Kakuna Image Editor

Melinnur KILIÇARSLAN
Gazi University Engineering Faculty
Eti, Yükseliş Sk. No. 5, 06570
Çankaya/Ankara, TURKEY
+905069877752
melinnur.kilicarslan@gmail.com

İsa GÜLEN
Gazi University Engineering Faculty
Eti, Yükseliş Sk. No. 5, 06570
Çankaya/Ankara, TURKEY
+905318347111
isa.gulenn@gmail.com

Fazıl Kürşad TİYEKLİ
Gazi University Engineering Faculty
Eti, Yükseliş Sk. No. 5, 06570
Çankaya/Ankara, TURKEY
+905317761415
kursadtiyekli@gmail.com

ABSTRACT
Nowadays, social media platforms are getting more populer day by day. People share their opinions, videos and images. These image posts can be modified as the users' wish and moods. People needs softwares to edit their images. In this paper, we describe an image editing tool, which name is 'Kakuna Image Editor', its' creative filters and effects in details.

Keywords
Image editing, Image Filters, Image Effects, Image Operations

1. INTRODUCTION
An original image which is taken by camera, can not be in a good form because of many reasons like the lights' angle of incidence, angle of camera etc. The image may be look better with a few changes on color pixels or transformations. To edit images, users need to softwares or social media platforms like, Instagram, Snapchat, include image editing tools in their apps. According to the statistics, the total number of daily active Instagram users and daily Instagram stories are over than 500 million. Every second 995 photos are uploaded[1]. Only 10% of photos on Instagram have #nofilter hashtag [2]. Of course, this does not mean that 90% of the photos on Instagram are filtered. However, we can conclude that most of them are filtered. The highest downloaded image editing app has nearly 500 million downloads. This means people love using filters and improvements are needed in this area. Kakuna Image Editor has many options and operations to enchance the images. People can apply transformations, filters and effects to images. The point that makes Kakuna different from the other image editing apps is the filters it has. The Kakuna's filters are unique and creative. Thus, Kakuna provides users more options to enchance and edit their images.

2. APPROACH
Kakuna Image Editor is an image editing tool that provides the users to apply transformations, filters and effects on the images.

Kakuna Image Editor has a simple and easy to use interface. It's landing page is as in the Figure 1. To create a GUI application, we used the Python binding for the QT toolkit,known as PyQt5. PyQt consists of a number of Python bindings for cross-platform applications that combine all the strengths of Qt and Python [3]. It is designed using QtDesigner, that is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. It allows you to compose and customize windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test them using different styles and resolutions [4].

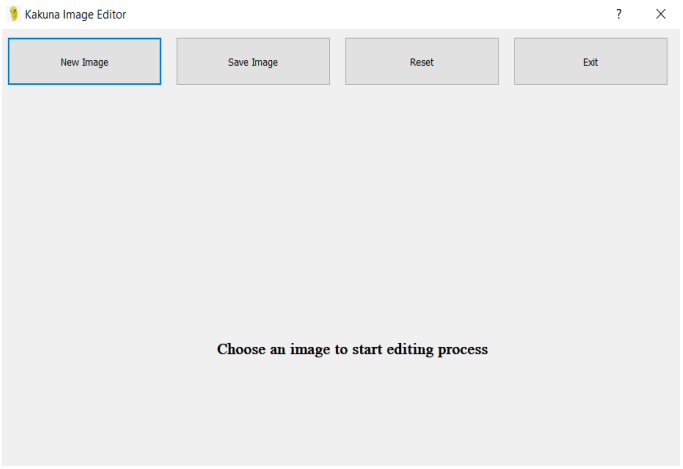

Figure 1. Kakuna Image Editor Landing Page

Buttons, that are located on the top of window, and editing tools are activated when an image is browsed.

Users can browse an image from a file, to edit with "New Image" button. After this action, editing tools will be activated.

Users can save the images to any file, that they want, with "Save Image" button.

With "Reset" button, all of the changes made to the image, are reseted and image returns to its original form.

"Exit" buttton provides the users exit the program and close the window.

