"CARTOONING AN IMAGE "

A MINI PROJECT REPORT SUBMITTED

by

K.Sindhuja

Enroll.No .2320200

Faculty Incharge

Dr.P.S.Latha Kalyampudi
Assistant Professor
Department of Computer science

DEPARTMENT OF COMPUTER SCIENCE

CENTRAL TRIBAL UNIVERSITY OF ANDHRA PRADESH

VIZIANAGARAM, ANDHRA PRADESH-535003

2025

DECLARATION

I,_____ ,bearing Enrollment Number_____here by declare that this Mini project report on "**CARTOONING AN IMAGE USING MACHINE LEARNING IN PYTHON**"is my own work except where specifically stated to the contrary and that it is not substantially the same as any project report, which has been submitted to any other University.

Place: Vizianagaram

Date: _____

Student Name:

_____

Signature:

_____

CERTIFICATE

This is to certify that the Mini Project Report entitled, **"CARTOONING AN IMAGE USING MACHINE LEARNING IN PYTHON"** submitted by_____, bearing Enrollment Number_____ is a Project report work done by the candidate during the period of study under my supervision and the dissertation has not previously formed the basis for the award to the candidate of any degree.

Place: Vizianagaram

Date: _____

Dr.P.S.Latha Kalyampudi ,
Assistant Professor
Department of Computer Science

ACKNOWLEDGEMENT

I express my sincere thanks to **Prof. Tantravahi Srinivasan**, Hon'ble Vice-Chancellor, Central Tribal University of Andhra Pradesh, Vizianagaram for permitting me to carry out this Lab Report, his valuable blessings, and ready cooperation during the progress of my project work.

It is my duty to say thanks to my teachers **Dr.Bonthu Kotaiah, Associate Professor cum Head, Department of Computer science** for their excellent teaching and ready help throughout the semester for completion of this project work.

It is my duty to say thanks to my teachers **Dr. P.S. Latha Kalyampudi, Assistant Professor, Department of Computer science** for their excellent teaching and ready help throughout the semester for completion of this project work.

I am acknowledging all the Authors, Publishers, Research Scholars & Internet sources of information which I have collected and presented in this lab work. Lastly, I am thankful to all my family members, friends and well-wishers, who directly or indirectly helped me a lot in completion of this Lab Work Successfully

STUDENT NAME: _____

ENROLLMENT NO.: _____

TABLE CONTENT

ABSTRACT

Cartooning an image is a creative application of computer vision and image processing techniques that transforms real-life photos into cartoon-like representations. The primary objective of this project is to develop a Python-based program using the **OpenCV library** to convert normal images into cartoon versions with enhanced visual effects. The cartooning process involves multiple stages of image transformation, including grayscale conversion, noise reduction, edge detection, and bilateral filtering for smooth color regions.

OpenCV provides a powerful set of tools for image processing, making it an ideal choice for implementing this project. The algorithm identifies the key outlines and features in an image using adaptive thresholding and combines them with color simplification techniques to achieve a stylized cartoon effect. This project helps in understanding fundamental image processing operations such as edge detection, color quantization, and masking. The output is a clear and smooth cartoon-like image that retains essential details while giving a simplified, aesthetic appearance.

This project demonstrates how fundamental computer vision algorithms can be applied for fun, artistic, and educational purposes. The implementation is simple, cost-effective, and can be extended to real-time video streams for cartoonifying webcam feeds or creating creative filters for social media applications.

1. INTRODUCTION

In the digital era, image processing has emerged as one of the most dynamic and essential fields within computer science and artificial intelligence. It focuses on enabling computers to analyze, interpret, and manipulate visual information in the same way that humans perceive and understand images. Through image processing, a wide range of applications have been developed—spanning from medical imaging, object recognition, and autonomous vehicles to entertainment and creative media design.

One of the most engaging and visually appealing applications of image processing is cartooning an image, a process that transforms a real-life photograph into a stylized, cartoon-like representation. This involves simplifying colors, emphasizing edges, and reducing fine details to create an artistic visual effect that mimics hand-drawn or comic-style illustrations. The final result retains the essence of the original image while giving it a distinctive and visually striking cartoon appearance.

The technique of cartooning an image is not merely for artistic purposes—it also serves as an excellent way to understand fundamental computer vision operations such as edge detection, noise reduction, image smoothing, and thresholding. By learning how these operations interact, students and researchers can gain deeper insights into how computers process visual data at the pixel level to extract meaningful patterns and features.

With the advent of powerful open-source libraries such as OpenCV (Open Source Computer Vision Library), implementing such image transformations has become easier and more accessible. OpenCV provides a wide range of functions that perform various image operations such as filtering, color space transformation, morphological operations, and feature detection. Its efficiency, speed, and cross-platform support make it one of the most preferred libraries for computer vision applications.

In this project titled "Cartooning an Image using OpenCV in Python," the primary objective is to create a Python-based system that can automatically convert an input image into a cartoon-like version by applying sequential image processing techniques. The cartooning process is achieved by combining multiple transformations in a step-by-step manner, each contributing to a specific visual enhancement:

1. **Grayscale Conversion** – The first step involves converting the input image from RGB (color) to grayscale to simplify intensity variations and prepare it for edge detection.

