ECS 174 Final Project Report

Jingdan Hu, Thae Su (Catherine) Win, Maggie Kuang

University of California, Davis

ECS174 Computer Vision, Spring 2024

June 10, 2024

Topic

Cartoonizing Reality

Introduction

Cartoonization is an image processing technique that converts photographs into cartoon-like images. This project aims to develop an easy-to-use website for users to cartoonize their images. This report details the methods used for this project, including the implementation of two distinct styles, and presents the experimental results obtained from applying these methods.

Problem

Cartoonizing an image normally takes a long time to draw and it requires people to have high drawing skills. Developing an automated cartoonization tool can simplify the process of drawing it ourselves and make it accessible to a wider audience, letting people transform their photos into cartoon-like representations without requiring users to have technical knowledge and artistic skills.

Motivation

Cartoonizing reality, turning real-world images into cartoon-like versions, serves various practical purposes. It is a fun way to express creativity and entertain people which is often used in digital art, animation, and storytelling. Cartoonization also helps protect privacy by hiding faces or sensitive details in photos or videos. In education, it simplifies complex concepts, making them easier to understand. It can also be utilized to visually engage viewers when educating them about scientific ideas or historical events through educational videos. Moreover, businesses use cartoonization for marketing and branding, creating memorable content that attracts customers' attention and promotes products effectively.

Dataset

For this project, we find that a dataset is not necessary since we create a Tkinter-based GUI that allows users to upload images from their own laptop. Some sample images are provided to let users test our program if users are not feeling comfortable to upload their own images.

Method

We mainly used OpenCV functions and the K-Means clustering algorithm to perform image processing and cartoony them. Gaussian blur, median blur, and bilateral filter were also used to blur the images before we transform them into cartoon-like.

Cartoonization1: This method follows a series of steps to transform a regular image into a cartoon-like representation:

1. Smoothing the image
    - The method begins by applying a Gaussian blur to the input image. This blur smooths out the image by averaging pixel values within a defined neighborhood, reducing noise and finer details.
    - Then, a median blur is applied to further smooth the image by replacing each pixel's value with the median value of its neighboring pixels. Unlike Gaussian blur, median blur preserves edges while reducing noise.
    - Lastly, edge-preserving smoothing is achieved using a bilateral filter. Because it averages pixel values based on both spatial closeness and intensity similarity, this filter smooths the image while maintaining edges.
2. Edge Detection
    - Laplacian filtering is employed to detect edges in images by highlighting regions of rapid intensity change. The result is an edge-detected image where edges are emphasized.
3. Threshold and Invert
    - Otsu's thresholding method helps to separate the edges we've detected from the rest of the image. This method automatically calculates an optimal threshold value to separate the foreground from the background, enhancing the clarity of the edges.
    - The thresholded image is inverted so that the edges appear white on a black background.
4. K-means clustering
    - Color quantization, which reduces the amount of unique colors in the image, is done using K-means clustering to simplify the color palette and generate a cartoonish effect.

Finally, the color-quantized image is further simplified through binarization, reducing color depth and enhancing the cartoon effect. The edges are then combined with the simplified color regions using a bitwise AND operation.

Cartoonization2:

1. Load the image using `cv2.imread` and resize it to a shape that is a multiple of 4 for better processing.
2. Downsample the image using `cv2.pyrDown` to reduce the resolution of the image to simplify details.
3. Apply the bilateral filter multiple times to smooth the image while preserving the edges.
4. Use `cv2.pyrUp` to restore the image back to its original size.
5. Convert the image to grayscale, apply a median blur to reduce noise, and then use `cv2.adaptiveThresholding` to detect edges.
6. Convert the image to RGB format then combine the color image with the edge mask.

Tkinter-based GUI

1. Setup GUI
    - Build a main window with a size of 1000x700 and set up a background to attract users.

- Sample output and a simple instruction are located on the left side to guide the users.
- An upload button is placed at the top to let users upload images.
- A drop-down menu is located below the upload button to let users select the cartoonization method.
- A start button is placed at the bottom to start cartoonizing the image.

