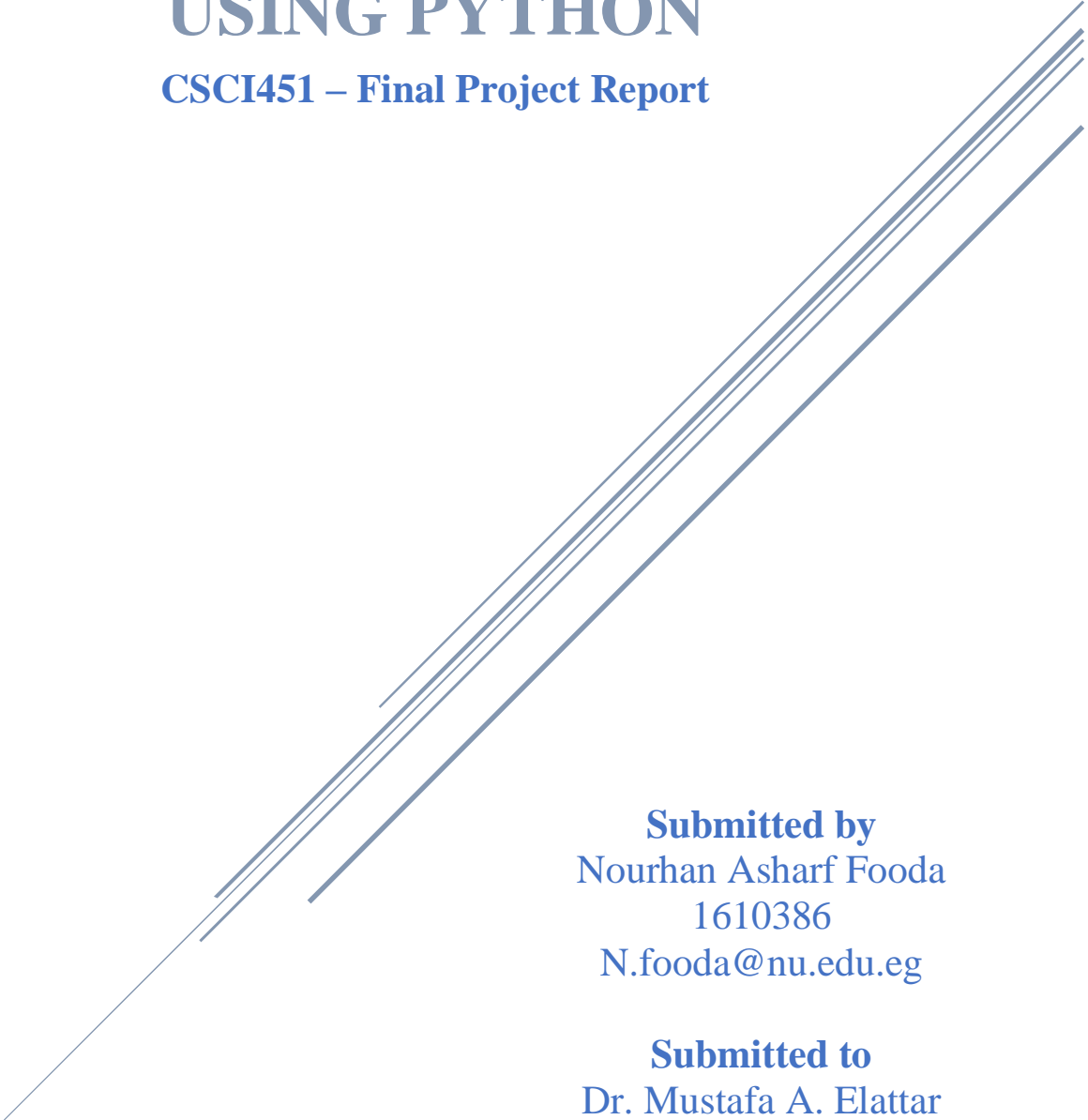


CARTOONING OF AN IMAGE USING PYTHON

CSCI451 – Final Project Report



Submitted by
Nourhan Asharf Fooda
1610386
N.fooda@nu.edu.eg

Submitted to
Dr. Mustafa A. Elattar
Eng. Asmaa Elhadidy

Abstract

As the technology is advancing rapidly, cartooning real images is no longer restricted to artists because image processing algorithms can do this job quite well nowadays. There has been an essential need for the image processing algorithms in such a field due to its low cost, almost free, which is a drawback in the professional cartooning programs that require payment. This project converts real photos into cartoon images using OpenCV python. This is mainly done through applying edge detection mechanism to produce bold silhouettes and some filters such as the bilateral filter to reduce the color palette and median blur filter to remove the noise. The dataset used for testing images is a subset from “1 Million Fake Faces” dataset on Kaggle and contains 1029 images. Since the cartoon effect is somehow subjective, the parameters controlling the resulting images were tuned to produce the maximum possible cartoon effect.

