OpenCV Image Filter: Simple application applying OpenCV

By

John Russell A. Caragay

Karen C. Escabarte

B.S. Computer Engineering

Technological Institute of the Philippines

About

An image filter is a method or procedure that modifies the colors, shade, hue, saturation, texture, and other aspects of a picture. Image filters can be applied to digital or analog pictures. Using filters, you can change the way an image looks for many reasons, including creative expression, aesthetics, and business. Image filters are used extensively on social media platforms in this generation. One of the best examples of Facebook, Snapchat, and Instagram providing a wide variety of filters to choose from is that people use various filters on their photographs in order to achieve the visual effects that they want.

OpenCV is a Python library that is available for free and may be used for various computational projects. It contains a large number of features and techniques, which are all adaptable to a broad range of applications. In order to achieve the filters and produce the effects, we are going to employ various methods of image processing.

These are the following filtering techniques that are used for image processing with the help of the application of opencv.

Detail Enhancement (HDR)

High-dynamic-range imaging, also known as HDR, is a technique which is used in image processing to generate a greater dynamic range of luminance. It enhances the amount of detail present in an image, the HDR effect can also use cv2.detailEnhance() function to make it easy.

Pencil Sketch

An additional example of image filtering is the pencil sketch effect. In fact, there is a built-in function that may be used to make its implementation quite easy and simple.

Grayscale

Images can be provided a black-and-white appearance by applying a filter known as greyscale. The removal of colored elements from the image is the primary focus in this filter. In order to convert the picture to grayscale, we will be implementing the cv2.cvtColor() function.

Brightness Adjustment (Brightness and Darkness tone)

This is one of the editing features of an image filtering. Sometimes we came across filters that significantly brighten the image, but at other times, we discover filters that decrease the brightness. These are the outcomes of applying filters that adjust the level of brightness. In order to accomplish this goal, we are going to make use of the cv2.convertScaleAbs() function. It is possible to adjust the beta value in order to get the desired outcomes. Because the function has been established at this point, the beta value will produce the expected outcomes. A value with a positive sign indicates an image with a brighter tone, whereas a value with a negative sign implies an image with a darker tone. The maximum Beta value is 255, where the minimum is 0. Then the value lower than 0 indicates a darker tone.

Invert Filter

Invert Filter is really simple to implement in python by the use of cv2.bitwise_not() function. This filter focuses on inverting the values of the pixels. This may be accomplished by subtracting the pixel values by 255.

Procedure and Output

Importing Necessary Libraries

```
In [1]: from collections import Counter
        from sklearn.cluster import KMeans
        from matplotlib import colors
        import matplotlib.pyplot as plt
        import numpy as np
        import cv2
```

Figure 1.1 Import Libraries

In this figure, we import the necessary libraries, which will help to run the program. These are some of the built-in libraries and modules that you can see in the figure above: counter, KMeans, colors, mathplotllib.pyplot, numpy, and cv2. The Python Counter serves as a container that stores and holds the count of each component. Next is the Kmeans, in which OpenCV is used in combination with the unsupervised machine learning process in order to separate the various components of an image. Meanwhile, matplotlib and matplot.lib are for the visualization of images. Then the numpy will be able to obtain and modify the pixel values by operating on the ndarray. Last is the implementation of cv2/OpenCV. It is a Python library for computer vision, such as videos, images, and more.

```
In [2]: #Importing HTML and CSS to Make the Output Align in Center.
        from IPython.core.display import HTML as Center

        Center(""" <style>
        .output_png {
            display: table-cell;
            text-align: center;
            vertical-align: middle;
        }
        </style> """)
Out[2]:
```

Figure 1.2 Centering the output

This figure illustrates the alignment of an output image. To ensure that all photos are centered, we import HMTL as the center. The figure shows the inline CSS code that you can also modify.

Importing the Testing Image

```
In [3]: test_img = cv2.imread('lenna.png')
        plt.imshow(test_img)

Out[3]: <matplotlib.image.AxesImage at 0x210a9324640>
```

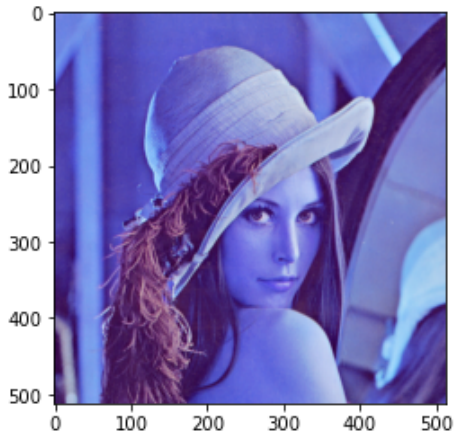


Figure 1.3 Importing the testing Image

Based on the above demonstration, the testing images with the filename lenna.png are loaded into the variable test_img. In order to display the image, the plt.imshow() function was implemented.

```
In [4]: #Since the default color in opencv is BGR, we changed the color to RGB.
        image = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
        plt.imshow(image)

Out[4]: <matplotlib.image.AxesImage at 0x210a94264c0>
```

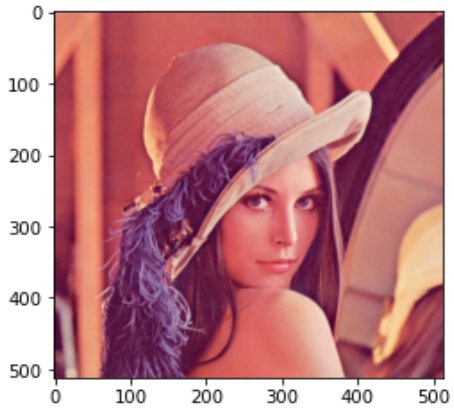


Figure 1.4 Changing testing Image into RGB

Since the default color in opencv is BGR, we have to modify it by using the cv2.COLOR BGR2RGB function. This cell in the program makes it possible for the image to return to its original color. This will help maintain the color of an image as close to its original form.

