# **Cartoonify Image Using OpenCV and Python**

## Introduction

In this project, we will build one interesting application that will cartoonify the image provided to it. To build this cartoonifyer application we will use python and OpenCV.

we convert image to cartoon byusing OpenCV and [python](https://www.analyticsvidhya.com/blog/2022/06/video-game-clustering-using-python/) as the programming language.

## OpenCV

There is a library called [OpenCV](https://opencv.org/about/) which provides a common infrastructure for computer vision applications and has optimized-machine-learning algorithms. It can be used to recognize objects, detect, and produce high-resolution images.

## Requirements

Python: We use python as a programming language for building the application.

cv2: We use cv2 for image processing.

Numpy: Mainly NumPy is used for dealing with arrays. Here the images that we use are stored in the form of arrays. So for that, we use NumPy.

easygui: easygui is a module used for GUI programming in python. In our application easygui is used to open the file box to upload images from the local system.

Imageio: Imageio is a python library that reads and writes the images.

Matplotlib: Matplotlib is used for visualization purposes. Here we plot the images using matplotlib.

## **Implementation**

To create a cartoon effect, we need to pay attention to two things; **edge** and **color palette**. Those are what make the differences between a photo and a cartoon. To adjust that two main components, there are four main steps that we will go through:

1. Load image
2. Create edge mask
3. Reduce the color palette
4. Combine edge mask with the colored image

# **1. Load Image**

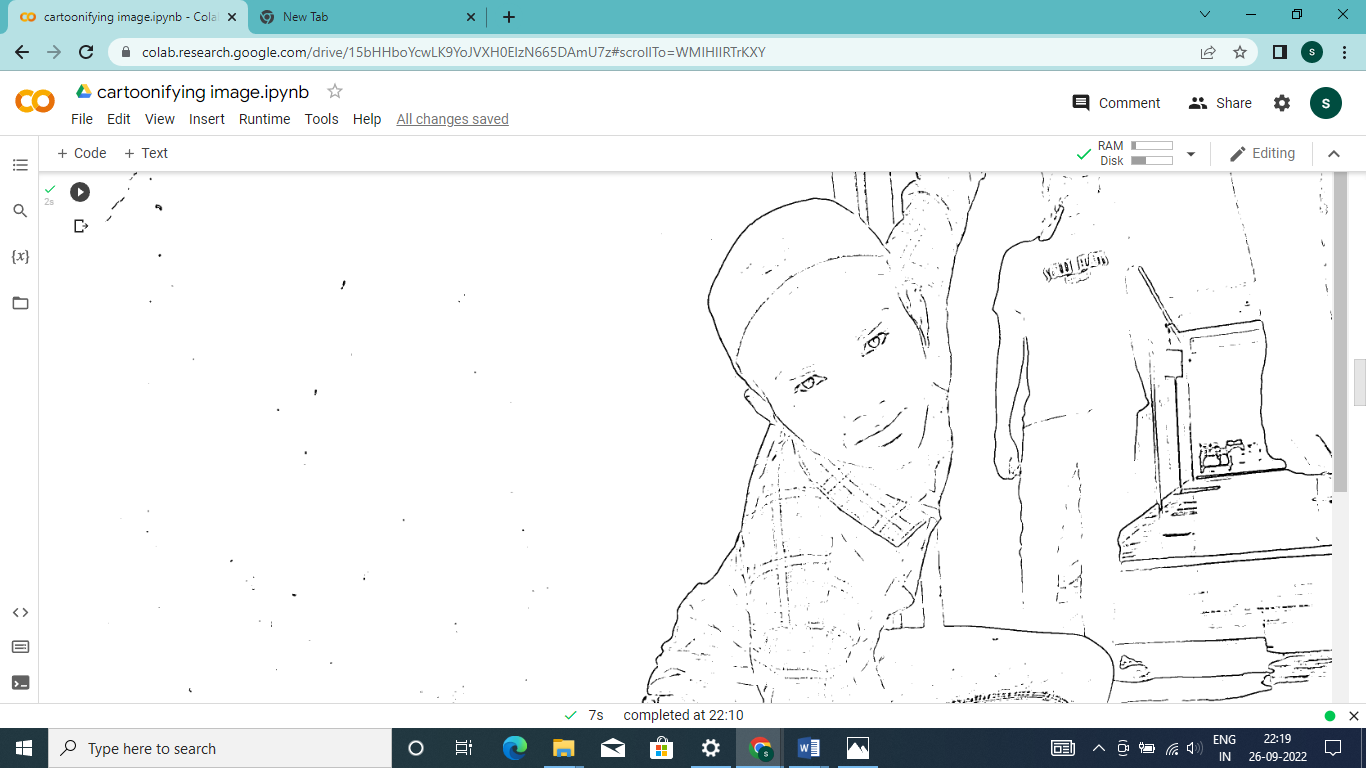
The first main step is loading the image. Define the **read\_file** function, which includes the **cv2\_imshow** to load our selected image in Google Colab.

