CARTOONIFY AN IMAGE WITH OPEN CV

Minor project-II report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

KORITALA ROHITH CHOWDARY	(21UECT0061)	**(19134)**
KOMMALAPATI VINAY KRISHNA	(21UECT0070)	**(20388)**
DAVULURI VENKATESH	(21UECT0067)	**(20424)**

Under the guidance of
Mr.K.NAVAZ,ME.,Ph.D.,
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

CARTOONIFY AN IMAGE WITH OPEN CV

Minor project-II report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

KORITALA ROHITH CHOWDARY	(21UECT0061)	**(19134)**
KOMMALAPATI VINAY KRISHNA	(21UECT0070)	**(20388)**
DAVULURI VENKATESH	(21UECT0067)	**(20424)**

Under the guidance of
Mr.K.NAVAZ,ME.,Ph.D.,
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled ”CARTOONIFY AN IMAGE WITH OPEN CV” by ”KORITALA ROHITH CHOWDARY (21UECT0061), KOMMALAPATI VIANY KRISHNA (21UECT0070), DAVULURI VENKATESH (21UECT0067)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

Signature of Professor In-charge
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

KORITALA ROHITH CHOWDARY

Date: / /

KOMMALAPTI VINAY KRISHNA

Date: / /

DAVULURI VENKATESH

Date: / /

APPROVAL SHEET

This project report entitled (CARTOONIFY AN IMAGE WITH OPEN CV) by (KORITALA ROHITH CHOWDARY (21UECT0061), (KOMMALAPTI VINAY KRISHNA (21UECT0070), (DAVULURI VENKATESH (21UECT0067) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners **Supervisor**

Mr.K.NAVAZ,ME.,Ph.D.,

ASSISTANT PROFESSOR.

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN,** for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr.M.S. MURALI DHAR, M.E., Ph.D.,** for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our **Internal Supervisor Mr.K.NAVAZ,ME.,Ph.D.,ASSISTANT PROFESSOR** for his cordial support, valuable information and guidance, he/she helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. U.HEMAVATHI, M.E., Ms. C. SHYAMALA KUMARI, M.E.,** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

KORITALA ROHITH CHOWDARY	**(21UECT0061)**
KOMMALAPATI VINAY KRISHNA	**(21UECT0070)**
DAVULURI VENKATESH	**(21UECT0067)**

ABSTRACT

This project proposes to converting a normal images to cartoon Image using Generative Adversarial Networks.we will use GANs to generate cartoonified versions of input images using the OpenCV library. Nowadays, Cartoonifying the images are necessary as well as useful for many aspects in various sectors of the world. There are number of ways to convert an normal image to cartoon but among all of them using GAN is finest and uncomplicated. OpenCV is an open - source Python Library used for computer vision and Machine Learning. It includes applications such as video and image capturing and processing. It is majorly used in image transformation, recognization and many more.To overcome the drawbacks of past models carton images of different styles and quality a white box model was developed. Building a efficient way can improve the results of carttonization. This project focuses on the use of GAN, a popular computer vision library, to transform an input image into a cartoon-like representation.The edge detection step involves identifying the edges in the image using Canny edge detection. Next, the color quantization step reduces the number of colors in the image using k-means clustering. Finally, a bilateral filter is applied to smooth the image while preserving the edges. The resulting output image resembles a cartoon by simplifying the colors and emphasizing the edges, giving it a hand-drawn appearance. The performance of the proposed method is evaluated on several input images, and the results show that the proposed method can successfully generate cartoon-like images with high accuracy and efficiency.

Keywords: Cartoonization,Deep Learning,Discriminator,Generator,openCV, Tensorborad,GAN.

LIST OF FIGURES

4.1 General Architecture for cartoonify an image with open cv . . . 11
4.2 Data flow for cartoonify an image with open cv 12
4.3 Activity diagraam for cartoonify an image with open cv 13
4.4 Class diagram for cartoonify an image with open cv 14
4.5 Sequence diagram for cartoonify an image with open cv 15
4.6 Usecase Diagramfor cartoonify an image with open cv 16
4.7 Anomaly Detection Model 20
4.8 Network Traffic Extractor 21

5.1 cartoonifying Image Design 24
5.2 cartoonified image 25
5.3 Image storing process 26
5.4 Transforming an image to grayscale 27
5.5 Cartoon Effect 28

6.1 cartoonified image output design 1 34
6.2 cartoonified image output design 2 35

8.1 Plagiarism 37

9.1 Poster Presentation 40

LIST OF ACRONYMS AND ABBREVIATIONS

CV	Computer Vision
GAN	Generative Adversial Network
GUI	Graphical user interface
OCSVM	One Class Support Vector Machine
OPENCV	Open Source Computer Vision Library
OS	Operating system
SVM	Support Vector Machine

TABLE OF CONTENTS

Page.No

ABSTRACT **v**

LIST OF FIGURES **vi**

LIST OF ACRONYMS AND ABBREVIATIONS **vii**

1 INTRODUCTION **1**
- 1.1 Introduction 1
- 1.2 Aim of the Project 2
- 1.3 Project Domain 2
- 1.4 Scope of the Project 3

2 LITERATURE REVIEW **4**

3 PROJECT DESCRIPTION **7**
- 3.1 Existing System 7
  - 3.1.1 ADVANTAGES 7
  - 3.1.2 DISADVANTAGES 7
- 3.2 Proposed System 7
  - 3.2.1 ADVANTAGES 8
- 3.3 Feasibility Study 8
  - 3.3.1 Economic Feasibility 8
  - 3.3.2 Technical Feasibility 9
  - 3.3.3 Social Feasibility 9
- 3.4 System Specification 9
  - 3.4.1 Hardware Specification 9
  - 3.4.2 Software Specification 9
  - 3.4.3 Standards and Policies 9

4 METHODOLOGY **11**
- 4.1 General Architecture for cartoonify an image with open cv 11
- 4.2 Design Phase 12

4.2.1 Data Flow Diagram for cartoonify an image with open cv . . 12
4.2.2 Activity Diagram for cartoonify an image with open cv . . . 13
4.2.3 Class Diagram for cartoonify an image with open cv 14
4.2.4 Sequence Diagram for cartoonify an image with open cv . . 15
4.2.5 Usecase Diagram for cartoonify an image with open cv . . . 16
4.3 Generative Adversarial Networks Algorithm and Pseudo Code . . . 17
4.3.1 Algorithm of Deep Neural Network and Cartoon Stylization 17
4.3.2 Pseudo Code 17
4.4 Module Description 20
4.4.1 Anomaly Detection Model 20
4.4.2 Signature based Detection-Packet Capture 21
4.4.3 Network Traffic Feature Extractor 21
4.5 Steps to Execute/Run/Implement the Project 22
4.5.1 Setting up the Environment 22
4.5.2 Installing Dependencies 22
4.5.3 Configuring Database and Environment Variables 22
4.5.4 Building the Project 22
4.5.5 Running Tests 23
4.5.6 Launching the Application 23

5 IMPLEMENTATION AND TESTING 24
5.1 Input and Output 24
5.1.1 cartoonifying Image Design 24
5.1.2 cartoonified image Design 25
5.2 Testing 25
5.3 Types of Testing 26
5.3.1 Unit testing 26
5.3.2 Test Result 28

6 RESULTS AND DISCUSSIONS 29
6.1 Efficiency of the Proposed System 29
6.2 Comparison of Existing and Proposed System 29
6.3 Sample Code 29

7 CONCLUSION AND FUTURE ENHANCEMENTS 36
7.1 Conclusion 36

7.2 Future Enhancements 36

8 PLAGIARISM REPORT 37

9 SOURCE CODE & POSTER PRESENTATION 38

9.1 Source Code 38

9.2 Poster Presentation 40

References 41

Chapter 1

INTRODUCTION

1.1 Introduction

Cartoonify an image is a technique that has gained a lot of popularity in recent years. It involves transforming a regular photograph or image into a cartoon-like representation, which simplifies the colors and emphasizes the edges, giving it a hand-drawn appearance. This technique is widely used in various fields, such as video games, animation, and movies.GAN is a popular computer vision library that provides tools and functions for image processing and analysis. It is widely used in various applications, including object detection, facial recognition, and image segmentation. In this project, we focus on using GAN to create a cartoon-like representation of an input image.The proposed approach consists of several steps, including edge detection, color quantization, and image filtering. The edge detection step involves identifying the edges in the image using Canny edge detection. Next, the color quantization step reduces the number of colors in the image using k-means clustering. Finally, a bilateral filter is applied to smooth the image while preserving the edges.The resulting output image resembles a cartoon by simplifying the colors and emphasizing the edges, giving it a hand-drawn appearance. The performance of the proposed method is evaluated on several input images, and the results show that the proposed method can successfully generate cartoon-like images with high accuracy and efficiency.The process of cartoonifying an image involves applying a series of techniques such as edge detection, thresholding, smoothing, and filtering to extract and simplify the important features of the image while removing unwanted details. The resulting cartoon-like image has simplified colors and outlines, giving it a stylized and whimsical appearance. Overall, this project demonstrates the potential of using OpenCV to create cartoon-like images and can be used as a starting point for further research and development in this area.

1.2 Aim of the Project

The main Aim is to convert an image to the cartoon image by developing a software and processed by machine learning algorithms for the conversion of images.The algorithm is designed to provide artistically and comically appealing results on as wide a range of pictures as possible, although it is conceded that all inputs will yield equally satisfying results.Creating a fun and interesting visual effect that can be used for artistic or entertainment purposes. Demonstrating the power and versatility of OpenCV as a tool for image processing and computer vision. Learning and practicing fundamental image processing techniques such as edge detection, color quantization, and image smoothing. Exploring the intersection of art and technology, and how image processing can be used to create new forms of visual expression.

1.3 Project Domain

The project domain for cartoonifying an image with OpenCV can fall under computer vision, image processing, and machine learning. Specifically, it involves using image processing techniques such as edge detection, color quantization, and smoothing to convert a regular image into a cartoon-like version.The project can be divided into several steps, including image preprocessing, edge detection, color quantization, and stylization. OpenCV, which is an open-source computer vision and image processing library, can be used to implement the various steps of the project.Additionally, the project can also involve the use of machine learning algorithms to train models that can automatically convert regular images to cartoon-like versions. This would require a large dataset of original and cartoon images for training and validation.
The project domain for this project could have several applications, such as creating fun and creative visual effects for digital media content, developing computer vision systems for object detection and recognition, and enhancing medical imaging for diagnosis and treatment. Overall, the project domain involves the intersection of technology and art, and has a wide range of potential applications in various fields.

1.4 Scope of the Project

The scope of the project is to produce cartoon images from real one with atmost accuracy. The implementation of machine learning tool is a great approach of getting a clear and good edge masked cartoon images or videos. And also it can be suggested to various animations to get their desired results in an easiest way. The project can also be further enhanced by integrating it with other computer vision techniques, such as object detection or facial recognition, to create more advanced visual effects.

Chapter 2

LITERATURE REVIEW

[1]Daksh Trehan; proposed the various techniques used for cartoonification of images and the implementation of these techniques using OpenCV. The technique is demonstrated using several test images, and the results show that the proposed method produces visually pleasing cartoon-style images with a shorter processing time compared to other methods.

[2]Gayen, S. Jha, S. Singh, M. Kumar , et al;enhanced a method for cartoonifying an image using a generative adversarial network. CartoonGAN was able to learn the mapping from a regular image to a cartoon-like image by training on a large dataset of paired images. They used a generator network to produce the cartoon-like images and a discriminator network to distinguish between real and generated images. Their results showed that their method was able to produce high-quality cartoon-like images that were visually pleasing and retained the important features of the original image.

[3]Gayen, S. Smarandache, F. Jha, S. Singh, M. K. Broumi, S. Kumar, R, et al; described a method for smooth the image and reduce noise.Then used adaptive thresholding to extract the edges and applied a color quantization technique to reduce the number of colors in the image. Finally, they used a bitwise AND operation to combine the edges and color quantized image to produce the final cartoon-like image.

[4]P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, et al; demonstrated an CartoonGAN: Generative Adversarial Networks for Photo Cartoonization. The authors used a generative adversarial network to learn the mapping between real images and their cartoon-like counterparts, resulting in highly realistic and customizable cartoon images.