GENERATING COLOR PALETTE

```
In [5]: #This function is used to convert the rgb color to hexadecimal.
        def rgb_to_hex(rgb_color):
            hex_color = "#"
            for i in rgb_color:
                i = int(i)
                hex_color += ("{:02x}".format(i))
            return hex_color
```

Figure 1.5 RGB to Hex Function

The default output in generating the color palette are different values of red, green, and blue which are quite hard to understand. With hex value, we can just copy the hex code and get the exact color in the picture. Therefore in this function, it will convert the rgb values that will be generated to hex value

```
In [6]: #This function is used to reshape and resize the image for color processing.
        def prep_image(raw_img):
            modified_img = cv2.resize(raw_img, (900, 600), interpolation = cv2.INTER_AREA)
            modified_img = modified_img.reshape(modified_img.shape[0]*modified_img.shape[1], 3)
            return modified_img

In [7]: #Using K-Means to identify the top colors of the image.
        def color_analysis(img):
            #The n_clusters will generate the number of top colors. It can be changed to any desired number.
            clf = KMeans(n_clusters = 5)
            color_labels = clf.fit_predict(img)
            center_colors = clf.cluster_centers_
            counts = Counter(color_labels)
            ordered_colors = [center_colors[i] for i in counts.keys()]
            hex_colors = [rgb_to_hex(ordered_colors[i]) for i in counts.keys()]
            plt.figure(figsize = (13, 5))
            plt.title("Color Palette")
            return plt.pie(counts.values(), labels = hex_colors, colors = hex_colors)
```

Figure 1.6 Image Preparation and Color Analysis

This figure shows the functions that will prepare the image and will identify the color by using KMeans. In the prep_image function, we resize and reshape the image to make our model work properly. Resizing is optional, however, reshaping the image is required to define the clusters. The color_analysis function on the other hand is the main function for analyzing the colors. The first thing that we did is to specify the clusters or the number of top colors that we will generate. Then the counter function will generate the container of elements as dictionary keys In simple terms, the colors. Lastly, we converted the rgb colors to hex values using the rgb_hex function, and plotted it using pie.

```
In [8]: #This function is used to call and create the pie chart.
        def color_palette (img):
            modified_image = prep_image(img)
            color = color_analysis(modified_image)
            return color
```

```
In [9]: #This code will call the color_palette function and will send the image to get the pie chart of
        #top colors.
        color_palette(image)
        plt.show()
```

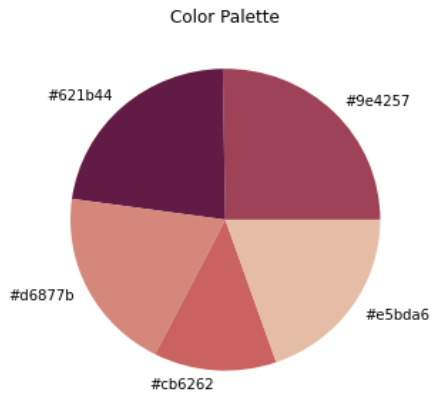


Figure 1.7 Color Palette Output

After creating the important functions, the last function will now be created so that we can just call the main function if we want to generate the color palette of other images. The color_palette function will first call the prep_image function for the preparation of the image and then the color_analysis function for generating the pie chart. In cell 9, we can see that we just only need to call the function.

GRAYSCALE FILTER

```
In [10]: #This function is used to convert the original image to grayscale.
         def grayscale (img):
             gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
             return cv2.cvtColor(gray, cv2.COLOR_BGR2RGB)
```

Figure 1.8 Grayscale Filter

This figure displays a grayscale filter. Images can be given a black-and-white appearance by using this technique. This filter is primarily concerned with removing colored components from a picture. The cv2.cvtColor() method will be implemented in order to convert the image to grayscale. In terms of code, we used the function name grayscale with one parameter. Inside the function, you will be able to see the gray variable container where the image processing happened. The cv2.color_BGR2GRAY was used in this line to implement OpenCV. Then the return will be converted to RGB. We used this because the program will first read a BGR image color.

```
In [11]: #Calling the grayscale function by sending the original image.
         grayscale_test = grayscale(image)
         plt.imshow(grayscale_test)

Out[11]: <matplotlib.image.AxesImage at 0x2f454fe8c40>
```

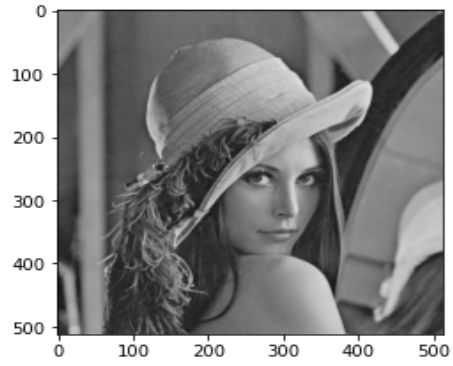


Figure 1.9 Grayscale Filter Output

In this figure, the grayscale function is called. In calling this function, the image that needs to be filtered should be added in the parameter. Then, using the plt.imshow to plot the image using matplotlib.

DETAIL ENHANCEMENT

SIGMA_S = SMOOTH_SCALE (values: 0-200)

SIGMA_R = EDGE_SCALE (values: 0-1)

```
In [12]: #This function is used to enhance the original image.
         def enhance (img,smooth_scale,edge_scale):
             enhance_img = cv2.detailEnhance(img, sigma_s = edge_scale, sigma_r = edge_scale)
             return enhance_img

In [13]: #Calling the enhance function, and specifying the smoothness,and the desired edge scale for the image.
         enhance_test1 = enhance(image,10,0.12)
         plt.imshow(enhance_test1)

Out[13]: <matplotlib.image.AxesImage at 0x210a9531cd0>
```

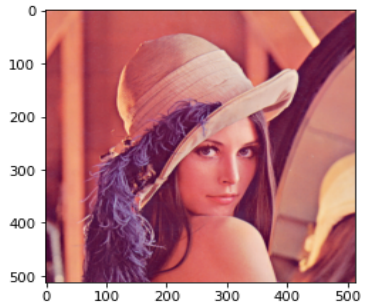


Figure 1.10 Detail Enhancement

In enhancing the details of an image using opencv, the function cv2.detailEnhance() is needed. It has three parameters namely: img source, sigma_s value, and sigma_r value. Img source is the name of the image, then sigma_s is on how smooth you want the image that scales from 0 to 200, and sigma_r for

the edge scale that has 0 to 1 value. As we can see in the figure, we created a function that has three parameters so that we can specify the values without using the cv2.detailEnhance repetitively. Cell 13, is an example on how we iteratively call the enhance function.

```
In [14]: #Using different smoothness and edge scale.
         enhance_test2 = enhance(image,200,1)
         plt.imshow(enhance_test2)

Out[14]: <matplotlib.figure.AxesImage at 0x210a9456760>
```

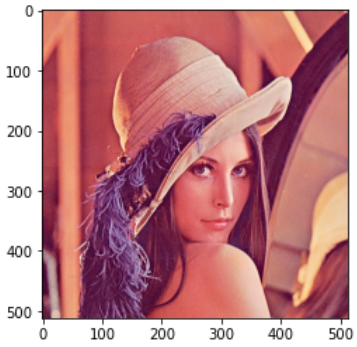


Figure 1.11 Detail Enhancement Test 2

In this figure, maximum values for the smooth value and edge value are used. It is visible that the image is very smooth, and the edges are not much seen.

