MARRI LAXMAN REDDY GROUP OF INSTITUTIONS

MLR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE

Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

NBA NATIONAL BOARD OF ACCREDITATION

NAAC

A

REAL-TIME/FIELD BASED RESEARCH PROJECT

Report

On

PICTOON -USING OPENCV

Submitted in partial fulfillment of the

Requirements for the award of the degree of

Bachelor of Technology

In

COMPUTER SCIENCE AND ENGINEERING-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

By

R. LAWRENCE -22R21A66B2

Under the guidance of

Mrs.N. JAYASREE

Assistant Professor

Department of

COMPUTER SCIENCE AND ENGINEERING-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

2022-2026

MARRI LAXMAN REDDY GROUP OF INSTITUTIONS

MLR INSTITUTE OF TECHNOLOGY

(UGC AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE

Laxman Reddy Avenue, Dundigal, Hyderabad-500 043, Telangana, India

NBA NATIONAL BOARD OF ACCREDITATION

NAAC

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

CERTIFICATE

This is to certify that the project entitled **"**PICTOON-using OpenCV**"** has been submitted by **R. LAWRENCE- 22R21A66B2** in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering-Artificial Intelligence & Machine Learning from Jawaharlal Nehru Technological University, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Internal Guide **Project-Coordinator**

Head of the Department

COMPUTER SCIENCE AND ENGINEERING-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DECLARATION

We hereby declare that the project entitled **"**PICTOON- using Open CV**"** is the work done during the period from **Feb 2024 to July 2024** and is submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in **Computer Science and Engineering-Artificial Intelligence & Machine Learning** from Jawaharlal Nehru Technology University, Hyderabad. The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

R. LAWRENCE- 22R21A66B2

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we now have the opportunity to express our guidance for all of them. First of all, we would like to express our deep gratitude towards our internal guide**, Mrs.N.Jayasree, Assistant Professor,Computer Science and Engineering- Artificial Intelligence & Machine Learning** for support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. K. Sai Prasad, HOD, Department of Computer Science and Engineering-Artificial Intelligence & Machine Learning** for providing the facilities to complete the dissertation.We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

R. LAWRENCE- 22R21A66B2

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

ABSTRACT

Machine Learning with traditional art forms like Cartoonization will continue to thrive, permeating various domains. Inspired by the success of Instagram and the rising demand for user-friendly photo effects apps, we embark on a project aimed at Cartoonizing digital images using Open CV. Our innovative approach distinguishes itself from previous methods by employing a layered technique, meticulously crafted to deliver a balanced output reminiscent of classic cartoons. Throughout the project, we explore a range of techniques, carefully orchestrated to harmonize and enhance the Cartoonization process. Our solution not only empowers users to capture images but also enables sophisticated processing, ensuring an engaging and seamless experience. By seamlessly blending artistry with technological advancements, our project redefines the boundaries of Cartoonization, offering a glimpse into the future of image processing applications. It is capable of producing exactly the effect specified above, and a wide range of input images will yield satisfactory results.

LIST OF FIGURES & TABLES

INDEX

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Our project focuses on Cartoonizing digital images using OpenCV, inspired by the popularity of Instagram and the increasing demand for user-friendly photo effects apps. Unlike previous methods, our innovative approach uses a layered technique to create balanced outputs reminiscent of classic cartoons.

We employ a range of techniques that harmonize to enhance the Cartoonization process, allowing users to capture images and perform sophisticated processing for an engaging and seamless experience. By blending artistry with technological advancements, our project redefines the boundaries of Cartoonization, showcasing the future of image processing applications and ensuring satisfactory results across a wide range of input images.

1.2 PURPOSE OF THE PROJECT

The purpose of implementing a **PICTOON**-using Open CV encompasses various objectives and potential benefits. Here are some common purposes:

1. **Operational Efficiency:**

   Convert images of different formats i.e. jpg, JPEG into cartoon style selected filter.

2. **User Experience:**

   Personalize interfaces and services based on user preferences.

3. **Data Security and Privacy:**

   Ensure responsible and secure handling of facial data.

1.3 MOTIVATION

Our aim was to address a recent incident where individuals from the dark web took profile pictures from Instagram and other social media platforms, manipulated them, and used the altered images for malicious purposes which caused many people catastrophic results.

CHAPTER 2

LITERATURE SURVEY

We conducted a thorough literature survey by reviewing existing systems for Cartoonization of images using OpenCV . Research papers, journals and publications have also been referred in order to prepare this survey.

2.1 EXISTING SYSTEM

Prisma

Prisma is a widely known mobile application that leverages neural networks and AI to apply artistic effects to photos, including various cartoon styles. Its primary features include a broad selection of artistic filters, real-time processing, and easy sharing options. The technology behind Prisma relies on deep learning algorithms and cloud-based processing, allowing for sophisticated transformations. However, the requirement for an internet connection for processing and limited customization options are notable limitations.

ToonMe

ToonMe is another mobile app that transforms photos into cartoon and vector art styles using AI-powered techniques. It offers a variety of cartoon styles and an intuitive user interface. While it provides a fun and easy way to cartoonify images, some features require in-app purchases, and the quality of the output can sometimes be inconsistent.

Cartoon Photo Editor

Cartoon Photo Editor is designed specifically to apply cartoon effects to photos on mobile devices. It features multiple cartoon and sketch filters within an easy-to-use interface. This app relies on straightforward image processing algorithms suitable for quick edits. However, it is limited to predefined filters and lacks advanced customization options that users might desire for more unique outputs.

BeFunky

BeFunky is an online photo editor that includes a range of artistic effects, including cartoon filters. As a web-based tool, it provides browser-based editing capabilities, various filters and effects, and a collage maker. While convenient for quick edits, some features are behind a paywall, and performance can depend heavily on internet speed and browser capabilities.

DeepArt.io

DeepArt.io is a web application that uses neural networks to transform photos into artistic pieces, including cartoon styles. It offers high-quality artistic transformations and allows users to upload their own styles for unique effects. This service relies on deep neural networks and cloud computing, which means it requires an internet connection, and processing times can be lengthy, especially for high-resolution images.

Adobe Photoshop

Adobe Photoshop is the gold standard in professional photo editing software, offering a comprehensive range of filters and effects, including cartoonification. It provides advanced editing tools, customizable filters, and high-quality output, suitable for professional use. However, its subscription model can be expensive, and it has a steep learning curve, making it less accessible for casual users.

2.2 LIMITATIONS OF EXISTING SYSTEM

Processing Time: High-quality cartoonification requires significant computational power, leading to longer processing times, especially for high-resolution images.

Device Limitations: Performance may vary depending on the device's hardware capabilities, with lower-end devices struggling to produce high-quality results. There are such applications like a limited no.of images are only cartoonified by a single user or for a single day.

Consistency and Accuracy: The quality and style of cartoonification can vary widely between images, even within the same application, leading to inconsistent results.

Ease of Use: While many tools are designed for general users, achieving professional-quality cartoonification often requires some level of expertise and familiarity with image editing.

Use Case Limitations:

Not Suitable for All Images: Some images, especially those with complex backgrounds or low contrast, do not convert well to a cartoon style.

Professional Use: The cartoonified images may not meet the standards required for professional or commercial use, limiting their applicability. Due to shortage of different effects there are few issues to convert the image into required or necessary output.

Privacy and Security:

Data Privacy & Security Risks : Uploading images to online services for cartoonification can raise privacy concerns, as users need to trust that their images are not misused or stored without consent. Online applications might expose users to security risks, such as data breaches or malicious software.

Cultural Representation : Cartoon styles may not accurately or respectfully represent people from diverse cultural backgrounds, leading to potential issues of misrepresentation or insensitivity.

Lack of Versatility: Many tools are limited to a few cartoon styles and cannot replicate a wide range of artistic styles, such as different cartoon genres or hand-drawn aesthetics. Many applications can only provide only few effects for the images.

CHAPTER 3

PROPOSED SYSTEM

3.1 PROPOSED SYSTEM

The proposed system is a user-friendly web application developed with Streamlit. It enables users to convert images into cartoonized versions and perform various image manipulations. Upon launching, users can either upload their own images or select from predefined datasets, including categories like Pets, Flowers, Actors, Cartoons, Cars, and Buildings. The app offers interactive features for resizing images to specified dimensions and applying various filters with adjustable intensity. Users can view both the original and processed images side-by-side for easy comparison. After achieving the desired transformation, users can download the processed image directly to their device. The app's design emphasizes simplicity and ease of use, making it accessible for users with different technical skills. Combining a clean interface with powerful functionality, the system is a versatile tool for image manipulation and creative expression.

3.2 OBJECTIVES OF PROPOSED SYSTEM

The objectives of the proposed system include the following:

Advanced Edge Detection and Stylization : To accurately identify and stylize edges, mimicking the look of hand-drawn lines and enhancing the overall cartoon effect**.**

Deep Learning and Neural Networks : To leverage advanced machine learning techniques for producing high-quality, stylized cartoon images that can adapt to various input types and styles.