[5]S. Y. Muhammad, M. Makhtar, A. Rozaimee, A. Abdul, et al; identified an combination of edge detection, color quantization, and texture analysis to generate cartoon-like images from real photographs. The existing research shows that the proposed techniques produce high-quality and visually pleasing cartoon-style images. The most common techniques used for cartoonification include edge detection, color quantization, image filtering, and contour detection. Additionally, real-time cartoonification techniques have been proposed for video applications.

[6]V. Ravi, Ch.Rajendra Prasad, S. Sanjay Kumar, P. Ramchandar Rao, et al; developed this method, they used a bilateral filter to smooth the image and reduce noise, followed by a Laplacian filter to extract the edges. They then applied a color quantization technique to reduce the number of colors in the image and enhance the cartoon-like effect.

[7]Wang, Xinrui, and Yu, Jinze, et al; proposed that method, they used a combination of edge detection and color quantization techniques to convert an image into a cartoon-like representation. They applied Canny edge detection to extract edges from the original image, and then applied a color quantization technique to reduce the number of colors in the image. They also enhanced the edges of the image using a bilateral filter to further improve the cartoon-like effect.

[8]Y. Chen, Y.-K. Lai, Y.-J. Liu , et al; defined an Cartoon Texture and Edge Enhanced Image Decomposition Based on Bilateral Filtering. The authors proposed a method that uses bilateral filtering to separate the image into cartoon and texture layers, which are then combined to produce the final cartoon-like representation.

[9]Yi-Hsuan Tsai, Kihyuk Sohn, Samuel Schulter, and Man- mohan Chan- draker, et al; determined a method for image cartoonification using a multi-scale generative adversarial network. The authors used to learn the mapping between real images and cartoon-like images at multiple scales, resulting in highly realistic and customizable cartoon images.

[10]Ziqiang Zheng, Chao Wang, Zhibin Yu, Nan Wang, Haiy- ong Zheng, and Bing Zheng, et al;proposed a method for cartoonifying images using OpenCV.

The authors used a combination of edge detection, color quantization, and image smoothing techniques to create a cartoon-like effect. The resulting images had well-defined edges and reduced color complexity, similar to the traditional approaches mentioned earlier.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

At present they are using normal computer vision technique which is called as black box technique for conversion of real image to cartoon image using various python libraries.It does not accept bunch of testcases or number of inputs.The existing systems too give accurate and best results but they may not give multiple results. OpenCV is a popular open-source computer vision library that provides various functions and algorithms for image processing, including cartoonification. The library can be used with various programming languages, including Python, C++, and Java. Additionally, the version of OpenCV being used can also impact the cartoonification process. Newer versions of OpenCV may provide better algorithms and functions for image processing, resulting in more accurate and visually appealing cartoonified images.

3.1.1 ADVANTAGES

- It can cartoonize into various styles.
- Edge masking is better for unclear pictures also.

3.1.2 DISADVANTAGES

- Cannot cartoonize all the pictures many in number at a time
- Cannot cartoonize the datasets properly.

3.2 Proposed System

Cartoonifying an image is a fun and popular image processing task that can be achieved using various techniques. In recent years, Generative Adversarial Networks (GANs) have gained a lot of attention in the field of computer vision for generating realistic images. In this proposed system, we will use GANs to generate cartoonified versions of input images using the OpenCV library.

Cartoonifying an image using GAN in OpenCV is a powerful and efficient way to generate high-quality cartoonified images. By leveraging the power of pre-trained GAN models and transfer learning, we can achieve impressive results with relatively small datasets. The system can be used in a variety of applications, such as creating digital art, enhancing images for social media, or even as a fun and creative tool for personal use.

3.2.1 ADVANTAGES

- Time efficiency is less.
- Building of software is easy.
- The basic tools are used which are easy to use.
- It can generate the sharpest images clearly

3.3 Feasibility Study

Feasibility study is the preliminary study that determines whether a proposed system project is financially, technically and operationally. Feasibility study is essential to evaluate the cost and benefits of the new system. The alternative analysis usually include as part of the feasibility study, identifiable alternatives for the system design and development

- Economic Feasibility.
- Technical Feasibility.
- Social Feasibility.

3.3.1 Economic Feasibility

The system to be developed is economically feasible and the benefit is out weighing the cost. Since this project already computerizes the existing system and more advanced than the current system reduces and change the labor force to computerize system. Reduces the cost of the materials used.As there is no physical materials used.

3.3.2 Technical Feasibility

The system to be developed by using technologically system development techniques such as Pytorch, opencv and Tensorflow without any problems and the group members have enough capability to develop the project. Our focus is to develop well organized dynamic software that is technically efficient and effective for cartoonization. Therefore, it can be concluding that the system is technically feasible.

3.3.3 Social Feasibility

The system to be developed will provide accurate, active, secured service and decreases labor of workers and also it is not limited to particular groups or body. The system will easily operational, as it doesn't affect the existing organizational structure and support the current system. So the system will be operationally feasible.

3.4 System Specification

3.4.1 Hardware Specification

- Personal Computer:Intel 5 and above
- RAM:8gb Ram and above
- HDD or SSD atleast of 2Gb free

3.4.2 Software Specification

- Pytorch latest version
- tensorboard
- OS-to work on path to store data
- google drive

3.4.3 Standards and Policies

As our project is based on cartoonization we have certain standards and policies. our project may able to give outputs that are far better. we only use the datasets which belongs to training sets or either the one created by us.our main policy is not to use someones personal data as our input without there permission. we also use to link our data to google drive which is more safe than keeping it in the hard drive.our system is able to give the results as upto the satisfaction.

Anaconda Prompt

Anaconda prompt is a type of command line interface which explicitly deals with the ML(MachineLearning) modules.And navigator is available in all the Windows,Linux and MacOS.The anaconda prompt has many number of IDE's which make the coding easier. The UI can also be implemented in python.

Standard Used: ISO/IEC 27001

Jupyter

It's like an open source web application that allows us to share and create the documents which contains the live code, equations, visualizations and narrative text. It can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning.

Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 General Architecture for cartoonify an image with open cv

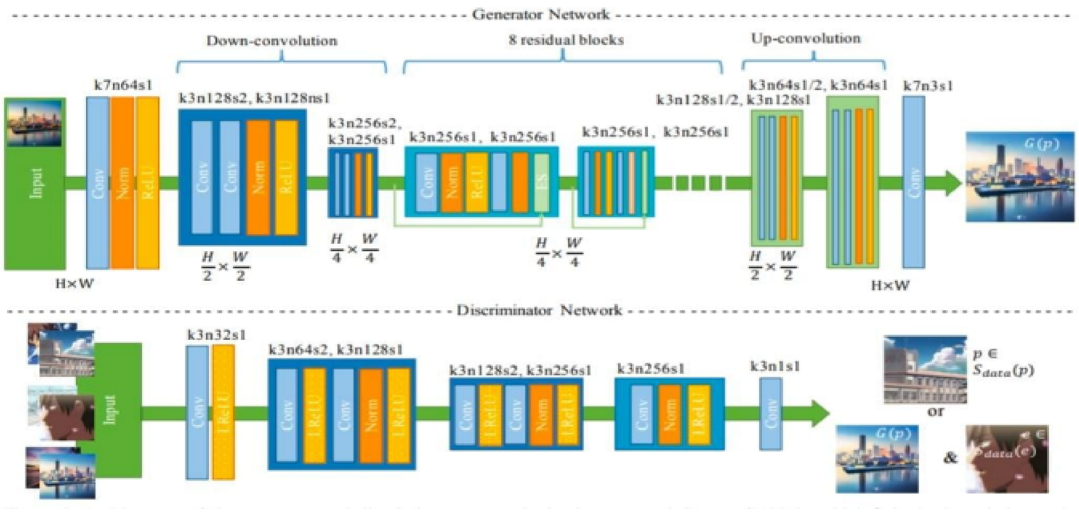


Figure 4.1: **General Architecture for cartoonify an image with open cv**

Figure 4.1 represents the architecture diagram depicts a system that people use to abstract the software system's overall outline and build constraints, relations, and boundaries between components. Image preprocessing,edge detection,color quantization,image filtering,contour detection. The resulting image will have a cartoon-like appearance. Depending on the implementation, additional steps may be involved, such as color mapping or texture mapping. The specific implementation details may vary, but the general architecture for cartoonifying an image with OpenCV typically involves these stages.

4.2 Design Phase

4.2.1 Data Flow Diagram for cartoonify an image with open cv

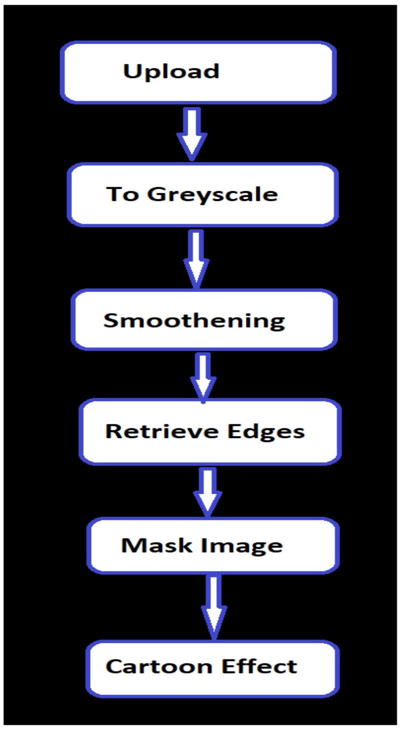


Figure 4.2: **Data flow for cartoonify an image with open cv**

Figure 4.2 represents the process of cartoonifying an image involves a series of steps, such as reducing the number of colors, detecting edges, and simplifying the details of the image. These steps typically involve various image processing operations, such as filtering, thresholding, and morphological operations. To ensure efficient data flow, it is important to carefully consider the order in which these steps are performed. For example, it may be more efficient to first detect edges using an edge detection algorithm, such as the Canny edge detector, before simplifying the details of the image. In addition, it is important to consider the data types and sizes used in each step, as well as the memory requirements of the overall process. Using smaller data types or breaking the image into smaller regions can help reduce memory usage and improve performance. Overall, optimizing data flow is an important consideration in any image processing task, including cartoonifying an image with OpenCV.

4.2.2 Activity Diagram for cartoonify an image with open cv

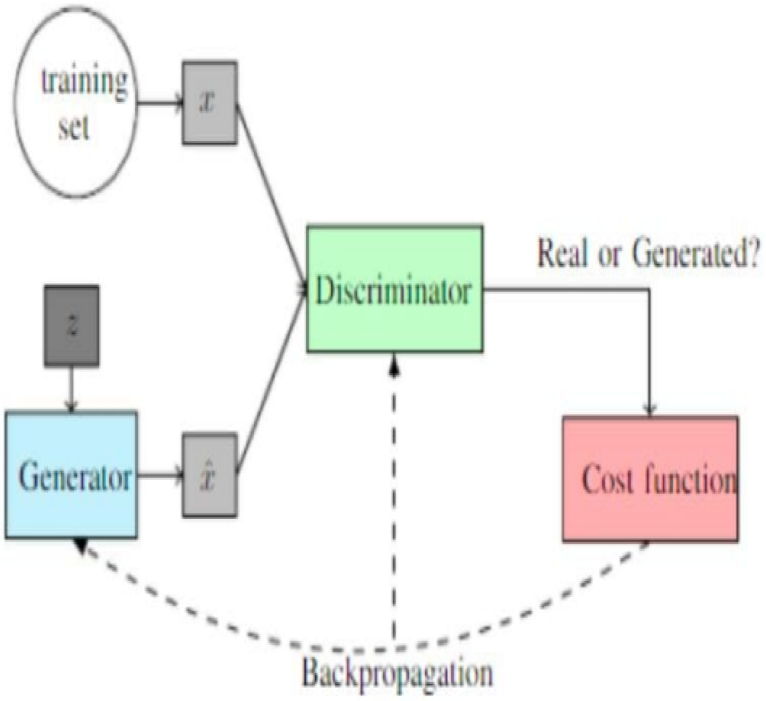


Figure 4.3: **Activity diagraam for cartoonify an image with open cv**

Figure 4.3 represents the static view of an application. In this diagram, we have an Image Input class that represents the original image that we want to cartoonify. The Cartoonify Processor class is responsible for performing the various processing steps needed to cartoonify the image, including edge detection, detail simplification, and color reduction. The Cartoonify Processor has a one-to-many relationship with the Image Input, as it will process many images. The Cartoonified Image class represents the resulting image after it has been processed by the Cartoonify Processor. The Cartoonify Processor has methods for edge detection, detail simplification, and color reduction, which are used to process the input image and produce the output image. These methods could be implemented using OpenCV functions or custom algorithms. This UML diagram is just one example, and the actual implementation of a cartoonify algorithm may differ depending on the specific requirements and design decisions.

