanced methods work with slightest difference in their output. Hope we make our motivation successful.

1.5 Organization of the project

The remaining sections of this study are organized as follows:

- Chapter 2: Literature Review – provides about Literature Survey in which the existing methods for conversion of original image to cartoon image.
- Chapter 3: Methodology - presents a detailed description of the process, algorithm of this Image to Cartoon conversion using MATLAB.
- Chapter 4: This chapter describes the MATLAB code editor and code of this Image to Cartoon Conversion.
- Chapter 5: This chapter describes Experimental Results of the Image Conversion Process.
- Chapter 6: This chapter, describes the conclusion and the future scope of this Entire Project.

CHAPTER II

2 Literature Survey

- Cartoon Photo Effect Application; Kevin Dade
It Create a user-friendly Android app for applying a cartoon effect with artistically appealing results on various images. The Tonify algorithm provides a user-friendly way to apply cartoon effects on Android devices, leveraging the computational power of OpenCV4Android for efficient image processing. The algorithm exhibits unpredictability in its performance across different types of input images, particularly struggling with portrait images and high-detail scenes. This lack of consistency suggests limitations in its applicability to diverse image types. Tonify is considered a success, capable of producing the cartoon effect. Acknowledges limitations and suggests future improvements, such as an adaptive algorithm based on image types.
- Image Cartoonization Methods Using LBG, KPE, KMCG Quantization; Dr. Archana B. Patankar, Ms. Purnima A. Kubde, Ms. Ankita Karia. This paper discusses methods for generating cartoonized painterly effects on images using vector quantization algorithms (LBG, KPE, KMCG), comparing results based on time and effect quality. It comprehensively explores various cartoonization methods using vector quantization, providing valuable insights for image stylization applications. But absence of a detailed discussion on the limitations or potential challenges associated with the proposed methods limits a full understanding of their applicability.
- Edge Detection of human face; Jilin Dong, Jinhuizi He, Haoxuan Wang. This Applied Prewitt, Sobel, Roberts, Canny operators. Visual comparison of edge detection results. The paper provides a comprehensive comparison of edge detection methods (Prewitt, Sobel, Roberts, and Canny) for human face recognition, highlighting the superiority of the Canny edge detector in achieving clearer results. It lacks detailed exploration of the limitations or potential challenges

associated with the discussed edge detection methods, limiting a thorough understanding of their applicability in diverse scenarios. Canny detector most effective for face recognition which is Suggested for future work.

- Cartoonify an Image using OpenCV in Python; Saurabh Kumar, Utkarsh Bhardwaj, T. Poongodi. This paper mainly focuses on converting real-world images into cartoon prints using Cartoon GAN (Generative Adversarial Network) with glad and inimical loss functions. Utilizes OpenCV in Python for implementation. This methodology effectively utilizes Generative Adversarial Networks (GANs), specifically Cartoon GAN, to produce high-quality cartoon-style images from real-world photographs. But the paper lacks explicit discussion on potential limitations or challenges faced during the implementation, leaving room for a more comprehensive understanding of the method's effectiveness and potential drawbacks. Finally, it Summarizes successful implementation, discusses future prospects for enhancing image resolution, and acknowledges the role of OpenCV in achieving cartoonification.
- Image cartoonization algorithm without deep learning; Shanshan Wang, Jiangyuan Qi, Wuhan, Changjiang. This paper proposes a traditional image processing-based algorithm for cartoonization. Uses K-means clustering for color reduction and contours for edge extraction. It provides a focused and informative analysis of the need for a more comprehensive exploration of limitations and challenges in existing edge detection methods, emphasizing the importance of understanding their applicability in diverse scenarios. This paper's lack of detailed exploration may be subjective, as it depends on the specific goals and scope defined by the authors, potentially overlooking the depth of the analysis within the intended context. It concludes with algorithm's satisfactory results, outperforming existing methods. Emphasizes simplicity, efficiency, and avoidance of deep learning complexities.

CHAPTER III

3 METHODOLOGY

The proposed methodology utilizes image processing techniques including noise removal with Median filter, edge detection with Canny edge detector, dilation for boldening edges, averaging filter for color reduction, and color quantization using a proposed formulation. By combining these steps, the algorithm efficiently generates high-quality cartoon images with enhanced contrast and reduced computational complexity compared to deep learning-based methods, making it suitable for real-time applications. Experimental results demonstrate the effectiveness of the approach, showcasing visually appealing cartoon images while preserving contextual information at a lower-computational costs.

3.1 About Image

The most important and valuable resource to a human is their vision and the moment of capture i.e. an image. Some moments are needed to be captured into cameras, portraits etc.

The Image is spread in two directions i.e. Horizontal-direction, Vertical Direction. The product of both the directions of an image gives the number of **pixels**. These pixels are the basis for image and we perform several operations on that and one of them is Image to Cartoon.

This Image to Cartoon Conversion is the process of Conversion of Pixels into similar type such as making a group of data into a similar one. This conversion is further said in detail. Images are of many types such as JPEG (Joint Photographic Expert Group), PNG (Portable Network Graphics), SVG (Scalable Vector Graphics), WebP (Web Picture Format) and so on...

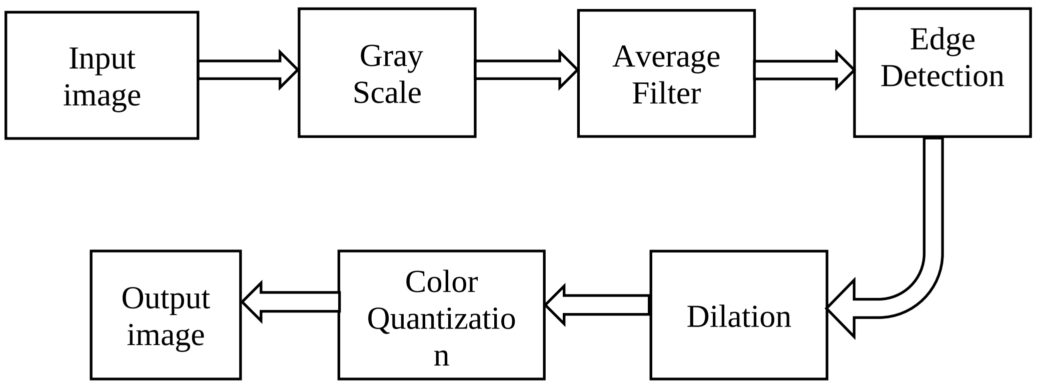
1. JPEG - JPEG (pronounced JAY-peg) is a graphic image file compressed with lossy compression using the standard developed by the ISO/IEC Joint Photographic Experts Group.

2. PNG - The PNG file format is widely used on websites to display high-quality digital images. Created to exceed the performance of GIF files, PNGs offer not just lossless compression, but also a much broader and brighter color palette.
3. SVG - The SVG file format is a popular tool for displaying two-dimensional graphics, charts, and illustrations on websites. Plus, as a vector file, it can be scaled up or down without losing any of its resolution.
4. WebP - WebP is a modern image format that provides superior lossless and lossy compression for images on the web. Using WebP, webmasters and web developers can create smaller, richer images that make the web faster. WebP lossless images are 26% smaller in size compared to PNGs.

3.2 Methodology Flow chart

The following figure shows the methodology flow chart, it describes the way of converting original images to cartoon images using image processing technique.

Block Diagram:



1. Noise Removal: Begin by applying a Median filter to eliminate any salt and pepper noise present in the image, preserving edges. So, we use the Median filter to remove any salt and pepper noise present in the image without blurring the edges.

2. Edge Detection: It’s used for detecting the edges for a given image which is essential for cartoon images. Here it converts the noise-filtered image to grayscale and use a Canny edge detector to detect precise edges. So, in our project this step helps in identifying the outlines and boundaries of objects in the image.
3. Dilation: Dilation helps to make the edges bold and thick which helps to give the image cartoon effect. So, we apply morphological operations, like dilation, to enhance and thicken the detected edges which helps to resembling cartoon-style edges.
4. Averaging Filter: Apply an averaging filter to smooth the image. By using average filter helps in reducing color variations and preparing for color quantization. So, an averaging filter is applied to the original image in our project to blur out its contents, reducing color quantities and simplifying shapes.
5. Color Quantization: Implement color quantization by dividing each pixel value by a constant (a-b), where a and b are predefined values. The resulting value is rounded down and multiplied back by (a-b). Reducing the number of distinct colors in the image.

