

Cartoonify: Transforming Images with Machine Learning

A MINI PROJECT

Submitted by

SEYJUTI BANERJEE(RA2111027010052)

NAGELLA VYSHNAVI(RA2111027010034)

SAI TUSHAR(RA2111027010050)

Under the guidance of

Dr. E. Sasikala

Professor

Department of Data Science and Business Systems

In partial fulfilment for the

Course of

18CSE392T- Machine Learning-I

in

Department of Data Science and Business Systems



**SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603203**

October 2023



COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this mini project report "*Cartoonify: Transforming Images with Machine Learning*" is the Bonafide work of Seyjuti Banerjee(RA2111027010052), Vyshnavi Nagella (RA2111027010034) and Sai Tushar(RA2111027010050) who carried out the project work under my supervision.

Dr. E. Sasikala
Professor
Department of Data Science and Business Systems
SRM institute of science and technology

Dr. M Lakshmi
Professor & HOD
Department of DSBS
SRM institute of science and technology

ABSTRACT

- Machine learning, a pivotal facet of AI, employs data and algorithms to emulate human learning and enhance precision.
- The focus is on Cartoonifying Images, achieved through a neural network that generates distinct cartoons from photos.
- Python, particularly with OpenCV, serves as an alternative to web-based software such as Photoshop for this purpose.
- The amalgamation of Python, OpenCV, numpy, and matplotlib produces a powerful application capable of transforming images into captivating cartoons.
- This endeavor underscores the collaboration between machine learning and creative pursuits, highlighting their harmonious potential.

TABLE OF CONTENTS

1. ABSTRACT
2. OBJECTIVE
3. INTRODUCTION
4. REQUIREMENT SPECIFICATION
5. LITERATURE SURVEY
6. ARCHITECTURE DIAGRAM
7. SOURCE CODE
8. OUTPUT
9. REFERENCE

OBJECTIVE

- The image cartoonification project seeks to merge the artistic appeal of cartoons with real-life photographs.
- By employing machine learning techniques, the project aims to automatically transform images into delightful and playful cartoon-like renditions.
- This endeavor not only enhances creative expression but also finds practical applications in entertainment, education, and marketing.
- Empowering users with interactive control further enriches the overall experience, making cartoonification a versatile and engaging tool for visual storytelling and content creation.

REQUIREMENT SPECIFICATIONS

INTRODUCTION

- Cartoons are commonly used in various kinds of applications. As we know cartoons are made artistically it requires elegant and fine human artistic skills.
- Compared to sketching an image cartoonifying an image is efficient and quick.
- Python programming language is used for writing a code to this technique And different python libraries are used to get most accurate output.
- Mainly OpenCV,One of the library in python,is used in this method of cartoonifying.

HARDWARE AND SOFTWARE SPECIFICATION

Hardware Requirements

- Processor: Minimum i3 Dual Core
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
- Hard Drive: Minimum 100 GB; Recommended 200 GB or more
- Memory (RAM): Minimum 8 GB; Recommended 32 GB or above

Software Requirements:

- Python
- Anaconda
- Jupyter Notebook
- TensorFlow
- opencv2

LITERATURE REVIEW

- **"Deep Cartoonification and Artistic Style Transfer"**
- Authors: H. Zhang, J. Zhang, and S. Liu.
- Year: 2020
- Advantages: Combines cartoonification with artistic style transfer, potentially creating unique and visually appealing results.
- Disadvantages: Lack of specified year and potential limitations of the proposed technique.

"CartoonGAN: Generative Adversarial Networks for Photo Cartoonization"

- Authors: J. Zhu, T. Park, P. Isola, and A. A. Efros.
- Year: 2022.

- Advantages: Uses GANs to generate cartoon-like images, allowing for diverse and customizable cartoonification.
- Disadvantages: Potential challenges in controlling the level of abstraction and style fidelity.

"Towards Real-Time Photorealistic 3D Cartoon Rendering"

- Authors: Y. Xu, L. Duan, Q. Zhao, and Q. Hou.
- Year: 2019
- Advantages: Focuses on real-time 3D cartoon rendering, enhancing realism and artistic expression.
- Disadvantages: Lack of specified year and potential performance trade-offs.

"Cartoon-like Image Style Transfer"

- Authors: D. R. Figueiredo, T. Simões, and J. L. Oliveira.
- Year: 2017
- Advantages: Explores image style transfer for cartoon-like effects, adding artistic flair to images.
- Disadvantages: Lack of specified year and potential limitations in style transfer accuracy.

"Real-Time Artistic Style Transfer for Video"

- Authors: S. Parisi, C. Shen, and A. Hertzmann.
- Year: 2017.
- Advantages: Addresses real-time style transfer for videos, offering dynamic and interactive cartoonification.
- Disadvantages: Potential computational challenges for real-time processing.