Table of Contents

List of Figures	3
I. Introduction	4
II. Methodology	5
A. Loading Images	5
B. Cartooning Images.....	5
1. Reducing color palette using a bilateral filter	5
2. Converting the color image to grayscale and applying a median blur filter	5
3. Creating an edge mask from the grayscale image using adaptive thresholding.....	5
4. Combining the bilateral filtered image with the edge mask.	5
III. Results.....	6
IV. Discussion & Conclusion	9
V. References	10

List of Figures

FIGURE 1: ORIGINAL IMAGE	6
FIGURE 2: IMAGE AFTER THE DOWNSAMPLING	6
FIGURE 3: IMAGE AFTER APPLYING THE BILATERAL FILTER	6
FIGURE 4: IMAGE AFTER THE UPSAMPLING	6
FIGURE 5: ORIGINAL IMAGE AFTER CONVERTING TO GRAYSCALE	7
FIGURE 6: IMAGE AFTER APPLYING MEDIAN BLUR FILTER	7
FIGURE 7: EDGE MASK CREATED FROM THE BLURRED IMAGE	7
FIGURE 8: EDGE MASK AFTER CONVERTING TO COLORED MASK	7
FIGURE 9: FINAL CARTOON IMAGE VERSION 1	8
FIGURE 10: FINAL CARTOON IMAGE VERSION 2	8
FIGURE 11: CARTOON IMAGE WITH INDEX [30] VERSION 1	8
FIGURE 12: CARTOON IMAGE WITH INDEX [30] VERSION 2	8

I. Introduction

A cartoonist is an artist who has his or her expertise in cartooning. With the advent of computers and other technological advancements, the art of cartoon has seen a vast change from its original status. While professional cartoonists use their sketching skills on papers or through graphics software to create cartoons, amateurs use online websites or programs to convert a real photo into cartoon image.

As the cartoon industry is growing steadily, the world needs more automated tools that can do the cartoonizing effects within short time with less expertise. Over the last few years, professional cartooning programs have spread widely but most of them are paid. Thus, the need for image processing algorithms has been given the attention to do the mission of cartoonizing real images within short time and for free.

A cartoon image has two main features which are clear edges and homogenous colors. Thus, creating cartoon images using image processing mainly depends on applying edge detection mechanism and some filters to produce the cartoon effect from a real image.

A dataset containing 1029 images was used in the form of zipped file. It is a subset from a dataset named “1 million fake faces” on Kaggle presented by Tunguz Consulting.

II. Methodology

A. Loading Images

To load the image from the zipped file containing the dataset, the zipfile library was used to access the files(images) and store them in a list for an easy use. To open an image with index [n] from the list, PILLOW library was used. Then, the resulting PIL image was converted to an OpenCV BGR image, then an OpenCV RGB image to be used in the cartooning process.

B. Cartooning Images

1. Reducing color palette using a bilateral filter

A bilateral filter is used to convert an image into a cartoon because it smooths flat regions while keeping edges sharp. The three parameters in `cv2.bilateralFilter` control the diameter of the pixel neighborhood (d) and the standard deviation of the filter in color space (σ_{Color}) as well as coordinate space (σ_{Space}). To maximize the processing speed and overcome the slowness of the bilateral filter, the real image was down-scaled 3 times using Gaussian pyramid and a small bilateral filter was used 13 times repeatedly. Then, the filtered image was up-scaled to its original size again.

2. Converting the color image to grayscale and applying a median blur filter

The original RGB image was converted to a grayscale image using `cvtColor` to avoid interfering the colors in the blurring process as well as to be used for thresholding later. Then, a median blur filter with an eleven-pixel local neighborhood was applied to diminish the image noise as it replaces each pixel value with the median value of all the pixels in a small neighborhood.

3. Creating an edge mask from the grayscale image using adaptive thresholding.

To create the edge mask, the adaptive thresholding was applied on the grayscale blur image. The normal thresholding uses a threshold pixel value to convert a grayscale image into a binary image, so if a pixel value in the original image is above the threshold, then the pixel value in the final image will be 255. Otherwise, it will be 0. Besides the normal thresholding function, the adaptive thresholding detects the most significant features in each small neighborhood independently without regard to the global image nature. Thus, we can easily get bold, black outlines around objects and people in a cartoon. To remove any noise left, the `ADAPTIVE_THRESH_MEAN_C` algorithm with block size = 9 will ensure that the threshold is applied to the mean of a 9x9 neighborhood minus $C = 2$.

4. Combining the bilateral filtered image with the edge mask.

The image resulted from the bilateral filtering is combined with the edge mask resulted from the adaptive thresholding using `bitwise_and` to create the final cartoon image that we want.

III. Results



Figure 1: Original Image



Figure 2: Image after the downsampling



Figure 3: Image after applying the bilateral filter



Figure 4: Image after the upsampling



Figure 5: Original Image after converting to grayscale



Figure 6: Image after applying median blur filter



Figure 7: edge mask created from the blurred image



Figure 8: edge mask after converting to colored mask



Figure 9: Final cartoon image version 1



Figure 10: Final cartoon image version 2



Figure 11: Cartoon image with index [30] version 1



Figure 12: Cartoon image with index [30] version 2

IV. Discussion & Conclusion

The results presented in the previous section belongs to image with index [10] from our dataset.