2. **Noise Reduction using Median Blur** – Image noise, which can interfere with edge detection, is reduced using median blurring, which smoothens the image while preserving sharp edges.

3. **Edge Detection using Adaptive Thresholding** – The grayscale image is processed using adaptive thresholding to detect the prominent edges and contours that define the structure of objects in the image.

4. **Bilateral Filtering for Color Smoothing** – To achieve a cartoon-like effect, bilateral filtering is applied to the original color image. This technique smoothens color gradients while preserving sharp edges, giving the image a soft, painted look.

5. **Combining Edge Mask with the Smoothed Image** – Finally, the detected edge mask is overlaid on the filtered color image, producing a clear, artistic cartoon image that balances color and edge definition.

The outcome of this process is a visually pleasing cartoon version of the input image that highlights key contours while minimizing unnecessary textures or shadows.

This project provides an excellent learning experience for students and beginners in the field of computer vision, as it combines several foundational image processing concepts into one creative and practical implementation. Furthermore, it demonstrates how traditional computer vision algorithms—without the need for advanced deep learning models—can still produce impressive results that have real-world relevance in fields such as digital art, photography, animation, gaming, and social media filters.

By developing this project, learners gain hands-on experience in working with OpenCV functions, understanding image transformations, and building a mini application that bridges the gap between technical computation and artistic creativity. It also opens pathways to explore advanced topics such as real-time video cartooning, neural style transfer, and AI-based visual stylization in future research.

1.1. Problem Definition

In recent years, the exponential growth of **digital media and computer graphics** has transformed the way images are processed, edited, and presented. Visual content creation has become a central part of industries such as entertainment, advertising, education, and social media. In this context, **image stylization**—the process of converting photographs into artistic versions—has gained significant attention. Among various stylization effects, **cartooning an image** stands out as a popular technique that simplifies the visual complexity of an image while retaining its key structural details, producing a hand-drawn or animated appearance.

Traditionally, converting an image into a cartoon version required extensive manual work using complex software such as **Adobe Photoshop, CorelDRAW, or Illustrator**. Such manual techniques involve applying filters, adjusting edges, manipulating color saturation, and fine-tuning brightness and contrast. While these tools can produce high-quality results, they are often **time-consuming, require professional expertise, and lack automation**. As a result, there is a growing need for an **automated, efficient, and algorithmic approach** to transform regular images into cartoon-like representations with minimal human intervention.

The central challenge of this project is to **design and implement an algorithm** that can automatically detect essential features such as edges, textures, and colors within an image and then process them to create a simplified, smooth, and visually appealing cartoon version. The output

should retain important image characteristics—like shape and structure—while reducing unnecessary noise, color gradients, and textures.

To address this problem, the project leverages **OpenCV (Open Source Computer Vision Library)** in Python, which provides a rich set of tools for image manipulation and transformation. OpenCV's optimized functions make it possible to implement edge detection, filtering, and thresholding efficiently, even on standard computing systems without requiring advanced hardware or deep learning models.

The **problem statement** can therefore be formally defined as follows:

"To develop a Python-based application using OpenCV that automatically transforms a real-life image into a cartoon-like version by combining edge detection, noise reduction, and color smoothing techniques, while preserving important visual details and enhancing artistic appearance."

This problem involves several key image processing operations and principles:

1. **Noise Reduction using Blurring Techniques –** Real-world images often contain random noise due to lighting, texture, or camera quality. Before stylization, this noise must be minimized using **median blurring** or **Gaussian blurring**, ensuring a cleaner base image for further processing.

2. **Edge Detection for Extracting Outlines –** Cartoon-like effects depend heavily on bold and clear edges. The project uses **adaptive thresholding** or **Canny edge detection** to extract these outlines, which form the basis for cartoon-style borders.

3. **Bilateral Filtering for Color Simplification** – Instead of applying a uniform blur, **bilateral filtering** smoothens color transitions while preserving edges, resulting in flat, paint-like color regions that resemble hand-drawn artwork.

4. **Masking and Blending for Stylized Output** – The processed edge mask is combined with the color-simplified image using **bitwise operations** to merge outlines and colors. This step produces the final cartoon image, where edges remain prominent, and colors appear smooth and vibrant.

5. **Brightness and Contrast Enhancement (Optional) –** To make the cartoon image appear more vivid and visually clear, brightness and contrast adjustments can be applied to enhance the final output.

The ultimate goal of this project is to create an **automated system capable of generating high-quality cartoon images** without requiring manual intervention or professional editing skills. The system should be:

- **Accurate** – preserving major image details such as edges and shapes.

- **Efficient** – capable of processing images quickly on standard computers.

- **User-Friendly** – simple to use, with minimal setup or technical requirements.

By solving this problem, the project demonstrates how fundamental image processing techniques can be combined to perform creative visual transformations. It also highlights the potential of Python and OpenCV for developing practical, real-time computer vision applications that bridge the gap between art and technology.

