Image Filter Document

Farid Ghorbani
Azad University of Tehran
Tehran, Iran
June 2019

0. Tools:

In this project, we have utilized the OpenCV, imageio, and numpy libraries. Initially, we install these modules using the command pip3 install, and then by importing them into our program, they provide us with access to a myriad of functions.

1. Functions:

1.1. image_resize

The inputs to this function are the input image, the desired length, width, and the mode of the resize function.

It operates as follows: first, it adjusts the dimensions based on the height, and then it calls the resize function from OpenCV.

```
    r = height / float(h)
    dim = (int(w * r), height)
```

```
resized = cv2.resize(image, dim, interpolation = inter)
return resized
```

1.2. make_same_size

This function takes two input images and resizes the larger image to match the size of the smaller one. In simpler terms, it makes both images the same size.

```
def make_same_size(img1, img2):
    x = min(img1.shape[0], img2.shape[0])
    y = min(img1.shape[1], img2.shape[1])
    img1 = img1[0:x, 0:y]
    img2 = img2[0:x, 0:y]
    return (img1, img2)
```

1.3. edge_render

This function takes an input image and outputs an image that represents its edges using the algorithm described below:

Edge Detection Algorithm (Sobel Operator):

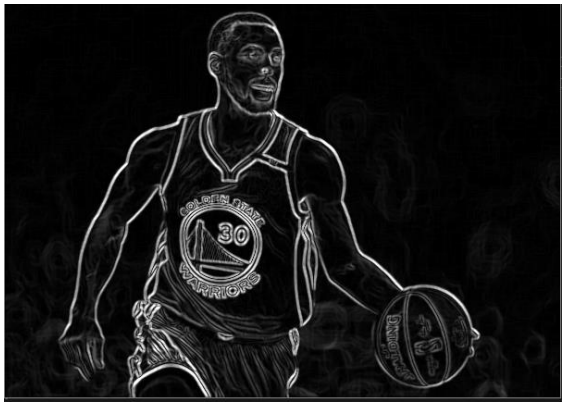
- The first step is to compute the derivative of the input image to find its edges. Edges are areas where pixel intensity changes suddenly, and by taking the derivative, we can determine the gradient of this change. In locations where the gradient of the derivative is greater than that of its neighbors, it can be inferred that an edge exists.

Implementation in the Function blackAndWhite_rende:

- Initially, the function applies the Sobel operator in both the x and y directions using OpenCV's built-in functions on the output image from the GaussianBlur function.
- The resulting grad_x and grad_y values represent the gradients in the x and y directions, respectively, at each pixel. Their combination at each point approximates the gradient at that point, denoted as grad.

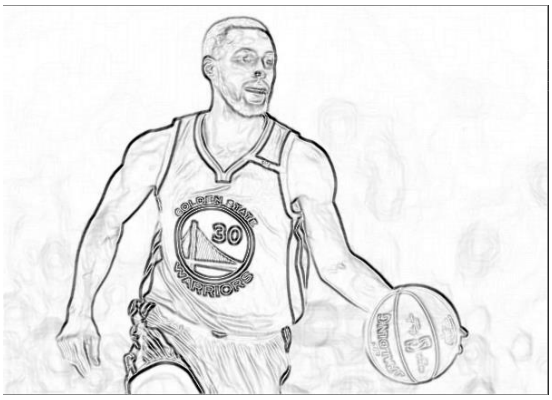
```
src = img_rgb
img = src

src = cv2.GaussianBlur(src, (3, 3), 0)
gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

grad_x = cv2.Sobel(gray, cv2.CV_16S, 1, 0, ksize=3, scale=1, delta=0, borderType=cv2.BORDER_DEFAULT)
grad_y = cv2.Sobel(gray, cv2.CV_16S, 0, 1, ksize=3, scale=1, delta=0, borderType=cv2.BORDER_DEFAULT)
abs_grad_x = cv2.convertScaleAbs(grad_x)
abs_grad_y = cv2.convertScaleAbs(grad_y)
grad = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
```

After computing the gradients (grad_x and grad_y), the function then displays the gradient magnitude, denoted as grad, which represents the overall intensity of the edges in the image. This visualization helps to understand the strength of the edges detected across the image.