4.2.3 Class Diagram for cartoonify an image with open cv

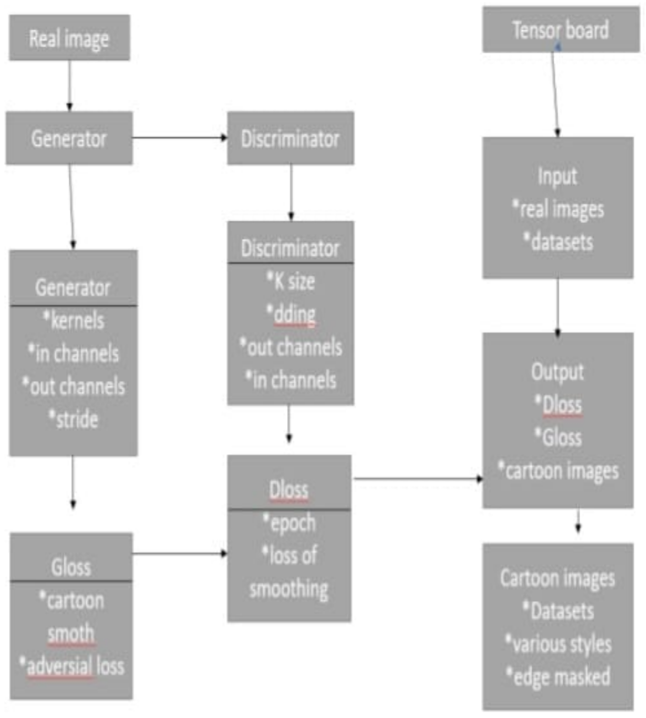


Figure 4.4: **Class diagram for cartoonify an image with open cv**

Figure 4.4 represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application. In this class diagram, we have an ImageProcessor class which represents the process of manipulating an input image to produce an output image. It has three private methods: edge detection(), detail simplification(), and color reduction(), which are used to apply different image processing techniques on the input image. The input image and output image are stored as mat objects. The CartoonifyProcessor class is a subclass of ImageProcessor and adds an additional public method called cartoonify image(). This method applies the edge detection, detail simplification, and color reduction techniques in a specific order to create a cartoon-style image. Finally, we have an ImageCartoonify class, which uses an instance of CartoonifyProcessor to process a given input image and produce the corresponding cartoon-style image. The process image() method takes an input image and returns the processed output image.

4.2.4 Sequence Diagram for cartoonify an image with open cv

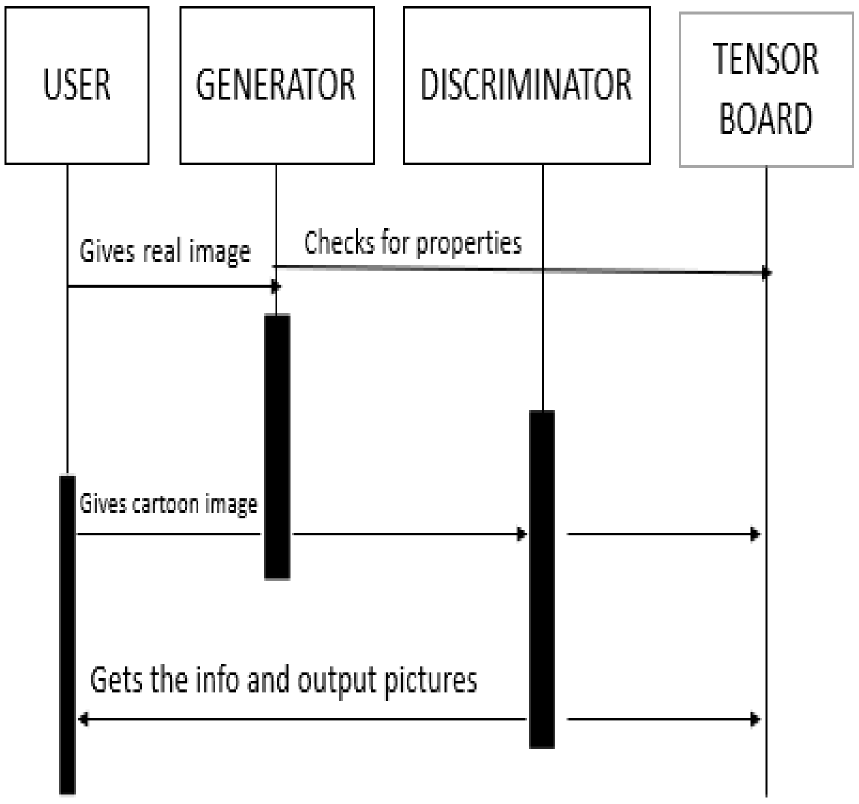


Figure 4.5: **Sequence diagram for cartoonify an image with open cv**

Figure 4.5 represents depicts of the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of scenario. A sequence diagram is a type of UML diagram that depicts interactions between objects or components in a system in a chronological sequence. In the context of a project like cartoonifying an image with OpenCV, a sequence diagram can help to illustrate the flow of data and operations between different components or modules of the system. For example, a sequence diagram for a cartoonifying project might show the steps involved in preprocessing an image, such as smoothing or edge detection, followed by the steps involved in applying a cartoon filter to the image, such as reducing color depth or adding a line drawing effect. The sequence diagram might also show how user input or parameters affect the processing steps, or how the system responds to errors or exceptions.

4.2.5 Usecase Diagram for cartoonify an image with open cv

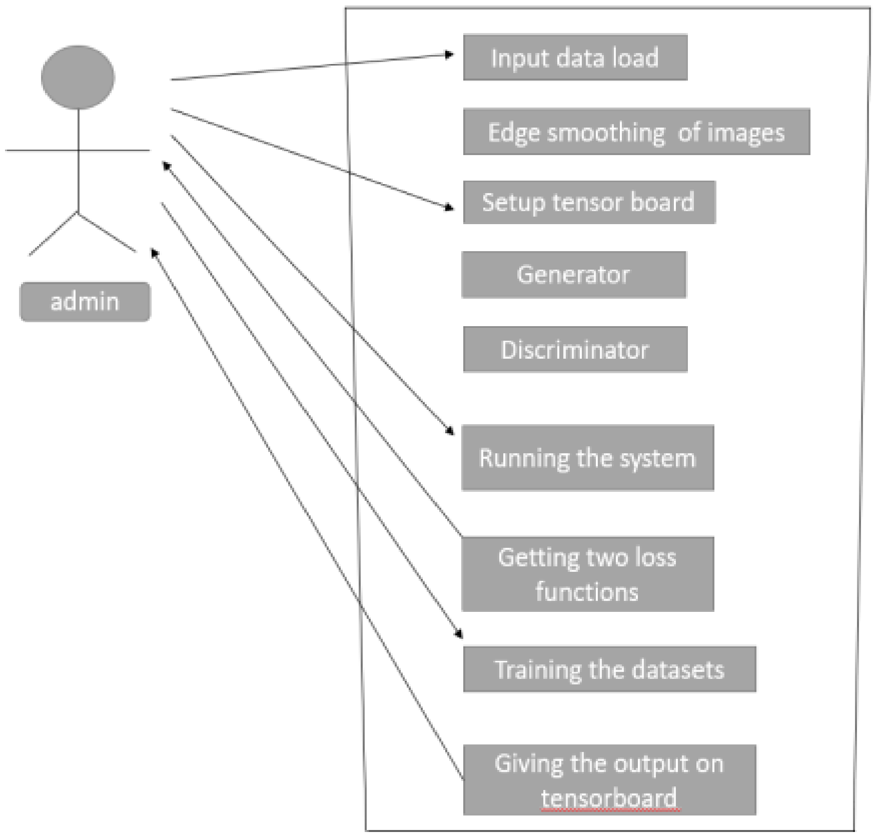


Figure 4.6: **Usecase Diagramfor cartoonify an image with open cv**

Figure 4.6 represents the main functionalities and interactions of a system, including a project like ”cartoonify an image with OpenCV”. Here are some potential use cases for this project: input data load,edge smoothing of images,generator,training the data sets and finally output on tensorboard.

4.3 Generative Adversarial Networks Algorithm and Pseudo Code

4.3.1 Algorithm of Deep Neural Network and Cartoon Stylization

Generative Adversarial Networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other (thus the "adversarial") in order to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation. A generative adversarial network (GAN) is a deep neural network framework which is able to learn from a set of training data and generate new data with the same characteristics as the training data. For example, a generative adversarial network trained on photographs of human faces can generate realistic-looking faces which are entirely new and previously unseen. Generative Adversarial Networks (GANs) are used in Cartoonify an Image. CartoonGAN is a GAN framework for cartoon stylization that takes unpaired photos and cartoon images for training. It proposes two novel losses suitable for cartoonization: a semantic content loss, which is formulated as a sparse regularization in the high-level feature maps of the VGG network; a style loss that encourages the generation of diverse styles. The CartoonGAN model is trained on a large dataset of real-world photos and cartoon images. The model learns to map photos to cartoon images by minimizing the difference between the generated image and the target image. In summary, GANs are used to learn the mapping between real-world photos and cartoon images. The CartoonGAN model is trained on a large dataset of real-world photos and cartoon images to learn this mapping. Once trained, the model can be used to generate cartoon images from real-world photos.

4.3.2 Pseudo Code

```
import scipy.signal as sig
Define the threshold value for the anomaly detection
THRESHOLD = 0.1
Define a function to detect anomalies in a signal
def detect anomalies(signal):
Compute the Fourier transform of the signal
freq, power spectrum = sig.periodogram(signal)
Normalize the power spectrum
normalized spectrum = power spectrum / power spectrum.max()
```

```
Compute the mean and standard deviation of the normalized spectrum
mean spectrum = normalized spectrum.mean()
std spectrum = normalized spectrum.std()
Check if the normalized spectrum contains any values above the threshold
if (normalized spectrum ¿ (mean spectrum + THRESHOLD * std spectrum)).any():
return True
else:
return False
Define a function to monitor the signal and detect anomalies
def monitor signal():
Read in the signal from a data source
signal = read signal()
Check if the signal contains any anomalies
if detect anomalies(signal):
An anomaly was detected - take action
take action()
Wait for a fixed amount of time before checking the signal again time.sleep(10)
Define a function to take action when an anomaly is detected
def take action():
Send an alert to a security team
send alert()
Take steps to contain the intrusion
contain intrusion()
Define a function to send an alert to a security team
def send alert():
Use a communication protocol to send an alert to a security team communication
protocol.send alert()
Define a function to contain an intrusion
def contain intrusion():
Take steps to contain the intrusion, such as blocking network access or shutting
down systems
take steps to contain intrusion()
Define a function to read in the signal from a data source
def read signal():
Read in the signal from a data source, such as a network traffic log or sensor data
```

```
signal = read signal from data source()
Preprocess the signal, if necessary, to prepare it for analysis
preprocessed signal = preprocess signal(signal)
return preprocessed signal
Define a function to preprocess the signal, if necessary def preprocess signal(signal):
Preprocess the signal, such as filtering out noise or removing outliers
preprocessed signal = preprocess(signal)
return preprocessed signal
Define a function to take steps to contain an intrusion
def take steps to contain intrusion():
Take steps to contain the intrusion, such as blocking network access or shutting down systems
steps to contain intrusion()
Log the steps that were taken to contain the intrusion
log steps to contain intrusion()
Define a function to log the steps that were taken to contain an intrusion
def log steps to contain intrusion():
Use a logging mechanism to log the steps that were taken to contain the intrusion
logging.log steps to contain intrusion()
Define a function to handle communication protocol for alerting
def communication protocol():
```