1.2. Objectives

The primary goal of the project titled **"Cartooning an Image using OpenCV in Python"** is to design and develop an automated system capable of transforming a real-world image into a visually appealing **cartoon-like version** using the principles of **image processing and computer vision**.

This project focuses not only on producing artistic results but also on enhancing the understanding of fundamental image processing operations that are widely used in the fields of artificial intelligence, multimedia, and visual computing. By applying multiple OpenCV techniques in sequence, the project provides an opportunity to explore how mathematical and algorithmic concepts can be used to achieve creative, real-world applications.

The detailed objectives of the project are as follows:

1. To understand and implement fundamental image processing techniques

One of the core objectives of this project is to help learners **gain hands-on experience with basic image processing operations** such as:

- **Grayscale conversion**, which simplifies images by reducing them to intensity levels.
- **Edge detection**, which identifies object boundaries and outlines that define the structure of an image.
- **Blurring and filtering**, which smoothen the image and remove unwanted noise or artifacts.

By implementing these techniques step by step, students develop a deeper understanding of how digital images are represented as matrices of pixels and how each transformation affects the visual output.

2. To design an efficient algorithm that converts real images into cartoon representations

The project aims to build a **systematic algorithmic pipeline** that can take any real-world image and convert it into a cartoon-like version automatically. The algorithm follows a logical sequence:

1. Reading and resizing the image for processing.

2. Converting the image to grayscale and applying median blur to reduce noise.

3. Detecting strong edges using adaptive thresholding.

4. Applying a bilateral filter to smooth color transitions while preserving edges.

5. Combining the filtered image with edge masks to create the final cartoon output.

This objective emphasizes algorithm design skills — focusing on accuracy, efficiency, and clarity of the final result. It encourages students to think about optimization and visual quality simultaneously.

3. To enhance the clarity and brightness of the final cartoon image

After generating the initial cartoon output, further refinement is achieved through **contrast enhancement and brightness adjustments**. These post-processing steps make the cartoon appear more vivid, colorful, and clear.
The goal is to ensure that the output image retains a professional and aesthetically pleasing look while maintaining a balance between simplicity and realism.
By experimenting with scaling factors (alpha) and brightness offsets (beta), students learn how fine-tuning parameters can impact the overall visual quality of an image.

4. To demonstrate the practical applications of computer vision in creative and entertainment domains

This project showcases how **computer vision**, a traditionally technical field, can be applied creatively in various industries such as:

- **Photography and graphic design**, for generating artistic effects.
- **Social media filters**, where real-time image cartooning can enhance user engagement.
- **Animation and gaming**, where cartoon-like textures can be used in 2D and 3D modeling.
- **Educational tools**, where image stylization helps in creating appealing learning materials.

The objective is to highlight the interdisciplinary potential of computer vision — merging technology, creativity, and visual art.

5. To develop a cost-effective, efficient, and user-friendly system

A major objective is to build a **lightweight and accessible solution** that can run on any standard computer without the need for expensive software or specialized hardware.
By using open-source tools such as **Python and OpenCV**, the system remains both cost-effective and scalable.
The project is designed to be simple enough for beginners yet functional enough to demonstrate real-world applications of image processing.

6. To provide an educational platform for practical learning

Beyond producing creative output, the project serves as an **educational tool** that allows students to integrate theoretical concepts with hands-on coding.
Through this project, learners gain practical experience in:

- Applying OpenCV functions effectively.
- Understanding how sequential processing steps interact.
- Analyzing how algorithmic changes affect visual outcomes.

This approach strengthens problem-solving skills and prepares students for more advanced topics in machine learning, artificial intelligence, and digital image analysis.

2. FEASIBILITY STUDY AND LITERATURE SURVEY

Feasibility Study

A feasibility study is conducted to evaluate the practicality, efficiency, and overall viability of implementing a project. It ensures that the proposed system can be developed within the available resources, time, and technical constraints. The **Cartooning an Image using OpenCV in Python** project has been analyzed under four major feasibility aspects: technical, economic, operational, and time feasibility.

Technical Feasibility

The project is **technically feasible** because it utilizes **Python** as the programming language and **OpenCV (Open Source Computer Vision Library)** for image processing operations. Both are **open-source, cross-platform, and well-supported** by the developer community, making them ideal for academic and research purposes.

Python's simplicity allows developers and students to write clear, readable code without complex syntax. Its large ecosystem of libraries—such as **NumPy** for numerical operations, **Matplotlib** for visualization, and **OpenCV** for computer vision—provides all the essential tools for implementing the cartooning algorithm.

OpenCV is optimized for real-time computer vision applications and includes prebuilt functions for:

- Image reading and resizing
- Color conversion
- Blurring and filtering
- Edge detection
- Thresholding and bitwise operations

These features enable smooth and efficient implementation of the cartooning algorithm on any **standard personal computer**. The system does not require advanced hardware like GPUs or high-end servers; a basic system with moderate specifications (4GB RAM or higher) is sufficient to process images effectively.