PENCILSKETCH FILTER

SIGMA_S = SMOOTH_SCALE (values: 0-200)

SIGMA_R = EDGE_SCALE (values: 0-1)

SHADE_FACTOR = SHADE (values:0-0.1)

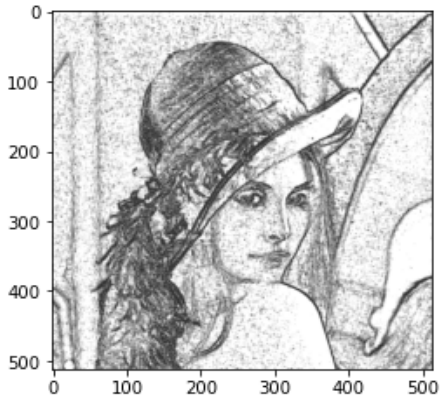
```
In [15]: #Creating two separate functions for each option.

         #Function that returns gray pencilsketch.
         def pen_sketch_gray (img,smooth_scale,edge_scale,shade):
             pen_gray, pen_color =  cv2.pencilSketch(img, sigma_s=smooth_scale, sigma_r=edge_scale, shade_factor=shade)
             return cv2.cvtColor(pen_gray, cv2.COLOR_BGR2RGB)

         #Function that returns colored pencilsketch.
         def pen_sketch_colored (img,smooth_scale,edge_scale,shade):
             pen_gray, pen_color =  cv2.pencilSketch(img, sigma_s=smooth_scale, sigma_r=edge_scale, shade_factor=shade)
             return pen_color
```

Figure 1.12 Pencil Sketch Filter

In creating a pencil sketch filter, it is important to use the function cv2.pencilSketch (). It has three parameters namely: img source, sigma_s, sigma_r, and shade_factor. The first three parameters are also the same with the parameters in detail enhancement, they are for image source, smooth value, and edge

scale value. However, the shade_factor is used to specify how dark or light the shade of the image is. In the function created, it is also observable that it has two variables, the pen_gray, and pen_color. The first variable, pen_gray returns a grayscale pens sketch, which is why it is also needed to convert from BGR to RGB using the cv2.COLOR_BGR2RGB function. The second variable, pen_color on the other hand returns a color pen sketch. We can see the outputs of these two functions on figure 1.13.

```
In [16]: #Calling the function that will return uncolored pencil sketch.
         pensketch_test1 = pen_sketch_gray(image,60,0.07,0.1)
         plt.imshow(pensketch_test1)

Out[16]: <matplotlib.image.AxesImage at 0x2f4550cd220>
```




```
In [17]: #Testing the colored pencil sketch functions.
         pensketch_test2 = pen_sketch_colored(image,60,0.07,0.1)
         plt.imshow(pensketch_test2)

Out[17]: <matplotlib.image.AxesImage at 0x2f455129a90>
```

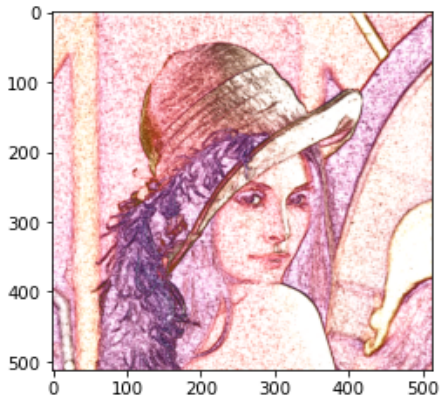


Figure 1.13 Pencil Sketch Filter Outputs

In this figure, we called the two functions to return a gray and colored pencil sketch. As we can see, it is also required to input the parameters for the filter.

BRIGHTNESS

BETA = BRIGHTNESS_VALUE (negative and positive values)

In [18]:
```
# brightness adjustment using cv2.convertScaleAbs() function.
# The beta Value will indicate if the image will be brighter or darker.
def bright(img, brightness_value):
    image_to_bright = cv2.convertScaleAbs(img, beta=brightness_value)
    return image_to_bright
```

In [19]:
```
# positive beta value (brighter image).
bright_test1 = bright(image, 60)
plt.imshow(bright_test1)
```

Out[19]: <matplotlib.image.AxesImage at 0x210a96d0940>

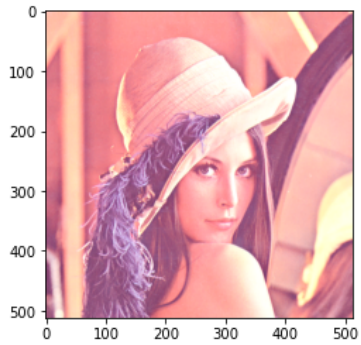


Figure 1.14 Brightness Adjustment

Image filtering application contains this technique as one of its editing features. In order for us to be successful in reaching our objective, we are going to make use of the cv2.convertScaleAbs() method. It is convenient to make adjustments to the beta value in order to get the outcomes which are desired. Due to the fact that the function has already been built at this stage, the beta value will yield the expected outcomes. A value that is preceded by a positive sign suggests that the associated image has a Brighter tone, whereas a value that is preceded by a negative sign suggests that the associated image has a darker tone. The least possible value for Beta is 0, while the greatest possible value is 255. A darker tone is indicated by a number that is less than 0. In terms of code in the cell, We used a function named bright with two parameters - for the image and desired brightness.

```
In [20]: # negative beta value (darker image).
         bright_test2 = bright(image, -60)
         plt.imshow(bright_test2)

Out[20]: <matplotlib.image.AxesImage at 0x210a972efd0>
```

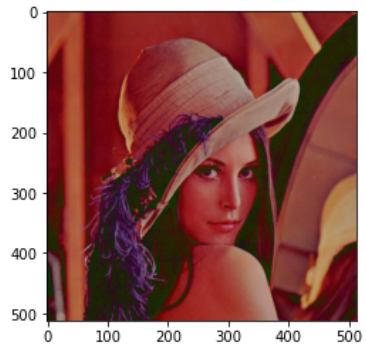


Figure 1.15 Brightness Adjustment Dark Output

In this figure, we used a negative brightness value (-60). As we can see the image is darker compared to the output shown in Figure 1.14.