# **2. Create Edge Mask**

Commonly, a cartoon effect emphasizes the thickness of the edge in an image. We can detect the edge in an image by using the cv2.adaptiveThreshold() function.

In that function, we transform the image into grayscale. Then, we reduce the noise of the blurred grayscale image by using cv2.medianBlur. The larger blur value means fewer black noises appear in the image. And then, apply adaptiveThreshold function, and define the line size of the edge. A larger line size means the thicker edges that will be emphasized in the image

—>after this output



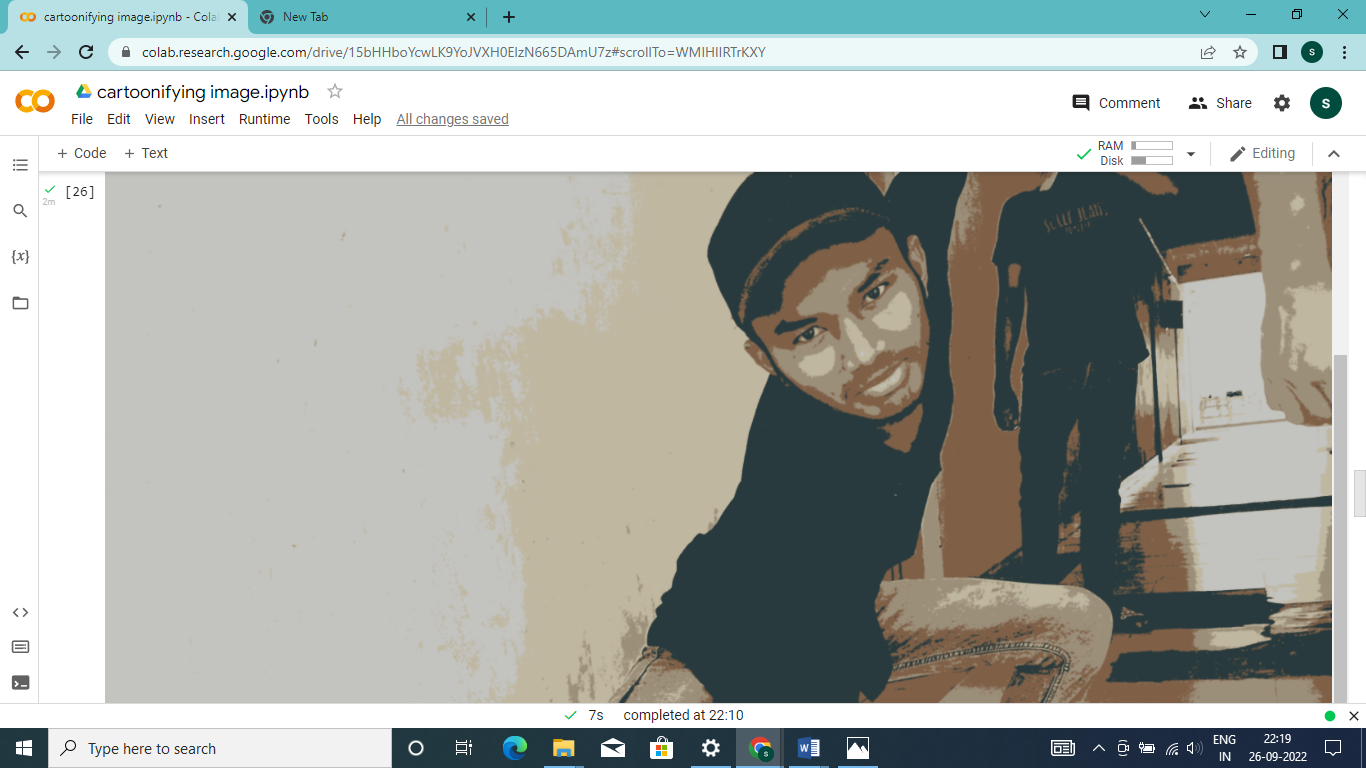
# **3. Reduce the Color Palette**

The main difference between a photo and a drawing — in terms of color — is the number of distinct colors in each of them. A drawing has fewer colors than a photo. Therefore, we use **color quantization** to reduce the number of colors in the photo.