Customizable Filters and Parameters : To provide users with the flexibility to adjust and fine-tune the cartoonification process, allowing for personalized and diverse artistic results.

Multi-Style and Genre Option : To offer a wide range of cartoon styles and genres, enabling users to choose and apply their preferred artistic aesthetic easily.

High-Resolution Output and Quality Enhancement : To ensure that the cartoonified images are of high resolution and quality, preserving essential details and textures for a polished look.

Privacy and Security Features : To safeguard user privacy and data security by offering local processing options and secure handling of images during cloud-based processing.

Accessibility and Ease of Use : To create an intuitive and accessible interface that caters to users of all skill levels, supplemented with helpful resources and guidance for effective use

.3.3 SYSTEM REQUIREMENTS

Here are the requirements for developing and deploying the application.

3.3.1 SOFTWARE REQUIREMENTS

Below are the software requirements for the application development:

1. The required language is python
2. Editor VSCode
3. OpenCV, Numpy, OS, IO Libraries for Model Building
4. StreamLit Platform.
5. Data sets.

3.3.2 HARDWARE REQUIREMENTS

Below are the hardware requirements for the application development:

1. Operating System : Windows
2. Processor : intel i3(min)
3. Ram : 4 GB(min)
4. Hard Disk : 250GB(min)(Not mandatory)

3.3.3 FUNCTIONAL REQUIREMENTS

Here are the functional requirements for a cartoonification of image application:

1. **Advanced Edge Detection and Stylization**

   The system shall detect edges in images using algorithms such as Canny, Sobel, and deep learning-based detectors. The system shall allow users to adjust edge thickness, smoothness, and intensity.

2. **Deep Learning and Neural Networks**

   The system shall use Convolutional Neural Networks (CNNs) to extract features from images. The system shall employ Generative Adversarial Networks (GANs) to generate stylized cartoon images. The system shall provide different pre-trained models for various cartoon styles.

3. **Customizable Filters and Parameters**

   The system shall offer filters for edge detection, color simplification, and texture application. The system shall allow users to adjust parameters such as saturation, contrast, brightness, and color palette through sliders or input fields.

4. **Multi-Style and Genre Options**

   The system shall include neural style transfer options for various cartoon genres (e.g., anime, Western cartoons, comic book styles). The system shall provide a library of predefined cartoon styles and templates for users to choose from.

5. **Interactive and Real-Time Processing**

   The system shall support real-time processing for instant preview of the cartoonification effect. The system shall include interactive editing tools that allow users to make adjustments and corrections on- the-fly.

6. **High-Resolution Output and Quality Enhancement**

   The system shall use super-resolution techniques to enhance the quality of low-resolution cartoonified images. The system shall apply detail preservation methods to maintain essential details and textures.

7. **Privacy and Security Features**

   The system shall offer local processing on the user's device to ensure privacy. For cloud-based processing, the system shall use secure, encrypted upload and processing protocols.

8. **Accessibility and Ease of Use**

   The system shall have a user-friendly interface that is easy to navigate for users of all skill levels. The system shall provide tutorials, tooltips, and guidance to help users understand and utilize the features effectively. The system shall support multiple languages to cater to a diverse user base.

General Functional Requirements

- The system shall support common image formats such as JPEG, PNG, and BMP for input and output.
- The system shall allow users to save and export the cartoonified images in high resolution.
- The system shall provide undo and redo functionality for user actions.
- The system shall allow users to compare the original image with the cartoonified version side-by-side.

3.3.4 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements specify the quality attributes and constraints of a system, defining how well the system performs its functions rather than what functions it performs. Here are some non- functional requirements for a image cartoonification :

1. Performance Requirements

The system shall process images and apply cartoonification effects within a maximum of 5 seconds for standard resolution images (e.g., 1920x1080). The system shall support real-time preview with minimal lag (less than 1 second delay) for changes in parameters.

2. Scalability Requirements

The system shall be able to handle multiple concurrent users without significant degradation in performance. The system shall efficiently scale to process high-resolution images (e.g., 4K) with acceptable performance.

3. Reliability Requirements

The system shall be available 99.9% of the time, ensuring minimal downtime. The system shall have robust error handling to manage and recover from failures gracefully.

4. Usability Requirements

The user interface shall be intuitive and easy to navigate for users of all experience levels. The system shall provide clear instructions and feedback for user actions.

5. Security Requirements

The system shall encrypt all user data during transmission and storage to protect privacy. The system shall implement secure authentication mechanisms for user access.

6. Compatibility Requirements

The system shall be compatible with major operating systems, including Windows, macOS, Android, and iOS. The system shall support major web browsers such as Chrome, Firefox, Safari, and Edge.

7. Maintainability Requirements

The system shall be designed with modular architecture to facilitate easy maintenance and updates. The system shall include comprehensive documentation for developers and maintainers.

8. Portability Requirements

The system shall be portable and able to run on various hardware configurations, from desktops to mobile devices. The system shall support deployment on both local devices and cloud platforms.

9. Legal and Compliance Requirements

The system shall comply with relevant data protection regulations (e.g., GDPR, CCPA) to ensure user privacy and data security. The system shall respect intellectual property rights, ensuring that any pre-trained models or datasets used are properly licensed.

10. Accessibility Requirements

The system shall be accessible to users with disabilities, following guidelines such as the Web Content Accessibility Guidelines (WCAG). The system shall provide features like keyboard navigation, screen reader support, and adjustable text size.

11. Localization Requirements

The system shall support multiple languages and regional settings to cater to a global user base. The system shall allow for easy addition of new languages through a modular localization frameworks.

3.4 CONCEPTS USED IN THE PROPOSED SYSTEM

3.5 Concepts Summary

1. User Interface Design with Streamlit

- **Sidebar Navigation**: The sidebar provides options for users to upload images or select from predefined datasets. It also includes controls for re-sizing images and applying filters.
- **Interactive Widgets**: The application uses radio buttons, file up-loaders, number inputs, sliders, and buttons to allow users to interact with the application.
- **Image Display**: Streamlit's st.image function is used to display images within the application.

2. Image Processing

- **Open CV Integration**: The application uses Open CV (cv2) for image processing tasks such as re-sizing and color space conversion.
- **PIL Integration**: The Pillow library (PIL) is used to handle image file uploads and conversions between image formats.
- **Custom Filters**: Filters from a custom module (filters.py) are applied to images. These filters are defined in a dictionary (FILTER_MAP), which maps filter names to their respective functions.

3. Datasets Management

- **Predefined Datasets**: The application includes several predefined datasets (e.g., Pets, Flowers, Actors, Cartoons, Cars, Buildings). Images are loaded from local directories.
- **Dynamic Image Loading**: Images from selected datasets are dynamically loaded and displayed in a grid layout, allowing users to select an image to cartoonify.

4. Filter Application

- **Filter Selection and Intensity Adjustment**: Users can select a filter from a drop-down menu and adjust its intensity using a slider. The selected filter is then applied to the image.
- **Real-Time Feedback**: The application provides real-time feedback by displaying the original and filtered images side-by-side.

5. Image Re-sizing

- **Dynamic Re-sizing**: Users can resize the selected image by specifying new dimensions using number inputs. The re-sized image is then displayed.

6. Download Functionality

- **Image Download**: After applying filters, users can download the filtered image. This is done by converting the image to a BytesIO buffer and using Streamlit's st.download_button.

7. Modular Code Design

- **Filter Functionality Encapsulation**: The apply_filter function abstracts the application of filters, making the code modular and easier to extend.
- **Reusable Functions**: Functions like resize_image and apply_filter encapsulate specific functionality, promoting code reuse and maintainability.

8. Performance Considerations

- **Efficient Image Handling**: Images are processed efficiently using OpenCV and PIL to ensure quick response times.
- **Real-Time Interactivity**: The application is designed to provide immediate feedback on user actions, enhancing the user experience.

3.5 DATA SET USED IN THE PROPOSED SYSTEM

1. **Pets**
    - Path: code/DATASETS/PETS
    - Description: Contains images of various pets.
2. **Flowers**
    - Path: code/DATASETS/FLOWERS AND NATURE
    - Description: Contains images of flowers and nature scenes.
3. **Actors**
    - Path: code/DATASETS/ACTORS
    - Description: Contains images of actors.
4. **Cartoons**
    - Path: code/DATASETS/CARTOONS AND ANIME
    - Description: Contains images of cartoons and anime characters.
5. **Cars**
    - Path: code/DATASETS/CARS
    - Description: Contains images of various cars.

6. **Buildings**
    - Path: code/DATASETS/BUILDINGS
    - Description: Contains images of buildings and architectural structures.

Loading Dataset Images

For each dataset, the code loads images with extensions .png, .jpg, and .jpeg from the respective directories and stores the paths in the DATASET_IMAGES dictionary.