INVERT FILTER

```
In [21]: # invert filter using cv2.bitwise_not().
         # This invert filter means inverting the pixel values by subtracting the
         # pixel by 255 or use the bitwsie function.
         def invert(img):
             convert_to_invertFilter = cv2.bitwise_not(img)
             return convert_to_invertFilter

In [22]: #Calling the function to invert the image color.
         invert_test = invert(image)
         plt.imshow(invert_test)

Out[22]: <matplotlib.image.AxesImage at 0x210a979b760>
```

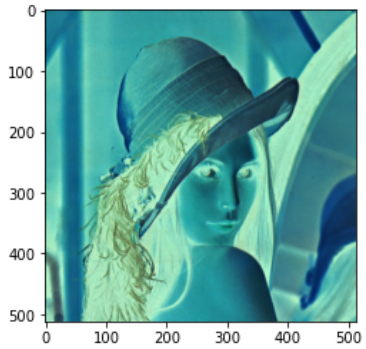


Figure 1.16 Invert Filter

Utilizing the cv2.bitwise not() method is something which is required to successfully create the invert filter in Python. This filter's primary purpose is to flip the values of the pixels in the image. This may

be done by dividing the pixel values by 255 and then removing those values. Here, we created a function named invert with one parameter for the image. Then Inside the function, you will be able to see the image processing from the invert filter by the use of the opencv. Lastly, it will return to the converted image.

SUMMARY OF TESTINGS:

```
In [23]: #Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Bri

#The plots created earlier are added inside the list
images = [image, grayscale_test, enhance_test1, pensketch_test1, pensketch_test2, bright_test1, bright_test2, invert_test]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

#Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```

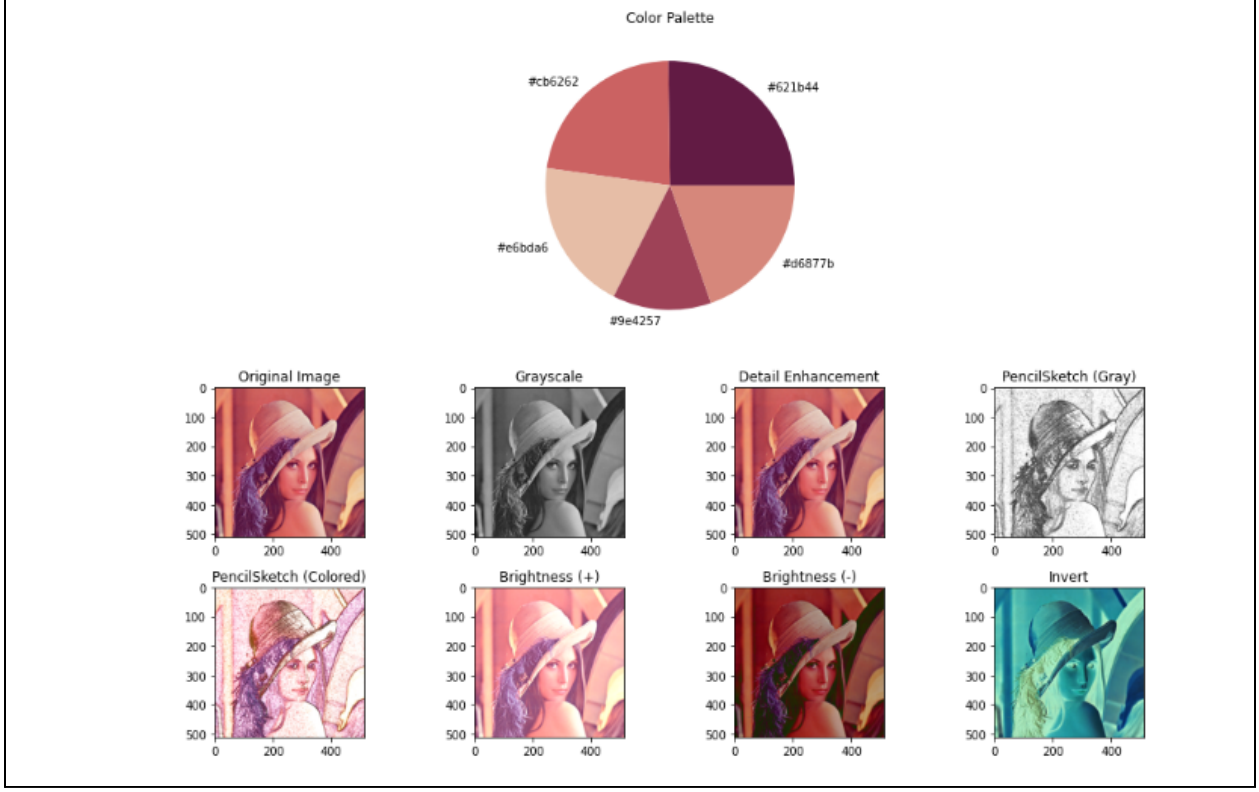


Figure 1.17 Summary of Testings

All of the image filters that we have used in the previous are shown in this figure. These filters include Grayscale, Detail Enhancement, PencilSketch(Gray), PencilSketch(Colored), Brightness(lighter tone), Brightness(darker tone), and Invert Filter. As can be seen, the color palette is also displayed within

the pie. The first item called in the code is the image's color palette, which displays the colors included in the testing image. The next line is meant for the titles of each of the eight plots contained within an array; this is useful for determining the label of each image. Next, we additionally construct a variable that will hold all of the filter images, and we name it as images. In addition, the figure size of the plot is specified using the values of 13 by 5.Overall, in order to access all the images included within the ndarray, we develop a for loop that displays them in a 2 row by 4 column format and shows them with the plt.show command ().

TESTING OTHER IMAGE

TEST 1