4.4 Module Description

4.4.1 Anomaly Detection Model

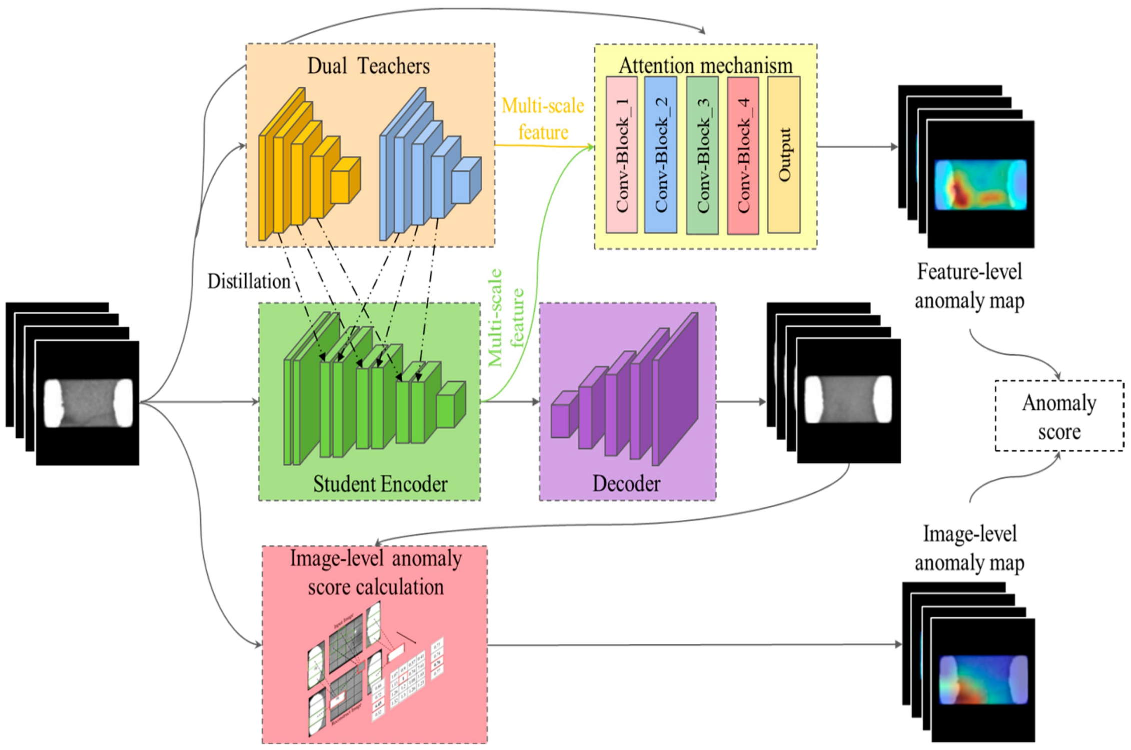


Figure 4.7: **Anomaly Detection Model**

Figure 4.7 represents the Image anomaly detection is a trending research topic in computer vision. The objective is to build models using available normal samples to detect various abnormal images without depending on real abnormal samples. It has high research significance and value for applications in the detection of defects in product appearance, medical image analysis, hyperspectral image processing, and other fields. This paper proposes an image anomaly detection algorithm based on feature distillation and an autoencoder structure, which uses the feature distillation structure of a dual-teacher network to train the encoder, thus suppressing the reconstruction of abnormal regions.

4.4.2 Signature based Detection-Packet Capture

We employ the knowledge base and the resulting attack rule base for signature-based detection. Known attack rules are kept in a knowledge base. We employ a Signature Apriori method (offline), which, as previously said, uses the recorded packet and known attack rules as input to construct derived attack rules. In the derived attack rule base, derived attack rules are kept. Here, Snort compares these packets to the rules in the attack rule base and knowledge base to see if there is any correlation. If it discovers any intrusive packets, it analyses the type of attack and provides a warning to the score calculation. Applying captured packets to the network traffic feature extractor follows.

4.4.3 Network Traffic Feature Extractor

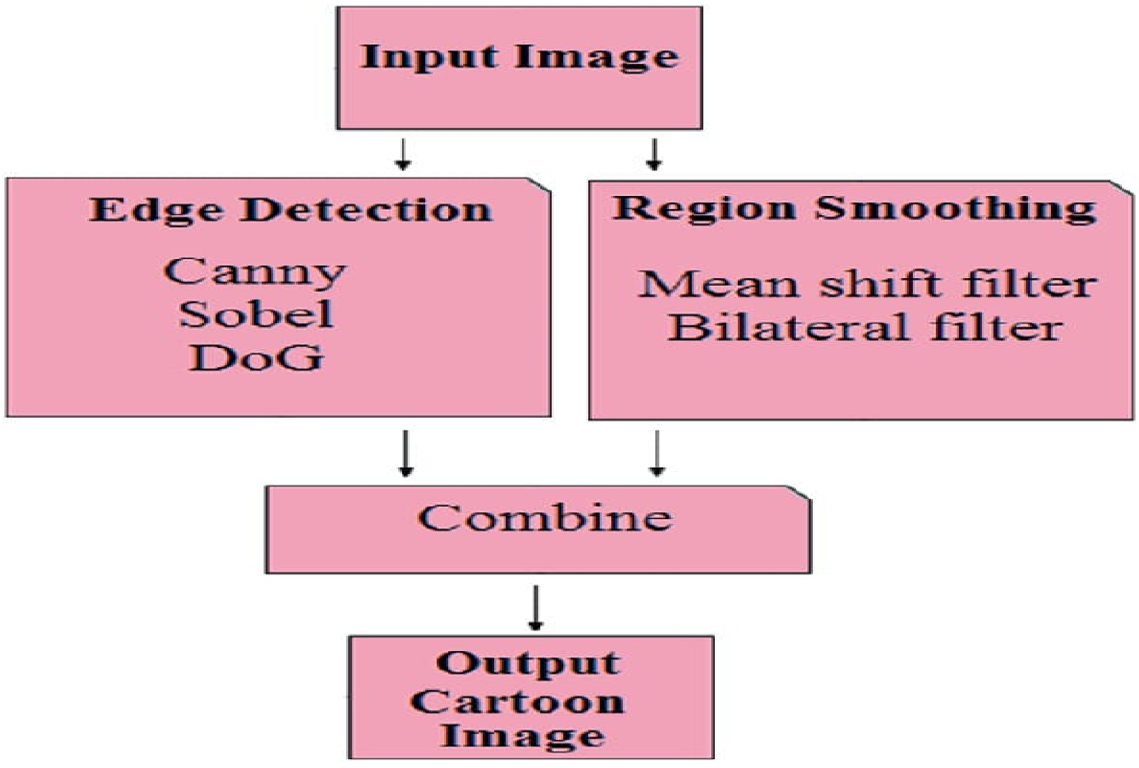


Figure 4.8: **Network Traffic Extractor**

From the network packets that were collected, it produces profiles of the network traffic. Every packet containing an AN is kept in the progress affiliation cache. An affiliation might be a series of packets moving to and from source to target while following a specific protocol and at a specific time. Any affiliation that has been finished (e.g., closed, reset, or terminated connection) will remain in the list of fully finished affiliations. For each successfully completed affiliation, a network profile is created by extracting the network options from the raw packets, including the basic choices, options relating to traffic (network traffic statistics), content (payload), and options related to virtualization.

4.5 Steps to Execute/Run/Implement the Project

4.5.1 Setting up the Environment

- Install the required programming languages and frameworks.
- Set up the development environment on your local machine.
- The project repository from the version control system

4.5.2 Installing Dependencies

- Identify project dependencies and requirements.
- Install necessary libraries, packages, or modules using the package manager.
- Ensure all external dependencies are correctly configured.

4.5.3 Configuring Database and Environment Variables

- Set up and configure the project's database.
- Create environment variables or configuration files for sensitive information.
- Test the connection to the database to ensure proper configuration.

4.5.4 Building the Project

- Run build scripts or commands to compile the project source code.
- Handle any build-time configurations or optimizations.
- Resolve any build errors or issues that may arise.

4.5.5 Running Tests

- Write and execute unit tests to ensure the functionality of individual components.
- Run integration tests to check the collaboration between different parts of the system.
- Resolve any failing tests and ensure a high test coverage.

4.5.6 Launching the Application

- Start the application locally to ensure it runs without errors.
- Verify that the user interfaces are responsive and functional.
- Check the application logs for any runtime issues.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 cartoonifying Image Design

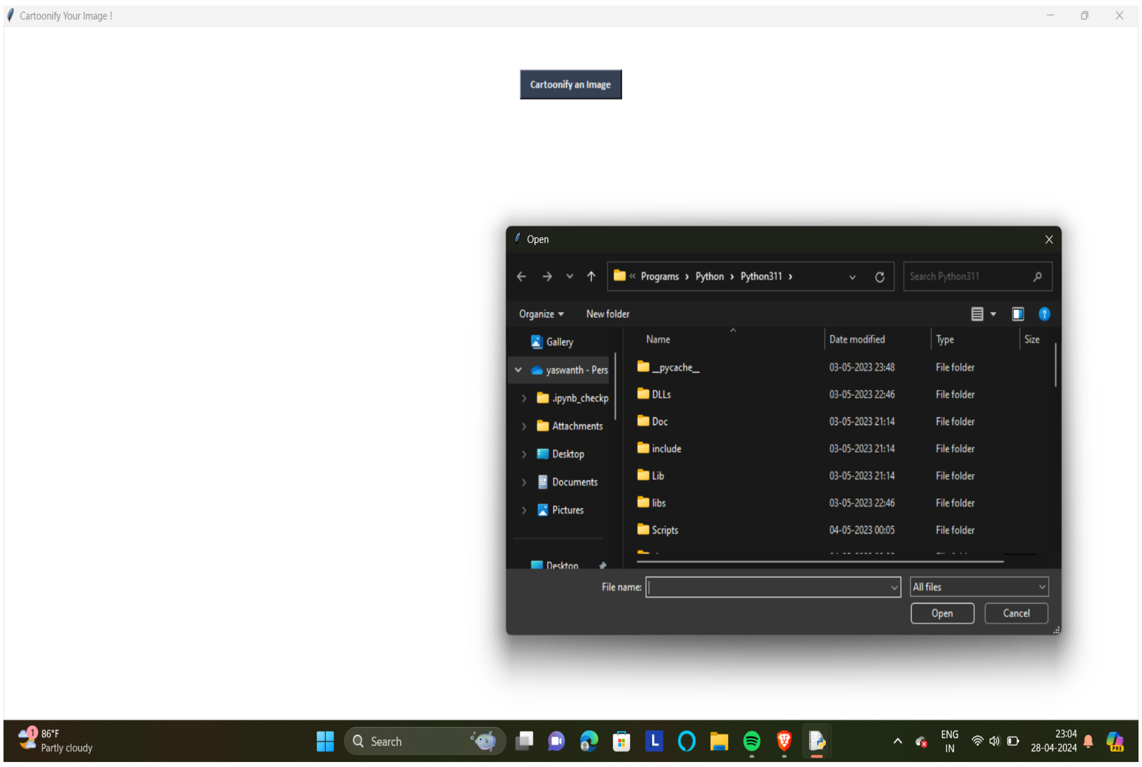


Figure 5.1: **cartoonifying Image Design**

Figure5.1represents the input which used for system is the dataset which consisting of real images. Either they maybe same pictures or else they can different pictures too.we can even give the cartoonized pictures dataset which can discriminated to real images.we had used various datasets.It contains process of original image, parameters for cartoonification,NumBilateralFilters,Edgethreshold,Colorreductionfactor,OpenCVs libraries these inputs, you can use OpenCV to cartoonify your image by applying a series of filters and transformations.