After identifying the edges using the Sobel operator, we subtract the value 255 from the gradient (grad) to obtain the resulting image, which represents the edges. This process effectively enhances the visibility of the edges in the image, producing the desired output of the function.

```
img_edge = 255-grad
```



2. Filters:

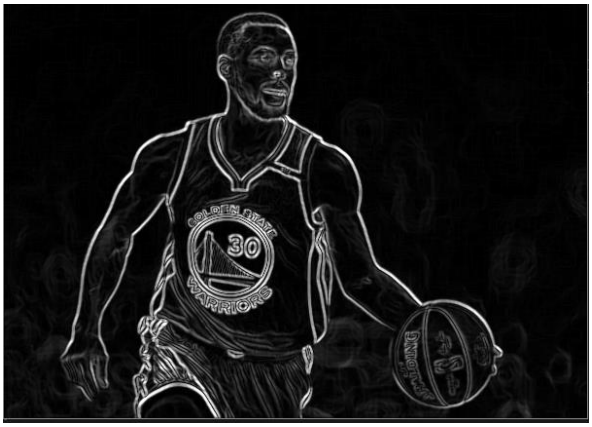
2.1. blackAndWhite

Algorithm:

1. Identify the edges of the input image (grad).
2. Iterate over all pixels of the gradient (grad).
3. For each pixel, check if its value is less than 85.
    - If the pixel value is less than 85, set the value to 0 for all three RGB channels in the input image.
    - If the condition is not met, leave the pixel value unchanged in the input image.

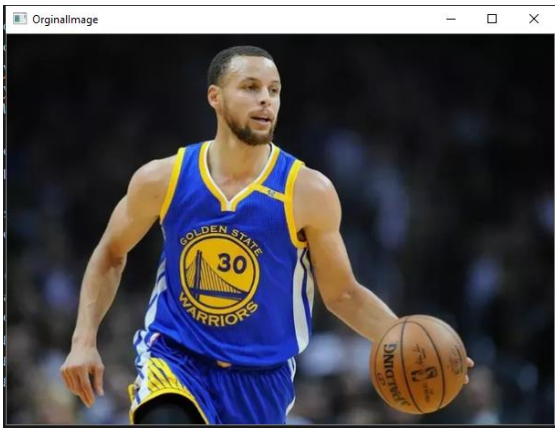
This algorithm modifies the pixel values of the input image based on the gradient magnitude (grad). Pixels with low gradient values are set to black, while the rest remain unchanged.

```
def blackAndColor_render (img_rgb):
    src = img_rgb
    img = src

    src = cv2.GaussianBlur(src, (3, 3), 0)
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

    grad_x = cv2.Sobel(gray, cv2.CV_16S, 1, 0, ksize=3, scale=1, delta=0, borderType=cv2.BORDER_DEFAULT)
    grad_y = cv2.Sobel(gray, cv2.CV_16S, 0, 1, ksize=3, scale=1, delta=0, borderType=cv2.BORDER_DEFAULT)
    abs_grad_x = cv2.convertScaleAbs(grad_x)
    abs_grad_y = cv2.convertScaleAbs(grad_y)
    grad = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
```



The threshold value of 85 has been applied empirically.

```
for i in range(0 ,grad.shape[0]):
    for j in range(0 ,grad.shape[1]):
        if grad[i][j] <= 85:
            img[i][j][0] = 0
            img[i][j][1] = 0
            img[i][j][2] = 0
```



Input Image

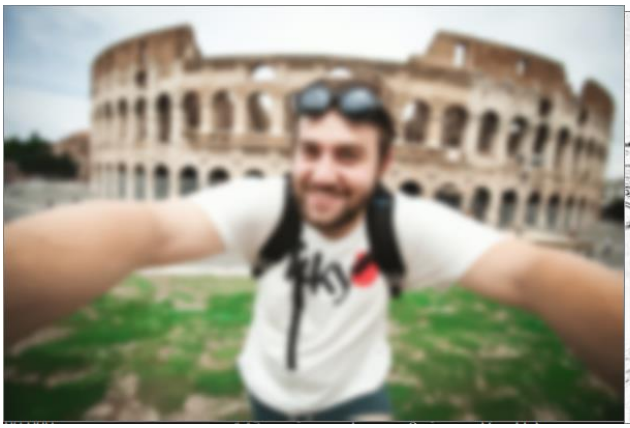
Filter Output

2.2. cartoon_render

A - The algorithm begins by smoothing the image using a bilateral filter, achieved through the functions cv2.pyrDown, cv2.bilateralFilter, and cv2.pyrUp.