```
In [24]: #Importing the first image that will be tested
         new_img = cv2.imread('bird.jpeg')

         #changing the color from bgr to rgb
         image = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
         plt.imshow(image)

Out[24]: <matplotlib.image.AxesImage at 0x28bae760850>
```

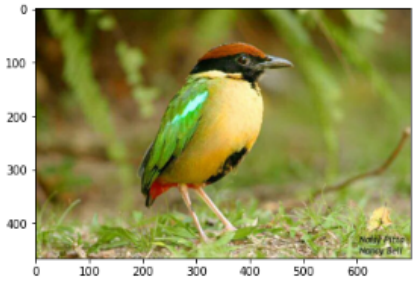


Figure 1.18 Test 1 image

Based on the figure, the new image with the filename bird.jpeg is loaded into the variable new_img. Since the default color that will be read is bgr, we need to convert the image into rgb by using the cv2.COLOR_BGR2RG function. This will help to maintain the original color of an image. In order to display the image, the plt.imshow() function was implemented.

In [25]:
```
#Calling the functions that created in the previous cells. Values of some parameters can be changed based on your own desires.
#Filters range can be changed here.
grayscale_img = grayscale(image)
enhance_img = enhance(image,10,0.12)
pensketch_gray = pen_sketch_gray(image,90,0.07,0.1)
pensketch_colored = pen_sketch_colored(image,60,0.07,0.1)
bright_posimg = bright(image, 60)
bright_negimg = bright(image, -60)
invert_img = invert(image)

#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Bri

#The plots created earlier are added inside the list
images = [image, grayscale_img, enhance_img, pensketch_gray, pensketch_colored, bright_posimg,bright_negimg, invert_img]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

    #Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```

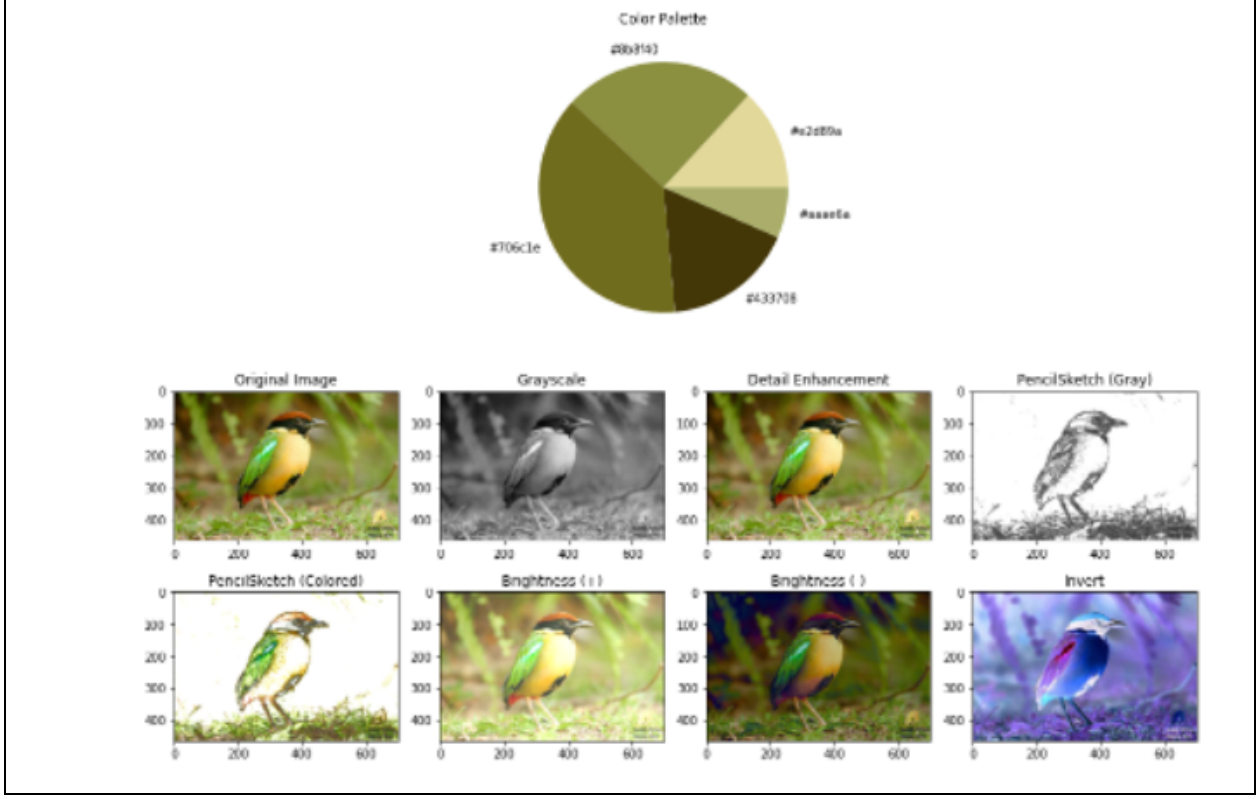


Figure 1.19 Test 1 image

The testing discussed in the previous discussion served as a basis for the implementation of a summary of a new image. We also built a variable that holds the various picture filtering techniques, and this can be accomplished by calling a variety of functions that we developed earlier on in the process. In the

output, the function for generating the color palette was called to display all color values that were contained inside the original image. After that, we proceed and generated a variable for the plot's image container as well as its labels. In addition, with the utilization of a loop, each image displays proportionately to the figure size. The differentiated results of all image filters will be displayed through output images.

TEST 2

```
In [26]: #Importing the second image that will be tested
         new_img = cv2.imread('wolf.jpg')

         #changing the color from bgr to rgb
         image = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
         plt.imshow(image)

Out[26]: <matplotlib.image.AxesImage at 0x28bae8e5b80>
```

Figure 1.20 Test 2 image summary

The new image with the filename wolf.jpg is imported into the variable new_img based on the figure. Since the default color that will be read is bgr, the cv2.COLOR BGR2RG function must be used to convert the image to rgb. This will enable them to maintain an image's natural color. For the image to be shown, the plt.imshow() function was implemented.