5.1.2 cartoonified image Design

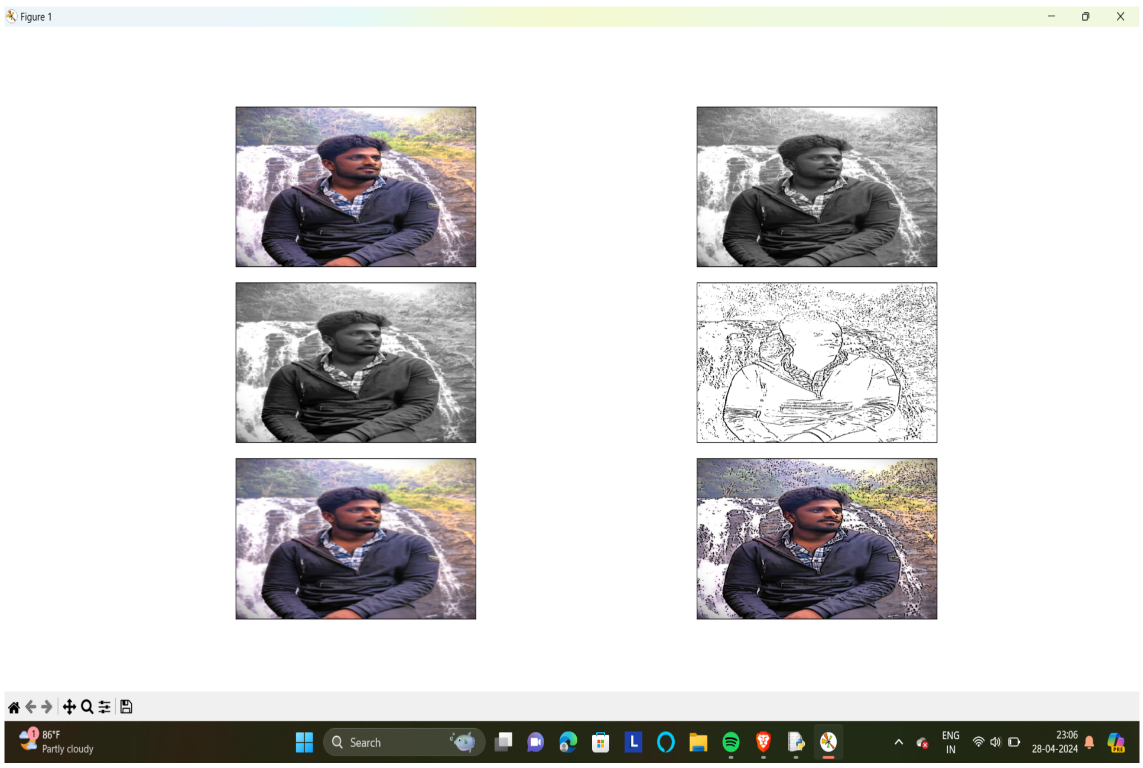


Figure 5.2: **cartoonified image**

Figure5.2 represents the output design which we get from the system is of mainly cartoonified image datasets.they of of different feauters and coloure cmpositions as based on there original features. Based on visual appeal,clarity,usability,customization ,speed and performance. This might involve optimizing the code to minimize processing time or reducing the size of the output to improve rendering performance.

5.2 Testing

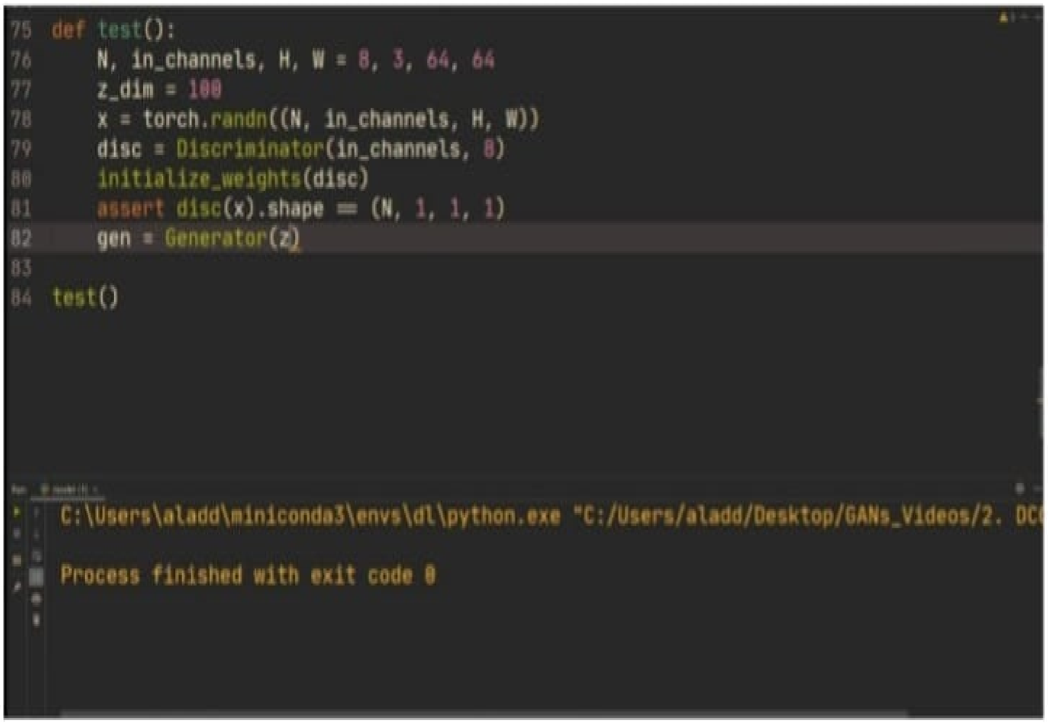
Testing to our system is done through various sources. It can be tested using Pytorch,jupyter notebook as well.The thing which need to be noticed is to import all the libraries required. It contains types of testing are input image testing,parameter testing,output testing,performance testing and user testing.

5.3 Types of Testing

5.3.1 Unit testing

A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system. In most programming languages, that is a function, a subroutine, a method or property. The isolated part of the definition is important. In his book ”Working Effectively with Legacy Code”, author Michael Feathers states that such tests are not unit tests when they rely on external systems: “If it talks to the database, it talks across the network, it touches the file system, it requires system configuration, or it can’t be run at the same time as any other test.”As our code is efficient for unit testing as we can test the units of the code to get the results accurately.

Test Result

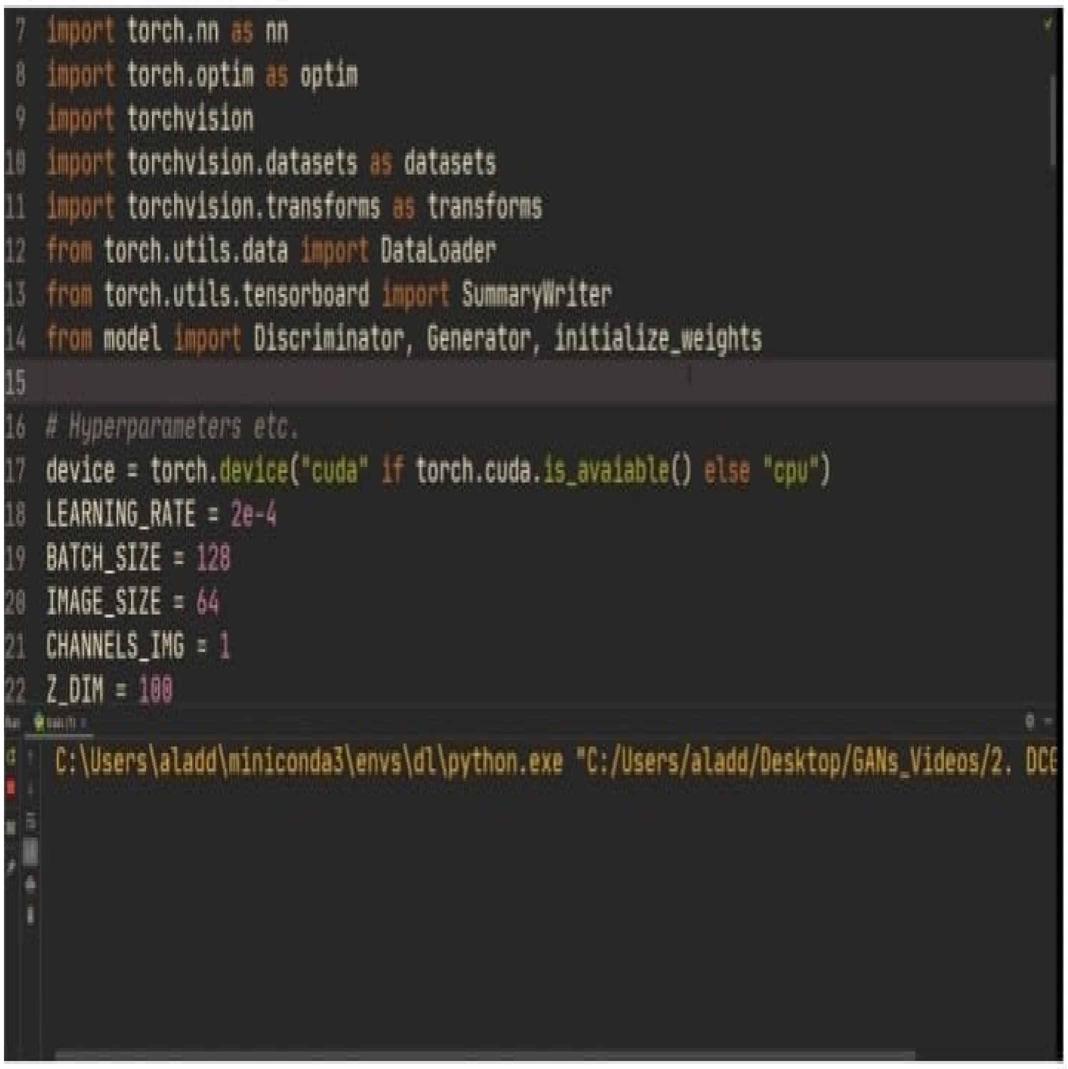


```
def test():
    N, in_channels, H, W = 8, 3, 64, 64
    z_dim = 100
    x = torch.randn((N, in_channels, H, W))
    disc = Discriminator(in_channels, 8)
    initialize_weights(disc)
    assert disc(x).shape == (N, 1, 1, 1)
    gen = Generator(z)

test()
```

```
C:\Users\aladd\miniconda3\envs\dl\python.exe "C:/Users/aladd/Desktop/GANs_Videos/2. DC

Process finished with exit code 0
```

Figure 5.3: **Image storing process**

Figure 5.1 represents a method imread in cv2 which is used to store images in the form of numbers. This helps us to perform operations according to our needs. The image is read as a numpy array, in which cell values depict R, G, and B values of a pixel.



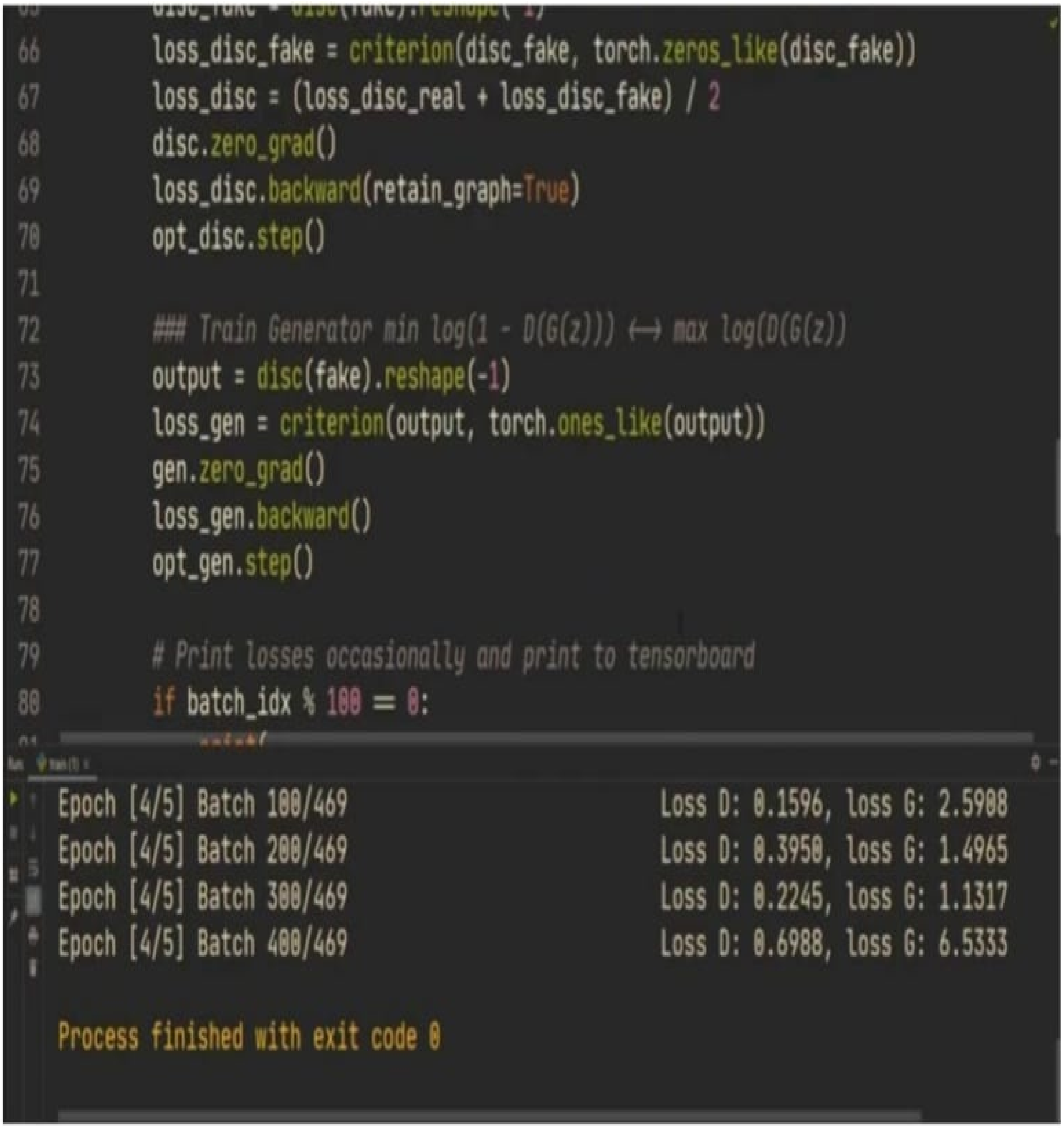
```
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
from model import Discriminator, Generator, initialize_weights

# Hyperparameters etc.
device = torch.device("cuda" if torch.cuda.is_avaiable() else "cpu")
LEARNING_RATE = 2e-4
BATCH_SIZE = 128
IMAGE_SIZE = 64
CHANNELS_IMG = 1
Z_DIM = 100
```

```
C:\Users\aladd\miniconda3\envs\dl\python.exe "C:/Users/aladd/Desktop/GANs_Videos/2. DCG
```