Thus, the technical setup required for this project is minimal, and the development process can be completed using freely available tools such as **VS Code**, **Jupyter Notebook**, or **Google Colab**.

Economic Feasibility

The project is **highly economical** because all the technologies used are **open-source and free of cost**. There are no expenses related to software licenses, dataset acquisition, or cloud resources. The project relies solely on:

- Python (open-source)
- OpenCV library (open-source)
- Standard computer hardware

Additionally, since the project can be executed on existing personal computers without any need for specialized devices, it significantly reduces implementation costs. The only investment required is time and basic technical knowledge of Python programming.

Compared to traditional commercial tools like Photoshop or Illustrator, which require paid licenses and skilled designers, this automated approach provides a **cost-efficient alternative** for generating cartoon images quickly and effectively.

Operational Feasibility

The **operational feasibility** of this project is very high because the system is simple, user-friendly, and easy to run. Once the user specifies the input image path, the program executes a sequence of predefined steps automatically—such as grayscale conversion, blurring, edge detection, and color smoothing—without requiring manual intervention.

The system displays both the **original image** and the **cartooned output**, allowing users to visually compare results. The process is intuitive, requiring minimal technical expertise. Even users with basic knowledge of Python can execute the program successfully.

Furthermore, the algorithm can be modified or extended to include additional functionalities like:

- Real-time cartooning using webcam input.
- Adjustment of cartooning intensity using slider controls.
- Saving outputs in various formats (JPEG, PNG, etc.).

The project's simplicity and scalability make it suitable for educational purposes, small-scale applications, and even creative digital platforms where automated stylization is desired.

Time Feasibility

The **time feasibility** of the project is excellent. The development process, including code design, testing, and validation, can be completed within a few days to weeks, depending on the complexity of added features.

The cartooning algorithm itself is computationally lightweight and can process images within a few seconds. Since OpenCV functions are optimized in C/C++, they execute rapidly even for high-resolution images. This makes the project efficient and suitable for:

- Real-time applications (e.g., webcam cartooning).
- Batch image processing (multiple images at once).

In terms of learning objectives, students can complete and test the project comfortably within the duration of a mini-project course or laboratory session, making it an ideal choice for academic assignments.

Literature Survey

The process of **image stylization and cartooning** has been widely studied in the fields of **computer vision, digital art, and non-photorealistic rendering (NPR)**. Several research papers and implementations highlight various approaches:

1. **Traditional Techniques:**
   Early methods for image cartooning primarily focused on **edge detection** (e.g., Sobel or Canny operators) and **color quantization**. These techniques simplified the image by reducing the number of color shades and emphasizing prominent outlines.

2. **Enhanced Filtering Techniques:**
   With advancements in image processing, methods combining **bilateral filtering** (for color smoothing) and **adaptive thresholding** (for edge extraction) became popular. These approaches produce smooth and artistic outputs that better resemble cartoons or paintings.

3. **Deep Learning Approaches:**
   In recent years, researchers have experimented with **Convolutional Neural Networks (CNNs)** and **Generative Adversarial Networks (GANs)** for image stylization and **neural style transfer**. Although these techniques produce highly realistic results, they require large datasets, powerful hardware (GPUs), and long training times, which make them less feasible for small-scale or educational projects.

This project adopts a **classical computer vision approach** using OpenCV, which provides a balance between simplicity, efficiency, and output quality. It demonstrates how traditional image processing algorithms can still achieve excellent artistic results without the complexity of modern deep learning frameworks.

3. METHODOLOGY

The methodology adopted in this project consists of a **sequential series of image processing operations** designed to transform a real-life photograph into a stylized cartoon image. Each step performs a specific transformation on the input image, and the cumulative effect of these operations produces a simplified, smooth, and visually appealing cartoon output.

The major stages of the algorithm include **image acquisition, grayscale conversion, noise reduction, edge detection, color smoothing**, and **final image reconstruction** by combining the processed layers. The following subsections describe these modules in detail.

3.1 Modules

1. Image Acquisition

The first step in any image-processing pipeline is **image acquisition**. In this project, the image is read from the system using the cv2.imread() function of the **OpenCV** library.

Once loaded, the image is **resized** to a fixed dimension (for example, 800 × 800 pixels) using cv2.resize() to ensure uniform processing irrespective of the original image size.

Resizing not only standardizes the dataset but also reduces computational load during filtering and thresholding operations. This step ensures that subsequent algorithms perform consistently for all input images, whether small or high-resolution.

```
img = cv2.imread("your_image.jpg")

img = cv2.resize(img, (800, 800))
```

---

2. Grayscale Conversion

After acquisition, the color image (in RGB/BGR format) is converted into **grayscale** using cv2.cvtColor() with the color conversion code cv2.COLOR_BGR2GRAY.

This transformation simplifies the image by eliminating color information and retaining only intensity variations. Grayscale images are easier and faster to process since they contain a single channel instead of three (Red, Green, Blue).