In [27]:
```
#Calling the functions that created in the previous cells. Values of some parameters can be changed based on your own desires.
#Filters range can be changed here.
grayscale_img = grayscale(image)
enhance_img = enhance(image,10,0.12)
pensketch_gray = pen_sketch_gray(image,90,0.07,0.1)
pensketch_colored = pen_sketch_colored(image,60,0.07,0.1)
bright_posimg = bright(image, 60)
bright_negimg = bright(image, -60)
invert_img = invert(image)

#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Bri

#The plots created earlier are added inside the list
images = [image, grayscale_img, enhance_img, pensketch_gray, pensketch_colored, bright_posimg,bright_negimg, invert_img]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

    #Used to automatically adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```

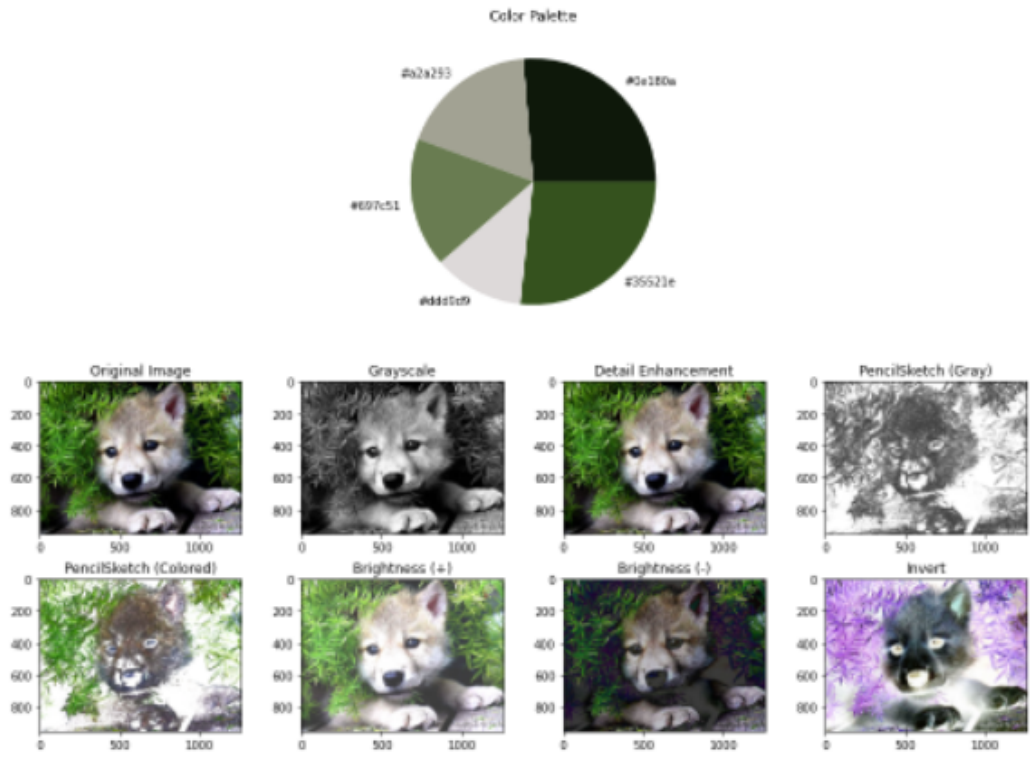


Figure 1.20 Test 2 image summary

The earlier testing helped implement a new image summary. We established a variable that holds the different photo filtering approaches by calling previous functions. In the output, the color palette function

displayed all original image color values. We next created variables for the plot's image container and labels. Each picture shows the figure size with a loop. Output images show all image filters' results.

TEST 3

```
In [28]: #Importing the first image that will be tested
         new_img = cv2.imread('zoro.jpg')

         #changing the color from bgr to rgb
         image = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
         plt.imshow(image)

Out[28]: <matplotlib.image.AxesImage at 0x28bb18ed220>
```

Figure 1.21 Test 3 image summary

Based on the figure, the new image with the filename zoro.jpg is loaded into the variable new_img. Since the default color that will be read is bgr, we need to convert the image into rgb by using the cv2.COLOR_BGR2RG function. This will help to maintain the original color of an image. In order to display the image, the plt.imshow() function was implemented.

In [29]:
```
#Calling the functions that created in the previous cells. Values of some parameters can be changed based on your own desires.
#Filters range can be changed here.
grayscale_img = grayscale(image)
enhance_img = enhance(image,10,0.12)
pensketch_gray = pen_sketch_gray(image,90,0.07,0.1)
pensketch_colored = pen_sketch_colored(image,60,0.07,0.1)
bright_posimg = bright(image, 60)
bright_negimg = bright(image, -60)
invert_img = invert(image)

#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Bri

#The plots created earlier are added inside the list
images = [image, grayscale_img, enhance_img, pensketch_gray, pensketch_colored, bright_posimg,bright_negimg, invert_img]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