2. Image Uploading
    - Open the file dialog to select an image file. If non-image input is selected, an error message will be shown.
    - The original image is displayed with a modified size in case the size of the uploaded image does not match the window size.
3. Choose methods to cartoonize the image
    - Create options for users to choose.
    - If none of the methods is selected or no image is received, an error message will be shown.
4. Start cartoonization
    - Apply the selected method to cartoonize images based on the users' decisions.
    - The cartoonized image is displayed and replaces the original image.

Final Outcome

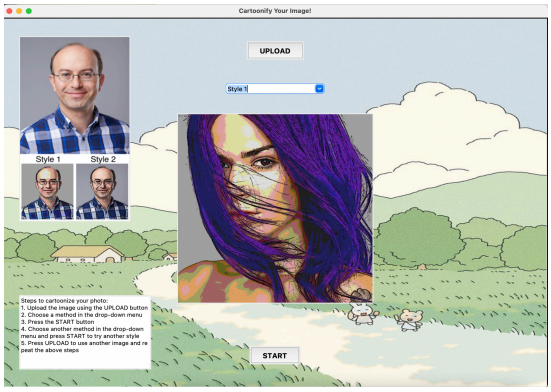
The final deliverable of this project will be a Tkinter-based GUI. Users will be able to upload their own images. The tool will provide real-time feedback, displaying the cartoonization results.
In our experiments, we applied both cartoonization techniques to a variety of input images, including portraits, landscapes, food, and objects.
After applying cartoonization1 to a variety of test images, I observed that the method effectively preserved fine details while producing sharp edges, resulting in cartoon-style images with clear outlines and vibrant colors. Images with intricate patterns and textures, such as portraits and landscapes, benefited particularly well from this method.
Cartoonization2 also gave us similar results but in a different way. The cartoon-style images produced by this method had a smoother appearance with softer edges like the cartoon you draw by hand. While not as detailed as the other method, it excelled in simplifying colors and textures, making it more suitable for images with fewer details and simpler compositions.
Overall, both methods demonstrated their effectiveness in transforming into visually appealing cartoon-style images, each catering to different preferences and requirements.

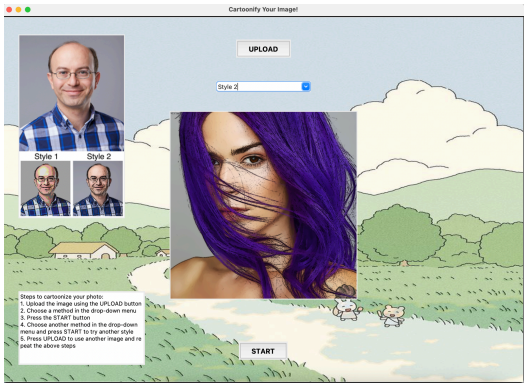
Demonstration

To launch the auto-cartoonization demo, users simply run 'Final_project.ipynb' with visual studio code. If users want to use the provided sample images, make sure the images are in the same directory as the ipynb file.

Original image:

Cartoonized images after processing:





Conclusion

Both cartoonization methods effectively transformed into cartoon-like images, each with its strengths and weaknesses. While Method 2 produced a smoother and more abstract cartoon image, Method 1 was better at maintaining details and creating sharp edges. The particular needs of the application, such as computing efficiency or detail preservation, determine which approach is best.

Contribution

All members participated in the Project report, slides, and presentation video.

Jingdan Hu: Mainly focus on developing Cartoonization 2, participated in adjustments on Cartoonization 1 and Tkinter-based GUI

Thae Su (Catherine) Win: Mainly focus on developing Cartoonization 1, participated in adjustments on Tkinter-based GUI

Maggie Kuang: Mainly focus on developing Tkinter-based GUI, participated in adjustments on Cartoonization 1

Presentation

https://youtu.be/MnQJrl8i5WM

References

https://medium.com/nerd-for-tech/cartoonize-images-with-python-10e2a466b5fb
https://data-flair.training/blogs/cartoonify-image-opencv-python/