The grayscale representation helps focus on essential structural information such as edges, contours, and shading—elements critical for cartoon effect generation.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

---

3. Noise Reduction

Real-world images often contain unwanted variations called **noise** due to lighting, shadows, or sensor imperfections. Noise can interfere with edge detection and produce irregular boundaries.

To minimize this, the **median blur filter** is applied using cv2.medianBlur(). Unlike simple averaging filters, the median blur replaces each pixel value with the **median of neighboring pixel intensities**, effectively reducing salt-and-pepper noise while preserving edge sharpness.

The kernel size (commonly 5 or 7) determines the strength of smoothing. This filtering step enhances the clarity of the image and prepares it for more accurate edge detection.

```
gray_blur = cv2.medianBlur(gray, 7)
```

---

4. Edge Detection

Edges are vital to achieving the **cartoon outline effect**. They define object boundaries and create the visual impression of drawing lines.

In this project, edges are extracted using **Adaptive Thresholding**, which dynamically adjusts the threshold value for smaller regions of the image, thereby handling variations in illumination.

OpenCV's cv2.adaptiveThreshold() function is used to convert the grayscale image into a binary edge mask, where white pixels represent detected edges and black pixels represent smooth regions.

This method ensures that even fine details are captured, producing bold and consistent outlines for the cartoon appearance.

```
edges = cv2.adaptiveThreshold(
    gray_blur, 255,
    cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY, 9, 9
)
```

---

5. Color Smoothing (Bilateral Filter)

To give the image a **painted or smooth texture**, the **Bilateral Filter** is applied to the original color image.

The bilateral filter is unique because, unlike regular blurring filters, it **preserves edges** while reducing color variation within homogeneous regions. It achieves this by considering both **spatial proximity** and **intensity similarity** when averaging pixel values.

As a result, flat regions of color (like skin tones or backgrounds) become smooth and uniform, while edges remain sharp and well-defined—characteristics typical of cartoon artwork.

color = cv2.bilateralFilter(img, d=9, sigmaColor=250, sigmaSpace=250)

---

6. Combining Results

Once edges and color-smoothed images are prepared, they are **merged** to produce the final cartoon effect.

First, the binary edge mask is converted into a 3-channel image using cv2.cvtColor() so it can be combined with the color image. Then, a **bitwise AND operation** (cv2.bitwise_and()) is applied to overlay the detected edges on the filtered color image.

This operation ensures that only the color regions within the detected boundaries are retained, giving the final image a comic-like outline and vibrant colors.

edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

cartoon = cv2.bitwise_and(color, edges_colored)

---

7. Display and Save Output

Finally, the cartooned image is displayed using cv2.imshow() along with the original image for comparison.

To enhance visual clarity, brightness and contrast can be adjusted using cv2.convertScaleAbs() with parameters **alpha** (contrast) and **beta** (brightness). The resulting image can then be **saved** to the system using cv2.imwrite() for future use.

These steps make the system interactive, allowing users to visualize the transformation in real time and preserve their results.

cartoon = cv2.convertScaleAbs(cartoon, alpha=1.2, beta=15)

cv2.imshow("Original Image", img)

cv2.imshow("Cartoon Image", cartoon)

cv2.imwrite("cartoon_output.jpg", cartoon)

cv2.waitKey(0)

cv2.destroyAllWindows()

