DIGITAL IMAGE

By

NAME: Amr Mahmoud Abdelghany mohamed

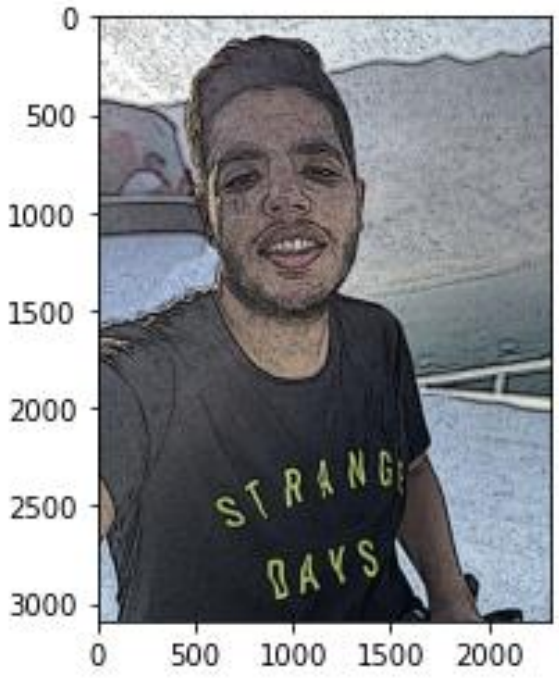
Project

Supervisor by

DR\ Shady Zahran

The original photo:

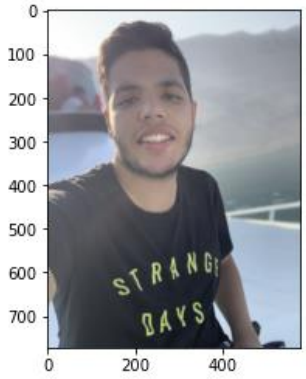
The cartooning photo:



Code:

```
num_down = 2 # number of downsampling steps
```

```
for _ in range(num_down):
    img_color = cv2.pyrDown(img_color)
```



Downsample: Reduce image size by half after each smoothing

Bilateral Filter: Is highly effective in noise removal while keeping edges sharp

```
for _ in range(num_bilateral):
    img_color = cv2.bilateralFilter(img_color, d=4, sigmaColor=4, sigmaSpace=7)
```

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}},$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|).$$

W_p = normalization factor

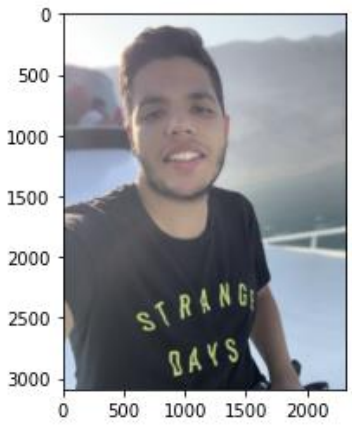
$G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)$ = space weight

$G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$ = range weight

```
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB)
```

cv2.cvtColor: method is used to convert an image from one color space to another

```
for _ in range(num_down):
    img_color = cv2.pyrUp(img_color)
```

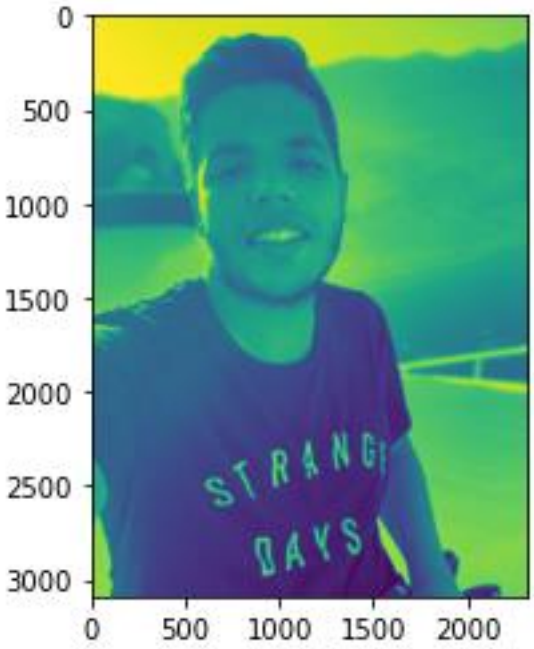


When we use cv2.pyrDown We Reduce image size by half so we use cv2.pyrUp to return image to the original size

pyrUp: Reduce image size by double

```
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
```