## **Color Quantization**

To do color quantization, we apply the K-Means clustering algorithm which is provided by the OpenCV library

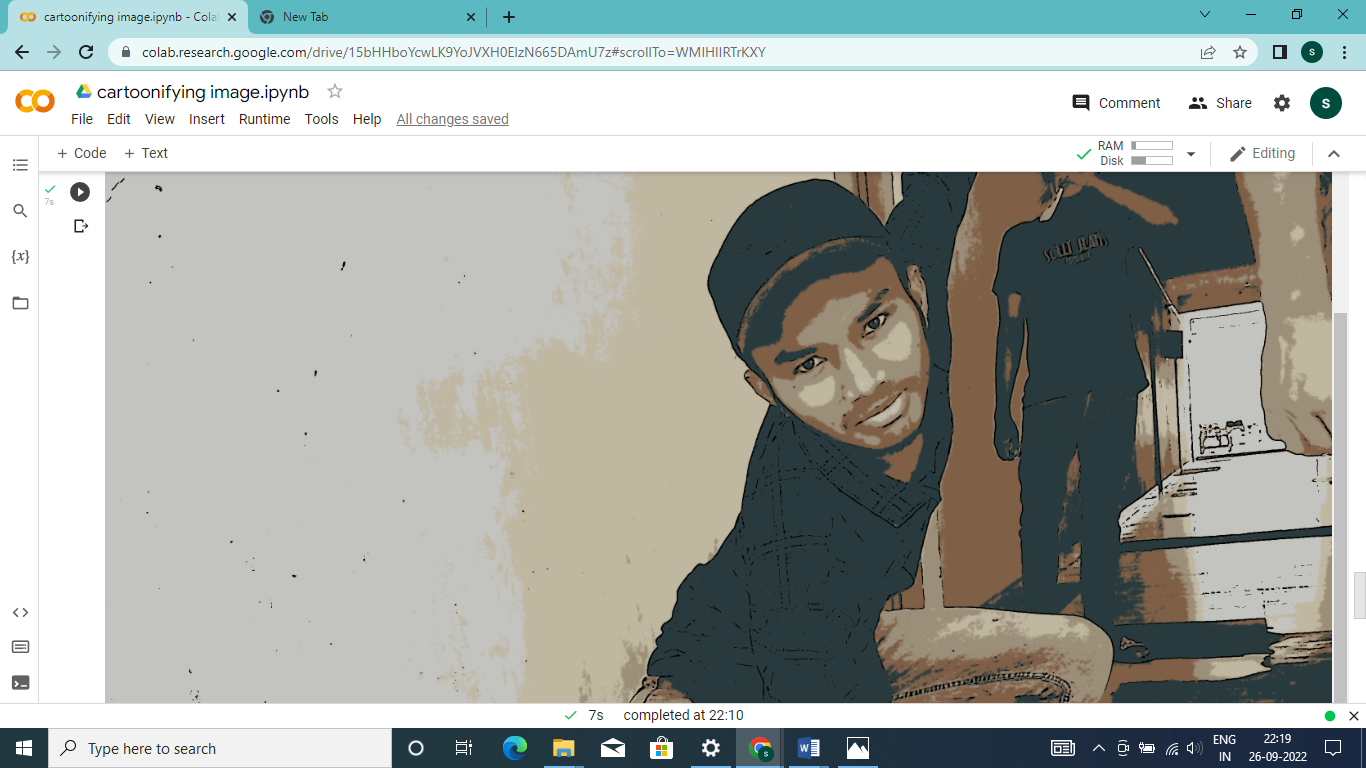
–>after this output:



# **4. Combine Edge Mask with the Colored Image**

The final step is combining the edge mask that we created earlier, with the color-processed image. To do so, use the cv2.bitwise\_and function.

→The final cartoon image:



YUGANDHAR

S180234

T.SHYAM

S180222