

CARTOONIZE THE INPUT IMAGE

For the course

OEC1903- Digital Image processing

Submitted by

1. ATHISTA VIGNESH P S (191001013)

2. BHARATH KUMAR K (191001015)

***Department of Information Technology****,*

**Rajalakshmi Engineering College, Thandalam**

**Anna University, Chennai.**

Bonafide Certificate

This is to certify that the project work titled “**CARTOONIZE THE INPUT IMAGE**” done by “**ATHISTA VIGNESH P S & BHARATH KUMAR K”** of **Department of Information Technology** , is a record of bonafide work carried out by him/her under my supervision.

Head of the Department/IT Faculty in-charge

INDEX

|  |  |  |
| --- | --- | --- |
| **C.NO** | **Chapter heading** | **P.No.** |
|  | Abstract | 3 |
|  | List Of Abbreviations | 3 |
|  | List Of Figures | 3 |
| 1 | Introduction | 4 |
|  | 1.1 Problem Statement | 4 |
|  | 1.2 Objective of the project | 4 |
|  | 1.3 Organization of the report | 4 |
| 2 | System Design | 5 |
|  | 2.1 UML Modelling | 5 |
|  | 2.1.1 Use Case Diagram | 5 |
|  | 2.1.2 Activity Diagram | 7 |
|  | 2.2 System specification | 9 |
|  | 2.2.1 Software requirements | 9 |
|  | 2.2.2 Hardware requirements | 9 |
|  | 2.2.3 Tools / Platforms | 9 |
| 3 | System architecture | 9 |
|  | 3.1 Framework | 9 |
|  | 3.2 Algorithms | 10 |
| 4 | Implementation and Results | 10 |
|  | 4.1 Inputs | 10 |
|  | 4.2 Outputs | 12 |
| 5 | Conclusion | 13 |
|  | Bibliography | 13 |
|  | Appendix A: Coding | 14 |

**Abstract:**

This report represents different techniques of converting image to cartoon. Using any one of below mentioned techniques it is possible to convert all types of captured images to cartoon such as images of person, mountains, trees, flora and fauna etc. There are several other techniques for image to cartoon conversion such as using photoshop, adobe illustrator, windows MAC, paint.net and much more.

**LIST OF ABBREVATIONS :**

* IDLE - Integrated Development and Learning Environment
* RAM – Random Access Memory
* GB – Giga Byte
* HDD – Hard Disk Drive
* MB – Megabyte
* JPG or JPEG – Joint Photographic Experts Group
* PNG – Portable Network Graphics
* RGB – Red, Green, Blue

**LIST OF FIGURES :**

* 2.1 A - Use Case Diagram
* 2.1 B - Activity Diagram
* 4.1 A - Sample Source Image - 1
* 4.1 B - Sample Source Image - 2
* 4.2 A - Sample Cartoon Image - 1
* 4.2 B - Sample Cartoon Image – 2

**CHAPTER 1**

**INTRODUCTION**

* 1. **Problem Statement**

With the recent success of Instagram, the popularity of simple and fun photo effects apps has been on the rise. The computer platform presents a unique arena for these applications by connecting users with both the means to capture images, and the computational power to perform sophisticated processing on these images. Tonify seeks to leverage the existing OpenCV library in order to emulate a particular effect known as Bilateral Effect in the computer graphics world.

* 1. **Objective Of the Project**

Creating a cartoon like effect is time and space consuming. Existing solutions to provide cartoon like effect to images are complex. Some solutions involve installing complex photo editing software like photoshop and other involve performing some tasks by user. Our research shows a website to carry out the task of Applying effects is more suitable, space efficient and takes minimum user efforts, for example toon photos is an existing website to perform such task but it is difficult to use as user has to markdown points & lines on the image to apply effects which is not user friendly also the options are limited. Hence there is a dire need for a simple program/application which is user friendly and performs the task of applying effects to images very well.

* 1. **Organization Of the Report**

The report is divided into four parts. Each part deals with the different aspects of Cartooning Image . Each part has various chapters explaining in detail.

**Part II : SYSTEM DESIGN**

This part discusses the important component system design behind Cartooning.

Chapter 2.1 discusses the UML Modelling and it consists of 2 sub chapters .

Chapter 2.1.1 describes the USE CASE diagram

Chapter 2.1.2 describes the ACTIVITY diagram.

Chapter 2.2 describes the SYSTEM SPECIFICATION required to run the program and it consists of 3 sub chapter.

Chapter 2.2.1 discuss the SOFTWARE REQUIREMENT.

Chapter 2.2.2 discuss the HARDWARE REQUIREMENT.

Chapter 2.2.3 discuss the Tools/Platform.