Summary of Dataset Handling in the Code

- **Dataset Definition**: The DATASETS dictionary defines the categories and their corresponding folder paths.
- **Image Loading**: Images are loaded from these directories and stored in the DATASET_IMAGES dictionary.
- **User Selection**: Users can choose a category from the sidebar, and the app displays images from the selected dataset.
- **Display and Selection**: Images from the chosen dataset are displayed in a grid layout, allowing users to select an image for cartoonification.

This setup allows the application to manage and utilize multiple image datasets, providing users with a diverse set of images to work with.

CHAPTER 4

SYSTEM DESIGN

4.1 COMPONENTS OR USERS IN THE PROPOSED SYSTEM

The proposed Cartoonizer App system comprises several key components, each responsible for specific functionalities within the application. Here are the main components:

1. User Interface (UI)

- **Streamlit Interface**: Utilizes Streamlit to create a web-based interactive interface.
- **Sidebar Navigation**: Allows users to upload images, select from predefined datasets, resize images, and apply filters.
- **Image Display Area**: Shows the selected or uploaded image, before and after applying filters, and displays all available filters with download options.

2. Dataset Management

- **Predefined Datasets**: A collection of images categorized into Pets, Flowers, Actors, Cartoons, Cars, and Buildings.
- **Dynamic Image Loading**: Loads images from local directories based on user selection and displays them for user interaction.
- **DATASETS Dictionary**: Defines the paths to the datasets.
- **DATASET_IMAGES Dictionary**: Stores the loaded images for quick access.

3. Image Processing

- **OpenCV Integration**: Used for image processing tasks such as resizing and applying filters.
- **Pillow Integration**: Handles image file uploads and conversions between formats.
- **Image Resizing Function**: Resizes images based on user input dimensions.
- **Filter Application Function**: Applies selected filters to images using a predefined map of filters (FILTER_MAP).

4. Filters

- **FILTER_MAP**: A dictionary mapping filter names to their respective functions.
- **Filter Functions**: Individual functions that apply specific image processing effects.
- **Filter Selection and Intensity Adjustment**: Allows users to choose a filter and adjust its intensity before applying it to the image.

5. File Handling

- **Image Upload**: Allows users to upload images in PNG, JPG, or JPEG formats.
- **Image Download**: Provides functionality for users to download the filtered images.

6. Auxiliary Functions

- **Apply_filter Function**: Applies the selected filter to the image.
- **Resize_Image Function**: Resizes the image based on user-provided dimensions.

7. Display and Interaction

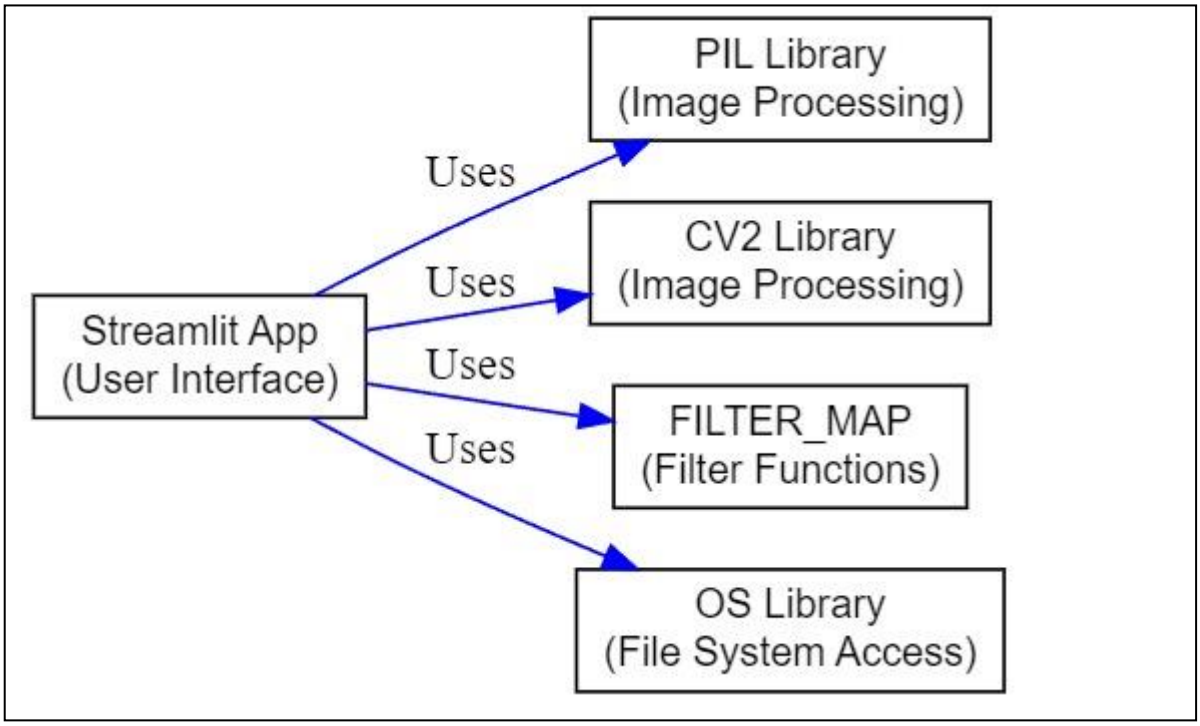
- **Dynamic Layouts**: Uses Streamlit's layout features to display images in a grid format and side-by-side comparisons.

Download Buttons: Allows users to download images after filters are applied.

4.2. PROPOSED SYSTEM ARCHITECTURE

This architecture ensures a modular and scalable design, allowing for easy maintenance and potential future enhancements, such as adding new filters or integrating cloud-based storage solutions.

1. **User Interface**: Users interact with the Streamlit interface to upload images or select from datasets, resize images, and apply filters.
2. **Application Logic**: The UI components trigger functions in the Application Logic Layer to process images.
3. **Data Management**: Handles the loading of dataset images and user-uploaded images, and manages the storage of processed images.
4. **Integration**: Utilizes OpenCV for image processing tasks and Pillow for image file handling.
5. **Storage**: Images are stored locally, either as part of predefined datasets or user-uploaded files.



Streamlit App (User Interface)

The Streamlit App serves as the main user interface for the Cartoonizer application. It manages user interactions such as uploading images, selecting filters, and downloading processed images. Users can choose to upload their own images or select from predefined datasets. The app also provides options to resize images and apply various filters, making it a versatile tool for image cartoonification.

PIL Library (Image Processing)

The Python Imaging Library (PIL) is used for fundamental image processing tasks within the application. It handles opening, manipulating, and saving images in various formats. When a user uploads an image or selects one from the dataset, PIL processes these images to ensure they are in the correct format for further manipulation and filtering.

CV2 Library (Image Processing)

OpenCV's CV2 library offers more advanced image processing capabilities compared to PIL. It is particularly useful for operations like resizing images and converting between different color spaces. This library is crucial for preparing images for filter application and ensuring they are displayed correctly within the app.

FILTER_MAP (Filter Functions)

FILTER_MAP is a custom module or dictionary that maps filter names to their corresponding filter functions. This component provides the specific image filters that users can apply to their images. By selecting a filter from the user interface, the app retrieves the appropriate function from FILTER_MAP and applies it to the image, transforming it according to the chosen style and intensity.

OS Library (File System Access)

The OS library is essential for accessing and managing files within the local filesystem. It enables the application to list files in directories, construct file paths, and check for file existence. This library is used extensively to load images from the predefined datasets and save the processed images for user download.

Workflow Integration

The workflow begins with user interaction through the Streamlit App, where users upload images or select them from datasets. The PIL and CV2 libraries then process these images, ensuring they are ready for filtering. FILTER_MAP provides the necessary functions to apply the chosen filters, and the OS library manages file access and storage. This integration of components allows the Cartoonizer App to function seamlessly, providing a user-friendly experience for transforming images into cartoon-style artwork.

4.2 ARCHITECTURE:

Creating UML diagrams for the Cartoonizer App involves representing the system's components and their interactions in a visual format.