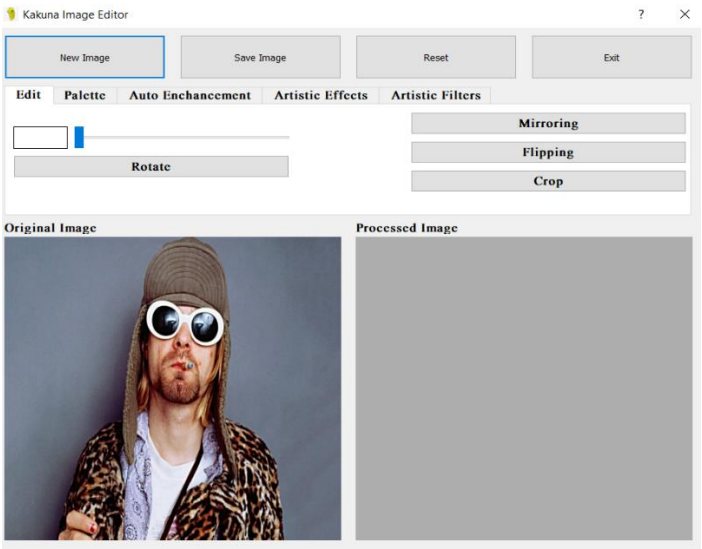

Figure 2. Kakuna Image Editor Edit Page

All of the elements should have a Signal-Slot connection to create an action. This connection can be supplied in different ways. You can make Signal-Slot connection on Qt Designer or you can do this with writing code. We prefferred the second method. Codes are written on PyCharm IDE.

You can see the edit panel in Figure 2. This panel includes basic operations like rotating, mirroring, flipping and cropping.

For rotating process, we have three element on interface. QLabel, QSlider and a button. User slides the bar (QSlider) to determine how much the image rotates, and this rotation value displays on QLabel. After clicking the "Rotate" button, image will be rotated as much as user wanted.

For mirroring, flipping and cropping events, there is only one button on interface. After clicking the buttons, the relevant process will be done.

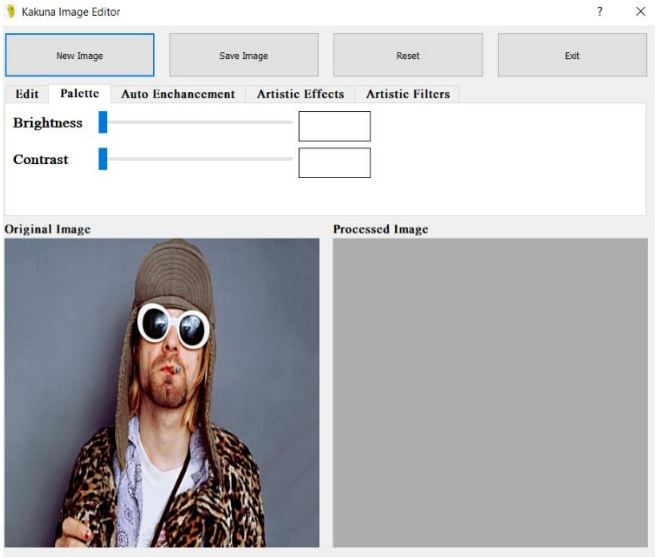

Figure 3. Kakuna Image Editor Palette Page

Palette page is as in Figure 3. Palette section has two components as brightness and contrast. User can set how much they want to apply brigthness and contrastness on image by sliding the Qslider bars.

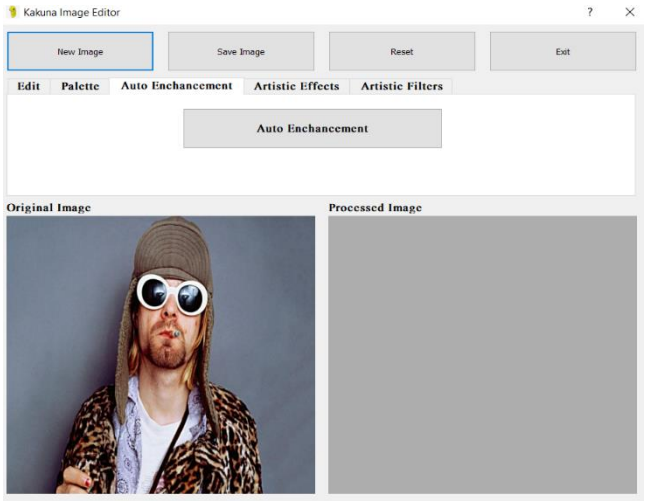

Figure 4. Kakuna Image Editor Auto Enchancement Page

In auto enchancement part, some enchancement methods are applied on images. This section is as in Figure 4. There is a button named as "Auto Enchancement". User can enchance the images automatically by clicking this button. In our program we used histogram normalization and contrast enchancement methods to enchance images. Histogram normalization method is demonstrated as a simple and widely accepted technique for contrast enchancement of images. It achieves overall image contrast enchancement by expanding the dynamics range of the histogram using cumulative distribution function as a transformation function [5].