B - Next, we utilize the edge_render(img) function, as explained earlier, to detect the edges of the image. The first image shows the output of the bilateral filter function, and the second image displays the output of edge_render(img).

```python
def cartoon_render (img_rgb):
    img_edge = edge_render(img_rgb)
    numDownSamples = 2
    numBilateralFilters = 1

    img_color = img_rgb
    for _ in range(0 ,numDownSamples):
        img_color = cv2.pyrDown(img_color)
    for _ in range(0 ,numBilateralFilters):
        img_color = cv2.bilateralFilter(img_color, 9, 9, 7)
    for _ in range(0 ,numDownSamples):
        img_color = cv2.pyrUp(img_color)
```



Output bilateral filter

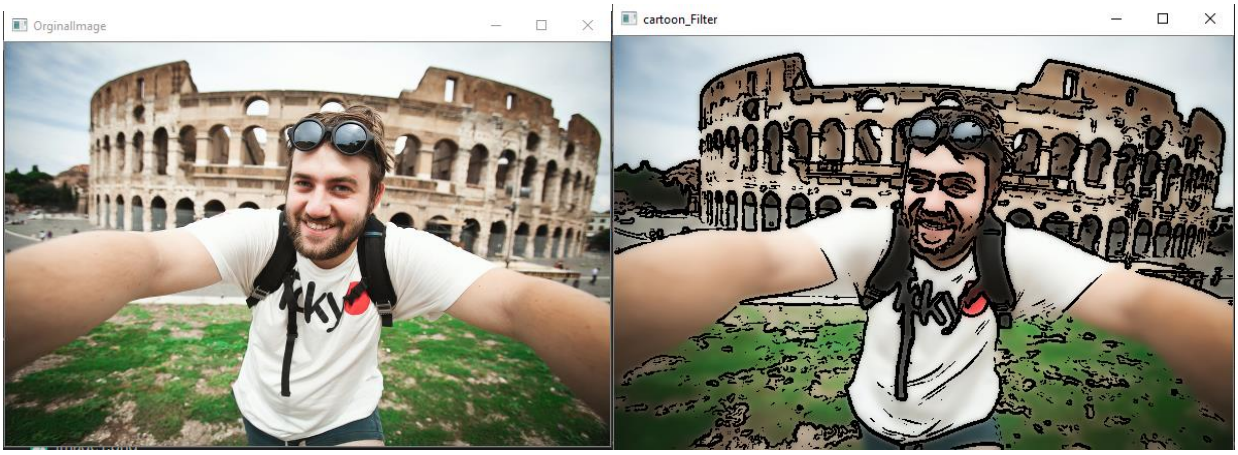
Output edge_render(img)

C - We resize both images to the same dimensions using our custom function described previously. After this step, we perform a bitwise AND operation between these two images, which I manually implemented instead of using a function.

```python
img_color, img_edge = make_same_size(img_color, img_edge)
```

D - We determine a threshold value empirically. Then, we iterate over all pixels in img_edge, setting the value to 0 in the corresponding pixel of img_color if the pixel in img_edge is below the threshold. The output of our program is the img_color image.

```python
for i in range(0 ,img_edge.shape[0]):
    for j in range(0 ,img_edge.shape[1]):
        if img_edge[i][j] <= 190 :
            img_color[i][j][0] = 0
            img_color[i][j][1] = 0
            img_color[i][j][2] = 0
return img_color
```