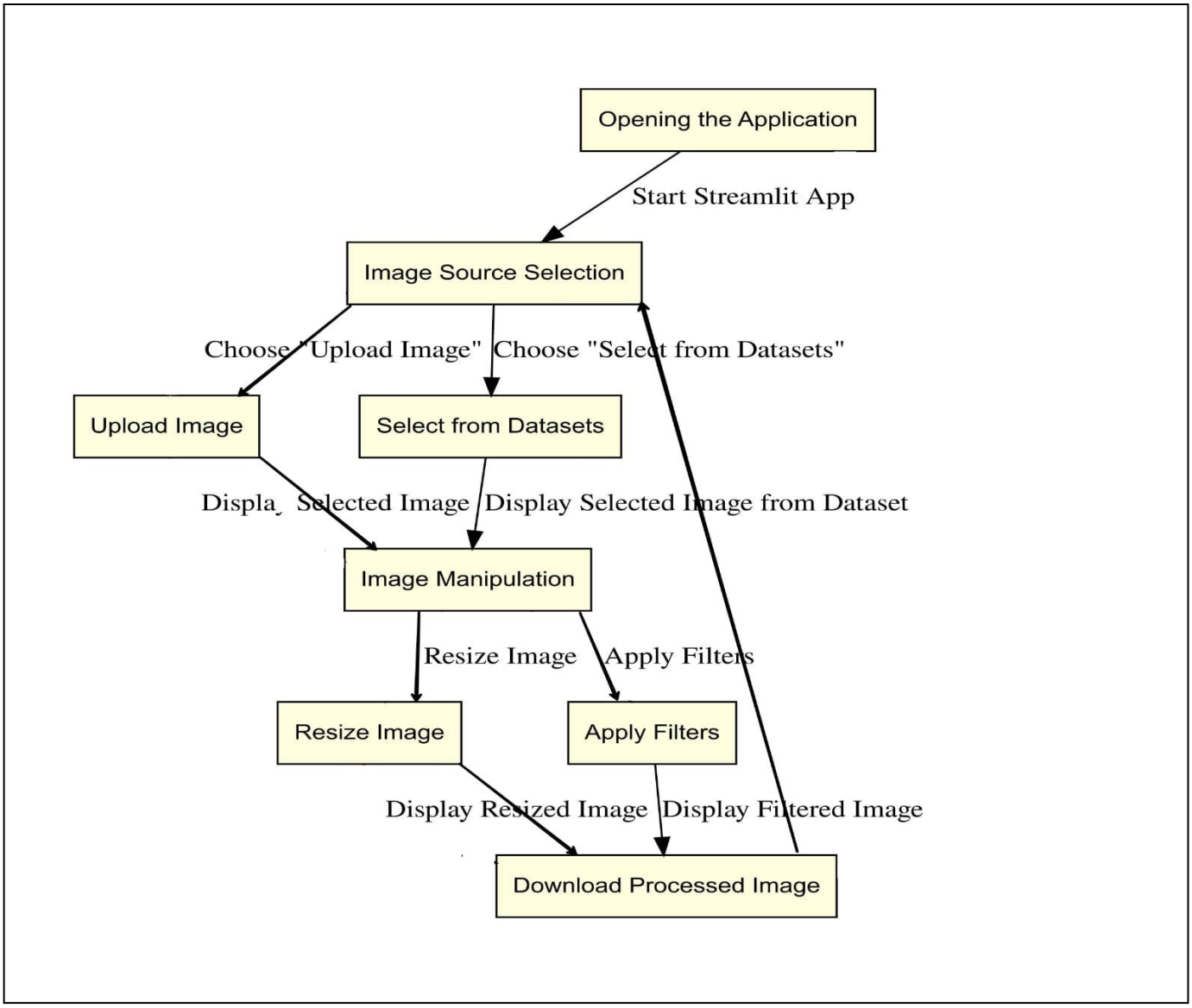


Fig.1.ARCHITECTURE DIAGRAM OF PICTOON

4..3 UML DIAGRAMS:

4.3.1 Use Case Diagram:

- **Purpose**: Illustrate the functionalities of the system from a user's perspective.
- **Components**:
  - **Actors**: User
  - **Use Cases**: Upload Image, Select Image from Dataset, Resize Image, Apply Filter, Download Image
- **Interactions**: Show the interactions between the User and each use case.

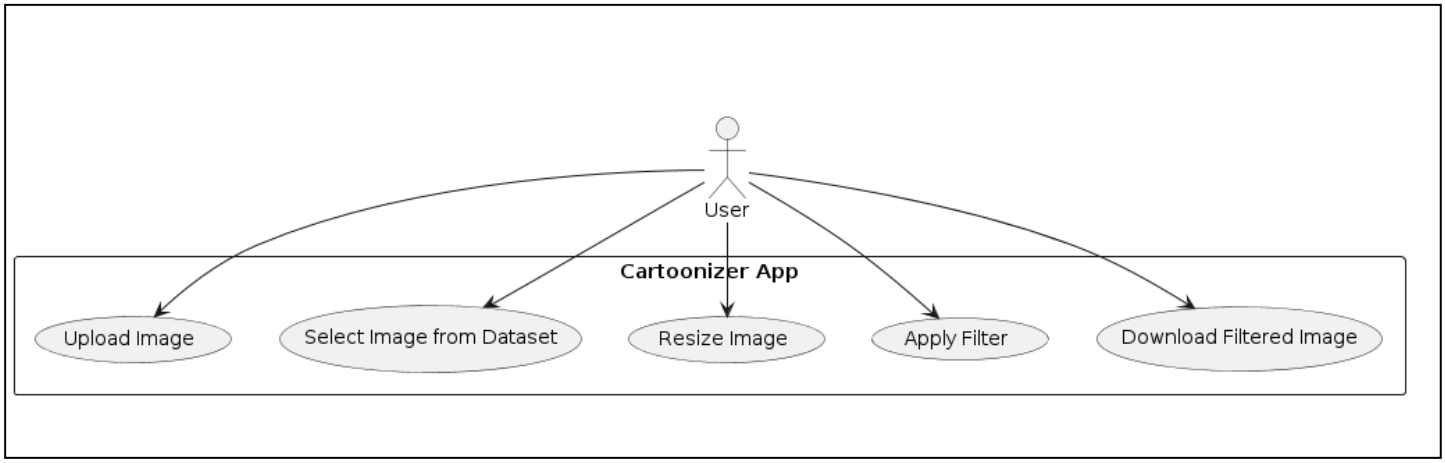


Fig.2.USECASE DIAGRAM

4.3.2 Class Diagram:

- **Purpose**: Represent the structure of the system by showing its classes and relationships.
- **Components**:
  - **Classes**:
    - CartoonizerApp (main class managing the app flow)
    - DatasetManager (handles datasets)
    - ImageProcessor (applies filters and resizes images)
    - UIHandler (manages user interface elements)
- **Attributes and Methods**: Show key attributes and methods for each class.

- **Relationships:** Illustrate associations between classes (e.g., Cartoonizer App uses Dataset Manager and Image Processor).

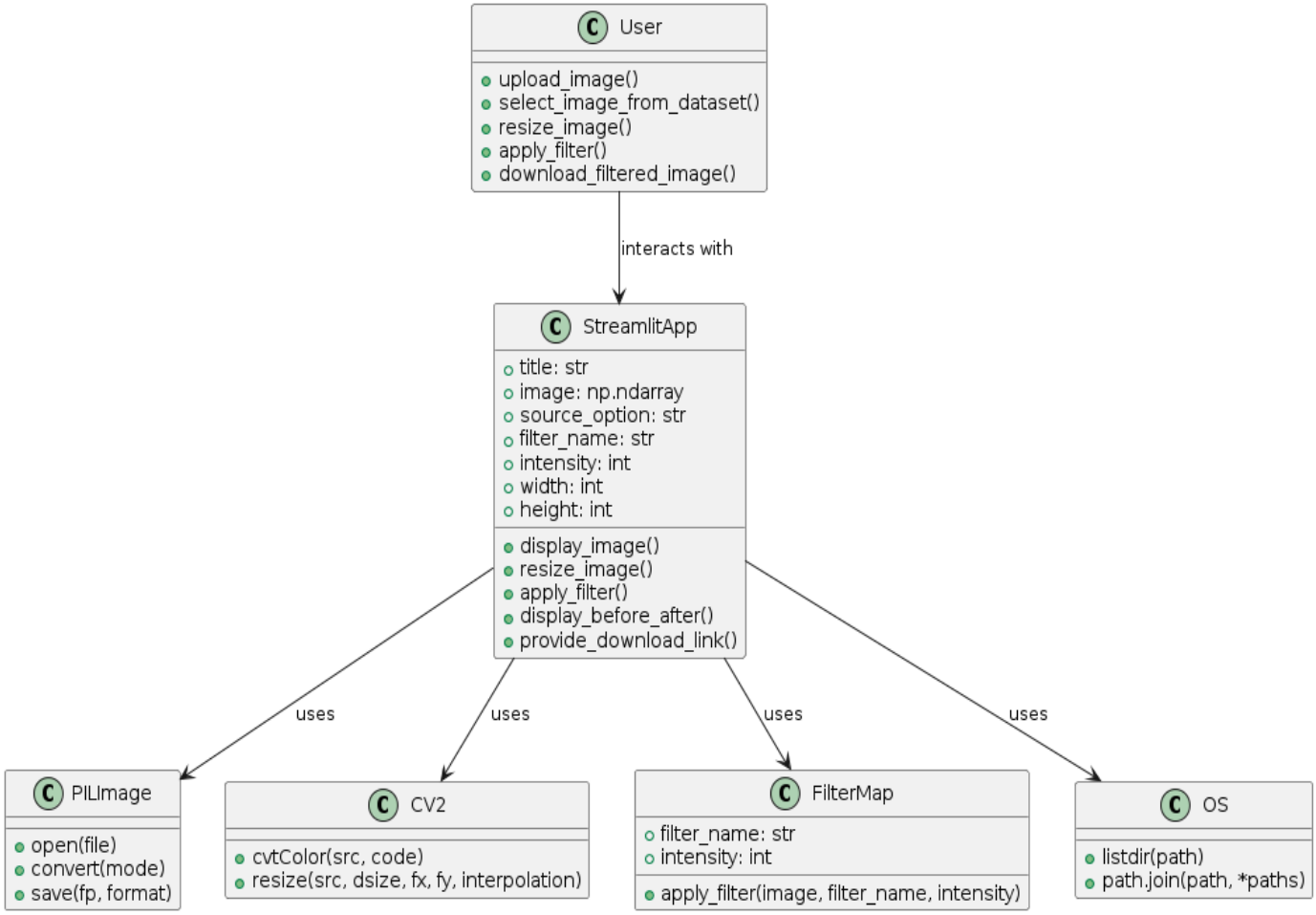
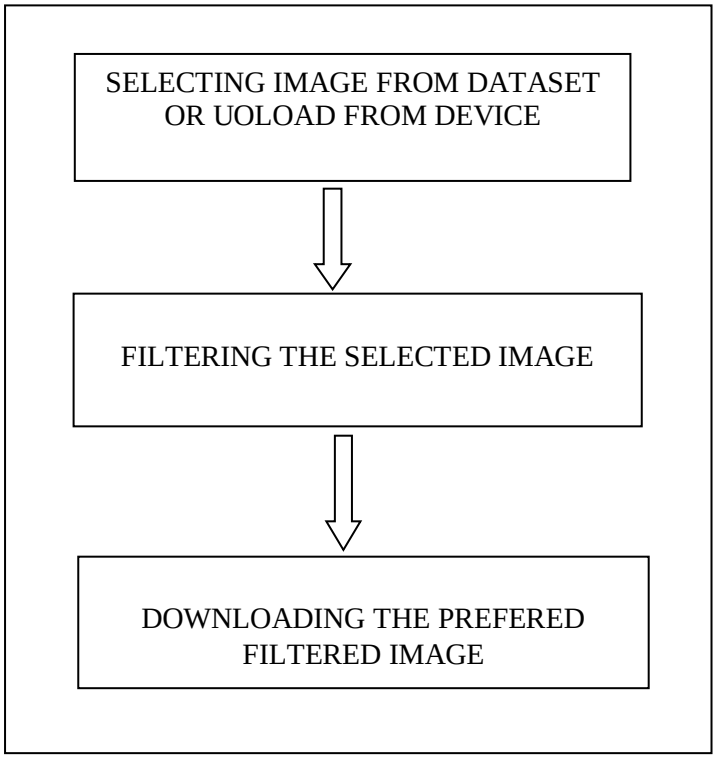


