Modern University
for technology & information

AI Cartoonify Image with Machine Learning

Presented to:

DR. hesham el-deeb

Presented by:

1-Abdelrahman Ghoniem Ghoniem g4

2-Mohammed Osama Abdelghany g6

3-Mohammed Abdelmotaleb Mohammed g7

4-Eyad Ahmed Ali Emam (G2)

5-Bassant Mohammed Mohammed (G2)

6-Adham Rabeh mohammed tolba (G2)

7-Yousef Elsayed Mohamed Mohamed Ahmed (G8)

Table of Contents

1-Introduction

In the realm of digital image processing, the quest for creative manipulation techniques has always been a fascinating endeavor. Among the plethora of techniques, the art of cartoonification stands out as a unique and captivating approach that transforms ordinary photographs into whimsical and expressive cartoons reminiscent of hand-drawn illustrations. This Python script represents a foray into this realm, offering an interactive and intuitive platform for users to embark on their journey of image cartoonification.

Cartoonification, at its core, is about simplifying the complexities of reality while accentuating key features to evoke a sense of artistic interpretation. By leveraging a combination of image processing algorithms and computational techniques, this script endeavors to emulate the distinct characteristics of traditional cartoon artistry. From exaggerated outlines to vibrant color palettes, the process seeks to imbue images with a playful and surreal charm, transcending the boundaries of mere representation.

The algorithmic underpinnings of this endeavor are multifaceted, encompassing a symphony of techniques orchestrated to orchestrate the transformation from raw images to stylized cartoons. Beginning with the selection of an input image, the journey unfolds through a series of meticulously crafted steps. Preprocessing stages such as resizing, grayscale conversion, and edge detection lay the foundation for subsequent transformations. The heart of the process lies in the fusion of sophisticated filters and adaptive thresholding techniques, culminating in the creation of cartoon-like renditions that capture the essence of the original while infusing it with an unmistakable artistic flair.

Central to the functionality of the script are a diverse array of libraries, each contributing its unique capabilities to the creative tapestry. OpenCV, renowned for its prowess in image processing, serves as the backbone, providing essential functionalities such as reading, manipulation, and filtering of images. NumPy, with its powerful array operations, lends its computational prowess to the mix,

facilitating efficient data manipulation and transformation. Matplotlib emerges as the canvas upon which the visual narrative unfolds, offering a versatile platform for image visualization and interpretation. Complementing these are EasyGUI and Tkinter, which orchestrate the user interface, facilitating seamless interaction and engagement.

In essence, this script epitomizes the marriage of art and technology, offering users a glimpse into the transformative power of computational creativity. Beyond its utility as a tool for image manipulation, it embodies the spirit of exploration and experimentation, inviting users to embark on a voyage of artistic discovery. Whether as a means of unleashing creative expression or as a tool for entertainment and amusement, the art of cartoonification continues to captivate and inspire, transcending boundaries and fostering a deeper appreciation for the boundless possibilities of digital imagery.

2-Algorithm

The algorithm follows these main steps:

1. **Image Selection**: The user selects an image using a file dialog provided by EasyGUI.
2. **Preprocessing**:
    - The selected image is read using OpenCV.
    - It is resized for processing efficiency.
    - Conversion to grayscale is performed.
    - Gaussian blur is applied to smoothen the image.
    - Edges are detected using adaptive thresholding.
3. **Cartoonification**:
    - A bilateral filter is applied to the image after quantizing color levels.
    - The filtered image is masked with the edges detected earlier to emphasize edges and flatten colors, giving a cartoon effect.
4. **Visualization**:
    - The original and cartoonified images are displayed using Matplotlib.
    - Additionally, intermediate steps such as grayscale conversion, edge detection, etc., are visualized for better understanding.

3-Used Libraries

1. **OpenCV (cv2)**: Used for image processing tasks like reading, resizing, converting colors, blurring, and edge detection.
2. **NumPy**: Utilized for numerical computations and array manipulation, especially for handling image data.
3. **Matplotlib (plt)**: Employed for visualizing images and intermediate steps of the cartoonification process.
4. **EasyGUI**: Facilitates the creation of a file dialog for image selection.
5. **Tkinter**: Used for creating the graphical user interface (GUI) to interact with the user, including a button to upload images.
6. **PIL (Python Imaging Library)**: Utilized for handling images and converting between different image formats.
7. **Sys**: Used for system-specific operations, such as exiting the script gracefully in case of errors or invalid inputs.

This combination of libraries enables efficient image processing, user interaction, and visualization, making the cartoonification process interactive and intuitive.