Figure 5.4: **Transforming an image to grayscale**

Figure 5.2 Is used to transform an image into the colour-space mentioned as‘flag’. Here, our first step is to convert the image into grayscale. Thus, we use the BGR2GRAY flag. This returns the image in grayscale. A grayscale image is stored as gray scale image. After each transformation, we resize the resultant image using the resize() method in cv2 and display it using imshow() method. This is done to get more clear insights into every single transformation step.

5.3.2 Test Result



```
        loss_disc_fake = criterion(disc_fake, torch.zeros_like(disc_fake))
        loss_disc = (loss_disc_real + loss_disc_fake) / 2
        disc.zero_grad()
        loss_disc.backward(retain_graph=True)
        opt_disc.step()

        ### Train Generator min log(1 - D(G(z))) <-> max log(D(G(z))
        output = disc(fake).reshape(-1)
        loss_gen = criterion(output, torch.ones_like(output))
        gen.zero_grad()
        loss_gen.backward()
        opt_gen.step()

        # Print losses occasionally and print to tensorboard
        if batch_idx % 100 == 0:
```

```
Epoch [4/5] Batch 100/469                    Loss D: 0.1596, loss G: 2.5908
Epoch [4/5] Batch 200/469                    Loss D: 0.3950, loss G: 1.4965
Epoch [4/5] Batch 300/469                    Loss D: 0.2245, loss G: 1.1317
Epoch [4/5] Batch 400/469                    Loss D: 0.6988, loss G: 6.5333

Process finished with exit code 0
```

Figure 5.5: **Cartoon Effect**

Figure 5.3 represents the combination of two specialties. This will be done using masking. We perform bitwise and on two images to mask them. Images will be just numbers. So that's how we mask edged image on our "BEAUTIFY" image.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The system which was proposed by us is the efficient in many ways to cartoonize the real pictures.And the technique used to build the software is soo cost less as well as easy to build just some good effort is needed. GAN can be the best choice for us to choose to cartoonify the images because even it is hard to build but it gives the results accurately and effectively.As we can implement many types of datasets on this system.

6.2 Comparison of Existing and Proposed System

The main thing to compare between the existing and our system is the technique used to build the software. they used the regular techniques to cartoonize but we had changed it by introducing GAN to build a software which can give more outputs.Time consuming is less compared to the old systems.the clarity and edge has been developed using some more properties.

6.3 Sample Code

```
import os
import io
import uuid
import sys
import yaml
import traceback

with open('./config.yaml', 'r') as fd:
    opts = yaml.safe_load(fd)

sys.path.insert(0, './white_box_cartoonizer/')

```

```python
import cv2
from flask import Flask, render_template, make_response, flash
import flask
from PIL import Image
import numpy as np
import skvideo.io
if opts['colab-mode']:
    from flask_ngrok import run_with_ngrok #to run the application on colab using ngrok


from cartoonize import WB_Cartoonize

if not opts['run_local']:
    if 'GOOGLE_APPLICATION_CREDENTIALS' in os.environ:
        from gcloud_utils import upload_blob, generate_signed_url, delete_blob, download_video
    else:
        raise Exception("GOOGLE_APPLICATION_CREDENTIALS not set in environment variables")
    from video_api import api_request
    # Algorithmia (GPU inference)
    import Algorithmia

app = Flask(name)
if opts['colab-mode']:
    run_with_ngrok(app)   #starts ngrok when the app is run

app.config['UPLOAD_FOLDER_VIDEOS'] = 'static/uploaded_videos'
app.config['CARTOONIZED_FOLDER'] = 'static/cartoonized_images'

app.config['OPTS'] = opts

## Init Cartoonizer and load its weights
wb_cartoonizer = WB_Cartoonize(os.path.abspath("white_box_cartoonizer/saved_models/"), opts['gpu'])

def convert_bytes_to_image(img_bytes):
    """Convert bytes to numpy array

    Args:
        img_bytes (bytes): Image bytes read from flask.

    Returns:
        [numpy array]: Image numpy array
    """

    pil_image = Image.open(io.BytesIO(img_bytes))
    if pil_image.mode=="RGBA":
        image = Image.new("RGB", pil_image.size, (255,255,255))
        image.paste(pil_image, mask=pil_image.split()[3])
    else:
        image = pil_image.convert('RGB')

    image = np.array(image)

    return image

@app.route('/')
@app.route('/cartoonize', methods=["POST", "GET"])
def cartoonize():
    opts = app.config['OPTS']
    if flask.request.method == 'POST':
        try:
            if flask.request.files.get('image'):
                img = flask.request.files["image"].read()

                ## Read Image and convert to PIL (RGB) if RGBA convert appropriately
                image = convert_bytes_to_image(img)

                img_name = str(uuid.uuid4())

                cartoon_image = wb_cartoonizer.infer(image)

                cartoonized_img_name = os.path.join(app.config['CARTOONIZED_FOLDER'], img_name + ".
                    jpg")
                cv2.imwrite(cartoonized_img_name, cv2.cvtColor(cartoon_image, cv2.COLOR_RGB2BGR))

                if not opts["run_local"]:
                    # Upload to bucket
                    output_uri = upload_blob("cartoonized_images", cartoonized_img_name, img_name +
                        ".jpg", content_type='image/jpg')

                    # Delete locally stored cartoonized image
                    os.system("rm " + cartoonized_img_name)
                    cartoonized_img_name = generate_signed_url(output_uri)


                return render_template("index_cartoonized.html", cartoonized_image=
                    cartoonized_img_name)

            if flask.request.files.get('video'):

                filename = str(uuid.uuid4()) + ".mp4"
                video = flask.request.files["video"]
                original_video_path = os.path.join(app.config['UPLOAD_FOLDER_VIDEOS'], filename)
                video.save(original_video_path)

                modified_video_path = os.path.join(app.config['UPLOAD_FOLDER_VIDEOS'], filename.
                    split(".")[0] + "_modified.mp4")

                ## Fetch Metadata and set frame rate
                file_metadata = skvideo.io.ffprobe(original_video_path)
                original_frame_rate = None
                if 'video' in file_metadata:
                    if '@r_frame_rate' in file_metadata['video']:
                        original_frame_rate = file_metadata['video']['@r_frame_rate']

                if opts['original_frame_rate']:
                    output_frame_rate = original_frame_rate
                else:
                    output_frame_rate = opts['output_frame_rate']

                output_frame_rate_number = int(output_frame_rate.split('/')[0])

                #change the size if you want higher resolution :
                ############################
                # Recommnded width_resize  #
                ############################
                #width_resize = 1920 for 1080p: 1920x1080.
                #width_resize = 1280 for 720p: 1280x720.
                #width_resize = 854 for 480p: 854x480.
                #width_resize = 640 for 360p: 640x360.
                #width_resize = 426 for 240p: 426x240.
                width_resize=opts['resize-dim']

                # Slice, Resize and Convert Video as per settings
                if opts['trim-video']:
                    #change the variable value to change the time_limit of video (In Seconds)
                    time_limit = opts['trim-video-length']
                    if opts['original_resolution']:
                        os.system("ffmpeg -hide_banner -loglevel warning -ss 0 -i '{}' -t {} -filter:v scale=-1:-2 -r {} -c:a copy '{}'".format(os.path.abspath(original_video_path), time_limit, output_frame_rate_number, os.path.abspath(modified_video_path)))
                    else:
                        os.system("ffmpeg -hide_banner -loglevel warning -ss 0 -i '{}' -t {} -filter:v scale={}:-2 -r {} -c:a copy '{}'".format(os.path.abspath(original_video_path), time_limit, width_resize, output_frame_rate_number, os.path.abspath(modified_video_path)))
                else:
                    if opts['original_resolution']:
                       os.system("ffmpeg -hide_banner -loglevel warning -ss 0 -i '{}' -filter:v scale=-1:-2 -r {} -c:a copy '{}'".format(os.path.abspath(original_video_path), output_frame_rate_number, os.path.abspath(modified_video_path)))
                    else:
                        os.system("ffmpeg -hide_banner -loglevel warning -ss 0 -i '{}' -filter:v scale={}:-2 -r {} -c:a copy '{}'".format(os.path.abspath(original_video_path), width_resize, output_frame_rate_number, os.path.abspath(modified_video_path)))

                audio_file_path = os.path.join(app.config['UPLOAD_FOLDER_VIDEOS'], filename.split(".")[0] + "_audio_modified.mp4")
```

```
                os.system("ffmpeg -hide_banner -loglevel warning -i '{}' -map 0:1 -vn -acodec copy -strict -2 '{}'".format(os.path.abspath(modified_video_path), os.path.abspath(audio_file_path)))

                if opts["run_local"]:
                    cartoon_video_path = wb_cartoonizer.process_video(modified_video_path, output_frame_rate)
                else:
                    data_uri = upload_blob("processed_videos_cartoonize", modified_video_path, filename, content_type='video/mp4', algo_unique_key='cartoonizeinput')
                    response = api_request(data_uri)
                    # Delete the processed video from Cloud storage
                    delete_blob("processed_videos_cartoonize", filename)
                    cartoon_video_path = download_video('cartoonized_videos', os.path.basename(response['output_uri']), os.path.join(app.config['UPLOAD_FOLDER_VIDEOS'], filename.split(".")[0] + "_cartoon.mp4"))

                ## Add audio to the cartoonized video
                final_cartoon_video_path = os.path.join(app.config['UPLOAD_FOLDER_VIDEOS'], filename.split(".")[0] + "_cartoon_audio.mp4")
                os.system("ffmpeg -hide_banner -loglevel warning -i '{}' -i '{}' -codec copy -shortest '{}'".format(os.path.abspath(cartoon_video_path), os.path.abspath(audio_file_path), os.path.abspath(final_cartoon_video_path)))

                # Delete the videos from local disk
                os.system("rm {} {} {} {}".format(original_video_path, modified_video_path, audio_file_path, cartoon_video_path))

                return render_template("index_cartoonized.html", cartoonized_video=final_cartoon_video_path)

        except Exception:
            print(traceback.print_exc())
            flash("Our server hiccuped :/ Please upload another file! :)")
            return render_template("index_cartoonized.html")
    else:
        return render_template("index_cartoonized.html")

if name == "main":
    # Commemnt the below line to run the Appication on Google Colab using ngrok
    if opts['colab-mode']:
        app.run()
    else:
        app.run(debug=False, host='127.0.0.1', port=int(os.environ.get('PORT', 8080)))
```