3.2. Sample Code

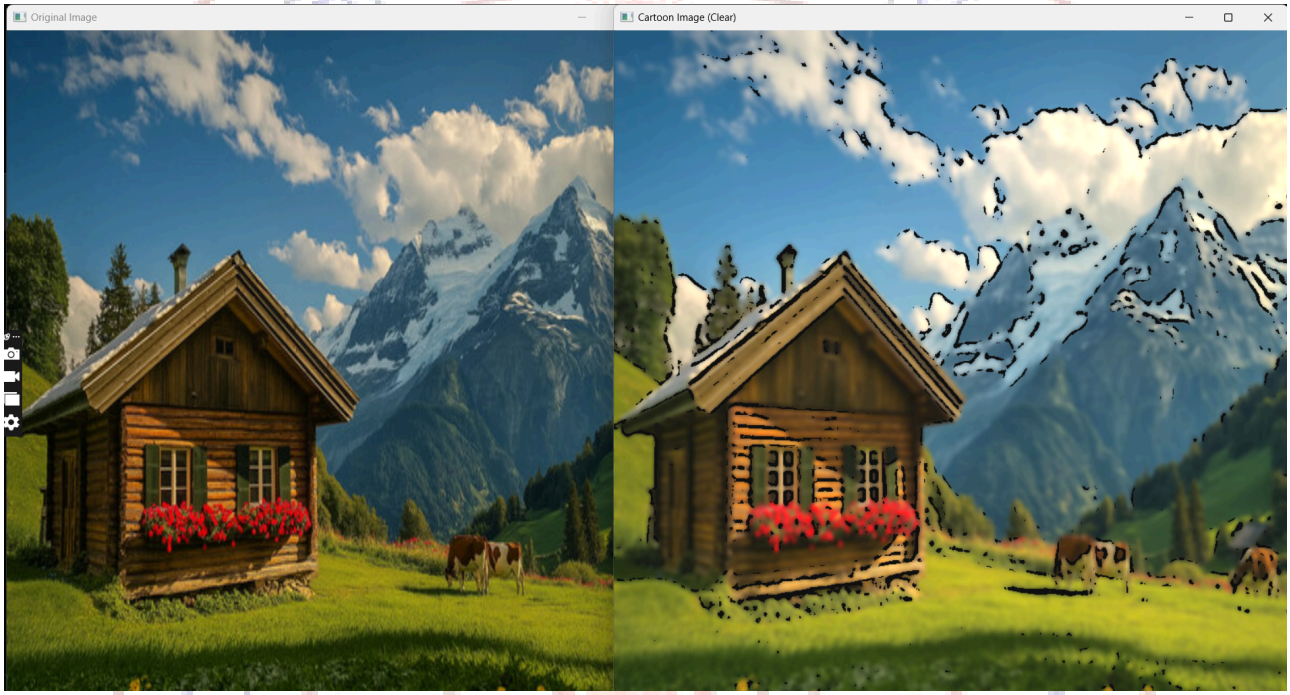
```
# Import required libraries

import cv2

import numpy as np



# ---------------------------------------------------------

# Step 1: Read the Image

# ---------------------------------------------------------

img = cv2.imread("Savithri.jpg.jpeg")



# Resize the image for uniform processing

img = cv2.resize(img, (800, 800))



# ---------------------------------------------------------

# Step 2: Convert to Grayscale

# ---------------------------------------------------------

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

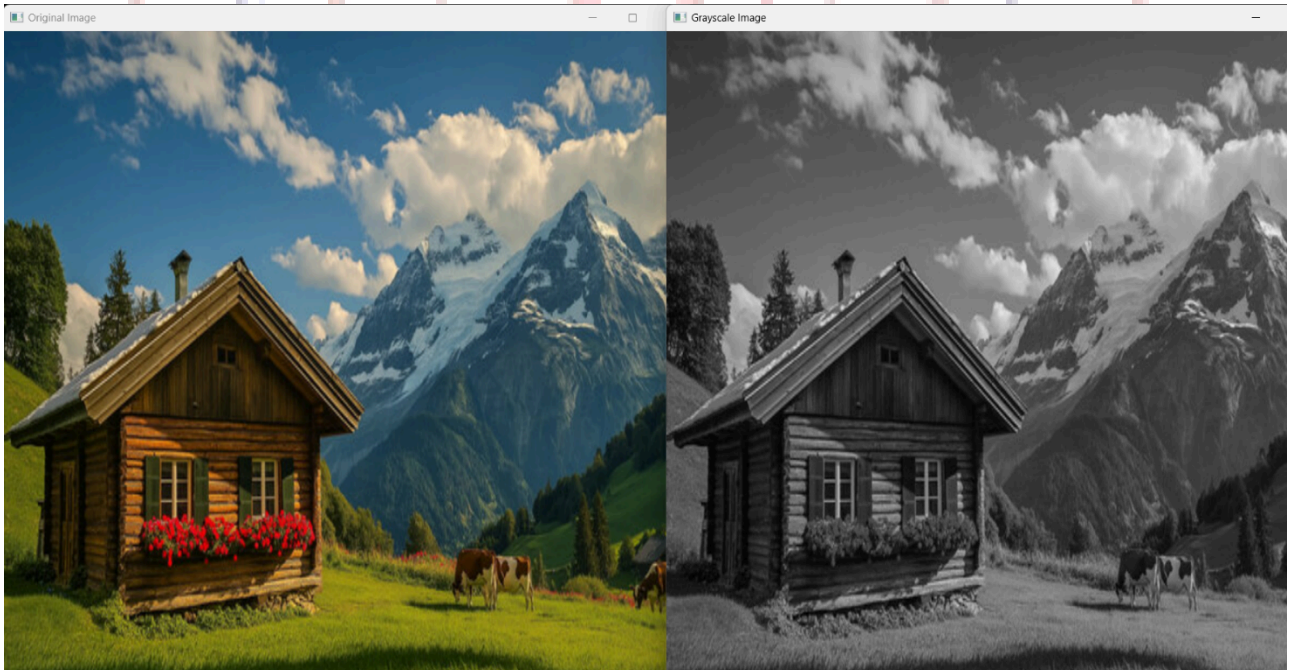
```
# ---------------------------------------------------------

# Step 3: Apply Median Blur to Reduce Noise

# ---------------------------------------------------------

gray_blur = cv2.medianBlur(gray, 7)



# ---------------------------------------------------------

# Step 4: Detect Edges (Refined)

# ---------------------------------------------------------

edges = cv2.adaptiveThreshold(gray_blur, 255,

                              cv2.ADAPTIVE_THRESH_MEAN_C,

                              cv2.THRESH_BINARY,

                              9, 9)