Fig.3.CLASS DIAGRAM

- Illustrate the interactions between objects over time.
- Show the sequence of messages exchanged during a face recognition process
- Describe the workflow or process flow within the face recognition system.
- Activities may include "Selecting Image," Downloading Image" etc.

- Model the different states a face recognition system might go through.
- States may include "Idle," "Selecting Image," Filtering Image ","Downloading Image" etc.

DATAFLOW:



- Illustrate the high-level components and their relationships in the system.
- Components may include "Selecting of image," "Filters Module," "Datasets," etc.

4.3.3 ACTIVITY DIAGRAM

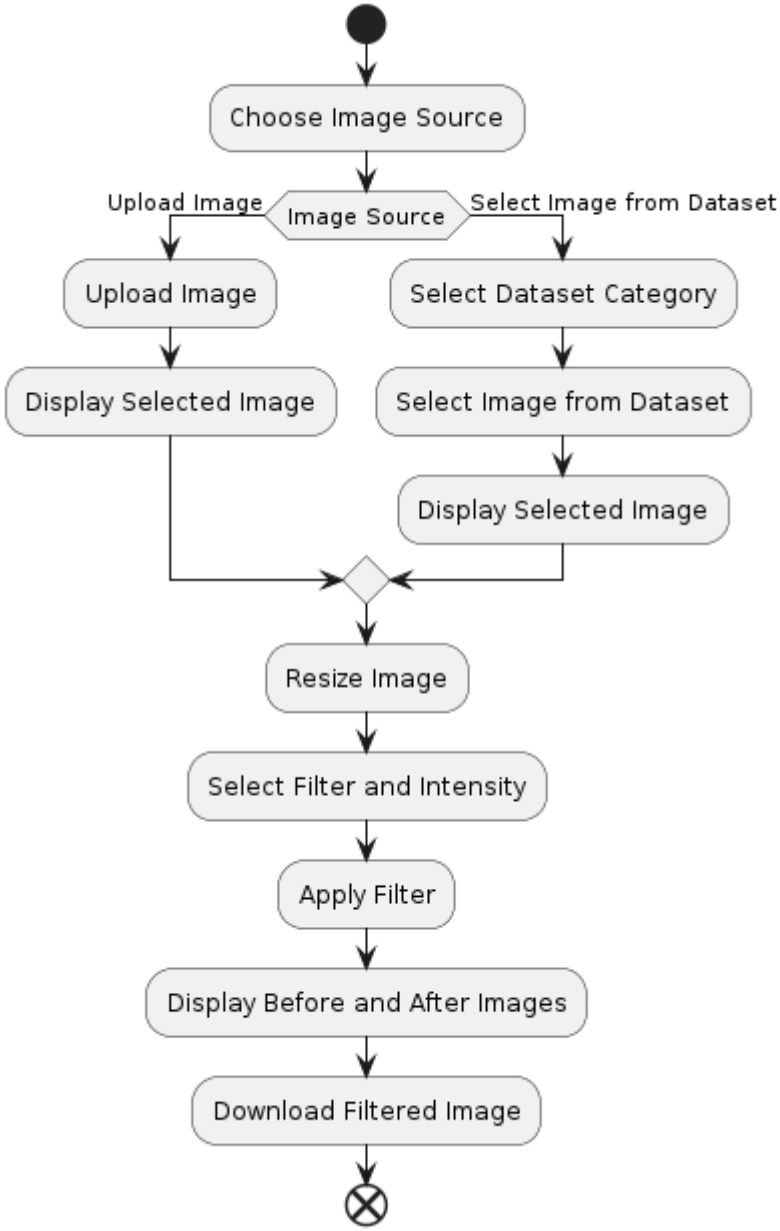


Fig.4.ACTIVITY DIAGRAM

The diagram begins at the "Start" point, indicating the initiation of the process.

Display Image Source Options

The application displays options for the user to choose the image source (either upload an image or select from a dataset).

- Branch: User Chooses to Upload Image:

  - Upload Image: The user uploads an image.

- Branch: User Chooses to Select Image from Dataset:

  -Select Image from Dataset: The user opts to select an image from a dataset.

  - Display Dataset Categories: The application displays categories of datasets.

- Download Filtered Image:

  -Download Filtered Image: The user opts to download the filtered image.

- End:

  - The diagram ends at the "End" point, indicating the completion of the process.

This activity diagram captures the workflow of the application from the user's perspective, detailing each step involved in uploading or selecting an image, applying filters, and downloading the final result.