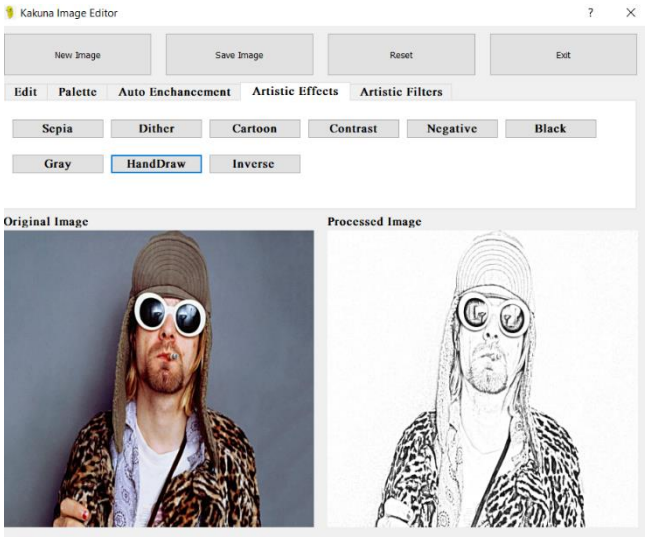

Figure 5. Kakuna Image Editor Artistic Effects Page

Kakuna Image Editor also has the frequently used filters besides its creative filters. This artistic effects page is as in Figure 5. This section has 9 filters as Sepia, Dither, Cartoon, Contrast, Negative, Black-White, Gray, HandDraw and Inverse. These filters are created by doing some processes on pixels' colors. A digital image is made up of multiple pixels coming together. Each pixel has its own color property. There are three main color as Red, Green and Blue. The other colors consists of mixtures of these main colors in specific proportions. Every filter has own mathematic to recolor the pixels to give a new look to the image. This part we used OpenCV library in a few filter to create. In the other filters, we used the mathematical operations, that was tested and used before, on pixels. We don't examine the mathematic or approach for all of the filters in this paper. But we will examine the sepia filter for an example. The other filters have the same logic.

Firstly, we determine the height and width values of the image. We will use this values in for loops to be able to access all of the pixels on the image. As we mentioned before, each pixel has own R,G,B value. If you change this values, the image has a new look. The pixels that has red color can be shown as (255,0,0) if the image format is RGB. This means there is no green and blue color on this pixel. In sepia filter, the pixels, that has red color, is replaced with (r * 0.393 + g * 0.769 + b * 0.189). Then all of the red pixels replaced with the color formed by mixing red, green and blue in the given proportions. Then the green pixels replaced with the color formed by mixing red,green and blue in the propotion as (r * 0.349 + g * 0.686 + b * 0.168). The last step is changing the blue pixels. The blue pixels replaced with the color formed by mixing red,green and blue in the proportion as (r *0.272 + g * 0.534 + b * 0.131).

Finally, sepia filter will be applied on the image. Not all but most of the filters have the same logic: just change the mixture proportion then get a new filter.

As we will explain the Kakuna's creative filters in details, we will examine each of them in titles.

2.1 Green Demon Filter

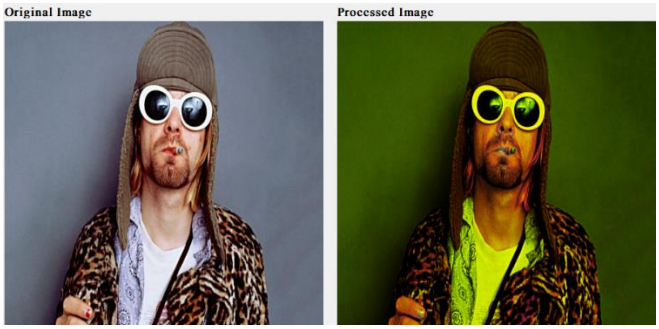

Figure 6. Green Demon Filter

In Green Demon Filter, we change the pixels' color values. For this change, we use this equations as below.

```
pixels[i, j] = min(255, int((green - blue) ** 2 / 128) + red), \
               min(255, int((red - green) ** 2 / 128) + blue), \
               min(255, int((blue - red) ** 2 / 128))
```

For the red ones, we differ the blue values from the green values. After that we calculated this difference exponent 2 and finally we divided this result by 128.Then we added the red pixel value to result. We made a comparison between 255 and this calculated value. Finally the minimum one replaced with the red pixels.