4-Shotscreens:-

User chooses an image from his local pc to apply apply cartoonize filter on it as simple as that for ex:-

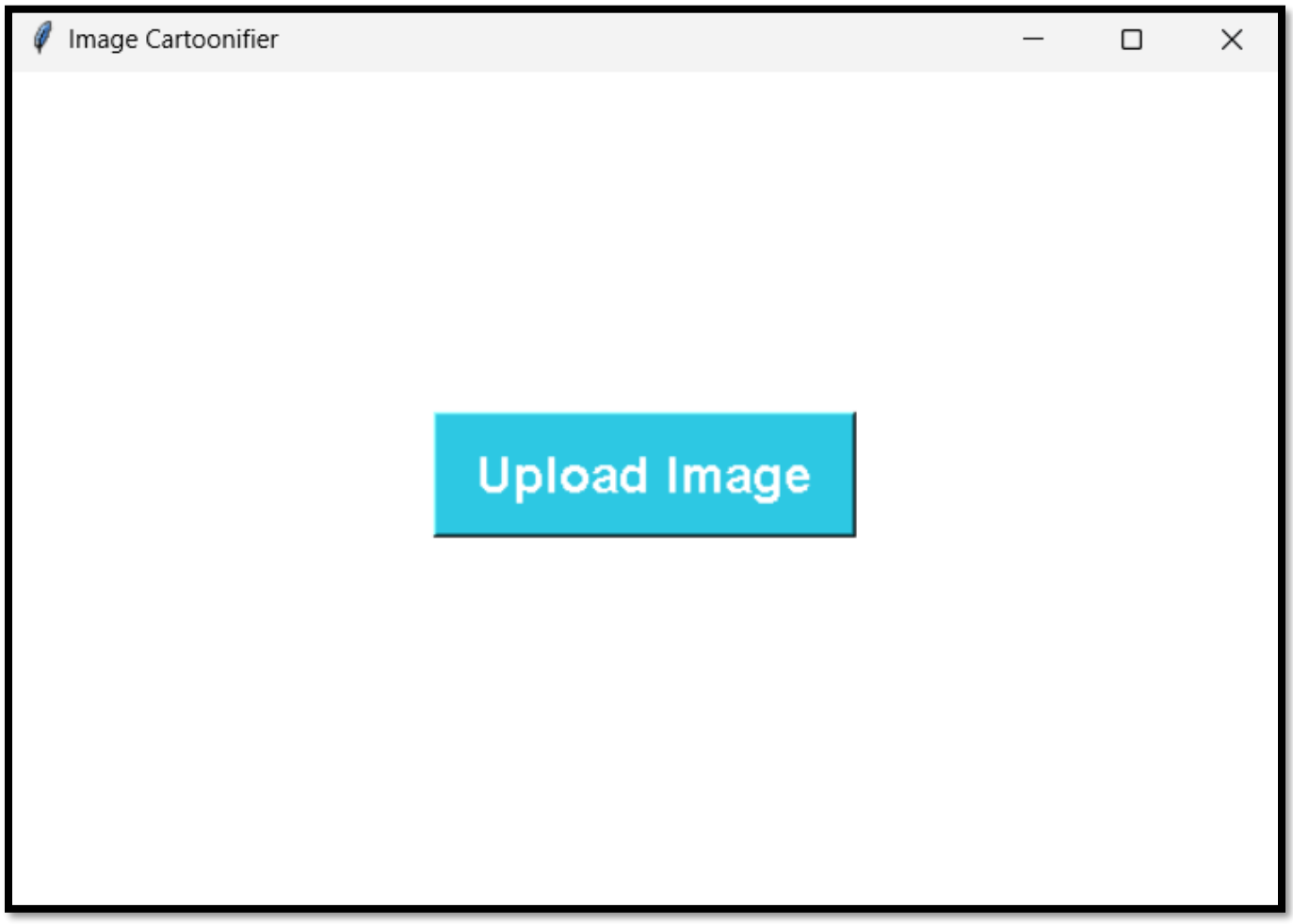
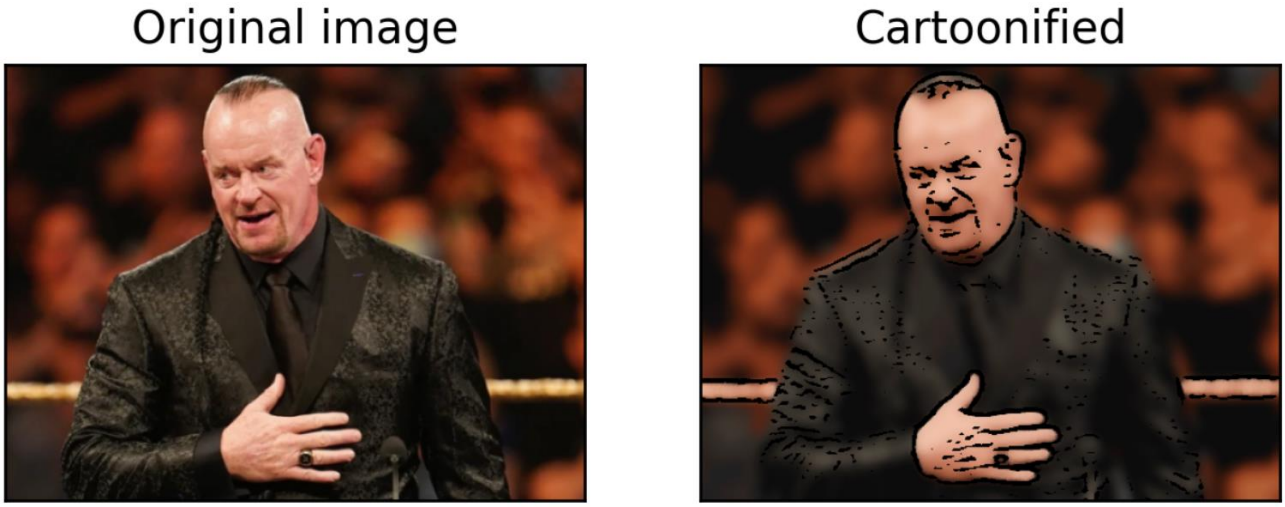