4.3.4 SEQUENCE DIAGRAM:

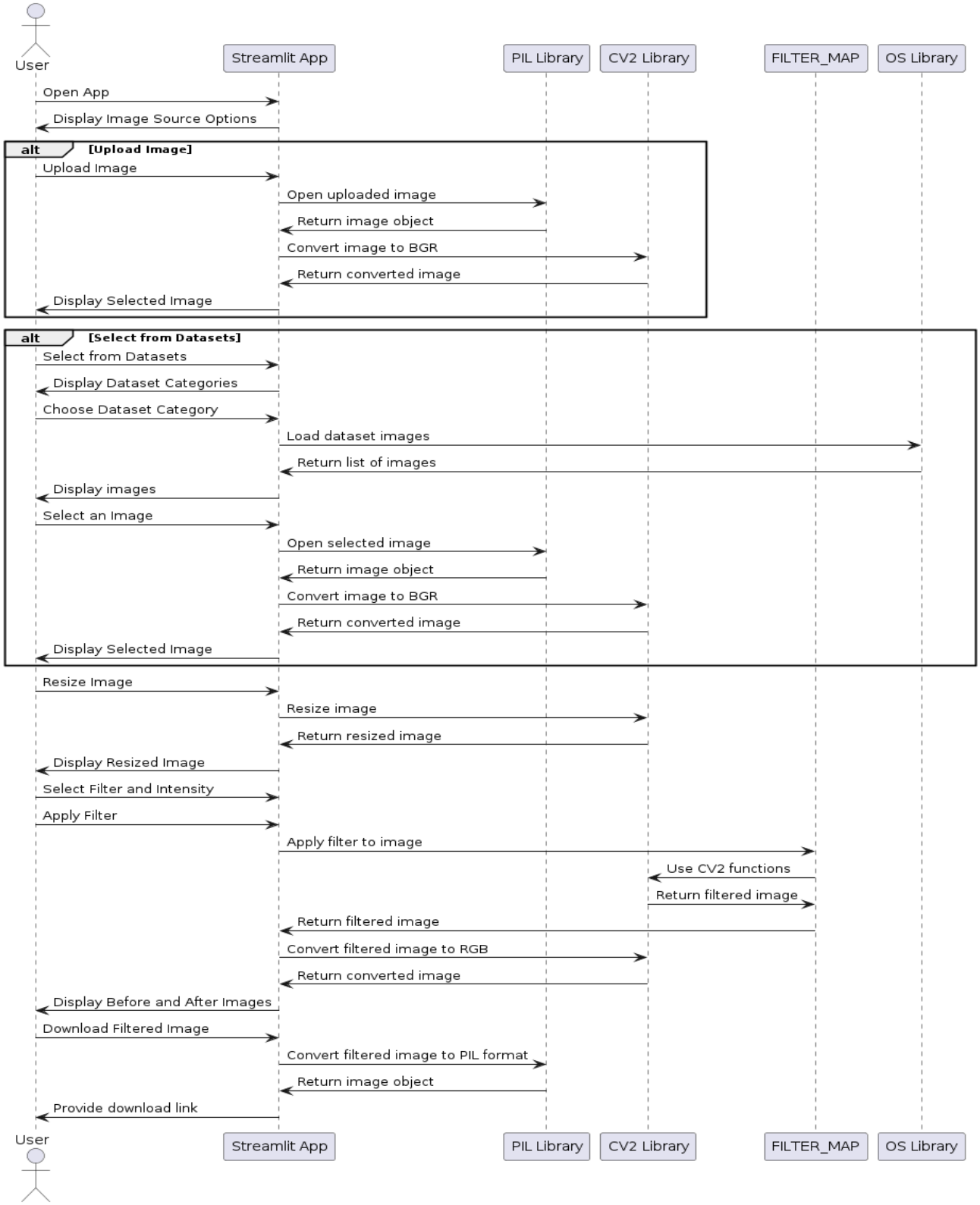


Fig.5.SEQUENCE DIAGRAM

The sequence diagram for the Cartoonizer App illustrates the detailed workflow of the application, showcasing interactions between the user, the app's main interface, and supporting libraries such as PIL, CV2, FILTER_MAP, and the OS library. The user can upload an image or select one from predefined datasets, which the app processes and displays using PIL and CV2 libraries. Users can resize the image and apply various filters, with the FILTER_MAP module handling the filter application using CV2 functions. The app then displays both the original and filtered images side-by-side. Finally, users can download the filtered image, which the app provides after converting it to the appropriate format using the PIL library. This diagram effectively captures the comprehensive and user-friendly functionality of the Cartoonizer App, emphasizing its ability to facilitate creative image processing and enhancement.

CHAPTER 5

IMPLEMENTATION

Implementing a Cartoonization involves several steps, ranging from data collection to model deployment. Below is a comprehensive guide with step-by-step instructions for building a Cartoonizing system using Python and popular libraries like OpenCV and PIL :

1. Data Collection:

Gather a diverse dataset containing images of different categories that you want the system to cartoonize such as buildings , flowers and etc.. Label each image with the corresponding identity.

2. Data Preprocessing:

Uploaded images are converted into RGB format using the PILLOW library.

Images are convereted to Numpy arrays to facilitate manipulation using OpenCV.

User can select a filtered images from available options and also ca resize the image.

3. Model Training:

Train the model using the preprocessed images. This involves feeding the images into the model, calculating the loss, and updating the model weights using back propagation.

4. Model Deployment:

Choose a deployment platform (e.g., local machine, cloud, edge device).

Integrate the trained model into your application or service.

5. Real-time Image Conversion :

We can either choose an image from the device or we select image from datasets preloaded in the site. Once the image is selected ,the image is filtered and given as output for user to save them.

6. .User Interface (Optional):

Create a user-friendly interface for interacting with the user to use the site. Display filtered images and additional information.Allow for user download and save those images.

7. Privacy and Security:

Implement privacy-preserving techniques (e.g., on-device processing, federated learning). Comply with data protection regulations and obtain user consent.

5.1 Source Code

5.1.1. MAIN CODE

```
import streamlit as st

import cv2

import numpy as np

from filters import FILTER_MAP  # Ensure this module exists

from PIL import Image

import os

import io

# Define datasets with representative icons

DATASETS = {

    "Pets": "C:/Users/Lawrence/OneDrive/Documents/MP/code/DATASETS/PETS",

    "Flowers": "C:/Users/Lawrence/OneDrive/Documents/MP/code/DATASETS/FLOWERS AND NATURE",

    "Actors": "C:/Users/Lawrence/OneDrive/Documents/MP/code/DATASETS/ACTORS",

    "Cartoons": "C:/Users/Lawrence/OneDrive/Documents/MP/code/DATASETS/CARTOONS AND ANIME",

    "Cars": "C:/Users/Lawrence/OneDrive/Documents/MP/code/DATASETS/CARS",

    "Buildings": "C:/Users/Lawrence/OneDrive/Documents/MP/code/DATASETS/BUILDINGS"

}

# Load all dataset images

DATASET_IMAGES = {}

for dataset_name, folder_path in DATASETS.items():


    images = [os.path.join(folder_path, file) for file in os.listdir(folder_path) if file.endswith(('png', 'jpg',
```

```
'jpeg'))]
    DATASET_IMAGES[dataset_name] = images
st.title("Cartoonizer App")
def apply_filter(image, filter_name, intensity):
    if filter_name in FILTER_MAP:
        filter_func = FILTER_MAP[filter_name]
        return filter_func(image, intensity)
    return image
def resize_image(image, width, height):
    return cv2.resize(image, (width, height), interpolation=cv2.INTER_AREA)
# User selection: Upload image or select from datasets
st.sidebar.title("Image Source")
source_option = st.sidebar.radio("Choose an option", ("Upload Image", "Select from Datasets"))
image = None
if source_option == "Upload Image":
    uploaded_file = st.sidebar.file_uploader("Upload your own image", type=["png", "jpg", "jpeg"])
    if uploaded_file:
        image = Image.open(uploaded_file)
        st.image(image, caption="Selected Image", use_column_width=True)
        image = np.array(image.convert("RGB"))
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

elif source_option == "Select from Datasets":
    st.sidebar.title("Select Dataset Category")
    dataset_category = st.sidebar.selectbox("Choose a category", list(DATASETS.keys()))
```

30

```
    if dataset_category:
        st.header(f"{dataset_category} Dataset")
        dataset_images = DATASET_IMAGES[dataset_category]
        selected_image_path = None
        num_cols = 3
        cols = st.columns(num_cols)
        for i, img_path in enumerate(dataset_images):
            col = cols[i % num_cols]
            with col:
                if st.button(f"Select {os.path.basename(img_path)}", key=img_path):
                    selected_image_path = img_path
                st.image(img_path, use_column_width=True)
        if selected_image_path:
            image = Image.open(selected_image_path)
            st.image(image, caption="Selected Image", use_column_width=True)
            image = np.array(image.convert("RGB"))
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
if image is not None:
    # Resize Options
    st.sidebar.title("Resize Image")
    width = st.sidebar.number_input("Width", min_value=1, value=image.shape[1])
    height = st.sidebar.number_input("Height", min_value=1, value=image.shape[0])
    if st.sidebar.button("Resize Image"):
        image = resize_image(image, width, height)
        st.image(image, caption="Resized Image", use_column_width=True)
```

```
# Filter Selection
st.sidebar.title("Filters")
filter_name = st.sidebar.selectbox("Choose a filter", list(FILTER_MAP.keys()))
intensity = st.sidebar.slider("Intensity", 1, 10, 5)
if st.sidebar.button("Apply Filter"):
    filtered_image = apply_filter(image, filter_name, intensity)
    filtered_image = cv2.cvtColor(filtered_image, cv2.COLOR_BGR2RGB)
    # Display Before and After images
    col1, col2 = st.columns(2)
    col1.header("Before")
    col1.image(image, use_column_width=True)
    col2.header("After")
    col2.image(filtered_image, use_column_width=True)
    # Download filtered image
    im_pil = Image.fromarray(filtered_image)
    img_buffer = io.BytesIO()
    im_pil.save(img_buffer, format='PNG')
    img_buffer.seek(0)
    st.download_button(label="Download Filtered Image",
                       data=img_buffer,
                       file_name="filtered_image.png",
                       mime="image/png")
# Display all filters with download buttons
st.header("All Filters")
all_filtered_images = {}
```


30

```
filter_names = list(FILTER_MAP.keys())
num_filters = len(filter_names)
for i in range(0, num_filters, 2):
    cols = st.columns(2)
    for j in range(2):
        if i + j < num_filters:
            fname = filter_names[i + j]
            filtered_img = apply_filter(image.copy(), fname, intensity)
            filtered_img = cv2.cvtColor(filtered_img, cv2.COLOR_BGR2RGB)
            all_filtered_images[fname] = filtered_img
            cols[j].subheader(fname)
            cols[j].image(filtered_img, caption=fname, use_column_width=True)
            im_pil = Image.fromarray(filtered_img)
            img_buffer = io.BytesIO()
            im_pil.save(img_buffer, format='PNG')
            img_buffer.seek(0)
            with cols[j]:
                st.download_button(label=f"Download {fname} Image",
                                   data=img_buffer,
                                   file_name=f"{fname}.png",
                                   mime="image/png")
```