The green and blue pixels were aslso replaced with the values obtained from the above equations.

After getting the new R,G,B values, we apply Image.point(lambda i: int(i ** 2 / 255)) as a final step. Image.point() method provides us to map image through function that is given in parentheses.

Original image and the image, that is applied Green Demon Filter, are as in Figure 6.

2.2 Blue Curacao Filter

Figure 7. Blue Curacao Filter

In Blue Curacao Filter, firstly we applied 2 different Image.point() method with the functions as given below.

```
image.point(lambda i: i ^ 0x8B if i < 128 else i)
image.point(lambda i: i ^ 0x3D if i < 256 else i)
```

We've mentioned what the image.point() method does. 0x8B and 0x3D is in hexadecimal format. In decimal format 0x8B equals 139 and 0x3D equals 61.

Then we replaced the red,green and blue colors with the same logic as the Green Demon Filter but with a different math. The equations are given below.

```
pixels[i, j] = min(100, int(green * blue / 255)), \
               min(100, int(blue * red / 255)), \
               min(255, int(red * green / 255))
```

After all of these calculations, we get the Blue Curacao Filter that is as in Figure 7.

2.3 Mai Tai Filter

Figure 8. Mai Tai Filter

Since there are many calculations in the Mai Filter, the steps will be discussed in general terms. While creating this filter, firstly the pixel values are assigned to a variable. Unlike the other filters, we obtained 4 pixel values in this filter because in this filter we work on RGBA images, not RGB. In this way, we have alpha parameter besides red, green and blue. Alpha parameter allows us to specify the opacity for a color.

Then we converted them to grayscale and calculated the saturation for each of them. This saturation value will be a benchmark for comparison in the next step. We determined some numbers to compare with the saturation value with if-elses. The correct code blocks will be runned as a result of comparison and the pixel values will be changed.

2.4 Black Russian Filter

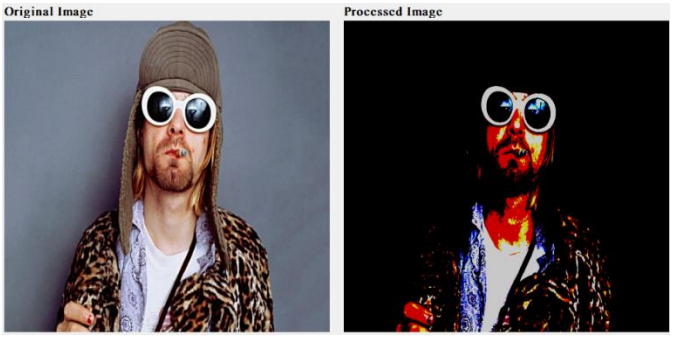

Figure 9. Black Russian Filter

In this filter, we determined a benchmark as 180. If the red,green and blue values bigger than this benchmark, then the pixel values equals to 200. If the red, green and blue values smaller than the benchmark value, then the values equals to 0.

In this way, we are getting darker image that is as in Figure 9.

2.5 Cobain Filter

Figure 10. Cobain Filter

When the Cobain filter is applied on the image, we get an image as seen in Figure 10. Blue color is used predominantly in this filter because of the filter's math.

```
red, green, blue, alpha = pixels[i, j]
pixels[i, j] = min(100, int(red ** .15 / 255)), \
               min(100, int(blue ** .45 / 255)), \
               max(75, int(red * green / 255)), \
               alpha
```

The filter's math is given above. We don't examine in details these calculations thus we did it before. But we will discuss why the image has blue density.As seen above equations, we take the minimum values on red and green values. Their benchmark is bigger than the blue's. In the calculation of blue value, we take the maximum value. So we got a blue color intensive image.

2.6 Carnaval Filter

Figure 11. Carnaval Filter

In the Carnaval filter, instead of working on all pixels of image, operations are applied on pixels in a specific region. We achieved this with a for loop using different incremental values on the x and y axes of the picture. The origin is considered as the point that is located on top-left of the image. X axes is width and y axes is height. A small part of code is given below.

```
for i in range(height):
    for j in range(width):
        image[j:x, x + 10:x + 20, 2] = 80
        image[i:init_value, init_value:x, 0] = 50
        image[x:x + 50, x + 50:x + 150, 1] = 180

        i += 10
        init_value += 30
        x += 30
        j += 10
```

The image, that is applied the Cobain Filter, is as seen in Figure 11.