As we use color quantization in project, helps to reduce the color palette of the image, by mapping groups of pixel values to specific pixel values. This process helps in achieving the limited color palette characteristic of cartoon images.

Threshold value:

In the proposed methodology for image to cartoon generation, a threshold value is utilized during the edge detection process.

This threshold value determines the sensitivity of the edge detection algorithm, influencing which edges are considered significant enough to be detected.

Through experimentation, a threshold value of 0.15,0.2,0.25 was found to provide satisfactory results, effectively delineating edges without overwhelming the output with noise.

Chapter IV

4 Software Description

4.1 MATLAB

MATLAB, renowned for its processing in technical computing, offers a wide variety for the integration of computation, Processing software's, and programming within a user-friendly interface. Evolving from a need to continuous access to matrix software developed by the LINPACK and EISPACK projects, MATLAB's very name, an acronym for "Matrix Laboratory," Develops its foundation in advanced matrix operations. What sets MATLAB different from other software's is its adaptiveness in handling matrices without the need for pre-dimensioning i.e no need to determine the matrix dimensions in before-hand, making it a biggest advantage to offer the solution for technical computing challenges, especially those present in matrices and vectors, with remarkable efficiency. A hallmark of MATLAB is its array of application-specific solutions, aptly named toolboxes. The toolboxes in MATLAB are very useful to MATLAB users, empowering them to delve into and apply higher technologies with ease. Each toolbox in MATLAB comprises an extensive list of libraries of MATLAB functions arranges to address specific classes of problems given to the MATLAB. From signal processing and control systems to neural networks, fuzzy logic, wavelets, simulation, and beyond, MATLAB's toolboxes provide a diverse array of disciplines, providing users with extensive resources for problem-solving and creation of new Products.

4.1.1 Typical uses of MATLAB:

The typical using areas of MATLAB are

Mathematics and computation.

Algorithm and development

Data acquisition

Data analysis, exploration and visualization

Scientific and engineering graphics

Modelling

Simulation

Prototyping

Application development and including graphical user interface building.

MATLAB serves as an interactive workbook with many tools around matrices as its fundamental data element, eliminating the need for dimension declaration. This characteristic checks the resolution of numerous technical computational challenges, particularly those revolve around in matrix and vector formulations, significantly out numbering the time required to code in scalar non-interactive languages like C or FORTRAN. A well-deserved feature of MATLAB's versatility lies in its assortment of supplementary, application-specific solutions known as toolboxes. These toolboxes play a vital role for most MATLAB users in its eco-friendly environment, facilitating the acquisition and application of specialized technologies. Each toolbox comprises an extensive compilation of MATLAB functions carefully created to augment the MATLAB environment, catering to distinct problem domains.

4.1.2 Features of MATLAB:

Cutting-edge algorithms designed for superior numerical computation, particularly in the realm of matrix algebra, enable MATLAB to excel in high-performance computing tasks.

MATLAB boasts an extensive repository of predefined mathematical functions alongside the capability for users to define their own functions, facilitating versatile problem-solving capabilities.

With robust support for both two and three-dimensional graphics, MATLAB empowers users to visualize and present data effectively through plotting and display functionalities.

Leveraging a potent, matrix or vector-oriented high-level programming language, MATLAB facilitates streamlined development of tailored applications, enhancing productivity and efficiency.

MATLAB's arsenal includes a diverse array of toolboxes tailored to address complex challenges across multiple application domains, providing users with specialized solutions for advanced problem-solving.

4.1.3 Basic building blocks of MATLAB:

At the core of MATLAB lies the matrix, serving as its foundational building block. Arrays represent the fundamental data type, encompassing vectors, scalars, real matrices, and complex matrices as distinct classes within this framework. MATLAB's built-in functions are meticulously optimized to streamline vector operations, enhancing efficiency and performance. Notably, MATLAB obviates the need for dimension statements when working with vectors or arrays, simplifying coding and computation tasks.

4.2 MATLAB Window:

The MATLAB works based on five windows

Command window

Work space window

Current directory window

Command history window

Editor window

Graphics window

Online-help window

4.2.1 Command Window:

The command window serves as the interface where users input MATLAB commands and expressions, indicated by the prompt (>>), and view the resulting output. It is automatically launched upon starting the MATLAB application program. Users interact with MATLAB by typing commands, including custom-written programs, directly into this window for execution, facilitating seamless interaction with the software environment.

4.2.2 Work Space Window:

Within MATLAB, the workspace encompasses the collection of variables generated by the user during a work session. The workspace browser provides a visual representation of these variables along with pertinent information. By double-clicking on a variable within the workspace browser, users can access the array editor, facilitating the retrieval and manipulation of data associated with that variable. This streamlined workflow enhances data exploration and analysis within the MATLAB environment.

4.2.3 Current Directory Window:

The "Current Folder" tab in MATLAB displays the contents of the current directory, with its path visible in the "Current Folder" window. For instance, in a Windows operating system, the path might be represented as follows: C:\MATLAB\work, indicating that the "work" directory is a subdirectory of the main directory "MATLAB" located in drive C.

Clicking on the arrow within the "Current Folder" window reveals a dropdown list of recently accessed paths.

MATLAB employs a search path to locate M-files and other MATLAB-related files. To execute any file in MATLAB, it must reside within the current directory or a directory listed on the search path. This ensures that MATLAB can locate and access the necessary files for execution seamlessly.

4.2.4 Command Window History:

The "Command History" window in MATLAB maintains a log of the commands entered by the user in the command window, spanning both the current session and previous MATLAB sessions. Users can review and select previously entered MATLAB commands from the command history window, facilitating their re-execution. Right-clicking on a command within the command history window provides users with various options in addition to executing the command, enhancing flexibility and productivity. This feature proves invaluable when experimenting with different commands during work sessions, allowing users to revisit and reuse commands efficiently.

4.2.5 Editor Window:

The MATLAB editor serves as both a specialized text editor for creating M-files and a graphical debugger. It can be displayed either in its own window or as a sub-window within the MATLAB desktop environment. Within this editor, users can write, edit, create, and save programs in files known as M-files.

Equipped with various pull-down menus, the MATLAB editor facilitates tasks such as saving, viewing, and debugging files. Its functionality includes performing basic checks and utilizing color to differentiate between different elements of code. As a result, this text editor is highly recommended as the preferred tool for writing and editing M-files, offering a user-friendly interface and enhanced productivity.

4.2.6 Graphics Window:

The "Command Window" in MATLAB serves as the interface where users input commands and expressions. However, it does not directly display graphical output generated by graphic commands. Instead, graphical output is typically displayed in separate windows or figures, which are generated by functions such as plot, imshow, or surf.

Once a graphical command is executed in the command window, MATLAB generates the corresponding output in a separate window, allowing users to view and interact with the graphical representation. These output windows are commonly referred to as figure windows and can be manipulated and customized using various MATLAB functions and tools.

4.2.7 Online Help Window:

MATLAB offers comprehensive online help for its built-in functions and programming language constructs through the MATLAB Help Browser. This browser is accessible as a separate window, either by clicking on the question mark symbol on the desktop toolbar or by entering "help browser" at the command window prompt.

The MATLAB Help Browser integrates a web browser directly into the MATLAB desktop environment, allowing users to access hypertext markup language (HTML) documents containing detailed information. The Help Browser consists of two main panes: the Help Navigator pane, utilized for locating specific information, and the Display pane, which presents the selected information.

In addition to these primary panes, there are self-explanatory tabs and tools available in the Help Browser, enabling users to perform searches and navigate through the wealth of available documentation efficiently. This comprehensive resource empowers users to access the information they need quickly and effectively, enhancing their understanding and utilization of MATLAB's features and capabilities.

4.3 MATLAB Files:

MATLAB has two types of files for storing information. They are

1. M-files

2. MAT – files

4.3.1 M-Files:

These are standard ASCII text files with a '.m' extension appended to the file name. Users create their own matrices using m-files, which are essentially text files containing MATLAB code. To generate these m-files, one can utilize the MATLAB Editor or any other text editor. The process involves entering the same statements as those typed at the MATLAB command line into the file and saving it under a name that ends with '.m'. There are two primary types of m-files: script files and function files.