Output

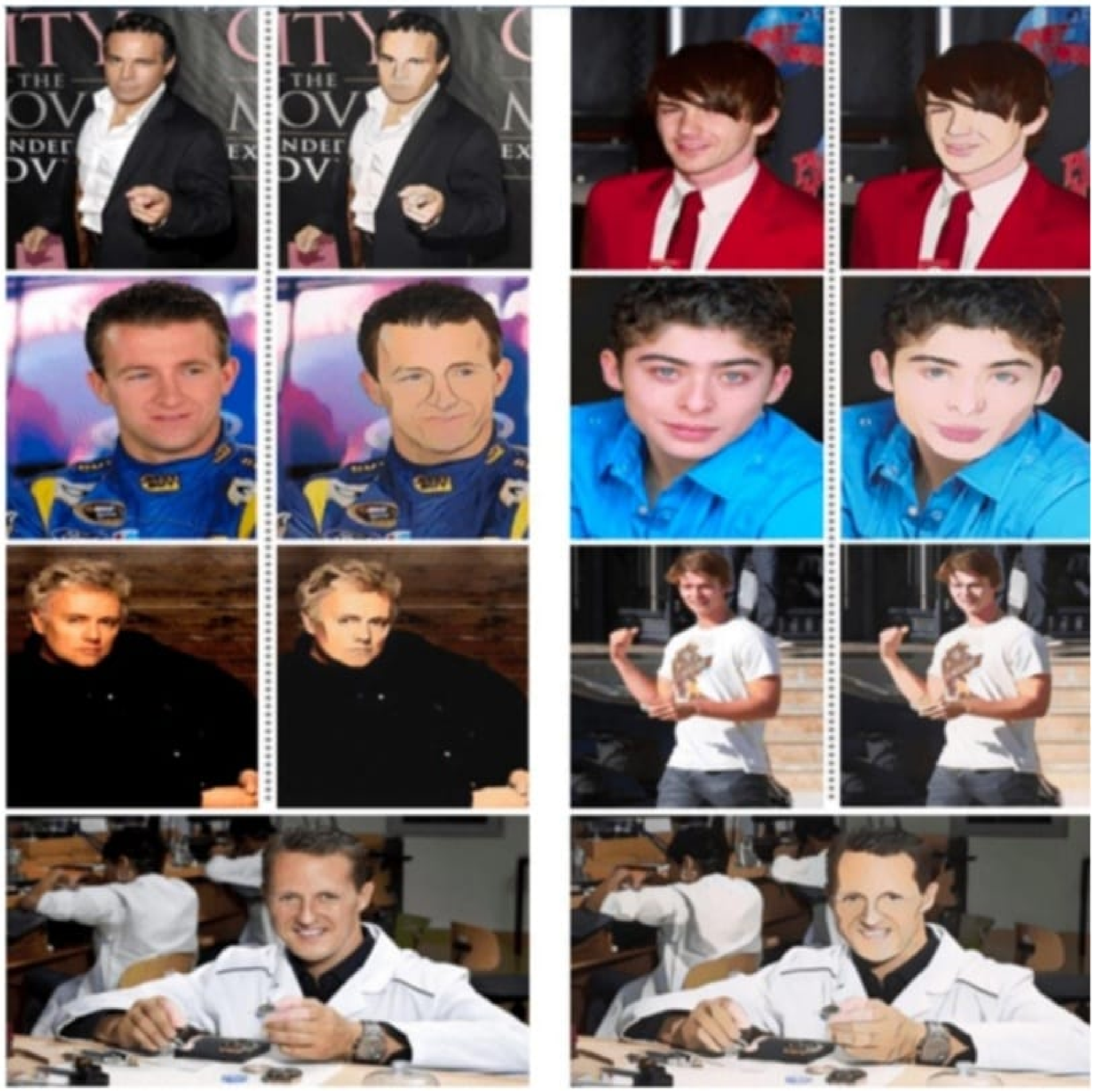


Figure 6.1: **cartoonified image output design 1**

Figure 6.1 represents the implementation details of the algorithm used for cartoonification.Finally general or blured image can transfered in animated images in all pictures.

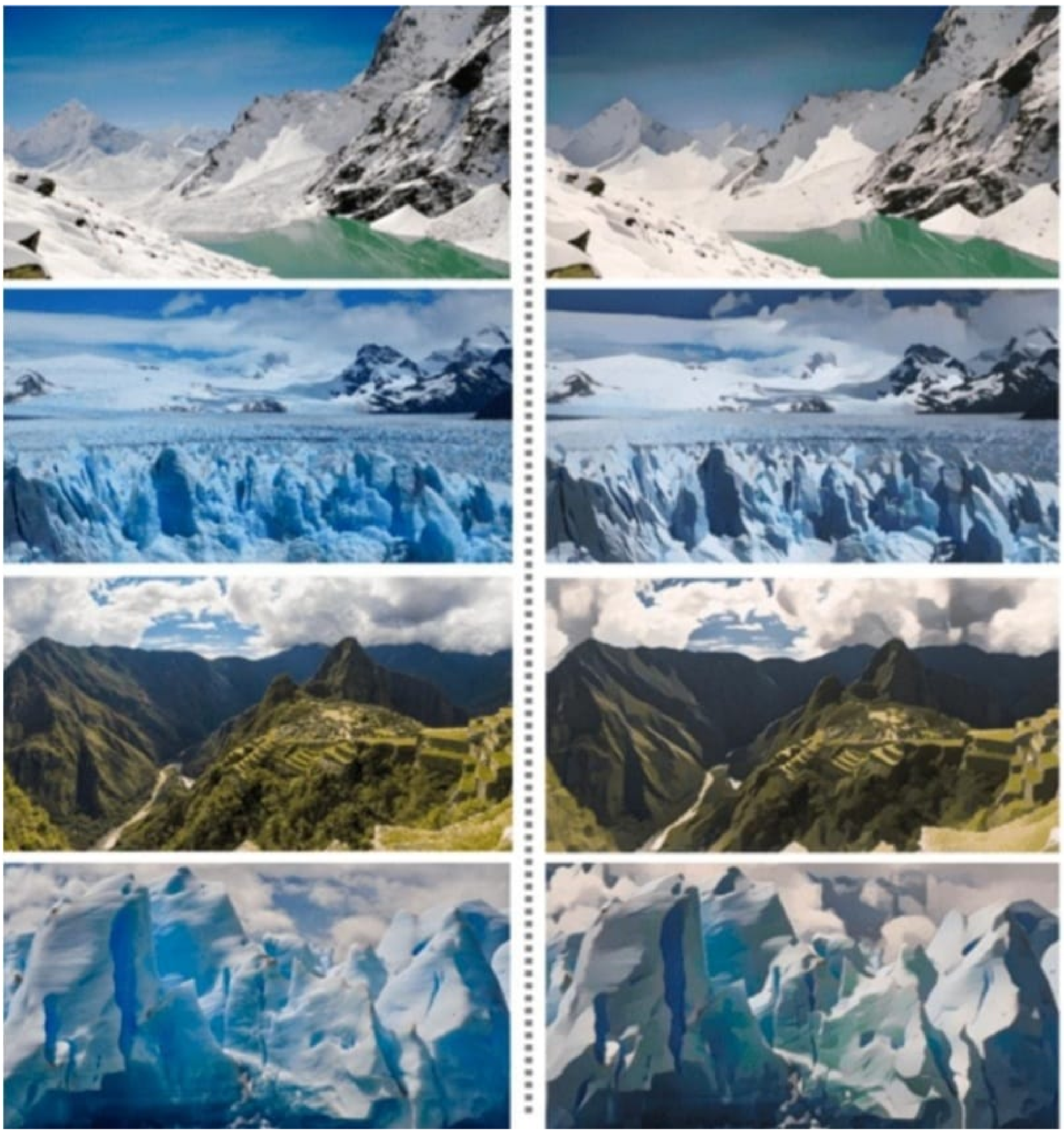


Figure 6.2: **cartoonified image output design 2**

Figure 6.2 represents the output of the project would be a cartoon-like version of the input image that emphasizes certain visual features and simplifies others, resulting in a stylized and visually appealing output.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

Many Companies analyzed that Advertising their products using Cartoon images will helps them to attract people. Nowadays many schools are introducing Cartoonified books to the children. As in the World which is rapidly Increasing Technology Cartoonizing plays a great role in various aspects. Nowadays, Cartoons are used primarily for conveying political commentary and editorial opinion in Newspapers and also used in social comedy and Magazines. Thus our project helps various sectors which includes cartoonization. It is important to preprocess the image before applying any filter or effect. This includes resizing the image to a smaller size, converting it to grayscale, and smoothing it to remove any noise or unwanted details. By carefully considering each step and its impact on the image, it is possible to create a wide range of unique and interesting cartoon-style images using OpenCV.

7.2 Future Enhancements

In future work it would like to focus more on converting longer videos, high quality videos/images and bigger file size. We would also like to work on adding feature that enables user to share the result on various social platforms.Our work would be helpful for future developers regarding this topic.As this work can helpful in various sectors in the future like animation and cartoonization using simplest form of techniques. There are many exciting opportunities for future enhancement of the cartoonify image process using OpenCV. By exploring these and other possibilities, it is possible to create even more advanced and realistic cartoon-style images.