2.7 Head Power Filter

Figure 12. Head Power Filter

The Head Power filter is a combination of some methods. Firstly, we determined a benchmark as 128. If the pixel values are smaller than this benchmark value, the pixels are equaled to 80.

After changing of the colors, we calculate radius value with some mathematical operations. Based on this radius value, we create a circular mask on the image. Then, the pixels that, is outside of this circular area, are made white color. The code is given below.

```
X, Y = np.ogrid[:rows, :cols]
centrow, centcol = rows / 2, cols / 2
distance = (X - centrow) ** 2 + (Y - centcol) ** 2
radius = (rows / 2) ** 2
circular_mask = (distance > radius)

image[circular_mask] = 255
```

Finally, the Head Power filter is created. The image that is applied this filter, is as Figure 12.

2.8 White Russian Filter

Figure 13. White Russian Filter

White Russian Filter is essentially same as Black Russian Filter. Difference is however determined benchmark is 20 this time. Values that are bigger than 20 is equals to 200, while values below 20 are equals to 0.

2.9 Ocean Filter

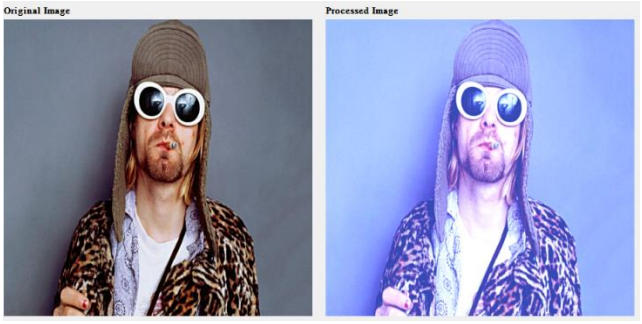

Figure 14. Ocean Filter

In ocean filter, we gave values for each color. 50 for red, 70 for green and 180 for blue. We combined these values on a single variable then, using *np.full* and *cv2.addWeighted* functions, we overlaid these values on the image of choice. We then proceed to convert the image from BGRA to BGR. Results can be seen on the Figure 14.

2.10 Magma Filter

Figure 15. Magma Filter

In magma filter, we used the kernel of [[1, 1, 0],[1, 0, -1],[0, -1, -1]] on the image, which is one of the kernels used while embossing an image. We proceed using *cv2.filter2D* function while also adding color map of COLORMAP_HOT from cv2.

2.11 Greenish Filter

Figure 16. Greenish Filter

As the name of the filter suggests, greenish filter is used to overlay a green filter on the image. We also used *medianBlur* on the image prior to splitting and merging (using *split*, and *merge* functions from cv2) the channels of the image.

2.12 Cotton Candy Filter

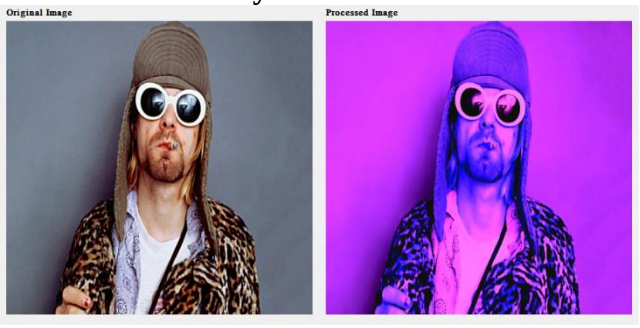

Figure 17. Cotton Candy Filter

In cotton candy filter, we did the same thing we did in greenish filter, though we also used *convertScaleAbs* and *bilateralFilter* functions of cv2 to make it look more aesthetic.

2.13 Dream Filter

Figure 18. Dream Filter

Dream filter was created by the use of *morphologyEx(), normalize(), stylization()* functions of cv2 and also split and merge operations we used in the other filters to change colors of the image. It was an experimental filter. We did some Morphological Transformations. In order to remove the noise we used *MORPH_OPEN* and *MORPH_ELLIPSE* to create elliptical kernels.

2.14 Focus Filter

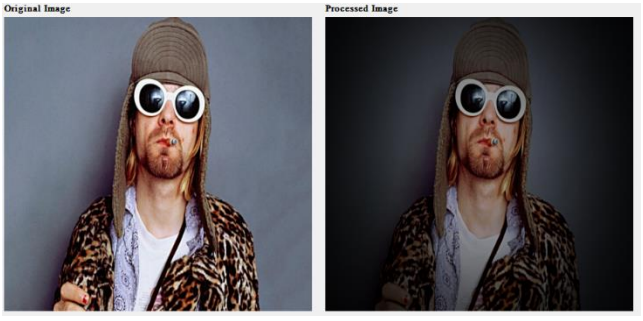

Figure 19. Focus Filter

Focus filter was done by the use of Gaussian kernels. We took the height and width values of the image first, following by using the getGaussianKernel() function with the Aperture size of height and width values. We did it for both of the values and then multiplied them for a kernel we can use. We then used numpy.linalg.norm to get the norm of the kernel and masked it on image. Resulting in the filter we can see in Figure 19.