4.3.2 Script Files:

These files contain a sequence of MATLAB commands that are executed sequentially when the file is run. They are useful for automating a series of tasks or calculations.

4.3.3 Function Files:

Function files define MATLAB functions that can be called from other scripts or functions. They typically accept input arguments and return output values. Function files allow for modularization and organization of code, promoting code reuse and maintainability.

4.3.4 Mat-Files:

These are binary files with a '.mat' extension, created by MATLAB when data is saved using the save command. The data is stored in a special format that only MATLAB can read. To load this data back into MATLAB, users utilize the load command. This allows for the preservation and retrieval of MATLAB variables and data structures, facilitating data sharing and analysis across different MATLAB sessions or environments.

4.4 MATLAB System:

The MATLAB system consists of five main parts:

4.4.1 Development Environment:

This comprehensive set of tools and facilities collectively aid users in utilizing MATLAB functions and files effectively. Many of these tools are graphical user interfaces (GUIs), designed to enhance the user experience. This ensemble encompasses:

MATLAB Desktop: Provides an integrated environment for accessing MATLAB's various features and functionalities.

Command Window: Serves as the primary interface for entering MATLAB commands and viewing their output.

Command History: Maintains a record of previously entered MATLAB commands, facilitating their recall and re-execution.

Editor and Debugger: Offers a specialized text editor for creating and editing MATLAB files (M-files) and includes debugging capabilities to troubleshoot code.

Help Browser: A browser integrated into the MATLAB desktop for accessing online help resources, including documentation, examples, and tutorials.

Workspace: Displays the current set of variables and their values created during a MATLAB session, providing insight into the state of the workspace.

File Browser: Allows users to navigate and manage files and directories within MATLAB.

Search Path: Specifies the directories in which MATLAB searches for files and functions, ensuring accessibility and usability of resources.

Collectively, these tools and facilities streamline the MATLAB workflow, empowering users to efficiently develop, debug, and execute MATLAB code while leveraging a wealth of resources and documentation for support and guidance.

4.4.2 MATLAB mathematical function:

MATLAB offers a vast collection of computational algorithms encompassing a wide range of functionalities. These algorithms cater to diverse mathematical and computational needs, spanning from elementary operations to advanced techniques. Some highlights include:

1. Elementary Functions: Basic mathematical operations such as addition, subtraction, multiplication, division, trigonometric functions (e.g., sine, cosine), and exponential functions.
2. Complex Arithmetic: Operations involving complex numbers, including addition, subtraction, multiplication, division, and complex exponentiation.
3. Matrix Operations: Comprehensive support for matrix operations, including matrix addition, subtraction, multiplication, division, transpose, and determinant calculation.
4. Matrix Inverse: Computation of the inverse of a matrix, essential for solving linear systems of equations and various mathematical applications.
5. Matrix Eigenvalues and Eigenvectors: Algorithms for computing the eigenvalues and eigenvectors of matrices, crucial for analyzing linear transformations and systems.
6. Bessel Functions: Special functions such as Bessel functions, which arise in many areas of applied mathematics, physics, and engineering, particularly in problems involving wave propagation and oscillations.
7. Fast Fourier Transforms (FFT): Efficient algorithms for computing the discrete Fourier transform (DFT) and its inverse, enabling analysis of signals and systems in the frequency domain.

These are just a few examples from the extensive array of computational algorithms available in MATLAB. The richness and versatility of MATLAB's computational capabilities make it a powerful tool for a wide range of scientific and engineering applications.

4.4.3 MATLAB language:

MATLAB is a high-level matrix or array language that offers a rich set of features for programming and computational tasks. These include:

1. Control Flow Statements: Constructs such as loops (for, while) and conditional statements (if, else) for controlling the flow of execution within a program.
2. Functions: The ability to define and call functions, enabling modularization and reuse of code.
3. Data Structures: Support for various data structures such as arrays, matrices, cell arrays, structures, and tables, facilitating efficient organization and manipulation of data.
4. Input and Output (I/O): Functions for reading data from files, writing data to files, and interacting with the user through input/output operations.
5. Object-Oriented Programming (OOP): MATLAB supports object-oriented programming paradigms, allowing users to create classes, objects, and methods for building modular and reusable code.
6. Programming in the Small: MATLAB is well-suited for rapidly prototyping and developing quick, ad-hoc solutions for small-scale problems.
7. Programming in the Large: MATLAB's features and capabilities also make it suitable for developing large and complex application programs, with support for structured programming techniques and software engineering principles.

Overall, MATLAB's combination of high-level syntax, powerful built-in functions, and extensive libraries make it a versatile tool for a wide range of programming tasks, from small-scale computations to large-scale application development.

4.4.4 GUI Construction:

MATLAB offers a robust suite of tools for visualizing vectors and matrices in graphical form, with extensive capabilities for annotation, printing, and customization. These include:

1. High-Level Functions: MATLAB provides high-level functions for generating two-dimensional and three-dimensional plots, enabling users to visualize data effectively. These functions cover a wide range of visualization needs, including line plots, scatter plots, surface plots, contour plots, and histograms.
2. Image Processing: MATLAB offers specialized functions for image processing tasks, allowing users to manipulate and analyze images with ease. These functions include image enhancement, filtering, segmentation, and feature extraction.
3. Animation and Presentation Graphics: MATLAB supports animation and presentation graphics, enabling users to create dynamic visualizations and slideshows to communicate their findings effectively.
4. Customization: MATLAB provides low-level functions that allow users to fully customize the appearance of graphics, including colors, line styles, markers, and fonts. This enables users to tailor visualizations to their specific needs and preferences.
5. Graphical User Interface (GUI) Development: MATLAB includes tools for building complete graphical user interfaces (GUIs) for MATLAB applications. Users can create interactive GUIs with buttons, sliders, menus, and other controls to enhance the usability and interactivity of their applications.

Overall, MATLAB's visualization capabilities empower users to explore and communicate their data effectively, whether through simple plots or interactive GUIs. With a rich set of functions and customization options, MATLAB provides a versatile platform for creating compelling visualizations for a wide range of applications.

4.4.5 MATLAB application program interface:

MATLAB provides a library known as the MATLAB Engine API, which enables the integration of C and FORTRAN programs with MATLAB. This library offers several key functionalities:

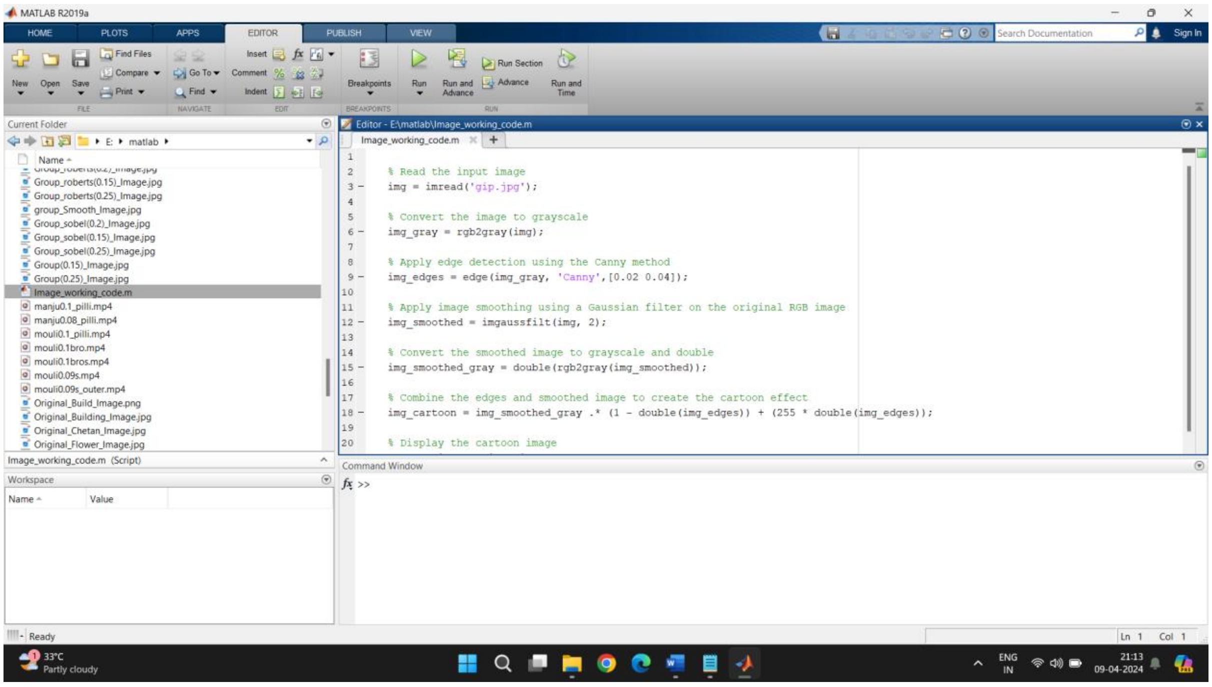
1. Calling MATLAB Functions: Users can call MATLAB functions from C or FORTRAN programs using the MATLAB Engine API. This allows for the utilization of MATLAB's extensive computational capabilities within C and FORTRAN code.
2. Using MATLAB as a Computational Engine: The MATLAB Engine API allows C and FORTRAN programs to treat MATLAB as a computational engine. This means that MATLAB can be invoked from within C or FORTRAN code to perform computations and return results.
3. Reading and Writing MAT-Files: MAT-files are binary files used by MATLAB to store data. The MATLAB Engine API provides facilities for reading and writing MAT-files from C and FORTRAN programs, enabling seamless data exchange between MATLAB and external code.