**Part III : SYSTEM ARCHITECTURE**

This part discusses the SYSTEM ARCHITECTURE for the program.

Chapter 3.1 discusses the FRAMEWORK

Chapter 3.2 discusses the ALGORITHM

**Part IV : IMPLEMENTATION AND RESULTS**

This part discusses the IMPLEMENTATION AND RESULTS of the problem mentioned.

Chapter 4.1 discusses the format and size of the input image.

Chapter 4.2 discusses the format and manner by which the output image is displayed.

**Part V : CONCLUSION**

Finally, this part concludes the project and discusses some future work.

**CHAPTER 2**

**SYSTEM DESIGN**

**2.1 UMI Design**

**2.1.1 Use Case Design**

The use case diagram is used to identify the primary elements and processes that form

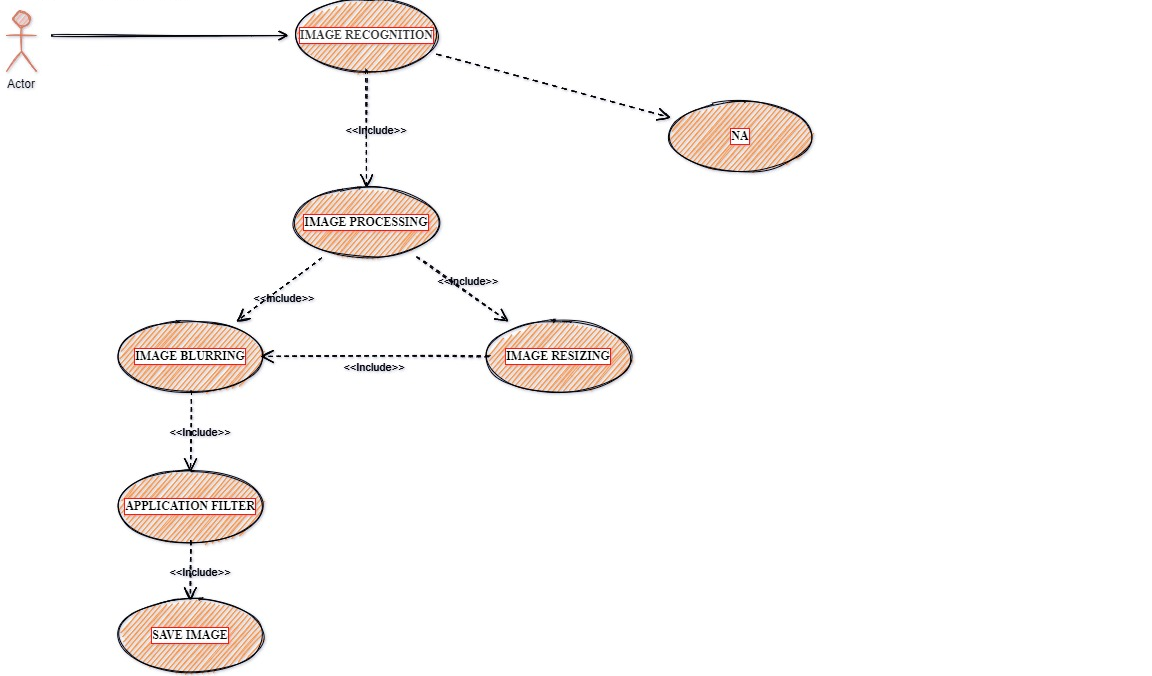
the system. It defines a goal-oriented set of interactions between external actors and

the system. The primary elements are termed as “actors” and the processes are called

use cases or actions. Actors are entities that will utilize the application in order to

complete a task. An actor maybe a class of users, roles users can play or other

systems. A primary actor is one having a goal requiring the assistance of the system. Since this project focuses more on the image processing phase, this particular use case diagram will focus on the actor i.e. the web camera and its part in the functionality of the software.

****

**FIGURE 2.1 A**

Figure 2.1 a) shows the use case diagram that includes a actor, where “user” is the primary actor. As said earlier, the diagram focuses more on the actor as this project deals mainly with the image processing phase

Below are the Use Cases, these go through each use case and detail what actions are performed by the actor and what outcome is expected.