2.15 Sharp Filter

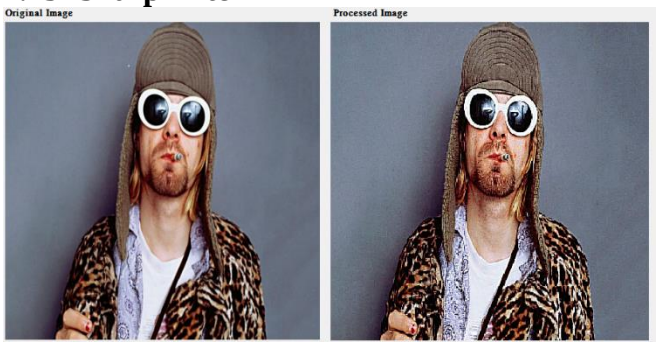

Figure 20. Sharp Filter

Sharp filter was created by the use of single experimental kernel. As we did with some of the other filters, we used *cv2.filter2D* and kernel of [[-1, 1, -1],[1, 1, 1],[-1, 1, -1]] . It makes pixels look more distinct. Thus we named it sharp filter.

2.16 Trippy Filter

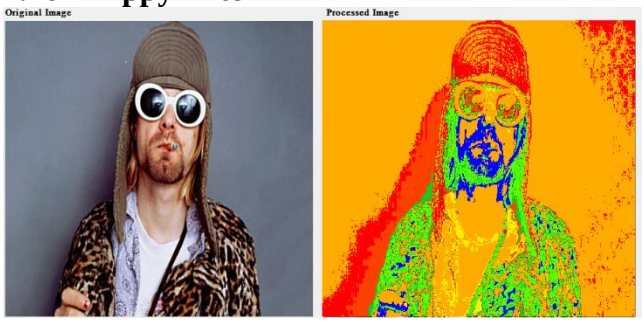

Figure 21. Trippy Filter

Trippy filter was create by using thresholding on the pixels of the image, followed by use of rainbow color map of cv2. In order to use thresholding we converted the image from BGR to HSV. We could've converted it to grayscale also but we wanted for it to be colorful and in HSV, it is easier to represent a color than BGR color space. After that with the use of threshold function of cv2, we set the thresholds to be 120 and 230. Adding the color map, our filter was complete.

2.17 Shaky Lemon Filter

Figure 22. Shaky Lemon Filter

When applied Shaky lemon filter on image, changed the color values of the pixels so that the yellow colors are dominant and ensuring the saturation of the picture. Firstly we use saturation function for to get a saturated picture and we highlighted the yellow color. This is how we use these equations for this change.

```
red, green, blue = pixels[i, j]
pixels[i, j]=min(255, int((green - blue) ** 2 / 128) + red), \
             min(255, int((red - green) ** 2 / 128) + blue), \
             min(255, int((blue - red) ** 2 / 128))
```

then, applied Image.point() method with the functions as given below.

```
image = image.point(lambda i: int(i ** 2 / 255))
```

The image, that is applied the Cobain Filter, is as seen in Figure 22.

2.18 Shape of Blue Filter

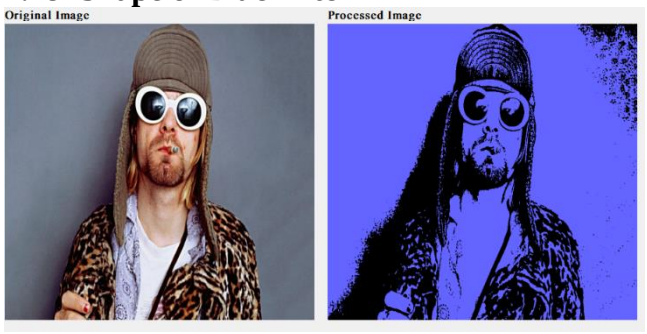

Figure 23. Shape of Blue Filter

Shape of Blue Filter is a combination of some methods. First of all with below method we take the images black and dominant parts;

```
r, g, b = pixels[x, y]
        if total > separator:
            image.putpixel((x, y), (255, 255, 255))
            image.putpixel((x, y), (0, 0, 0))
```

and than, for getting lightskyblue color of image we applied Image.point() method with the functions as given below.

```
image.point(lambda i: i ^ 0x8B if i < 135 else i)
```

finally, with below code we assign the blue color for complete picture.

```
pixels[i, j] = min(100, int(green * blue / 255)), \
                    min(100, int(blue * red / 255)), \
                    min(255, int(red * green / 255))
```

this filter represent shadows and blue color harmony. So we get shadow firstly then apply the blue color with given code.