By leveraging the MATLAB Engine API, users can harness the power of MATLAB within their C and FORTRAN applications, enabling tighter integration and enhanced computational capabilities. This facilitates the development of complex systems that combine the strengths of both MATLAB and traditional programming languages.

4.5 MATLAB working environment:

4.5.1 MATLAB desktop

The MATLAB desktop serves as the central hub for MATLAB operations, featuring several key components. These include the command window, workspace browser, current directory window, command history window, and figure windows for graphical outputs. In the command window, users input MATLAB commands and expressions, with the output displayed below the prompt (>>).



```

% Read the input image
img = imread('gip.jpg');

% Convert the image to grayscale
img_gray = rgb2gray(img);

% Apply edge detection using the Canny method
img_edges = edge(img_gray, 'Canny',[0.02 0.04]);

% Apply image smoothing using a Gaussian filter on the original RGB image
img_smoothed = imgaussfilt(img, 2);

% Convert the smoothed image to grayscale and double
img_smoothed_gray = double(rgb2gray(img_smoothed));

% Combine the edges and smoothed image to create the cartoon effect
img_cartoon = img_smoothed_gray .* (1 - double(img_edges)) + (255 * double(img_edges));

% Display the cartoon image
```

```
>>
```

Fig 4.5: MATLAB desktop

The workspace refers to the collection of variables created during a session, visible in the workspace browser. Double-clicking on a variable in the browser opens the array editor, facilitating information retrieval and editing of variable properties. Adjacent to the workspace browser, the current directory tab displays the contents of the active directory, as indicated in the current directory window. For instance, in a Windows environment, the path might appear as C:\MATLAB\work, signifying that the 'work' directory resides within the 'MATLAB' main directory on the C drive. Users can access recently used paths by clicking the arrow icon in the current directory window and can change the active directory by clicking the button next to it.

In MATLAB, a search path is utilized to locate M-files and related files, which are organized within directories in the computer's file system. For MATLAB to execute a file, it must reside in the current directory or be on the search path. By default, MATLAB includes files supplied with MATLAB and

MathWorks toolboxes in the search path. To view or modify the directories on the search path, users can select "Set Path" from the file menu on the desktop and use the Set Path dialog box. Adding commonly used directories to the search path is advisable to avoid the need to repeatedly change the current directory. The command history window maintains a record of commands entered by the user in the command window, spanning both the current and previous MATLAB sessions. Users can select and re-execute previously entered MATLAB commands from the command history window by right-clicking on a command or sequence of commands. This action prompts a menu to appear, offering various options in addition to executing the commands. Such functionality proves useful when experimenting with different commands during a work session.

4.5.2 Using MATLAB editor to create m-files:

The MATLAB Editor serves both as a specialized text editor for creating M-files and as a graphical MATLAB debugger. It can be displayed either as a standalone window or as a sub-window within the desktop environment. M-files are identified by the extension.m. The MATLAB Editor window features various pull-down menus for tasks such as saving, viewing, and debugging files. With its ability to perform simple checks and use color to distinguish different elements of code, this text editor is highly recommended for writing and editing M-functions. To open the editor, simply type "edit" at the prompt, followed by the name of the M-file (e.g., "edit filename.m"), which opens the specified file in an editor window ready for editing. It's important to note that the file must reside either in the current directory or in a directory included in the search path.

4.5.3 Getting help:

The primary method to access help online in MATLAB is through the MATLAB Help Browser, which can be opened as a separate window by clicking on the question mark symbol on the desktop toolbar or by typing "help browser" at the command window prompt. The Help Browser integrates a web browser into the MATLAB desktop, allowing users to view Hypertext Markup

Language (HTML) documents containing help information. The Help Browser comprises two main windows: the Help Navigator window, utilized for finding information, and the Display window, where the information is presented. Additionally, there are self-explanatory tabs, including the Navigator pane, which are used to perform searches and navigate through the available help documentation.

4.6 Code

Step by Step Process:

Step – 1: Input Image

Step – 2: Apply Median Filter for Noise Removal

Step – 3: Convert Image to Grayscale

Step – 4: Apply Averaging Filter for Color Reduction

Step – 5: Apply Canny Edge Detector for Edge Detection

Step – 6: Apply Dilation to Enhance Edges

Step – 7: Perform Color Quantization using Proposed Formulation

Step – 8: Combine Edge-Enhanced Image with Color Quantized Image

Step – 9: Output Cartoon Image

Code:

```
% Convert the image to grayscale

inputImage = imread('Original_Group_Image.Image_Format');

% Convert the image to grayscale

grayImage = rgb2gray(inputImage);

% Apply bilateral filtering to smooth the image while preserving edges
```

```
smoothedImage = imbilatfilt(grayImage);

% Perform edge detection using the Canny edge detector

edgesImage = edge(smoothedImage, "Canny',0.20);

se = strel('square', 2);

dilatedImage = imdilate(edgesImage, se);

% Combine the edges with the smoothed image to create a cartoon effect

cartoonFrame = inputImage;

cartoonFrame(repmat(edgesImage, [1, 1, size(inputImage, 3)])) = 220;

OutputImage = imfuse(smoothedImage, edgesImage, 'blend');

% This line indicates the 4 parameters which we would measure for the
images

[psnr, mse, maxerr, L2rat] = measerr(inputImage, cartoonFrame)

 Display the original and cartoon images side by side

 figure;

 imshow(inputImage);

 title('Original Image');

 figure;

 imshow(grayImage);

 title('Gray Image');

 figure;

 imshow(smoothedImage);

 title('smoothed Image');
```

```
figure;

imshow(edgesImage);

title('Edged Image');

figure;

imshow(dilatedImage);

title('dialte Image');

figure;

imshow(cartoonFrame);

title('Cartoon Image');
```

Above is the Code for Image to Cartoon Conversion. The steps follow the order of the Block diagram and here we didn’t use any specific color quantization technique, because this technique is less complex while compared to the original quantization techniques.