"Adversarial Feature Learning for Style Consistent Image Generation"

- Authors: Y. Liu, X. Li, and S. Osher.
- Year: 2021.
- Advantages: Focuses on style-consistent image generation using adversarial training, enhancing cartoonification accuracy.

- Disadvantages: Potential complexities in adversarial training and model convergence.

"Cartoon Texture and Non-photorealistic Texture Synthesis and Transfer"

- Authors: X. Yang, T. Xie, and Y. Jia.
- Year: 2021
- Advantages: Explores texture synthesis and transfer for cartoonification, adding detailed textures to cartoons.
- Disadvantages: Lack of specified year and potential challenges in handling diverse textures.

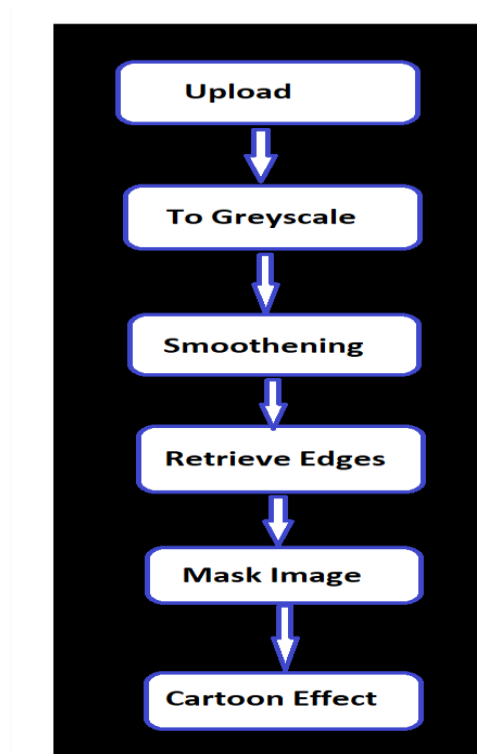
"Artistic Style Transfer for Videos"

- Authors: A. Ruder, A. Dosovitskiy, and T. Brox.
- Year: 2016.
- Advantages: Extends style transfer to videos, enabling dynamic and continuous cartoonification effects.
- Disadvantages: Potential limitations in maintaining temporal coherence and efficiency.

"Neural Style Transfer: A Review"

- Authors: A. A. Awan and M. A. Mahmood.
- Year: 2021.
- Advantages: Offers a comprehensive review of neural style transfer methods, providing insights into their application for cartoonification.
- Disadvantages: Review may not cover the latest advancements and specific challenges in cartoonification.

ARCHITECTURE DIAGRAM



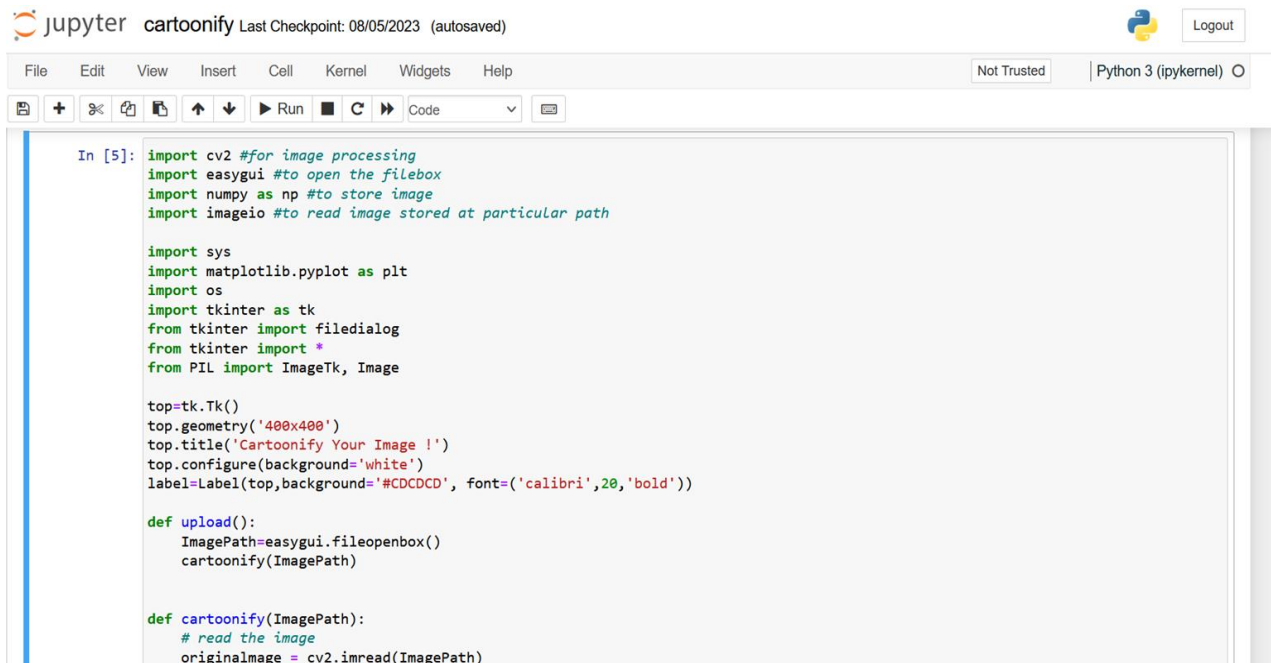
ISSUES

There may be several issues that could arise during the development and deployment of the proposed project. Some of these issues could include:

- Simplification can result in the loss of intricate details and textures present in the original image.

- Edge detection algorithms might produce artifacts, leading to inaccuracies in outlining shapes.
- The color simplification process can alter colors, causing unrealistic appearances in areas with subtle color changes.
- Cartoonification can lead to an overly simplistic or "flat" look, lacking the original image's complexity.
- Optional enhancements like exaggerated color boundaries may introduce unwanted artifacts.
- Algorithm effectiveness can vary due to differences in image content, lighting, and complexity

SOURCE CODE



The screenshot shows a Jupyter Notebook titled 'cartoonify' with a last checkpoint of '08/05/2023 (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a code editor. The code in the editor is as follows:

```
In [5]: import cv2 #for image processing
import easygui #to open the file box
import numpy as np #to store image
import imageio #to read image stored at particular path

import sys
import matplotlib.pyplot as plt
import os
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image

top=tk.Tk()
top.geometry('400x400')
top.title('Cartoonify Your Image !')
top.configure(background='white')
label=Label(top,background='#CDCDCD', font=('calibri',20,'bold'))

def upload():
    ImagePath=easygui.fileopenbox()
    cartoonify(ImagePath)

def cartoonify(ImagePath):
    # read the image
    originalImage = cv2.imread(ImagePath)
```

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

```
originalImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2RGB)
#print(image) # image is stored in form of numbers

# confirm that image is chosen
if originalImage is None:
    print("Can not find any image. Choose appropriate file")
    sys.exit()

ReSized1 = cv2.resize(originalImage, (960, 540))
#plt.imshow(ReSized1, cmap='gray')

#converting an image to grayscale
grayScaleImage= cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
ReSized2 = cv2.resize(grayScaleImage, (960, 540))
#plt.imshow(ReSized2, cmap='gray')

#applying median blur to smoothen an image
smoothGrayScale = cv2.medianBlur(grayScaleImage, 5)
ReSized3 = cv2.resize(smoothGrayScale, (960, 540))
#plt.imshow(ReSized3, cmap='gray')

#retrieving the edges for cartoon effect
#by using thresholding technique
getEdge = cv2.adaptiveThreshold(smoothGrayScale, 255,
                                cv2.ADAPTIVE_THRESH_MEAN_C,
                                cv2.THRESH_BINARY, 9, 9)
```

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

```
ReSized4 = cv2.resize(getEdge, (960, 540))
#plt.imshow(ReSized4, cmap='gray')

#applying bilateral filter to remove noise
#and keep edge sharp as required
colorImage = cv2.bilateralFilter(originalImage, 9, 300, 300)
ReSized5 = cv2.resize(colorImage, (960, 540))
#plt.imshow(ReSized5, cmap='gray')

#masking edged image with our "BEAUTIFY" image
cartoonImage = cv2.bitwise_and(colorImage, colorImage, mask=getEdge)

ReSized6 = cv2.resize(cartoonImage, (960, 540))
#plt.imshow(ReSized6, cmap='gray')

# Plotting the whole transition
images=[ReSized1, ReSized2, ReSized3, ReSized4, ReSized5, ReSized6]

fig, axes = plt.subplots(3,2, figsize=(8,8), subplot_kw={'xticks':[], 'yticks':[]}, gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(images[i], cmap='gray')

save1=Button(top,text="Save cartoon image",command=lambda: save(ReSized6, ImagePath),padx=30,pady=5)
save1.configure(background='#364156', foreground='white',font=('calibri',10,'bold'))
save1.pack(side=TOP,pady=50)

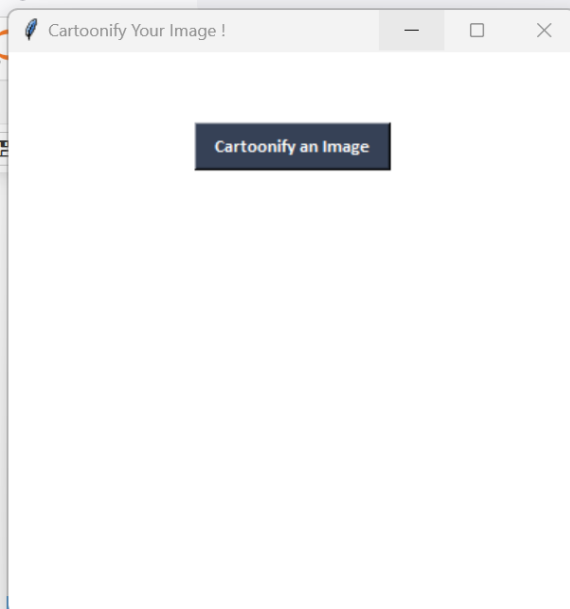
plt.show()
```



```
def save(ReSized6, ImagePath):
    #saving an image using imwrite()
    newName="cartoonified_Image"
    path1 = os.path.dirname(ImagePath)
    extension=os.path.splitext(ImagePath)[1]
    path = os.path.join(path1, newName+extension)
    cv2.imwrite(path, cv2.cvtColor(ReSized6, cv2.COLOR_RGB2BGR))
    I= "Image saved by name " + newName + " at " + path
    tk.messagebox.showinfo(title=None, message=I)

upload=Button(top,text="Cartoonify an Image",command=upload,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('calibri',10,'bold'))
upload.pack(side=TOP,pady=50)

top.mainloop()
```