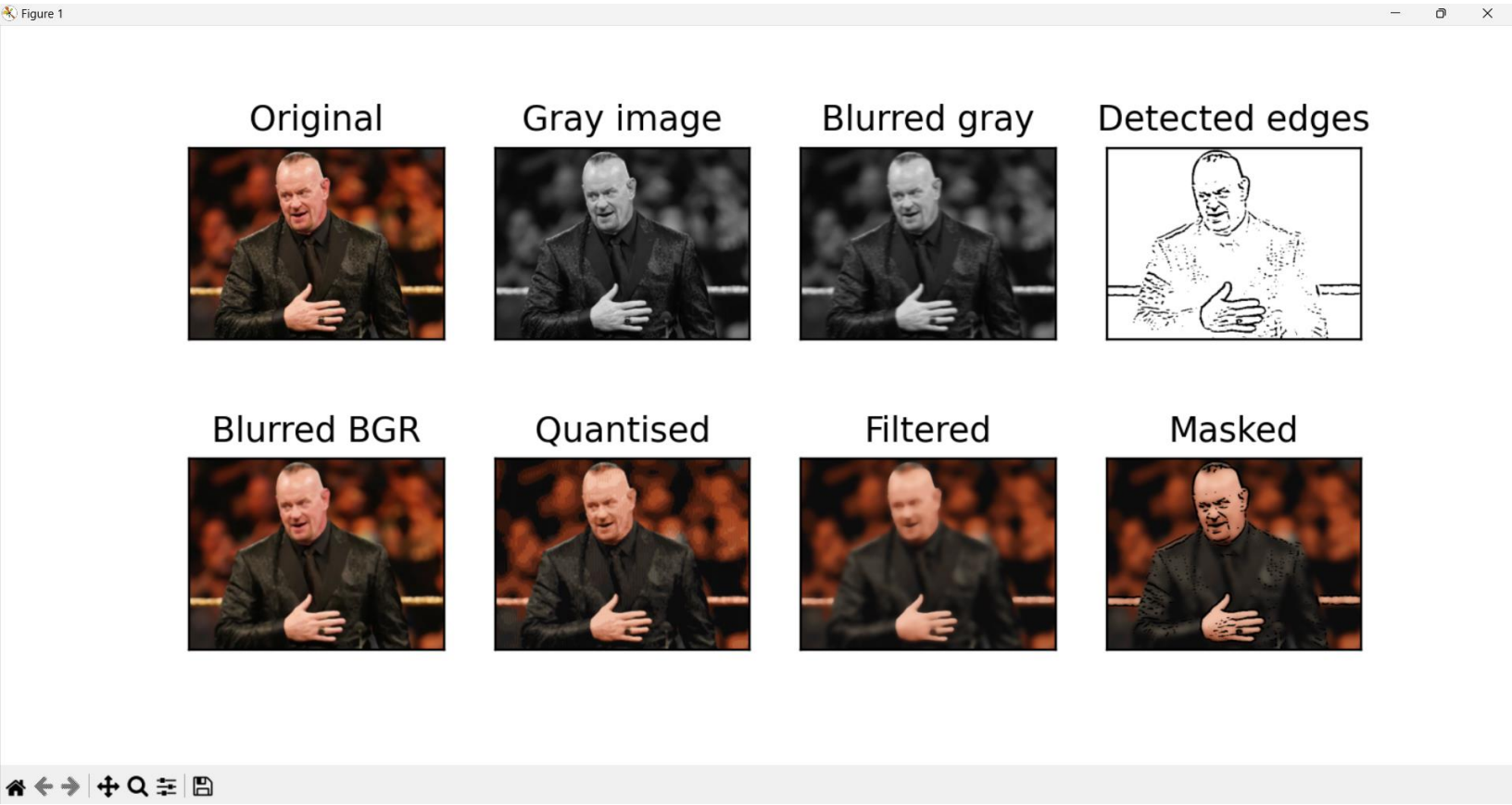
Figure 1



This is a photo of the undertaker before and after cartoonification process

..