    #Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```

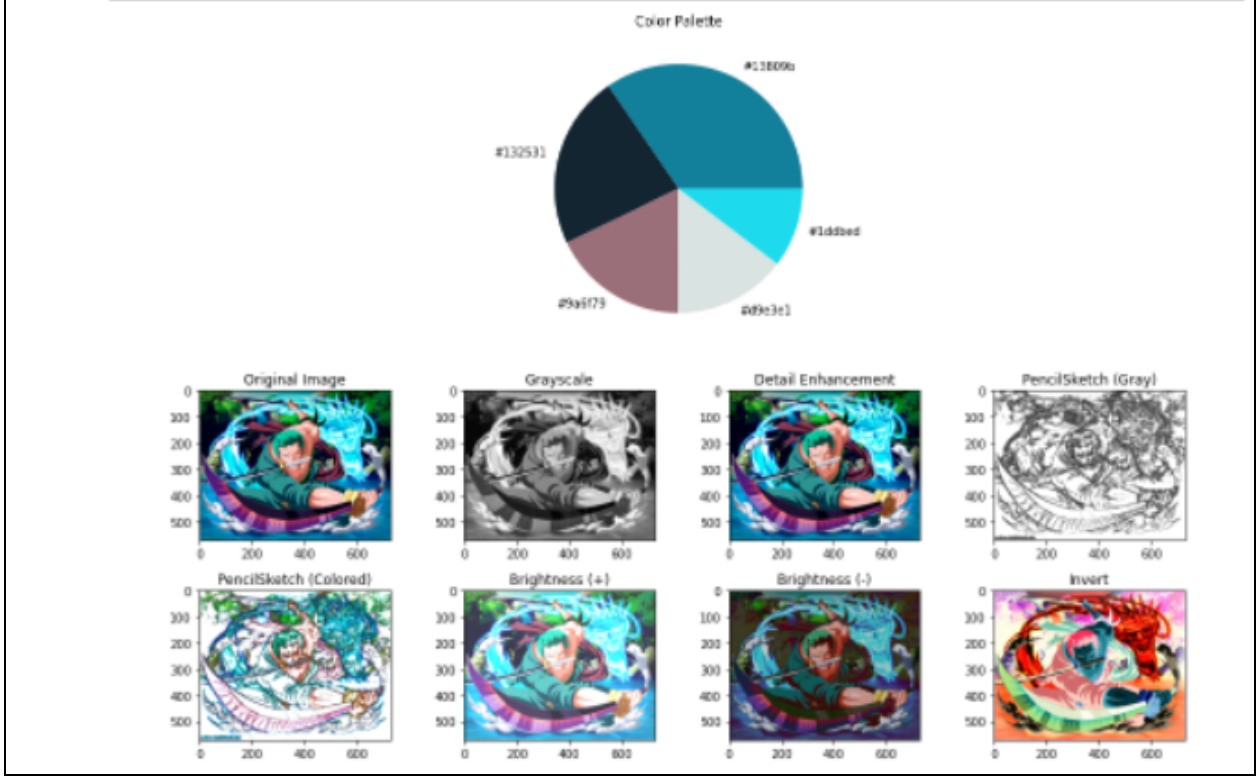


Figure 1.22 Test 3 image summary

The testing discussed in the previous discussion served as a basis for the implementation of a summary of a new image. We also built a variable that holds the various picture filtering techniques, and

this can be accomplished by calling a variety of functions that we developed earlier on in the process. In the output, the function for generating the color palette was called to display all color values that were contained inside the original image. After that, we proceed and generated a variable for the plot's image container as well as its labels. In addition, with the utilization of a loop, each image displays proportionately to the figure size. The differentiated results of all image filters will be displayed through output images.

Summary of Codes and Output

CARAGAY, JOHN RUSSELL A.
ESCABARTE, KAREN C.

Midterm Exam - Group 6

Importing Necessary Libraries

In [1]:
```
from collections import Counter
from sklearn.cluster import KMeans
from matplotlib import colors
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

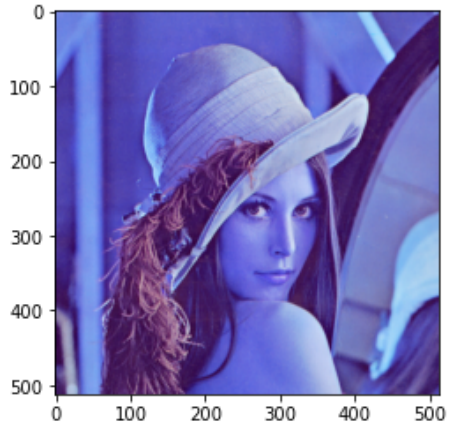
In [2]:
```
#Importing HTML and CSS to Make the Output Align in Center.
from IPython.core.display import HTML as Center