**Use Case 1:** Image Capture/Image Loader

**Actor:** User

**Goal:** To capture an image of the user from the video sequence or load an image.

**Overview:** The user feeds an image of the user that will be processed to extract useful information.

**Use Case 2:** Image Processing

**Actor:** User

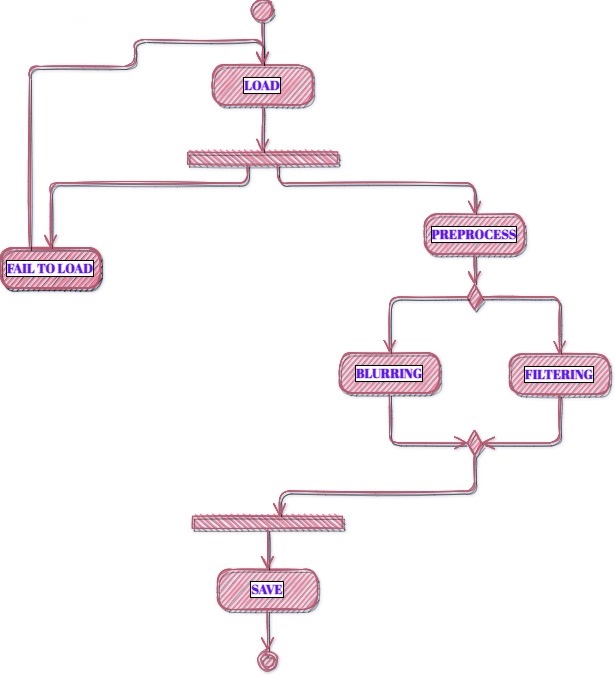
**Goal:** Process the image fed to the system.

**Overview:** Resize the image fed to the system and pre-process it using blur effect and filter effect.

**2.1.2 Activity Diagram**

The process flows in the system are captured in the activity diagram. An activity diagram consists of activities, actions, transitions, initial and final states and guard conditions.

In figure 2.1 B, first the image is loaded by the user is in RGB colour space. If the image is not found or if it corrupted, the user will be required to externally restart the program. The image is resized first and in order to eliminate the noise in the image loaded, first pre-processing on the image is done. The image is first blurred with any type of blurring ana then it is filtered using a filter method. After the image has been pre-processed, it is given threshold function and the outline of the image is obtained. Finally, the colour of the image is reduced and combined with outline image it is saved as a cartoon image.

****

**FIGURE 2.1 B**

**2.2. System specification**

**2.2.1. Software requirements**

* **IDLE:** Any IDLE that interprets python.
* **Python\* versions:** 2.7.X, 3.6.X
* **OpenCv**
* **Easygui**
* **Tkinter**

**2.2.2. Hardware requirements**

* **Processors:** Intel Atom® processor or Intel® Core™ i3 processor
* **Disk space:** 1 GB

**2.2.3. Tools / Platforms**

* **Operating systems:** Windows\* 7 or later, macOS, and Linux.

**CHAPTER 3**

**SYSTEM ARCHITECTURE**

**3.1 Framework**

The Waterfall methodology—also known as the Waterfall model—is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, and testing, for example), with each phase completely wrapping up before the next phase begins.

It is said that the Waterfall methodology follows the adage to “measure twice, cut once.” The success of the Waterfall method depends on the amount and quality of the work done on the front end, documenting everything in advance, including the user interface, user stories, and all the features’ variations and outcomes. With the majority of the research done upfront, estimates of the time needed for each requirement are more accurate, and this can provide a more predictable release date. With a Waterfall project, if parameters change along the way, it’s harder to change course than it is with Agile methodology.

**3.2 Algorithm**

Cartooning an image is a responsive program as it involves constant adjustment to get the most cartooned image that the users will be needing. Here's some algorithms to play with:

* Median or repeated box blur filter to obtain cartoonish color palette.
* Bilateral filtering should suit your needs even better.
* Min filter (zeroth percentile) to enhance some types of edges
* Color image segmentation using either small sub cube or sphere in the RGB color cube or Kmeans algorithm.
* Generic edge enhancement on segmented image using edge detection such as Sobel kernels or 8-way edge tracing
* Composite blurred/median-filtered image with enhanced edges
* Image Threshold to adjust the image to the user’s preference.

These are fairly basic and all very easy to implement. Keep in mind that median and box blur filters can be implemented with linear time complexity with respect to the kernel radius.

**CHAPTER 4**

**IMPLEMENTATION AND RESULTS**

**4.1 Inputs**

A Pop-Up window will be asking the user to select the image which is needed to be cartooned. The input image can be in any size and any format (e.g., JPG, JPEG, PNG,….)



**FIGURE 4.1 A) SAMPLE SOURCE IMAGE -1**

****

**FIGURE 4.1 B) SAMPLE SOURCE IMAGE -2**

**4.2 Outputs**

A New button ‘Save image’ will appear as soon as the image finish processing. Clicking on this button will save the cartooned image in the same directory and in the same format.