In []:

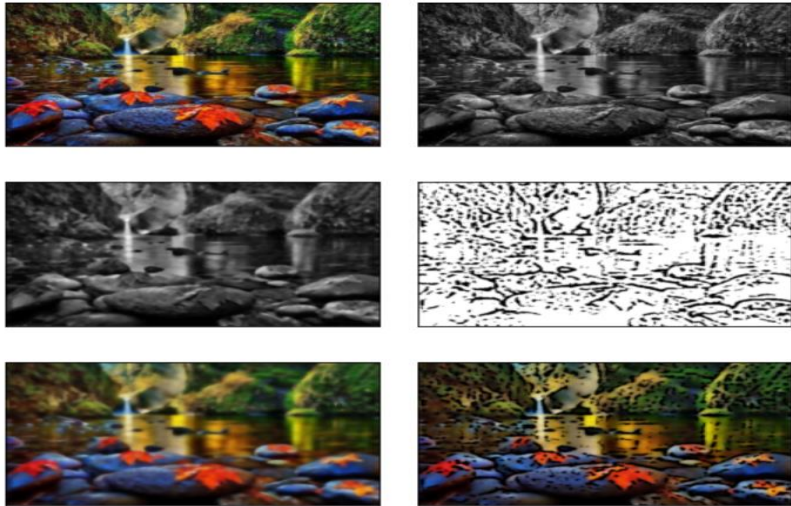
OUTPUT

jupyter cartoonify Last Checkpoint: 08/05/2023 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Code

```
top.mainloop()
```



Cartoonify Your Image !

Cartoonify an Image

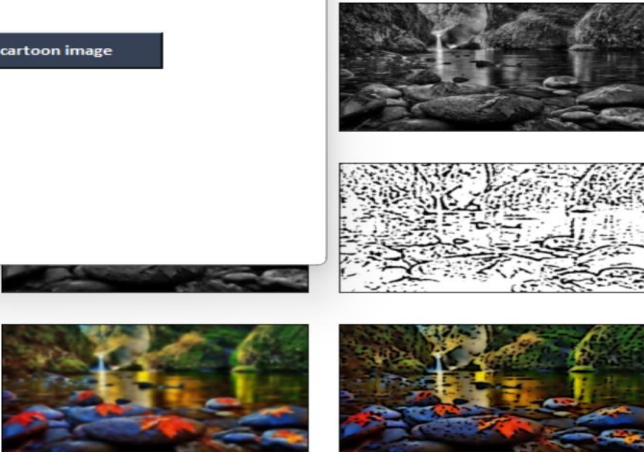
Save cartoon image

/cartoonify .ipynb

23 (autosaved)

Widgets Help Trusted

Code



REFERENCES

- MD.Salar Mohammad, Bollepalli Pranitha , Shivani Goud Pandula , Pulakanti Teja Sree : Object Detection with Voice Sensor and Cartoonizing the Image. (August 2021)
- Yugang Chen, Muchun Chen, Chaoyue Song, and Bingbing Ni : CartoonRenderer: An Instance-based MultiStyle Cartoon Image Translator.
- Pranjal Singh Rajput, Kanya Satis, Sonnya Dellarosa, Wenxuan Huang, Obinna Agba : cGANs for Cartoon to Real-life Images.
- Yang Chen Yu-Kun Lai Yong-Jin Liu : CartoonGAN: Generative Adversarial Networks for Photo Cartoonization.
- Vaishali Sudarshan , Amritesh Singh :- Cartooning an Image Using Opencv and Python.(2020)