The image, that is applied the Shape of Blue Filter, is as seen in Figure 23.

2.19 Shiny Wan Filter

Figure 24. Shiny Wan Filter

In the Shiny Wan filter, first of all the shades of black and white in the picture were first obtained, then the Black and white points were remapped with pale and bright silver colors according to their type. The amount of hue has been adjusted to achieve a subtle mix of original and tonal colors, or a completely bright and pale blend image.

r, g, b = pixels[x, y]

red = int(r * 0.796 + g * 0.769 + b * 0.189)

green = int(r * 0.349 + g * 0.686 + b * 0.500)

blue = int(r * 0.272 + g * 0.563 + b * 0.566)

The important thing here is that the red, green and blue tones come together properly to give the picture a bright pallor. For this, values are assigned to each pixel containing primary color as in the code example given above.

The image, that is applied the Shiny Wan Filter, is as seen in Figure 24.

2.20 Burgundy Shades Filter

Figure 25. Burgundy Shades Filter

It is aimed to create shades of burgundy and blue in the picture given with this filter. Here, if there are white colors in the picture given, a closed shade of blue is added to each pixel in the white parts. If white color is not available, it is provided to assign a color to the pixels in the rest of the picture so that burgundy colors dominate.

R = r * 0.125 + g * 0.102 + b * 0.135

G = r * 0.256 + g * 0.106 + b* 0.96

B = r * 0.565 + g * 0.300 + b * 0.206

To achieve this, the color model given above was applied to each pixel of the picture. Then, the color assignment given below was applied to remove the green color to reveal a shade of red and blue, burgundy, and to apply the blue color to the sharper edges.

R = 102 if R > 255 else B

G = 102 if G > 255 else R

B = 102 if B > 255 else R

The image, that is applied the Burgundy Shades Filter, is as seen in Figure 25.

2.21 Da Vinci Filter

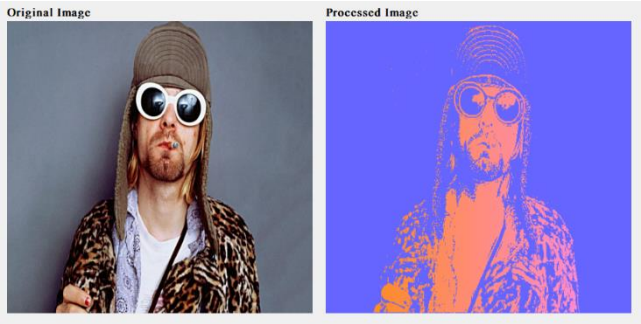

Figure 26. Da Vinci Filter

Leonardo Da Vinci frequently used the harmony of blue and orange in creating his works. This is quite evident in many of his famous works. While creating this filter, we wanted to use this feature, that is, the harmony of blue and orange. To create this harmony, orange color was assigned to the dark and edged parts of the picture and blue color was assigned to the remaining parts.

for i in range(width):

if i < width - 1 and (abs(pixels[j * width + i][0] - pixels[j * width + i + 1][0]) + abs(pixels[j * width + i][1] - pixels[j * width + i + 1][1]) + abs(pixels[j * width + i][2] - pixels[j * width + i + 1][2])) / 3 > 35:

row.append((255, 140, i // ((width + 255) // 255)))

To achieve this, firstly two different array were created. One of them is to contain the original form of the picture, the other is the array containing the edges in the picture to assign orange colors to the picture. Part of the method used to separate the closed colors from the light colors of the picture and assign them orange color is given above.

The image, that is applied the Da Vinci Filter, is as seen in Figure 26.