**FIGURE 4.2 A) SAMPLE CARTOON IMAGE -1**

****

**FIGURE 4.2 A) SAMPLE CARTOON IMAGE -2**

**CHAPTER 5**

**CONCLUSION**

**5.1 Conclusion**

All things considered, the Tonify algorithm is a success. It is capable of producing exactly the effect specified above, and a wide range of input images will yield satisfactory results. However, the algorithm is not perfect, and it does not respond predictably across all inputs. Given the variety of input images, it would be unrealistic to expect a one-size-fits-all approach to produce consistent results. In the future, an adaptive algorithm would probably be better suited to providing a consistent effect. Such an algorithm would recognize image types (portrait, indoor scene, outdoor scene for instance) and then change the edge detection parameters to better suit that particular image. Additionally, the algorithm may rely on an image derived from the hue, rather than the luminance, for edge detection. This might aid the algorithm in drawing contours around regions of different colours. Currently, the algorithm seems best suited for images that have a few large regions, each with relatively low colour diversity. Typically, the images that fail are those with a high amount of local colour variation and detail.

**BIBLOGRAPHY:**

[1]. A Neural Algorithm of Artistic Style, 2016 - Leon A. Gatys, Alexander S. Ecker, Matthias Bethge

[2]. Image Style Transfer Using Convolutional Neural Networks, 2016 - Leon A. Gatys, Alexander S. Ecker, Matthias Bethge

[3]. Perceptual Losses for Real-Time Style Transfer and Super-Resolution, 2016 - Justin Johnson, Alexandre Alahi, Li Fei-Fei

[4]. Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks, 2016 - Chuan Li, Michael Wand

[5]. Texture networks: Feed-forward synthesis of textures and stylized images, 2016 - D. Ulyanov, V. Lebedev, A. Vedaldi, V. Lempitsky

[6]. Demystifying Neural Style Transfer, 2017 - Yanghao Li, Naiyan Wang, Jiaying Liu, Xiaodi Hou

[7]. A Learned Representation For Artistic Style, 2017 - Vincent Dumoulin, Jonathon Shlens, Manjunath Kudlur

**APPENDIX A: CODING**

#CARTOONIZING AN IMAGE

import cv2

import numpy as np

import tkinter as tk

import os

from tkinter import \*

from PIL import Image

import easygui

def upload():

loc=easygui.fileopenbox()

cartoon(loc)

def save(img, ImagePath):

newName="cartoonified\_Image"

path1 = os.path.dirname(ImagePath)

extension=os.path.splitext(ImagePath)[1]

path = os.path.join(path1, newName+extension)

cv2.imwrite(path, img)

I= "Image saved by name " + newName +" at "+ path

tk.messagebox.showinfo(title=None, message=I)

top.destroy()

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 cartoon(loc):

img = cv2.imread(loc)

h,w,o = img.shape

if(w>2400):

img = cv2.resize(img, (int(w/3),int(h/3))

# Apply a bilateral filer on the image

img\_bf = cv2.bilateralFilter(img\_mb, 5, 80, 80)

# Use the laplace filter to detect edges

img\_lp\_al = cv2.Laplacian(img\_bf, cv2.CV\_8U, ksize=5)

# Convert the image to greyscale (1D)

img\_lp\_al\_grey = cv2.cvtColor(img\_lp\_al, cv2.COLOR\_BGR2GRAY)

# Remove some additional noise

blur\_al = cv2.GaussianBlur(img\_lp\_al\_grey, (5, 5), 0)

# Apply a threshold (Otsu)

\_, tresh\_al = cv2.threshold(blur\_al, 245, 255,cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

# Invert the black and the white

inverted\_Bilateral = cv2.subtract(255, tresh\_al)

# Reshape the image

img\_reshaped = img.reshape((-1,3))

img\_reshaped = np.float32(img\_reshaped)

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

K = 8

\_, label, center = cv2.kmeans(img\_reshaped, K, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)

center = np.uint8(center)

res = center[label.flatten()]

# Reshape it back to an image

img\_Kmeans = res.reshape((img.shape))

# Reduce the colors of the original image

div = 68

img\_bins = img // div \* div + div // 2

inverted\_Bilateral = cv2.cvtColor(inverted\_Bilateral, cv2.COLOR\_GRAY2RGB)

cartoon\_Bilateral = cv2.bitwise\_and(inverted\_Bilateral, img\_bins)

cartoon\_Bilateral = cv2.resize(cartoon\_Bilateral, (w,h))

save1=Button(top,text="Save cartoon image",command=lambda: save(cartoon\_Bilateral, loc),padx=30,pady=5)

save1.configure(background='#364156', foreground='white',font=('calibri',10,'bold'))

save1.pack(side=TOP,pady=50)

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()