When we close this page we can see how the process was done like shown here:-



Code

Imports:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import easygui
import tkinter as tk
from tkinter import filedialog, Label, Button, TOP
from PIL import ImageTk, Image
import sys
```

upload image:

```
# Function to upload image using easygui
def upload():
    # Open a file dialog to select an image
    ImagePath = easygui.fileopenbox()
    # Call the cartoonify function with the selected image path
    cartoonify(ImagePath)
```

Cartoonify image:

```
# Function to cartoonify the image
def cartoonify(ImagePath):
    # Read the image
    img = cv.imread(ImagePath)

    # Exit if image is not selected
    if img is None:
        print("Can not find any image. Choose appropriate file")
        sys.exit()
```

```
# Resize the image
resize = cv.resize(img, (480, 360), interpolation=cv.INTER_AREA)
# Convert the resized image to grayscale
gray = cv.cvtColor(resize, cv.COLOR_BGR2GRAY)
# Apply Gaussian blur to the grayscale image
blurred = cv.GaussianBlur(gray, (9, 3), 0)
# Detect edges using adaptive thresholding
edge = cv.adaptiveThreshold(
    blurred, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 7, 7)
# Apply median blur to the resized image
reblur = cv.medianBlur(resize, 5)
```

```
# Function to quantize the image
def quan(img, k):
    data = np.float32(img).reshape(-1, 5)
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 15, 0.01)
    ret, label, center = cv.kmeans(
        data, k, None, criteria, 10, cv.KMEANS_RANDOM_CENTERS)
    center = np.uint8(center)
    result = center[label.flatten()]
    result = result.reshape(img.shape)
    return result
```

```
# Call the quantization function
q = quan(reblur, 70)
# Apply bilateral filter to the quantized image
noise = cv.bilateralFilter(q, 15, 190, 190)
# Mask the filtered image with the detected edges
ci = cv.bitwise_and(noise, noise, mask=edge)
```

Plotting

```
plt.figure(figsize=(6, 4), dpi=250)
plt.subplot(1, 2, 1)
plt.imshow(cv.cvtColor(resize, cv.COLOR_BGR2RGB))
plt.title('Original image')
plt.xticks([])
plt.yticks([])

plt.subplot(1, 2, 2)
plt.imshow(cv.cvtColor(ci, cv.COLOR_BGR2RGB))
plt.title('Cartoonified')
plt.xticks([])
plt.yticks([])
plt.show()

plt.figure(figsize=(5, 4), dpi=210)
plt.subplot(2, 4, 1)
plt.imshow(cv.cvtColor(resize, cv.COLOR_BGR2RGB))
plt.title("Original")
plt.xticks([])
```

```
plt.yticks([])

plt.subplot(2, 4, 2)
plt.imshow(cv.cvtColor(gray, cv.COLOR_BGR2RGB))
plt.title('Gray image')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 4, 3)
plt.imshow(cv.cvtColor(blurred, cv.COLOR_BGR2RGB))
plt.title('Blurred gray')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 4, 4)
plt.imshow(cv.cvtColor(edge, cv.COLOR_BGR2RGB))
plt.title('Detected edges')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 4, 5)
```

```
plt.imshow(cv.cvtColor(reblur, cv.COLOR_BGR2RGB))
plt.title('Blurred BGR')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 4, 6)
plt.imshow(cv.cvtColor(q, cv.COLOR_BGR2RGB))
plt.title('Quantised')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 4, 7)
plt.imshow(cv.cvtColor(noise, cv.COLOR_BGR2RGB))
plt.title('Filtered')
plt.xticks([])
plt.yticks([])

plt.subplot(2, 4, 8)
plt.imshow(cv.cvtColor(ci, cv.COLOR_BGR2RGB))
plt.title('Masked')
plt.xticks([])
```

```
plt.yticks([])
plt.show()
```

```
# Tkinter GUI setup
top = tk.Tk()
top.geometry('600x400')
top.title('Image Cartoonifier')
top.configure(background='white')
label = Label(top, background='#15d6b9', font=('calibri', 18, 'bold'))
```

```
# Button to upload image
upload = Button(top, text="Upload Image", command=upload, padx=10, pady=5)
upload.configure(background='#2dc8e3', foreground='white', font=('Cooper Std Black', 18, 'bold'))
upload.pack(side=TOP, pady=160)

# Run the GUI
top.mainloop()
```