Result:

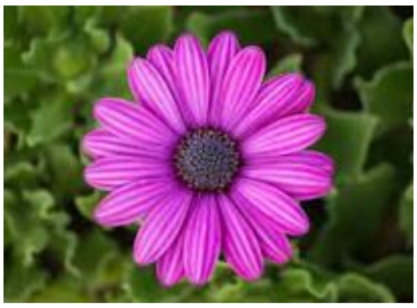


Fig 4.6.1 Original Image

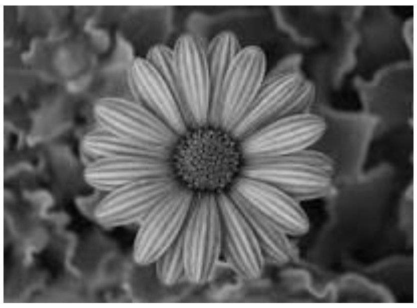


Fig 4.6.2 Gray Scale Image

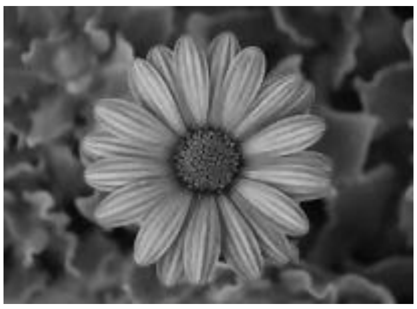


Fig 4.6.3 Smoothing Image

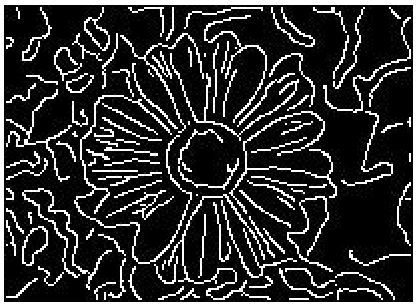


Fig 4.6.4 Edge Detection

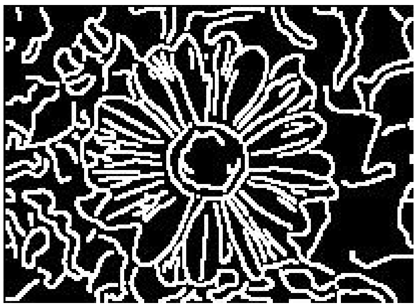


Fig 4.6.5 Dilate Image

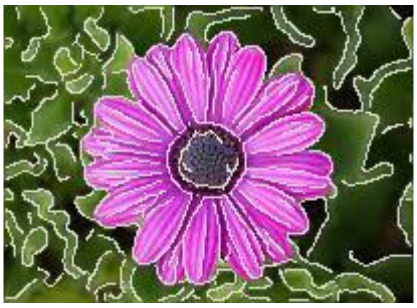


Fig 4.6.6 Cartoon Image

Table 4.6.1: Results of the image in all the stages of the image

Threshold Value	Edge Detection	PSNR	MSE	MAXERR	L2RAT
0..2	Canny	13.8763	2663.5	220	1.3576

Table 4.6.2: Parameters for the example image

This Histogram graph shows the histogram equalization of the above flower image. This graph shows thresholds at several points of the dimensions of the image. This is useful to compare many of the images with ease but the only problem is dimensions which vary from image to image.

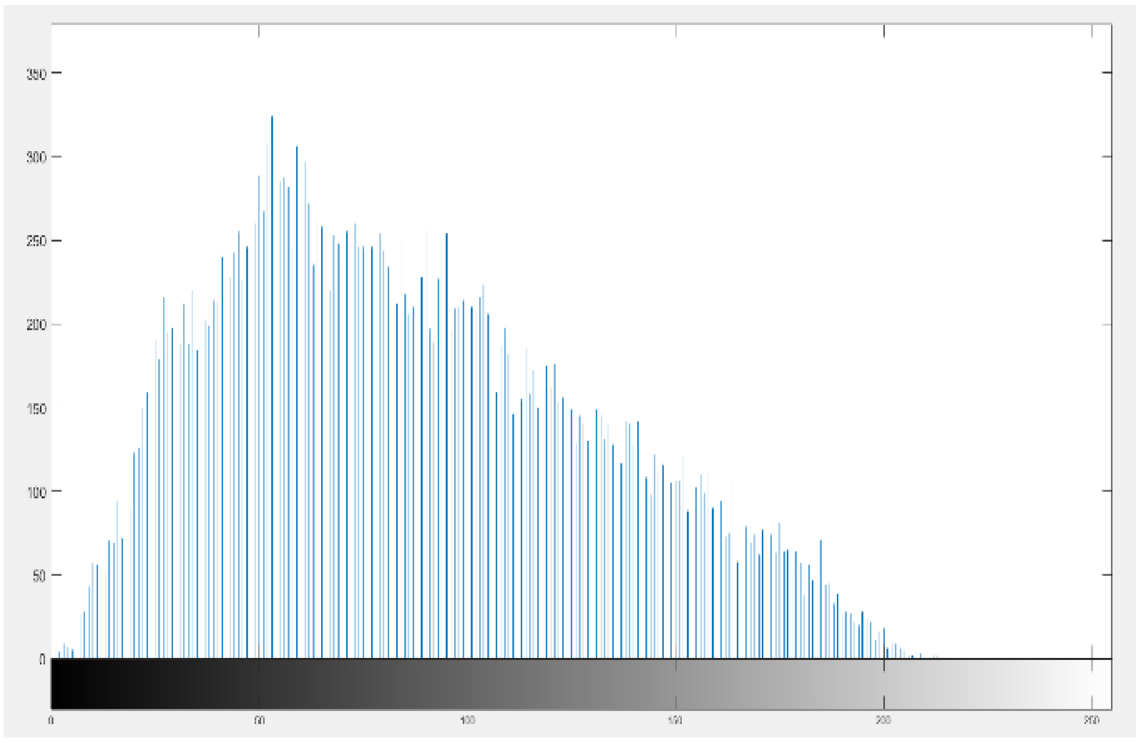


Fig.4.6.7 Histogram Equalization

Chapter V

5 Experimental Results

5.1 Performance Measures

The performance measures which are used to make the comparison between the images which gives the best of them among all the taken images. The performance measures which are taken here are PSNR, MSE, MAXERR, L2RAT.

5.1.1 PSNR

Peak Signal-to-Noise Ratio (PSNR) is a metric commonly used to assess the quality of reconstructed or compressed images. It quantifies the difference between the original and distorted images by comparing their peak signal strength to the level of noise introduced by the distortion.

5.1.2 MSE

Mean Squared Error (MSE) is a metric used to quantify the average squared difference between the pixel values of two images. It provides a measure of the overall distortion or error between the original and reconstructed images.

5.1.3 MAXERR

Maximum Error (MaxErr) is a metric used to quantify the maximum difference between the pixel values of two images. It provides a measure of the most significant distortion or error between the original and reconstructed images.

5.1.4 L2RAT

L2 ratio (L2rat) is a measure of the relative magnitudes of two vectors based on their L2 norms. Also known as the Euclidean norm, the L2 norm of a vector represents its length in a Euclidean space, computed as the square root of the sum of squared elements. The L2 ratio between two vectors provides insight into their similarity or dissimilarity in terms of magnitude.

5.2 Data Comparison Results for the Images

Here the comparisons are done for the four operators used for the edge detection they are

1. Canny Operator
2. Prewitt Operator
3. Robert Operator
4. Sobel Operator

5.2.1 Canny Operator

This Canny Operator is a multi-stage algorithm that aims to detect a wide range of edges in images while suppressing noise and minimizing false positives. This operator involves steps such as gaussian blurring to smoothening the image, gradient calculation to find edge strength and direction, non-maximum suppression to thin out edges, hysteresis thresholding to detect string and weak edges.