Chapter 8

PLAGIARISM REPORT

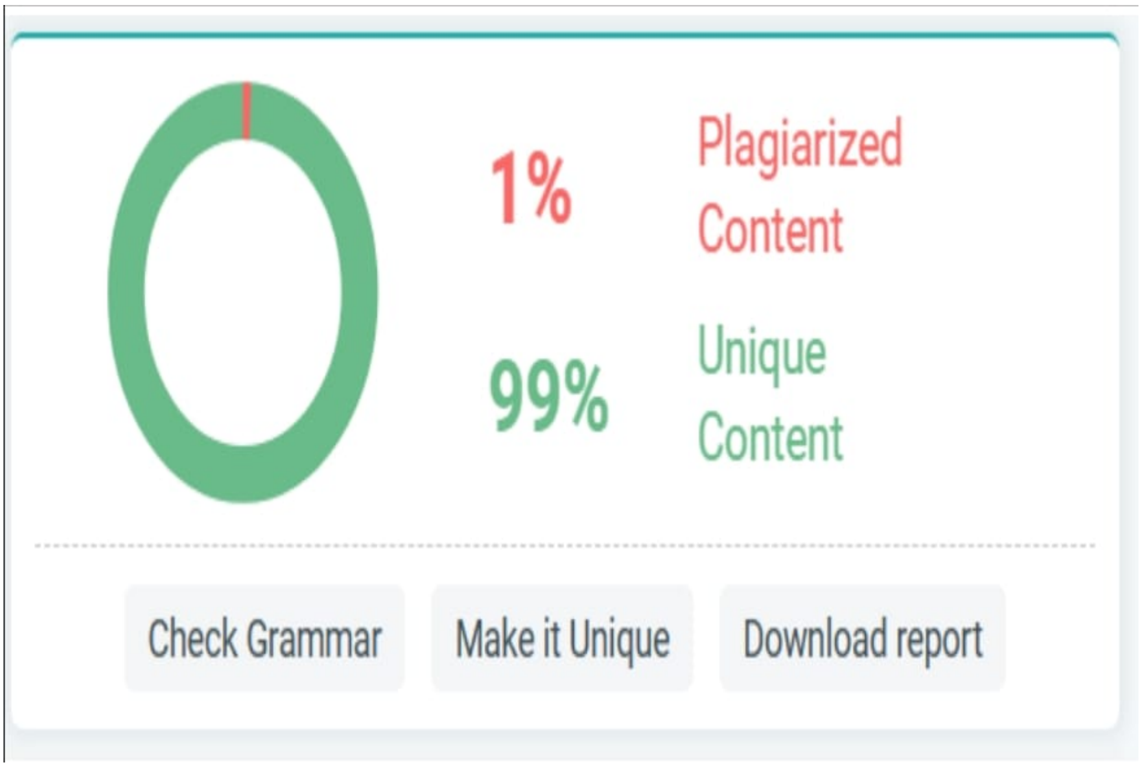


Figure 8.1: **Plagiarism**

Chapter 9

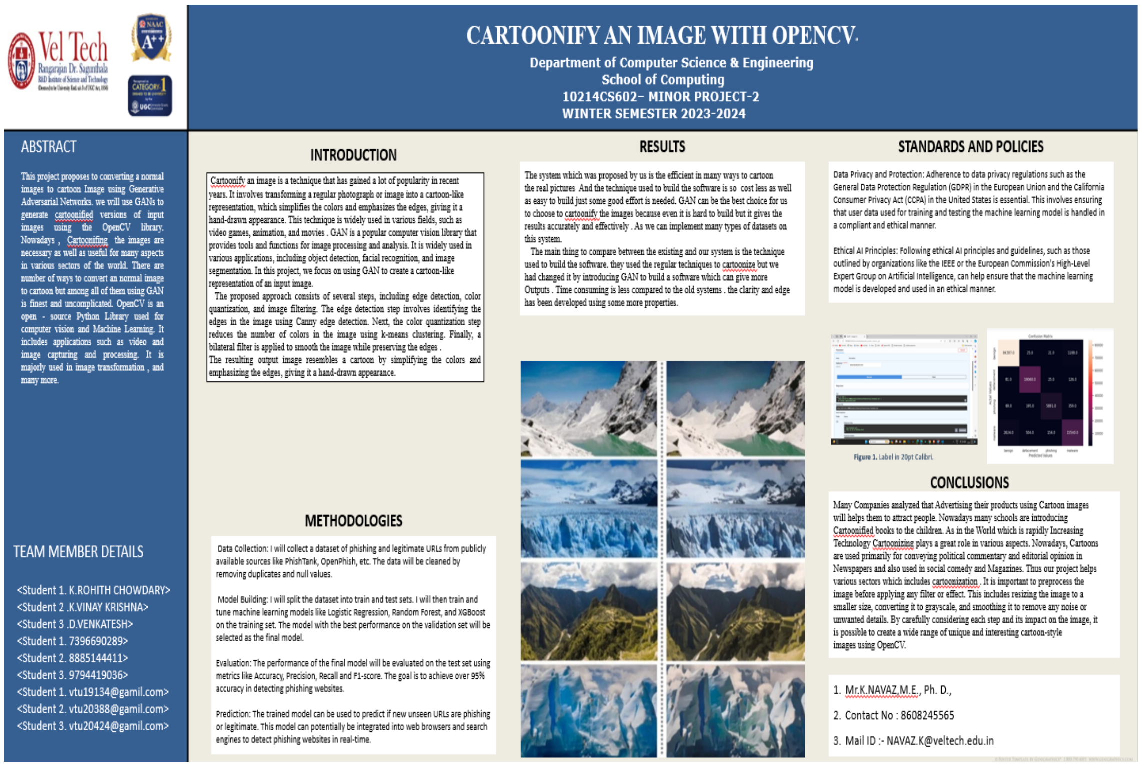
SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
from scapy . all import *
def ddos Test ( src , dst , iface , count ) :
pkt =IP ( src = src , dst = dst ) / ICMP( type =8 , id =678 ) / Raw ( load =     1234     )
send ( pkt , iface = iface , count =count )
pkt = IP ( src = src , dst = dst ) / ICMP( type =0 ) / Raw ( load =   AAAAAAAAAA    )
send ( pkt , iface = iface , count =count )
pkt = IP ( src = src , dst = dst ) / UDP( dport =31335 ) / Raw ( load =    PONG     )
send ( pkt , iface = iface , count =count )
pkt = IP ( src = src , dst = dst ) / ICMP( type =0 , id =456 )
send ( pkt , iface = iface , count = count )
src = str ( input (     Enter the src Ip :      ) )
dst = str ( input (     Enter the target Ip :      ) )
iface =     eth0
count =1
ddos Test ( src , dst , iface , count )
from scapy . all import *
import opt parse
from random import rand int
def exploit Test ( src , dst , iface , count ) :
pkt = IP ( src =src , dst = dst ) / UDP( dport =518 ) \
/ Raw ( load =    \ x01 \ x03 \ x00 \ x00 \ x00 \ x00 \ x00 \ x01 \ x00 \ x02 \ x02 \xE8      )
send ( pkt , iface = iface , count= count )
pkt = IP ( src =src , dst = dst ) / UDP( dport =635 ) \
/ Raw ( load =      \ xB0\ x02 \ x89 \ x06 \xFE\xC8\ x89F \ x04 \xB0\ x06 \ x89F      )
send ( pkt , iface = iface , count =count )
def scan Test ( src , dst , iface , count ) :
pkt = IP ( src =src , dst = dst ) / UDP( dport =7) \
/ Raw ( load =     cyber cop      )
send ( pkt )
pkt = IP ( src =src , dst = dst ) / UDP( dport =10080 ) \
/ Raw ( load =     Amanda      )
send ( pkt , iface = iface , count = count )
def main ( ) :
parser = opt pars e . Option Parser (     usageprog   +  −i iface  −s  src  −t   target   −c
        count
```

```
parser . add option (          i       , dest =      iface       , type =      string       , help =
     specify network interface       )
parser . add option (     s          , dest =      src       , type =      string       , help =      s pecify
      source address      )
parser . add option (          t       , dest =      tgt       , type =      string       , help =      specify
      target address      )
parser . add optio n (     c          , dest =      count      , type =      int      , help =      s pecify
      packet count      )
( options , args ) = parser . parseargs ( )
# if options . iface == None :
# iface =      eth0
# else :
iface = options . iface
# if options . src == None :
# sr c =      .      . join ( [ str ( rand int ( 1,254 ) ) for x in range ( 4 ) ] )
# else :
src = options . src
# if options . tgt == None :
# print ( parser . usage )
# exit ( 0 )
# else :
dst = options . tgt
# if options . count == None :
# count = 4
# else :
count = options . count
exploit Test ( src , dst , iface , count )s
can Test ( src , dst , iface , count )
if name
main ( )
==       main       :
```

9.2 Poster Presentation



CARTOONIFY AN IMAGE WITH OPENCV.

Department of Computer Science & Engineering
School of Computing
10214CS602– MINOR PROJECT-2
WINTER SEMESTER 2023-2024

ABSTRACT

This project proposes to converting a normal images to cartoon Image using Generative Adversarial Networks. we will use GANs to generate cartoonified versions of input images using the OpenCV library. Nowadays , Cartoonifing the images are necessary as well as useful for many aspects in various sectors of the world. There are number of ways to convert an normal image to cartoon but among all of them using GAN is finest and uncomplicated. OpenCV is an open - source Python Library used for computer vision and Machine Learning. It includes applications such as video and image capturing and processing. It is majorly used in image transformation , and many more.

TEAM MEMBER DETAILS

<Student 1. K.ROHITH CHOWDARY>
<Student 2 .K.VINAY KRISHNA>
<Student 3 .D.VENKATESH>
<Student 1. 7396690289>
<Student 2. 8885144411>
<Student 3. 9794419036>
<Student 1. vtu19134@gamil.com>
<Student 2. vtu20388@gamil.com>
<Student 3. vtu20424@gamil.com>

INTRODUCTION

Cartoonify an image is a technique that has gained a lot of popularity in recent years. It involves transforming a regular photograph or image into a cartoon-like representation, which simplifies the colors and emphasizes the edges, giving it a hand-drawn appearance. This technique is widely used in various fields, such as video games, animation, and movies . GAN is a popular computer vision library that provides tools and functions for image processing and analysis. It is widely used in various applications, including object detection, facial recognition, and image segmentation. In this project, we focus on using GAN to create a cartoon-like representation of an input image.

The proposed approach consists of several steps, including edge detection, color quantization, and image filtering. The edge detection step involves identifying the edges in the image using Canny edge detection. Next, the color quantization step reduces the number of colors in the image using k-means clustering. Finally, a bilateral filter is applied to smooth the image while preserving the edges .
The resulting output image resembles a cartoon by simplifying the colors and emphasizing the edges, giving it a hand-drawn appearance.

METHODOLOGIES

Data Collection: I will collect a dataset of phishing and legitimate URLs from publicly available sources like PhishTank, OpenPhish, etc. The data will be cleaned by removing duplicates and null values.

Model Building: I will split the dataset into train and test sets. I will then train and tune machine learning models like Logistic Regression, Random Forest, and XGBoost on the training set. The model with the best performance on the validation set will be selected as the final model.

Evaluation: The performance of the final model will be evaluated on the test set using metrics like Accuracy, Precision, Recall and F1-score. The goal is to achieve over 95% accuracy in detecting phishing websites.

Prediction: The trained model can be used to predict if new unseen URLs are phishing or legitimate. This model can potentially be integrated into web browsers and search engines to detect phishing websites in real-time.

RESULTS

The system which was proposed by us is the efficient in many ways to cartoon the real pictures And the technique used to build the software is so cost less as well as easy to build just some good effort is needed. GAN can be the best choice for us to choose to cartoonify the images because even it is hard to build but it gives the results accurately and effectively . As we can implement many types of datasets on this system.

The main thing to compare between the existing and our system is the technique used to build the software. they used the regular techniques to cartoonize but we had changed it by introducing GAN to build a software which can give more Outputs . Time consuming is less compared to the old systems . the clarity and edge has been developed using some more properties.

STANDARDS AND POLICIES

Data Privacy and Protection: Adherence to data privacy regulations such as the General Data Protection Regulation (GDPR) in the European Union and the California Consumer Privacy Act (CCPA) in the United States is essential. This involves ensuring that user data used for training and testing the machine learning model is handled in a compliant and ethical manner.

Ethical AI Principles: Following ethical AI principles and guidelines, such as those outlined by organizations like the IEEE or the European Commission's High-Level Expert Group on Artificial Intelligence, can help ensure that the machine learning model is developed and used in an ethical manner.

Figure 1. Label in 20pt Calibri.

CONCLUSIONS

Many Companies analyzed that Advertising their products using Cartoon images will helps them to attract people. Nowadays many schools are introducing Cartoonified books to the children. As in the World which is rapidly Increasing Technology Cartoonizing plays a great role in various aspects. Nowadays, Cartoons are used primarily for conveying political commentary and editorial opinion in Newspapers and also used in social comedy and Magazines. Thus our project helps various sectors which includes cartoonization . It is important to preprocess the image before applying any filter or effect. This includes resizing the image to a smaller size, converting it to grayscale, and smoothing it to remove any noise or unwanted details. By carefully considering each step and its impact on the image, it is possible to create a wide range of unique and interesting cartoon-style images using OpenCV.

1. Mr.K.NAVAZ,M.E., Ph. D.,
2. Contact No : 8608245565
3. Mail ID :- NAVAZ.K@veltech.edu.in

Figure 9.1: **Poster Presentation**

References

[1]An insider's "GUIDE TO CARTOONIZATION USING MACHINE LEARNING - Daksh Trehan - 2021 - July - 10.

[2] Gayen, S. Jha, S. Singh, M. Kumar, R. On a generalized notion of anti - fuzzy subgroup and some characterizations. International journal of Engineering and Advanced Technology 2021.

[3] Gayen, S. Smarandache, F. Jha, S. Singh, M. K. Broumi, S. Kumar, R. Introduction to plithogenic hyper soft subgroup. Neutrosophic Sets and Systems 2021.

[4] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image to image translation with conditional adversarial networks," arXiv preprint arXiv:1611.07004, 2018.

[5] S. Y. Muhammad, M. Makhtar, A. Rozaimee, A. Abdul, and A. A.Jamal, "Classification model for air quality using machine learning techniques," International Journal of Software Engineering and Its Applications, pp. 45-52, 2020.

[6] V. Ravi, Ch.Rajendra Prasad, S. Sanjay Kumar, P. Ramchandar Rao Image enhancement on OpenCV based on the tools : python 2.7 - Dept of ECE. SR College, Telangana, India - Feb 2022.

[7] Wang, Xinrui, and Yu, Jinze. Learning to cartoonize using White-box cartoon Representations. IEEE/CVF conference on computer vision and pattern recognition (CVPR).June 2022.

[8] Y. Chen, Y.-K. Lai, Y.-J. Liu, "Cartoon GAN: Generative Adversarial Network for photo cartoonization", International Conference on Image Processing, 2019.

[9] Yi-Hsuan Tsai, Kihyuk Sohn, Samuel Schulter, and Man- mohan Chandraker. Domain adaptation for structured output via discriminative representations. arXiv preprint arXiv:1901.05427, 2019.

[10] Ziqiang Zheng, Chao Wang, Zhibin Yu, Nan Wang, Haiy- ong Zheng, and Bing Zheng. Unpaired photo-to-caricature translation on faces in the wild. Neurocomputing, 355:71– 81,2020.