Figure [1] shows the original image, while Figure [2] shows the image after being downsampled 3 times using the Gaussian pyramid. Figure [3] shows the image after the bilateral filter. A small bilateral filter was applied 13 times with the following parameters: $d=9$, $\sigma_{\text{Color}}=9$, $\sigma_{\text{Space}}=9$. Then, the image was upsampled to its original size with the same number of steps used in the downsampling and the resulting image is shown in Figure [4]. The previous steps used to reduce the color palette in the image.

Figure [5] shows the original image after being converted to grayscale, while Figure [6] shows the grayscale image after applying a median blur filter with kernel size 11×11 to smooths the image as possible. To create the edge mask, the adaptive thresholding function was applied on Figure [6] with the following parameters: $\text{maxValue} = 255$, $\text{adaptiveMethod} = \text{ADAPTIVE_THRESH_MEAN_C}$, $\text{thresholdType} = \text{THRESH_BINARY}$, $\text{blockSize} = 9$, $C = 2$. Some of these parameters (adaptiveMethod , blockSize , C) were tuned different times later to reach the final desired output. Although the two adaptive methods which are $\text{ADAPTIVE_THRESH_MEAN_C}$ and $\text{ADAPTIVE_THRESH_GAUSSIAN_C}$ gave quite close results, but the first method seemed much more acceptable. Figure [7] shows the resulting edge mask from the adaptive thresholding. Figure [8] shows the grayscale edge mask after being converted to a colored one.

To create the final cartoon image, Images in figures [4] and [8] were combined together to produce the image shown in Figure [10]. Figure [9] shown a cartoon images resulted from the same combination as the previous one, but with different parameters which are: downsampling and upsampling steps = 2, number of bilateral filters applied = 7, size of median filter kernel = 7×7 . Those parameters values resulted in a noisy image as shown in figure [9], so they were tuned multiple times with different values to reach the desired output which is shown in figure [10].

To show the difference between the 2 versions of parameters on another image, image with index [30] from our dataset was used. Figure [11] shows the original image after being converted into a cartoon image with the same parameters used in figure [9], while figure [12] was created with the same parameters used in figure [10].

To conclude, the number of bilateral filters applied, and the kernel size of the median filter are significant parameters in the process of converting real images into cartoon images, and that clearly appears in the results section. Moreover, since the used functions in our cartooning process contain many parameters, the number of possible combinations between these parameters values are huge which results in too many different cartoon images from the same real image. That's why the cartoonish effect on image is quite subjective.

The used dataset can be accessed through this link: https://nileuniversity-my.sharepoint.com/:u:/g/personal/n_fooda_nu_edu_eg/ESmHCmrGfUpItlP0Jgl83OkB7smpffsJfK8PJ9Aof8Dbkg?e=DRkfn3, while some of the resulting images from both versions of parameters can be accessed through think link: https://nileuniversity-my.sharepoint.com/:u:/g/personal/n_fooda_nu_edu_eg/EQFDvCtvH29Pm3lXuDTGv-0BIOr2oGXOfAPN0z5k6O26TA?e=tgJ3lF

V. References

- “Cartooning an Image using OpenCV in Python?,” *tutorialspoint*. [Online]. Available: <https://www.tutorialspoint.com/cartooning-an-image-using-opencv-in-python>. [Accessed: 25-May-2020].
- I. Team, “Importance Of Cartoon Art And Cartooning,” *A Shoppers Guide To The Beautiful Naples*, 22-Apr-2019. [Online]. Available: <https://www.dessinateur-illustrateur.com/importance-of-cartoon-art-and-cartooning/>. [Accessed: 30-May-2020].
- “Image Filtering,” *Image Filtering - OpenCV 2.4.13.7 documentation*. [Online]. Available: <https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html>. [Accessed: 28-May-2020].
- “Image Module,” *Image Module - Pillow (PIL Fork) 3.1.2 documentation*. [Online]. Available: <https://pillow.readthedocs.io/en/3.1.x/reference/Image.html>. [Accessed: 29-May-2020].
- “Miscellaneous Image Transformations,” *Miscellaneous Image Transformations - OpenCV 2.4.13.7 documentation*. [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=adaptive_threshold. [Accessed: 29-May-2020].
- S. Tikoo, “Cartooning an Image using Open CV,” *Cartooning an Image using Open CV*, 04-May-2014. [Online]. Available: <https://www.skyfilabs.com/project-ideas/cartooning-an-image-using-open-cv>. [Accessed: 25-May-2020].
- Tunguz, B. 1 Million Fake Faces - 1, Version 3., Indiana, US: Tunguz Consulting, 2019. [Online Dataset]. Available: <https://www.kaggle.com/tunguz/1-million-fake-faces>. [Accessed: 27-May-2020].
- “zipfile - Work with ZIP archives,” *zipfile - Work with ZIP archives - Python 3.8.3 documentation*. [Online]. Available: <https://docs.python.org/3/library/zipfile.html>. [Accessed: 29-May-2020].