Code:

```
inputImage = imread('Original_Group_Image.jpg');

% Convert the image to grayscale

grayImage = rgb2gray(inputImage);

% Apply bilateral filtering to smooth the image while preserving edges

smoothedImage = imbilatfilt(grayImage);

% Perform edge detection using the Canny edge detector for threshold values
0.15, 0.2, 0.25

edgesImage = edge(smoothedImage, 'Canny',0.15);
edgesImage = edge(smoothedImage, “Canny',0.20);

edgesImage = edge(smoothedImage, “Canny',0.25);

se = strel('square', 2);

dilatedImage = imdilate(edgesImage, se);

% Combine the edges with the smoothed image to create a cartoon effect

cartoonFrame = inputImage;

cartoonFrame(repmat(edgesImage, [1, 1, size(inputImage, 3)])) = 220;

OutputImage = imfuse(smoothedImage, edgesImage, 'blend');

[psnr, mse, maxerr, L2rat] = measerr(inputImage, cartoonFrame)
```

The Comparison table for 5 types of images are given below:

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		15.4025	1876.3	220	1.2634
2.		21.5646	4535453	220	1.0820
3.		13.4844	2915	220	1.3886
4.		17.8554	1065.5	220	1.0949
5.		20.6665	557.7378	209	1.0580

Table 5.2.1.1: Parameters of the image for Canny at a threshold value of 0.15

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		16.1061	1593.9	220	1.2280
2.		22.8519	337.2008	220	1.0631
3.		13.8763	2663.5	220	1.3576
4.		18.3950	940.9801	220	1.0853
5.		21.5389	456.2358	202	1.0480

Table 5.2.1.2: Parameters of the image for Canny at a threshold value of 0.2

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		16.9133	1323.6	220	1.1926
2.		24.2426	244.8059	220	1.0486
3.		14.2960	2418.1	220	1.3258
4.		19.0059	817.5132	220	1.0752
5.		22.3872	375.2835	198	1.0398

Table 5.2.1.3: Parameters of the image for Canny at a threshold value of 0.25

5.2.2 Prewitt operator

This Prewitt operator computes the gradient approximation of an image intensity function. It is based on convolving the image with a pair of 3x3 convolution kernels, one predicting the gradient in the x-direction and the other in the y-direction in the y-direction. The edges are detected where the gradient is maximum.

Code:

```
inputImage = imread('Original_Group_Image.jpg');

% Convert the image to grayscale

grayImage = rgb2gray(inputImage);

 % Apply bilateral filtering to smooth the image while preserving edges

smoothedImage = imbilatfilt(grayImage);

 % Perform edge detection using the Prewitt edge detector for threshold values
0.15, 0.2, 0.25

edgesImage = edge(smoothedImage, ‘prewitt’,0.15);

edgesImage = edge(smoothedImage, ‘prewitt’,0.20);

edgesImage = edge(smoothedImage, ‘prewitt’,0.25);

 se = strel('square', 2);

dilatedImage = imdilate(edgesImage, se);

% Combine the edges with the smoothed image to create a cartoon effect

cartoonFrame = inputImage;

cartoonFrame(repmat(edgesImage, [1, 1, size(inputImage, 3)])) = 220;

OutputImage = imfuse(smoothedImage, edgesImage, 'blend');
```

[psnr, mse, maxerr, L2rat] = measerr(inputImage, cartoonFrame)

display("OutputImage")

The Comparison table for 5 types of images are given below:

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		24.6206	224.3988	220	1.3086
2.		35.6054	19.1663	220	1.0042
3.		26.9561	131.0591	220	1.0206
4.		24.5701	285.8021	220	1.0260
5.		37.7880	10.8212	156	1.0014

Table 5.2.2.1: Parameters of the image for Prewitt at a threshold value of 0.15

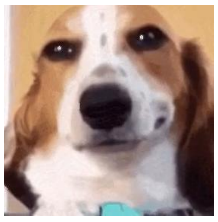
S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		30.4520	58.5976	220	1.0121
2.		38.9658	8.2509	175	1.0020
3.		31.6168	44.8124	220	1.0062
4.		27.4542	116.8588	220	1.0143
5.		43.0321	3.2350	123	1.0005

Table 5.2.2.2: Parameters of the image for Prewitt at a threshold value of 0.2

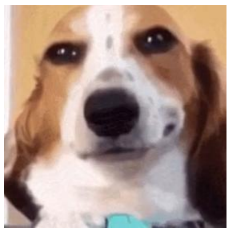
S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		36.1479	15.7865	208	1.0038
2.		42.7166	3.4787	154	1.0011
3.		40.2236	6.1762	184	1.0010
4.		30.1885	62.2643	209	1.0078
5.		INF	0	0	1

Table 5.2.2.3: Parameters of the image for Prewitt at a threshold value of 0.25

5.2.3 Robert operator

The Robert operator is simplest edge detection operators. It uses a pair of 2x2 convolution kernels to approximate the gradients in the x and y directions. It's less computationally intensive compared to Prewitt and Sobel operators but may not perform as well in all scenarios.

Code:

```
inputImage = imread('Original_Group_Image.jpg');

 % Convert the image to grayscale

grayImage = rgb2gray(inputImage);

 % Apply bilateral filtering to smooth the image while preserving edges

smoothedImage = imbilatfilt(grayImage);

 % Perform edge detection using the Robert edge detector for threshold values
0.15, 0.2, 0.25

edgesImage = edge(smoothedImage, ‘roberts’,0.15);

edgesImage = edge(smoothedImage, ‘roberts’,0.20);

edgesImage = edge(smoothedImage, ‘roberts’,0.25);

 se = strel('square', 2);

dilatedImage = imdilate(edgesImage, se);

% Combine the edges with the smoothed image to create a cartoon effect

cartoonFrame = inputImage;

cartoonFrame(repmat(edgesImage, [1, 1, size(inputImage, 3)])) = 220;

OutputImage = imfuse(smoothedImage, edgesImage, 'blend');

[psnr, mse, maxerr, L2rat] = measerr(inputImage, cartoonFrame)
```

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		24.3465	23.0191	220	1.0349
2.		37.0416	12.506	207	1.0028
3.		25.8375	169.5616	220	1.0237
4.		23.5701	285.8021	220	1.0260
5.		39.6745	7.008	173	1.008

Table 5.2.3.1 Parameters of the image for Robert at a threshold value of 0.15

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		29.9575	65.6639	220	1.0101
2.		39.708	6.9663	193	1.0017
3.		31.6168	44.8124	220	1.0062
4.		26.1339	158.3775	220	1.0144
5.		56.8210	1.0355	86	1.0000

Table 5.2.3.2: Parameters of the image for Robert at a threshold value of 0.2

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		35.8889	16.7566	220	1.0027
2.		41.0341	5.124	169	1.0014
3.		35.8327	16.9750	219	1.0021
4.		28.7769	86.1772	220	1.0074
5.		INF	0	0	1

Table 5.2.3.3: Parameters of the image for Robert at a threshold value of 0.25

5.2.4 Sobel operator

The Sobel Operator is similar to Prewitt operator, it uses a more sophisticated edge detection algorithm that uses convolution with 3x3 kernels to approximate the gradient of the image intensity function. This operator specifies regions of high spatial frequency in the image.

Code:

```
% Takes the input of the image

inputImage = imread('Input_Image.Image_Format');

 % Convert the image to grayscale

grayImage = rgb2gray(inputImage);

 % Apply bilateral filtering to smooth the image while preserving edges

smoothedImage = imbilatfilt(grayImage);

 % Perform edge detection using the Sobel operator for threshold values 0.15,
0.2, 0.25

edgesImage = edge(smoothedImage, ‘sobel’,0.15);

edgesImage = edge(smoothedImage, ‘sobel’,0.20);

edgesImage = edge(smoothedImage, ‘sobel’,0.25);

 se = strel('square', 2);

dilatedImage = imdilate(edgesImage, se);

% Combine the edges with the smoothed image to create a cartoon effect

cartoonFrame = inputImage;

cartoonFrame(repmat(edgesImage, [1, 1, size(inputImage, 3)])) = 220;

OutputImage = imfuse(smoothedImage, edgesImage, 'blend');
```

[psnr, mse, maxerr, L2rat] = measerr(inputImage, cartoonFrame)

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		24.2051	246.9528	220	1.0423
2.		35.1303	19.9551	220	1.0044
3.		26.2343	154.7651	220	1.0243
4.		24.3187	240.5543	220	1.0269
5.		37.3316	12.0203	156	1.0015

Table 5.2.4.1: Parameters of the image for Sobel at a threshold value of 0.15

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		29.8808	66.8340	220	1.0137
2.		38.9017	8.3735	220	1.0021
3.		32.2884	38.3918	208	1.0058
4.		27.2067	123.7121	220	1.0151
5.		42.9167	3.3221	123	1.0005

Table 5.2.4.2: Parameters of the image for Sobel at a threshold value of 0.2

S.no	IMAGE	PSNR	MSE	MAXERR	L2RAT
1.		35.5599	18.0757	208	1.0044
2.		42.7211	3.4751	143	1.0011
3.		39.4432	7.3919	184	1.0012
4.		29.8940	66.6315	209	1.0084
5.		INF	0	0	1