Input Image **Filter Output**

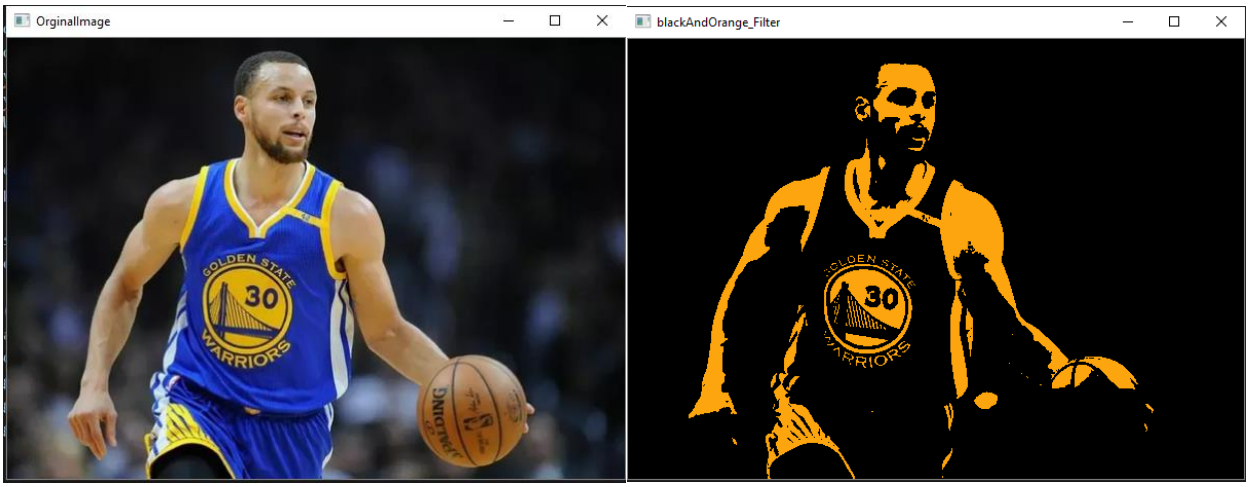
2.3. blackAndOrange

In this algorithm, we first make a copy of the input image. Then, we convert the image to grayscale to obtain the image mask. The threshold level for the mask function is determined empirically. We use the mode cv2.THRESH_BINARY_INV in the mask function, which inverts the result.

```
img = img_rgb.copy()
mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
_, mask = cv2.threshold(mask, 150, 255, cv2.THRESH_BINARY_INV)
```

Next, we iterate over all pixels in the mask image. If the pixel value is greater than 0, we set all channels of the copied image to 0. If it is equal to zero, we assign an orange color value.

```
for i in range(0 ,mask.shape[0]):
    for j in range(0 ,mask.shape[1]):
        if mask[i][j] <= 0:
            img[i][j][0] = 15
            img[i][j][1] = 165
            img[i][j][2] = 253
        else:
            img[i][j][0] = 0
            img[i][j][1] = 0
            img[i][j][2] = 0
return img
```



Input Image **Filter Output**

2.4. sketch_render

Our algorithm consists of 4 steps:

A - First, we convert the input image to grayscale using the grayscale function.

B - Then, we invert the grayscale image. This is achieved by subtracting the image from 255.

```
def sketch_render(img_rgb) :
    start_img = img_rgb
    img_gray = grayScale(start_img)
    img_inverted = 255-img_gray
```

C - Next, we apply a blur to the inverted image using the Gaussian filter function from the imageio library.

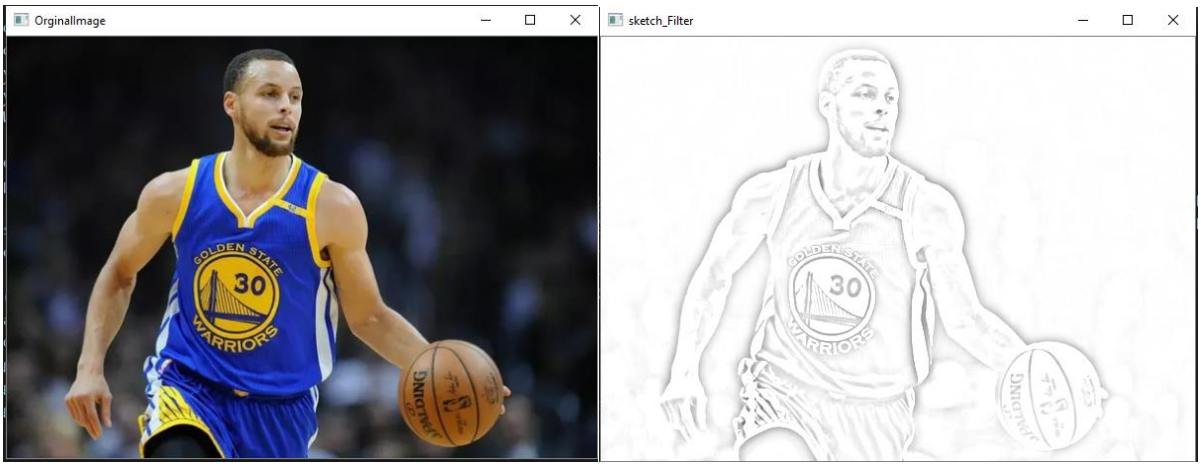
D - Finally, we blend the values of the blurred and grayscale images together using the result() function.

```
    img_blur = scipy.ndimage.filters.gaussian_filter(img_inverted,sigma=5)
    img_color= result(img_blur,img_gray)
    return img_color
```

```python
def grayScale(img_rgb):
    return np.dot(img_rgb[...,:3], [0.299, 0.587, 0.114])

def result(blur,gray):
    r=blur*255/(255-gray)
    r[r>255]=255
    r[gray==255]=255
    return r.astype('uint8')
```

The grayscale function converts the image to grayscale by taking the inner product of its pixel matrix.

In the result function, the value of the blurred image is combined with the value of the grayscale image, resulting in the output of our filter.



Input Image **Filter Output**

3. References:

[1] https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

[2] https://www.tutorialspoint.com/cartooning-an-image-using-opencv-in-python

[3] https://www.freecodecamp.org/news/sketchify-turn-any-image-into-a-pencil-sketch-with-10-lines-of-code-cf67fa4f68ce/