# ---------------------------------------------------------

# Step 5: Smoothen Colors using Bilateral Filter

# ---------------------------------------------------------

# Increase diameter and sigma values for smoother colors

color = cv2.bilateralFilter(img, d=9, sigmaColor=250, sigmaSpace=250)



# ---------------------------------------------------------

# Step 6: Combine Edges with the Smoothed Color Image

# ---------------------------------------------------------

# Convert edges to color for blending
```

```
edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)



# Blend the smoothed color image and edges for clarity

cartoon = cv2.bitwise_and(color, edges_colored)



# ------------------------------------------------------------

# Step 7: Enhance Brightness (Optional)

# ------------------------------------------------------------

# This step makes the cartoon appear more vivid

cartoon = cv2.convertScaleAbs(cartoon, alpha=1.2, beta=15)



# ------------------------------------------------------------

# Step 8: Display All Results

# ------------------------------------------------------------

cv2.imshow("Grayscale Image", gray)

cv2.imshow("Edge Mask", edges)

cv2.imshow("Cartoon Image", cartoon)



# Wait for key press and close all windows

cv2.waitKey(0)

cv2.destroyAllWindows()



# Optional: Save the cartoon image
```

```
cv2.imwrite("cartoon_output_clear.jpg", cartoon)

print("Cartoon image saved successfully as 'cartoon_output_clear.jpg'")
```

3.3. OUTPUT SCREENSHOTS





3.4.Tools & Platforms used

The development and implementation of the project **"Cartooning an Image using OpenCV in Python"** were carried out using a combination of open-source tools and software platforms. These tools provided an efficient and flexible environment for performing image processing operations, debugging, testing, and visualization.
The following table and descriptions summarize the tools, libraries, and platforms used in the project:

Tool / Library	Purpose and Description
Python	Python is the primary programming language used for implementing the project. It is a high-level, general-purpose language known for its simplicity, readability, and extensive library support. Python provides built-in packages for image processing, numerical computation, and visualization. Its large community and cross-platform compatibility make it ideal for academic and research-based projects.
OpenCV (cv2)	OpenCV (Open Source Computer Vision Library) is the core library used for performing image processing and computer vision operations. It provides a wide range of functions such as image reading, color conversion, edge detection, thresholding, blurring, and morphological transformations. In this project, OpenCV was used for all the major stages of the cartooning process—such as grayscale conversion, median blurring, bilateral filtering, adaptive thresholding, and combining layers to create the final cartoon image. Its optimized algorithms ensure real-time performance even on standard hardware.
NumPy	NumPy (Numerical Python) is used for performing mathematical and matrix operations efficiently. Since images are represented as multi-dimensional arrays (matrices of pixels), NumPy plays a vital role in handling pixel manipulation, numerical computation, and array-based data operations. It also supports interoperability with OpenCV, allowing fast and smooth execution of mathematical transformations required in image processing.
Matplotlib (optional)	Matplotlib is a Python library used for visualizing data and displaying images. Although OpenCV provides its own functions (cv2.imshow()), Matplotlib can be used for comparing images side by side, plotting histograms, or displaying intermediate results during processing. It is particularly helpful in analyzing how each step of the algorithm (such as filtering or thresholding) affects the output.
VS Code / Jupyter Notebook	Visual Studio Code (VS Code) and Jupyter Notebook were used as **Integrated Development Environments (IDEs)** for writing, testing, and executing the code. VS Code offers a lightweight and fast interface with Python extensions, making it ideal for developing structured Python programs. Jupyter Notebook provides an interactive coding environment that allows running code cells individually, visualizing outputs instantly, and documenting the process in an educational format. Both platforms enhance productivity and ease of debugging.

Additional Supporting Tools
In addition to the primary libraries, the following tools and resources contributed to the development process:

- **Operating System:** The project can be executed on Windows, macOS, or Linux systems without modification.
- **Command-Line Interface (CLI):** Used to install Python libraries (pip install opencv-python numpy matplotlib).
- **File System Access:** Used to load input images and save output results using OpenCV's I/O functions (cv2.imread() and cv2.imwrite()).

4. CONCLUSION AND FUTURE SCOPE

Conclusion

The project titled **"Cartooning an Image using OpenCV in Python"** demonstrates the successful implementation of a computer vision-based image stylization system that converts ordinary photographs into **cartoon-like representations**. Using a combination of **grayscale conversion, median blurring, edge detection, bilateral filtering, and image masking**, the algorithm produces outputs that closely resemble hand-drawn or comic-style artwork.

The approach relies on traditional **image processing techniques** rather than complex deep-learning models, making it **lightweight, fast, and efficient**. The use of OpenCV has significantly simplified the implementation process, as it provides optimized functions for performing operations like filtering, color space transformation, and thresholding with minimal computational overhead.