Table 5.2.4.3: Parameters of the image for Sobel at a threshold value of 0.25

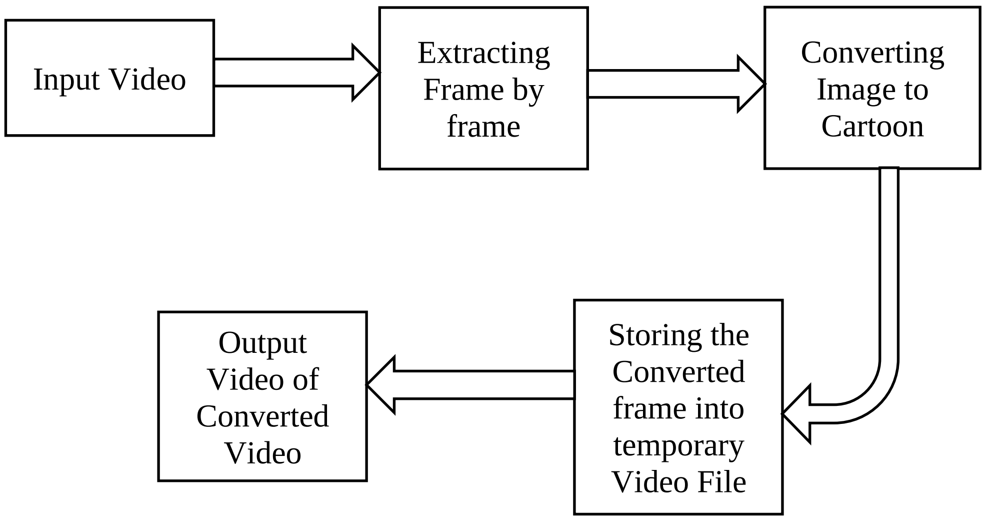
Original Video to Cartoon Video Conversion:

In the future scope, we extended this process from Video to Cartoon Video. We have additionally used two Algorithms which was created to finish this extension.

The Algorithm-1 goes as this:

- Take the video input
- Collect frame of this video to covert the frame into cartoon image and add the converted frame into another video file.
- Do this cycle until the information video is totally changed over into Cartoon video.

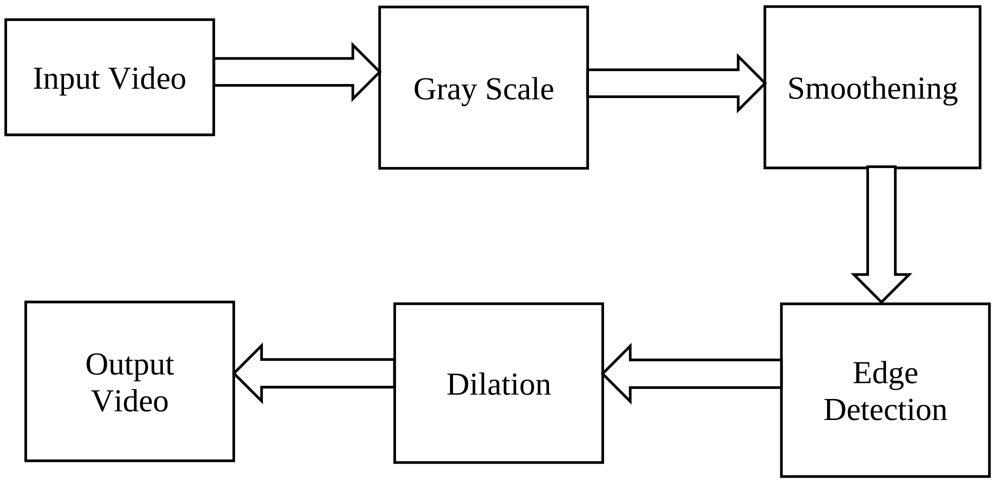
Block Diagram for Algorithm-1:


Input Video
Extracting Frame by frame
Converting Image to Cartoon
Storing the Converted frame into temporary Video File
Output Video of Converted Video

The Algorithm 2 goes as this:

- Take the video input
- Apply Gray Scaling, Smoothening, Edge Detection, Dilation processes individually to the input video.

Block Diagram for Algorithm-2:



The Difference between both the Algorithms:

Algorithm-1	Algorithm-2
It requires direct investment and additional room to save the video.	It gets some margin for multiple times and no additional room is taken.
As each casing is gathered and changed over it doesn't give that much errors.	As each cycle is applied single time totally it requires a lot of investment and it probably won't give the productive result.

Table 6.2.1: Difference between both the Algorithms

This provides the difference between the two algorithms. Thus, we picked the Algorithm-1 and accomplished the work to on it and got the result.

The Transformation time relies upon the length of the video, nature of the video.

The Video conversion may not be shown in the book, so we are attaching the drive link in which all data of the 3 video conversions is provided.

The naming conventions goes a follow:

"Video_Name_Input" is the Input Video

"Video_Name_Outputs" is the Output Video

This is the Link: **https://drive.google.com/drive/folders/1WHDMLhF3_C6kEfxnWNOe05B8I0OGgSdO?usp=drive_link**

The Video Conversion steps are mentioned in the algorithm. So, here the information which is not provided in the algorithm like memory management, time of conversion. It depends on the Video Quality and the dimensions of the video. Here is the data from the three videos which we have put in the link.

S.no	**Name of the Video**	**Size of the Video**	**Type of the Video**	**Conversion Time**
1.	Building_Input	15.6MB	MP4	15.7sec
	Building_Output	15.1MB	MP4	
2.	Water_Input	36.1MB	MP4	32.46sec
	Water_Output	30.7MB	MP4	
3.	Boy_Input	1.09MB	MP4	18.1sec
	Boy_Output	14.2MB	MP4	

Table 6.2.2: Comparison of the Inputs and Outputs of the Videos

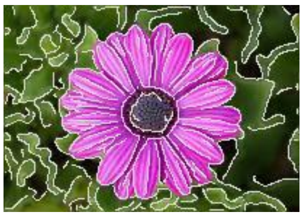
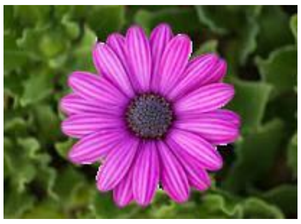
Chapter VI

6 Conclusion and Future Scope

6.1 Conclusion

After the processing on an image, they show that the comparison of different images with many thresholds for edge detection we ought to know that Canny is the best operator among all of them, and this color quantization is different for different images which is its advantage over the advanced image processing as it convert images according to the image quality.

The experimental results to show that canny is the best operator for the edge detection for our Conversions which are shown below and the comparisons are made by using the four operators (Canny, Prewitt, Robert, Sobel) at a threshold of 0.2:

S.no	Operator	Image output	PSNR	MSE	MAXERR	L2RAT
1.	Canny		13.8763	2663.5	220	1.3576
2.	Prewitt		33.0669	32.0912	208	1.0048

3.	Robert		31.6168	44.8124	220	1.0062
4.	Sobel		32.2884	38.3918	208	1.0058

Table 6: Cartoon Image outputs comparisons for different edge detectors

In conclusion, this presents an innovative image processing-based approach for generating cartoon images from Images taken by Humans. By using techniques such as noise removal, edge detection, dilation, averaging filtering, and color quantization, the proposed method effectively enhances contrast and simplifies color palettes to create visually appealing cartoon images. Here we have compared the image for different threshold values which are 0.15, 0.20, 0.25. Through comprehensive experimentation, the proposed approach demonstrates superior performance compared to existing image processing and deep learning-based methods, offering a practical and efficient solution for cartoon image generation. Overall, this contributes significantly to the field of image processing, providing a valuable technique for transforming ordinary photographs into engaging cartoon images.

After the handling on a picture, they show that the correlation of various pictures with numerous limits for edge location we should realize that Vigilant is the best administrator among every one of them, and this variety quantization is different for various pictures which is its benefit over the high-level picture handling as it converts pictures as per the picture quality.