5.1.2. FILTERS CODE

```
import cv2

import numpy as np

def caart(img, intensity=1):

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

    gray = cv2.medianBlur(gray, 7)

    edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 9, 10)

    color = cv2.bilateralFilter(img, 9, 300, 300)

    cartoon = cv2.bitwise_and(color, color, mask=edges)

    return cartoon

def pencil_sketch(img, intensity=1):

    gray, sketch = cv2.pencilSketch(img, sigma_s=60, sigma_r=0.07, shade_factor=0.05)

    return sketch

def cel_shading(img, intensity=1):

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

    edges = cv2.Laplacian(gray, cv2.CV_8U, ksize=5)

    ret, mask = cv2.threshold(edges, 100, 255, cv2.THRESH_BINARY_INV)

    quantized = cv2.bilateralFilter(img, 9, 75, 75)

    quantized = cv2.convertScaleAbs(quantized, alpha=(intensity + 1)/10)

    cel_shaded = cv2.bitwise_and(quantized, quantized, mask=mask)

    return cel_shaded

def sketch(img, intensity=1):

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

```
    inv = cv2.bitwise_not(gray)
    blur = cv2.GaussianBlur(inv, (21, 21), 0)
    inv_blur = cv2.bitwise_not(blur)
    sketch = cv2.divide(gray, inv_blur, scale=256.0)
    return sketch
def halftone(img, intensity=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    inv = cv2.bitwise_not(gray)
    circles = cv2.HoughCircles(inv, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=30,
minRadius=5, maxRadius=30)
    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            cv2.circle(img, (i[0], i[1]), i[2], (255, 255, 255), 3)
    return img

def toon(img, intensity=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.medianBlur(gray, 5)
    edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 9, 9)
    color = cv2.bilateralFilter(img, 9, 75, 75)
    cartoon = cv2.bitwise_and(color, color, mask=edges)
    return cartoon
def watercolor(img, intensity=1):
    img = cv2.edgePreservingFilter(img, flags=1, sigma_s=60, sigma_r=0.4)
```

```
    for i in range(2):
        img = cv2.bilateralFilter(img, d=9, sigmaColor=75, sigmaSpace=75)
    return img
def comic_book(img, intensity=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    edges = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    inverted_edges = cv2.bitwise_not(edges)
    return inverted_edges
def manga(img, intensity=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    ret, mask = cv2.threshold(edges, 100, 255, cv2.THRESH_BINARY_INV)
    manga = cv2.bitwise_and(img, img, mask=mask)
    return manga
def caricature(img, intensity=1):
    rows, cols, _ = img.shape
    center_x, center_y = int(cols / 2), int(rows / 2)
    for i in range(rows):
        for j in range(cols):
            offset_x = int((i - center_y) * (i - center_y) / center_y)
            offset_y = int((j - center_x) * (j - center_x) / center_x)
            img[i, j] = img[min(rows - 1, i + offset_x), min(cols - 1, j + offset_y)]
    return img
def cartoon_3d(img, intensity=1):
```

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 50, 150)
    edges = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    return cv2.addWeighted(img, 0.8, edges, 0.2, 0)
def flat_design(img, intensity=1):
    Z = img.reshape((-1, 3))
    Z = np.float32(Z)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    K = 8
    _, labels, centers = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
    centers = np.uint8(centers)
    res = centers[labels.flatten()]
    return res.reshape((img.shape))
def ink_wash(img, intensity=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
    return cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)
def pop_art(img, intensity=1):
    Z = img.reshape((-1, 3))
    Z = np.float32(Z)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    K = 4
    _, labels, centers = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
    centers = np.uint8(centers)
```

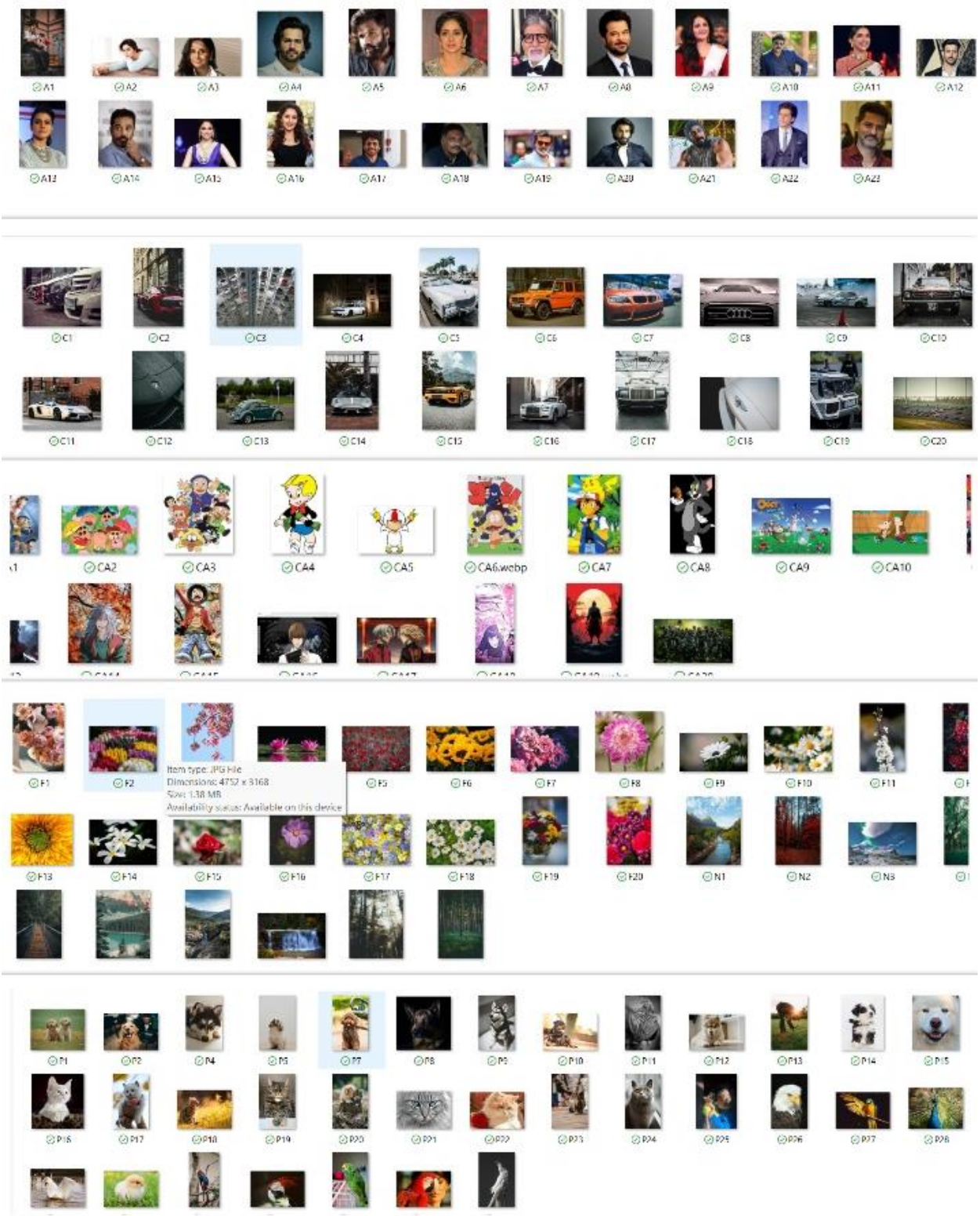
```
    res = centers[labels.flatten()]
    return res.reshape((img.shape))
def vector_art(img, intensity=1):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.equalizeHist(img)
    return cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
def pixel_art(img, intensity=1):
    height, width = img.shape[:2]
    temp = cv2.resize(img, (width // 10, height // 10), interpolation=cv2.INTER_NEAREST)
    return cv2.resize(temp, (width, height), interpolation=cv2.INTER_NEAREST)
def silhouette(img, intensity=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, silhouette = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY)
    return cv2.cvtColor(silhouette, cv2.COLOR_GRAY2BGR)
FILTER_MAP = {
    "Cartoon": caart,
    "Pencil Sketch": pencil_sketch,
    "Cel Shading": cel_shading,
    "Sketch": sketch,
    "Halftone": halftone,
    "Toon": toon,
    "Watercolor": watercolor,
    "Comic Book": comic_book,
    "Manga": manga,
    "Caricature": caricature,
```

```
    "3D Cartoon": cartoon_3d,
    "Flat Design": flat_design,
    "Ink Wash": ink_wash,
    "Pop Art": pop_art,
    "Vector Art": vector_art,
    "Pixel Art": pixel_art,
    "Silhouette": silhouette,
}
```