Through this project, the core concepts of **image enhancement, noise reduction, and feature extraction** have been practically demonstrated. The generated cartoon images exhibit **smooth color regions, clear edges, and minimal noise**, resulting in aesthetically pleasing visuals suitable for creative, educational, and entertainment purposes.

From an academic perspective, the project serves as a strong foundation for understanding fundamental **image processing pipelines** and how multiple transformations can be combined to achieve artistic outcomes. Moreover, it highlights the interdisciplinary nature of computer vision—merging technical computation with creative expression.

Overall, the project achieves its objective of developing a **simple, cost-effective, and user-friendly cartooning system** using OpenCV and Python. It can be easily extended or integrated into larger multimedia applications and serves as an excellent learning model for students interested in computer vision and digital art.

Future Scope

While the current system efficiently cartoonizes still images, it offers several possibilities for **enhancement and expansion**. The following points outline potential directions for future development:

1. **Real-Time Cartooning:**
   Extend the algorithm to process live video streams from a webcam, allowing users to see themselves in cartoon form in real time. This can be particularly useful for social media filters, live streaming, or creative video content generation.
2. **Graphical User Interface (GUI) Integration:**
   Develop a **user-friendly GUI** using frameworks like **Tkinter, PyQt, or Streamlit**, enabling users to upload images, adjust cartooning parameters (like filter intensity or edge thickness), and preview results without writing code.
3. **Advanced Stylization using Deep Learning:**
   Incorporate **neural style transfer** or **GAN-based models** to achieve more artistic and diverse stylization effects. Deep learning approaches can produce results resembling painting styles, sketches, or abstract art, enhancing creativity and realism.

4. **Mobile Application Development:**
   Implement the algorithm as a **mobile application** for Android or iOS platforms using libraries like **Kivy** or integrating Python with **Flutter** or **React Native** backends. This would make the cartooning effect instantly accessible to users on their smartphones.
5. **Cloud Integration and Web Deployment:**
   Host the cartooning service on cloud platforms such as **AWS**, **Azure**, or **Google Cloud** and deploy it as a **web application**. Users could upload photos online and receive cartoonized outputs instantly.
6. **Customization Options:**
   Allow users to select different cartoon styles (e.g., pencil sketch, watercolor, anime-style) and provide adjustable sliders for brightness, contrast, edge intensity, and color depth to personalize results.

5. REFERENCES

1. **OpenCV Documentation (2024).** ***Open Source Computer Vision Library.*** **[Online]. Available: https://docs.opencv.org**

2. **NumPy Developers (2024).** ***NumPy: Numerical Python Library.*** **[Online]. Available: https://numpy.org**

3. **Matplotlib Developers (2024).** ***Matplotlib: Visualization with Python.*** **[Online]. Available: https://matplotlib.org**

4. **Python Software Foundation (2024).** ***Python Programming Language.*** **[Online]. Available: https://www.python.org**

5. **GeeksforGeeks (2025).** ***Cartooning an Image using OpenCV in Python.*** **[Online]. Available: https://www.geeksforgeeks.org**

6. **TutorialsPoint (2024).** ***Image Processing with OpenCV.*** **[Online]. Available: https://www.tutorialspoint.com/opencv**

7. **Towards Data Science (2024).** ***Creating Cartoon Effects with OpenCV.*** **[Online]. Available: https://towardsdatascience.com**

8. **Stack Overflow (2024).** ***Discussions on Image Filtering and Adaptive Thresholding in OpenCV.*** **[Online]. Available: https://stackoverflow.com**

9. **ResearchGate (2023).** ***Non-Photorealistic Rendering and Image Stylization Techniques.*** **[Online]. Available: https://www.researchgate.net**

10. **Wikipedia (2024).** ***Edge Detection and Bilateral Filtering.*** **[Online]. Available: https://en.wikipedia.org/wiki**

11. **Senthil Kumar, S., & Kannan, E. (2020).** ***A Review on Image Filtering Techniques for Edge Preservation. International Journal of Computer Applications.***

12. **Gonzalez, R. C., & Woods, R. E. (2018).** ***Digital Image Processing.*** **Pearson Education.**

13. **Bradski, G. (2000).** ***The OpenCV Library. Dr. Dobb's Journal of Software Tools.***

14. **Jain, A. K. (2019).** ***Fundamentals of Digital Image Processing.*** **Prentice-Hall of India.**

15. **Patel, K., & Sharma, P. (2021).** ***Implementation of Image Cartoonization Using OpenCV. International Journal of Scientific Research in Engineering and Management (IJSREM).***

16. **Analytics Vidhya (2024).** ***A Complete Guide to Image Processing using OpenCV and Python.*** **[Online]. Available: https://www.analyticsvidhya.com**

17. **Medium (2024).** ***Fun with Image Cartoonization in OpenCV.*** **[Online]. Available: https://medium.com**

18. **TechVidvan (2023).** ***Cartoonify Image using OpenCV in Python.*** **[Online]. Available: https://techvidvan.com/tutorials/cartoonify-image-opencv**

19. **Kaggle (2024).** ***Computer Vision Projects and Datasets.*** **[Online]. Available: https://www.kaggle.com**

20. **YouTube (2024).** ***Cartoon Image Effect using OpenCV Python (Educational Tutorial).*** **[Online]. Available: https://www.youtube.com**