2.22 Dither Filter

Figure 27. Dither Filter

The dither filter is intended to create a flicker in the picture. In order to do this, using the height and width values of the picture, a navigation is provided on each pixel. Within the loop, pixels were read by skipping two by two according to the width and height values. First, the getpixel () method was used to obtain the pixels with a special method. This method is given below.

def get_pixel(self, image, i, j): (where i and j values are the pixel values from the picture.)

width, height = image.size

if i > width or j > height:

```
        return None
    pixel = image.getpixel((i, j))
    return pixel
```

The values from the getpixel () function are assigned to rgb values after a series of mathematical operations. As given below.

red = (p1[0] + p2[0] + p3[0] + p4[0]) / 4

green = (p1[1] + p2[1] + p3[1] + p4[1]) / 4

blue = (p1[2] + p2[2] + p3[2] + p4[2]) / 4

Then the saturation value was calculated using a function. Previously processed pixel values for the saturation value were re-processed, thereby providing image saturation. Finally, the values from the function were assigned as new pixels.

The image, that is applied the Dither Filter, is as seen in Figure 27.

3. RESULTS

The habit of using social media platforms that has become widespread today, has created new needs. People want to share their images by making them better and enchanced, not in the form they took from the camera. The Kakuna Image Editor allows the users to edit and manipulate the images. In this section, images obtained as a result of the operations of the Kakuna's components will be included.

The Kakuna Image Editor has basic operations as rotating, flipping and mirroring.

The image, that is rotated at an angle of 34 degrees, is as seen in Figure 26.

Figure 26. Rotated Image

After the rotating process, we applied mirroring operation on the image, that is given in Figure 27.

Figure 27. Mirrored Image

Then, we applied flipping on this image. The result is given in Figure 27.

Figure 28. Flipped Image

After the flipping operation, we cropped the image. When we cretate cropping operation, we used Qcrop library. User can select an area to crop with mouse. The cropped image is as seen in Figure 29.

Figure 29. Cropped Image

This is the basic operations in Kakuna Image Editor. The next process is carried out based on the last changes of the images. In rotating, mirroring and flipping operations, QTransform class is used. QTransform specifies 2D transformations of a coordinate system. A transformation specifies how to translate, scale, rotate.

Users can adjust the brightness and contrast of the image. User can adjust the image's luminosity level with brightness operation and the relative levels of dark and light regions on the image with contrast operation. The image, that is applied brightness operation when the slider value equals 20, is seen as Figure 30. The image that is applied contrast operation when the slider value equals 20, is seen as Figure 31.

Figure 30. Image After Brightness Operation

Figure 31. Image After Contrast Operation

In the page of artistic effects, we have frequently used 9 filters. These are Sepia, Cartoon, Contrast, Negative, Black, Gray, HandDraw, Inverse. These filters applied to the image one by one and the result images is given below, respectively.

Figure 32. Sepia Filter

Figure 33. Cartoon Filter

Figure 34. Contrast Filter

Figure 35. Negative Filter

Figure 36. Black Filter

Figure 37. Gray Filter

Figure 38. HandDraw Filter

The images created as a result of the Kakuna's creative filters will not be reshown in this section as they have been given before.

4. CONCLUSION

In this paper, we examined the Kakuna Image Editor, what it does and how it does. The aim of the project is to give different and brand new looks to the images according to the user's instructions

Both the filters frequently used and the filters, that created by the Kakuna developer team, were described in details.

One of the features that makes Kakuna different from other image editing tools is its creative filters, and the other is its' incremental advancement method. Incremental advancement method allows us to apply another filter on filtered image. Thus, more filter possibilities can be provided to the user than the filters that appear in the interface.

During this project, concepts about digital images, colors and filters were well understood. The part where we had the most problems, making the GUI components working and displaying the changed images on the interface. However, with these difficulties we have experienced, we have gained enough knowledge to develop interactive programs with the users with PyQt5.

5. REFERENCES

[1] Aslam,S.(06.01.2021). Instagram by the Numbers: Stats, Demographics & Fun Fact. Omnicore. https://www.omnicoreagency.com/instagram-statistics/

[2] Petterson, S.(20.10.2017). Statistics: How filters are used by Instagram's most successful users. Medium.

https://medium.com/@stpe/statistics-how-filters-are-used-by-instagrams-most-successful-users-d44935f87fa9

[3] V. Siahaan, R.H. Sianipar (2019, September 7). Learning PyQt5: A Step by Step Tutorial to Develop MySQL-Based Applications (pp. 1). Sparta Publishing.

[4] https://doc.qt.io/qt-5/qtdesigner-manual.html

[5] M.F. Khan, E. Khan, Z.A. Abbasi (2015, December). Image contrast enchancement using normalized histogram equalization. 126,24. 1. https://doi.org/10.1016/j.ijleo.2015.09.161.