Convert the image to gray



```
img_blur = cv2.medianBlur(img_gray, 7)
```

Applying median: the median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the *mean* of neighboring pixel values, it replaces it with the *median* of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value.

```
img_edge = cv2.adaptiveThreshold(img_blur, 255,
    cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY,
    blockSize=13,
    C=0.4)
plt.imshow(img_edge)
```

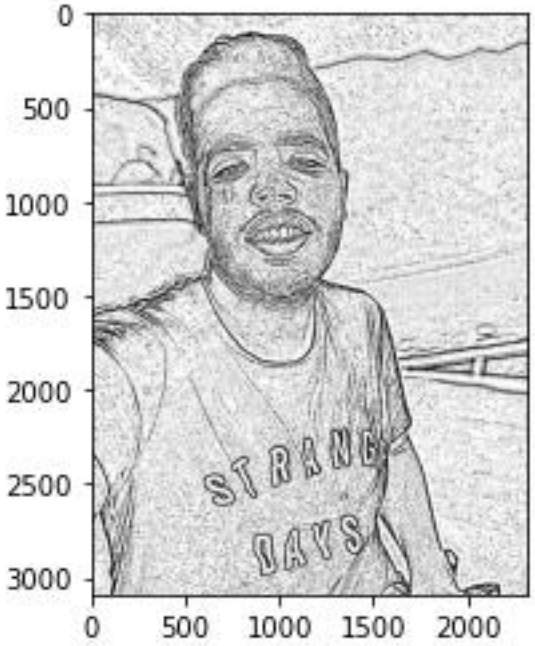
adaptiveThershold: calculate the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results

blockSize: It decides the size of neighborhood area

C: constant which is subtracted from the mean or weighted mean calculated.

```
img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)
```

Combine color image with edge mask

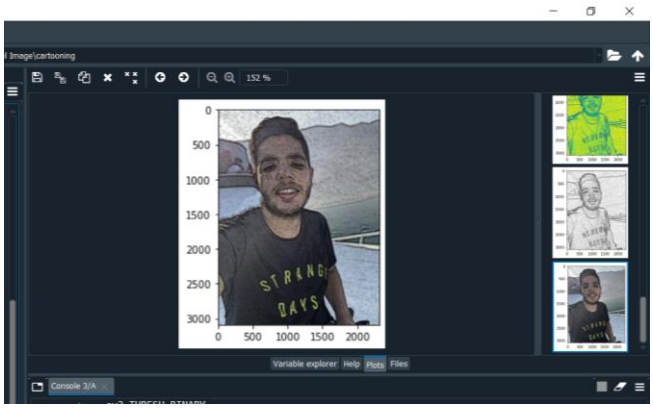



```
img_cartoon = cv2.bitwise_and(img_color, img_edge)
```

Convert back to color

“And finally it’s cartooning image”

All code:



```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 19 08:01:01 2020

@author: Amr Mahmoud
"""

#step 1
#Use bilateral filter for edge-aware smoothing.
import cv2
import matplotlib.pyplot as plt

num_down = 2 # number of downsampling steps
num_bilateral = 7# number of bilateral filtering steps

img_rgb = cv2.imread("a.jpeg")
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB)

# downsample image using Gaussian pyramid
img_color = img_rgb


for _ in range(num_down):
    img_color = cv2.pyrDown(img_color)
    plt.imshow(img_color)
```

```
# repeatedly apply small bilateral filter instead of
# applying one large filter
for _ in range(num_bilateral):
    img_color = cv2.bilateralFilter(img_color, d=4, sigmaColor=4, sigmaSpace=7)
    plt.imshow(img_color)

# upsample image to original size
for _ in range(num_down):
    img_color = cv2.pyrUp(img_color)

#STEP 2 & 3
#Use median filter to reduce noise
# convert to grayscale and apply median blur
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
img_blur = cv2.medianBlur(img_gray, 7)

plt.imshow(img_gray)
plt.imshow(img_blur)

#STEP 4
#Use adaptive thresholding to create an edge mask
# detect and enhance edges
img_edge = cv2.adaptiveThreshold(img_blur, 255,
   cv2.ADAPTIVE_THRESH_MEAN_C,
   cv2.THRESH_BINARY,
   blockSize=13,
   C=0.4)
plt.imshow(img_edge)
# Step 5
```

```
# Combine color image with edge mask & display picture
# convert back to color, bit-AND with color image
img_edge = cv2.cvtColor(img_edge, cv2.COLOR_GRAY2RGB)
img_cartoon = cv2.bitwise_and(img_color, img_edge)

# display
plt.imshow( img_cartoon)
```