- **Importing necessary packages**

```
import streamlit as st
import cv2
import numpy as np
from filters import FILTER_MAP  # Ensure this module exists
from PIL import Image
import os
import io
```

- **Creating datasets:**



CHAPTER 6

RESULTS

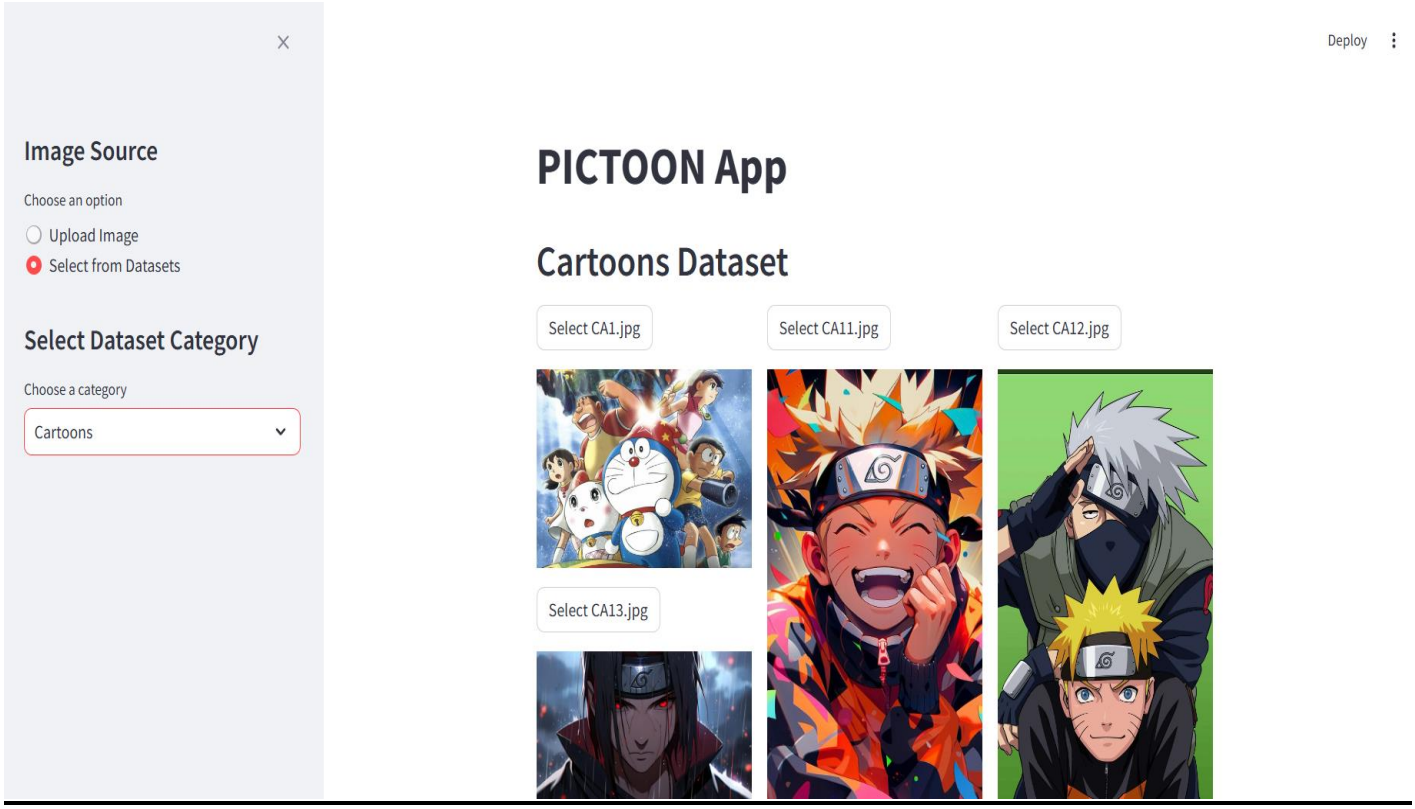
The results of a cartoonizing system typically involves conerting images of real time into different styles of cartoon styles. The outcomes can be categorized into several scenarios:

FILTERED IMAGE:

Description: The system correctly filters or converts images that user either upload or select from datasets available in the site

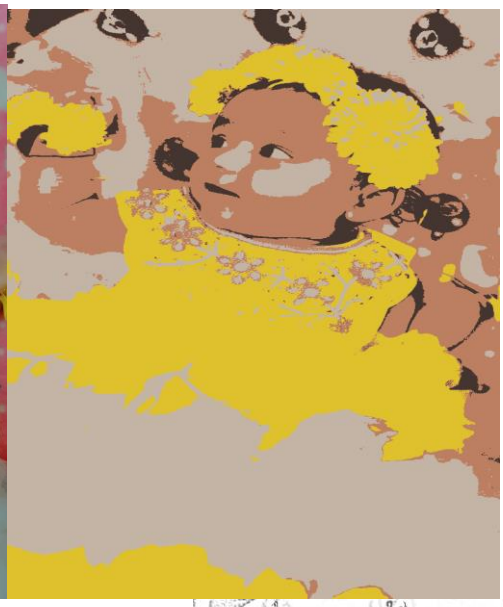
Use Case: Access control, security surveillance, authentication.

Code execution



INPUT IMAGE : 

OUTPUT FILTERED IMAGES:



CHAPTER 7

CONCLUSION

The Cartoonizer App is a pioneering application designed to democratize the process of transforming ordinary photographs into captivating cartoon-like images. Leveraging advanced image processing techniques, the app provides a seamless user experience, allowing individuals to apply a variety of artistic filters to their photos. Users can either upload their personal images or choose from a comprehensive set of curated datasets, encompassing categories such as pets, flowers, actors, cartoons, cars, and buildings. This flexibility caters to diverse interests and ensures that the app can be used for a wide array of creative and practical applications.

For professionals in fields such as marketing, social media, and graphic design, the Cartoonizer App provides a quick and efficient way to generate engaging content. The ability to easily transform images into cartoons can be used to create eye-catching visuals for advertisements, social media posts, and promotional materials, thus enhancing communication and outreach efforts.

In conclusion, the Cartoonizer App is not only a tool for personal and professional creative expression but also a valuable resource for educational and therapeutic purposes, making it a significant contributor to societal enrichment.

PERFORMANCE EVALUATION

The proposed system is evaluated based on the following parameters.

- Accuracy- how correctly the system provides the class label.
- Time- the amount of time the system takes to classify the input data.
- User-friendly Interface- how effectively the end users can understand and use the system.

FUTURE SCOPE

This project can further be developed into much efficient application by including many more features such as addition of more filters and many other features that reduces the effort of users such as we can add feature to restore the pixels and make the blur images much clear .Also can use AI to change background ogf the uploaded image as user wishes and further more.

REFERENCES

- Deep learning-based classification of the polar emotions of "moe"-style cartoon pictures .by Qinchen Cao; Weilin Zhang; Yonghua Zhu.

- Zhang, Jinglei, Hou, Ya-Wei. Image style migration based on improved recurrent generative adversarial network. Journal of Electronics and Information.

- https://github.com/Sun-Yize-SDUWH/Photo-Cartoonization

- Fan Ke-Yue, Liu S-G. Stylization method for portrait photo cartoons. Journal of Computer-Aided Design and Graphics.

- Wang Dechu. Research and application of automatic face caricature generation algorithm . South China University of Technology.

- Chen, Shuhuan, Wei, Yuke, Xu, Le, et al. A review of image style migration research based on deep learning . Computer Application Research.

- K. Lata, M. Dave and K. N. Nishanth, "Image-to-Image Translation Using Generative Adversarial Network," 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA).

- Yilin Ouyang; Yunbo Rao; Dawei Zhang; Jiajun Cheng “Cartoon Colorization with Gray Image Generated from Sketch” .

- Yuxiang Xie; Xidao Luan; Xin Zhang; Chen Li; Liang Bai “A Cartoon Image Annotation and Retrieval System Supporting Fast Cartoon Making”

- Xinyu Li, Wei Zhang, Tong Shen, Tao Mei “Everyone is a Cartoon-ist: Selfie Cartoonization with Attentive Adversarial Networks” .

- Henry Kang, Seungyong Lee, Charles K. Chui. “Flow Based Image Abstraction” IEEE Computer Society .

- Hamidreza Sadreazami, Amir Asif, Arash Mohammadi. In 12th Symposium on Operating Systems Design and Implementation.

- Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super resolution: Dataset and study. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, July 2017 “Iterative Graph-based Filtering for Image Abstraction and Stylization”.

- Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua and Sabine Susstrunk, “Slic superpix-els compared to state-of-the-art superpixel methods”, ¨ IEEE Transactions on Pattern Analysis and Machine Intelligence.

- Qingnan Fan, Jiaolong Yang, David Wipf, Baoquan Chen, and Xin Tong. “Image smoothing via unsupervised learning”. .

- Xinrui Wang, Jinze Yu, ByteDance, The University of Tokyo, Style2Paints Research “Learning to Cartoonize Using White-boox Cartoonize Representations”.