All in all, this presents an imaginative picture handling-based approach for creating animation pictures from Pictures taken by People. By utilizing procedures like commotion expulsion, edge recognition, enlargement, averaging sifting, and variety quantization, the proposed strategy successfully improves difference and works on variety ranges to make outwardly engaging animation pictures. Here we have analyzed the picture for various limit values which are 0.15, 0.20, 0.25. Through thorough trial and error, the proposed approach shows better execution looked at than existing picture handling and profound learning-based strategies, offering a commonsense and proficient answer for animation picture age. By and large, this contributes fundamentally to the field of picture handling, giving a significant procedure to changing normal photos into connecting with animation pictures.)

6.2 Future Scope

We've completed our extension to this project but we have faced some challenges in Video to Cartoon Conversion process. Some of the Challenges we faced were:

1. Memory management
2. Time of conversion

1. Memory management: It's based on videos. If the videos are taken by camera, then after cartoon conversion the memory management will be reduced. If the video is downloaded from other device or website, the converted video will have large memory to store compare to original video.
2. Time of conversion: It is based on the type of video, quality of the video, video length etc. Based on these the time of video conversion will be different.

If possible, we will try to reduce the challenges we faced in this Video Conversion.

Chapter VII

7 References

[1] A. B. Patankar, P. A. Kubde and A. Karia, "Image cartoonization methods," 2016 International Conference on Computing Communication Control and automation (ICCUBEA), Pune, India, 2016, pp. 1-7, DOI: 10.1109/ICCUBEA.2016.7860045.

[2] A. Jose, K. Deepa Merlin Dixon, N. Joseph, E. S. George and V. Anjitha, "Performance study of edge detection operators," 2014 International Conference on Embedded Systems (ICES), Coimbatore, India, 2014, pp. 7-11, DOI: 10.1109/EmbeddedSys.2014.6953040.

[3] Chinmay Joshi, Devendra Jaiswal and Akshata Patil, "Technical Paper Presentation on Application of Cartoon like effects to Actual Images", International Journal of Innovative Science and Research Technology, ISSN No: 2456-2165, Volume 4, Issue 3, March 2019.

[4] F. Andersson and S. Arvidsson, "Generative Adversarial Networks for photo to Hayao Miyazaki style cartoons". arXiv, 2020. DOI: 10.48550/arXiv.2005.07702.

[5] J. Dong, J. He and H. Wang, "Edge Detection of Human Face," 2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), Fuzhou, China, 2021, pp. 596-601, DOI: 10.1109/CEI52496.2021.9574525.

[6] Kevin Dade, "Toonify: Cartoon Photo Effect Application".

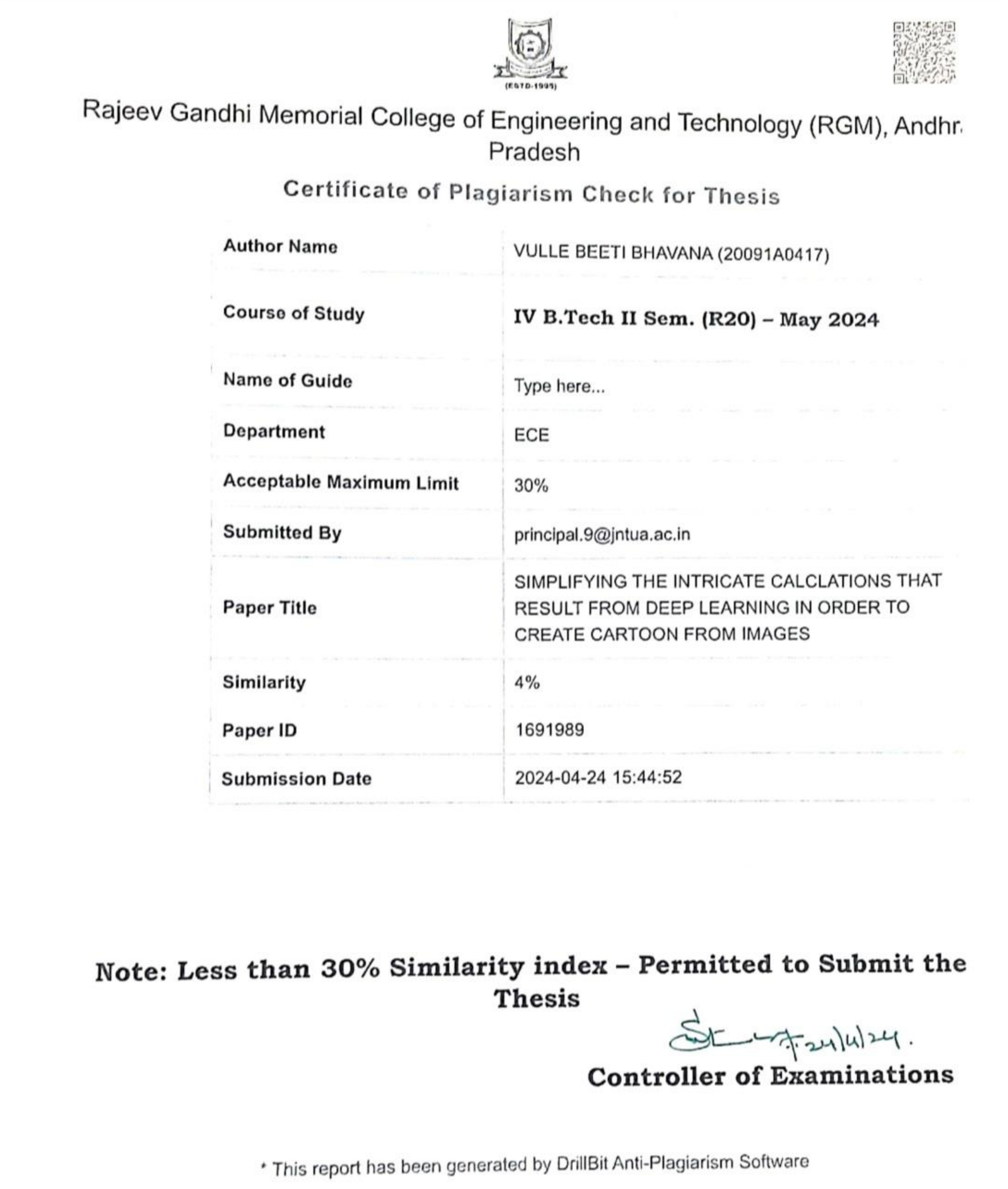
[7] Kumar, S., Sagar, V. & Punetha, D. A comparative study on facial expression recognition using local binary patterns, convolutional neural network and frequency neural network. Multimed Tools Appl (2023). https://doi.org/10.1007/s11042-023-14753-y.

[8] S. O. Abter and N. A. Z. Abdullah, "An efficient color quantization using color histogram," 2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT), Baghdad, Iraq, 2017, pp. 13-17, DOI: 10.1109/NTICT.2017.7976153.

[9] S. Wang and J. Qi, "A Novel Image Cartoonization Algorithm without Deep Learning," CIBDA 2022; 3rd International Conference on Computer Information and Big Data Applications, Wuhan, China, 2022, pp. 1-4.

[10] Vaishali Sudarshan and Amritesh Singh, "Cartooning an Image Using OpenCV and Python", ISSN: 2456-236X, Volume 4, Issue 2, 2020.

[11] Y. Chen, Y. -K. Lai and Y. -J. Liu, "Cartoon GAN: Generative Adversarial Networks for Photo Cartoonization," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 9465-9474, DOI: 10.1109/CVPR.2018.00986.



Rajeev Gandhi Memorial College of Engineering and Technology (RGM), Andhr
Pradesh

Certificate of Plagiarism Check for Thesis

Author Name	VULLE BEETI BHAVANA (20091A0417)
Course of Study	**IV B.Tech II Sem. (R20) – May 2024**
Name of Guide	Type here...
Department	ECE
Acceptable Maximum Limit	30%
Submitted By	principal.9@jntua.ac.in
Paper Title	SIMPLIFYING THE INTRICATE CALCLATIONS THAT RESULT FROM DEEP LEARNING IN ORDER TO CREATE CARTOON FROM IMAGES
Similarity	4%
Paper ID	1691989
Submission Date	2024-04-24 15:44:52

Note: Less than 30% Similarity index – Permitted to Submit the Thesis

[illegible] 24/4/24.

Controller of Examinations

* This report has been generated by DrillBit Anti-Plagiarism Software