Center(""" <style>
.output_png {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
}
</style> """)
```

Out[2]:

Importing the Testing Image

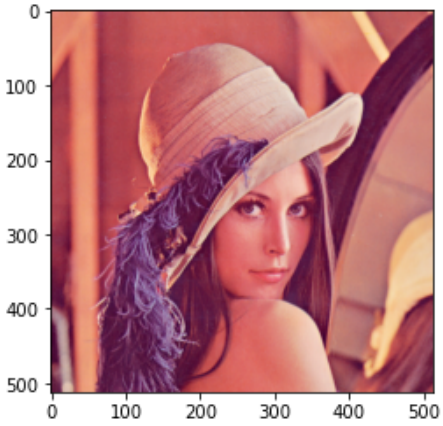
In [3]:
```
test_img = cv2.imread('lenna.png')
plt.imshow(test_img)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x2f454e45670>
```



In [4]:
```
#Since the default color in opencv is BGR, we changed the color to RGB.
image = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x2f454f46910>
```



GENERATING COLOR PALETTE

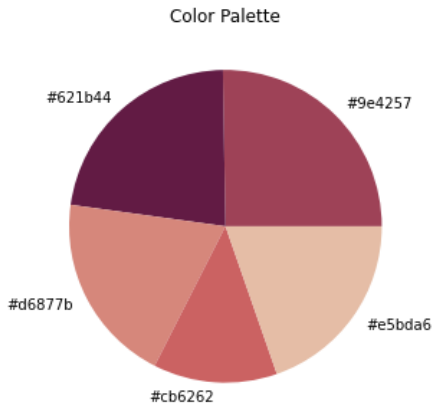
In [5]:
```
#This function is used to convert the rgb color to hexadecimal.
def rgb_to_hex(rgb_color):
    hex_color = "#"
    for i in rgb_color:
        i = int(i)
        hex_color += ("{:02x}".format(i))
    return hex_color
```

In [6]:
```
#This function is used to reshape and resize the image for color processing.
def prep_image(raw_img):
    modified_img = cv2.resize(raw_img, (900, 600), interpolation = cv2.INTER_AREA)
    modified_img = modified_img.reshape(modified_img.shape[0]*modified_img.shape[1], 3)
    return modified_img
```

In [7]:
```
#Using K-Means to identify the top colors of the image.
def color_analysis(img):
    #The n_clusters will generate the number of top colors. It can be changed to any desired number.
    clf = KMeans(n_clusters = 5)
    color_labels = clf.fit_predict(img)
    center_colors = clf.cluster_centers_
    counts = Counter(color_labels)
    ordered_colors = [center_colors[i] for i in counts.keys()]
    hex_colors = [rgb_to_hex(ordered_colors[i]) for i in counts.keys()]
    plt.figure(figsize = (13, 5))
    plt.title("Color Palette")
    return plt.pie(counts.values(), labels = hex_colors, colors = hex_colors)
```

In [8]:
```
#This function is used to call and create the pie chart.
def color_palette (img):
    modified_image = prep_image(img)
    color = color_analysis(modified_image)
    return color
```

In [9]:
```
#This code will call the color_palette function and will send the image to get the pie chart of
#top colors.
color_palette(image)
plt.show()
```

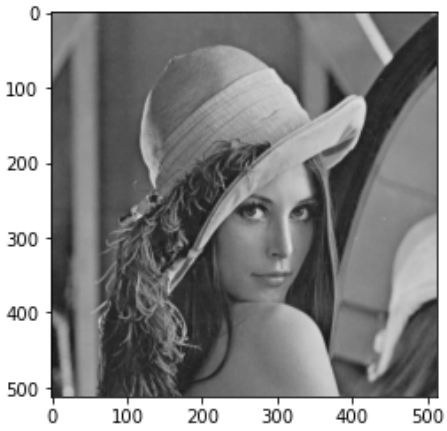


GRAYSCALE FILTER

In [10]:
```
#This function is used to convert the original image to grayscale.
def grayscale (img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return cv2.cvtColor(gray, cv2.COLOR_BGR2RGB)
```

In [11]:
```
#Calling the grayscale function by sending the original image.
grayscale_test = grayscale(image)
plt.imshow(grayscale_test)
```

Out[11]: <matplotlib.image.AxesImage at 0x2f454fe8c40>



DETAIL ENHANCEMENT

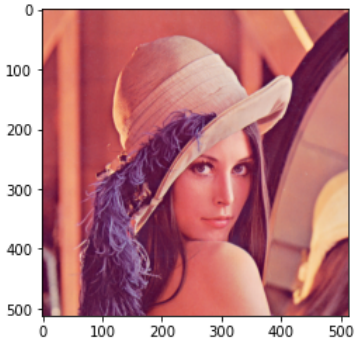
SIGMA_S = SMOOTH_SCALE (values: 0-200)

SIGMA_R = EDGE_SCALE (values: 0-1)

In [12]:
```
#This function is used to enhance the original image.
def enhance (img,smooth_scale,edge_scale):
    enhance_img = cv2.detailEnhance(img, sigma_s = edge_scale, sigma_r = edge_scale)
    return enhance_img
```

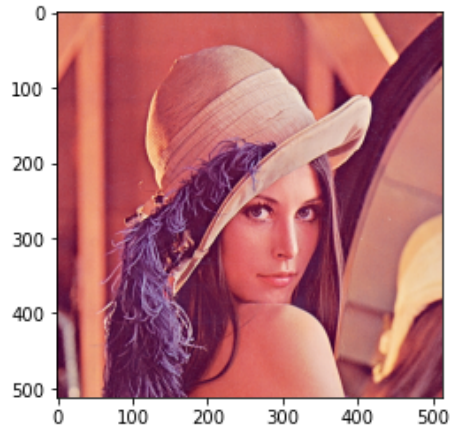
In [13]:
```
#Calling the enhance function, and specifying the smoothness,and the desired edge scale for the image.
enhance_test1 = enhance(image,10,0.12)
plt.imshow(enhance_test1)
```

Out[13]: <matplotlib.image.AxesImage at 0x2f455057640>



In [14]:
```
#Using different smoothness and edge scale.
enhance_test2 = enhance(image,200,1)
plt.imshow(enhance_test2)
```

Out[14]: <matplotlib.image.AxesImage at 0x2f454f57f40>



PENCILSKETCH FILTER

SIGMA_S = SMOOTH_SCALE (values: 0-200)

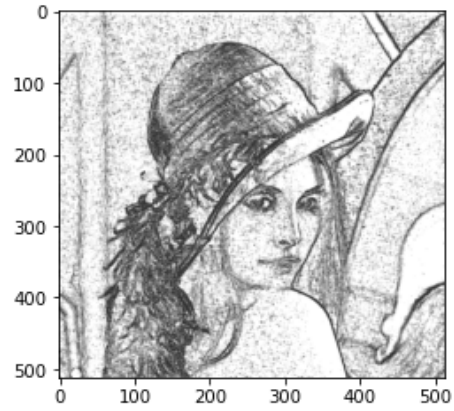
SIGMA_R = EDGE_SCALE (values: 0-1)

SHADE_FACTOR = SHADE (values:0-0.1)

In [15]:
```
#Creating two separate functions for each option.

#Function that returns gray pencilsketch.
def pen_sketch_gray (img,smooth_scale,edge_scale,shade):
    pen_gray, pen_color =  cv2.pencilSketch(img, sigma_s=smooth_scale, sigma_r=edge_scale, shade_factor=shade)
    return cv2.cvtColor(pen_gray, cv2.COLOR_BGR2RGB)

#Function that returns colored pencilsketch.
def pen_sketch_colored (img,smooth_scale,edge_scale,shade):
    pen_gray, pen_color =  cv2.pencilSketch(img, sigma_s=smooth_scale, sigma_r=edge_scale, shade_factor=shade)
    return pen_color
```

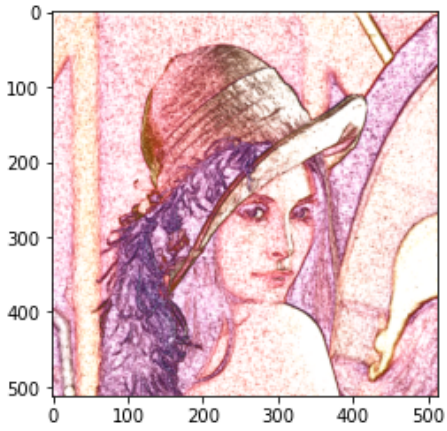
In [16]:
```
#Calling the function that will return uncolored pencil sketch.
pensketch_test1 = pen_sketch_gray(image,60,0.07,0.1)
plt.imshow(pensketch_test1)
```

Out[16]: <matplotlib.image.AxesImage at 0x2f4550cd220>



In [17]:
```
#Testing the colored pencil sketch functions.
pensketch_test2 = pen_sketch_colored(image,60,0.07,0.1)
plt.imshow(pensketch_test2)
```

Out[17]: <matplotlib.image.AxesImage at 0x2f455129a90>



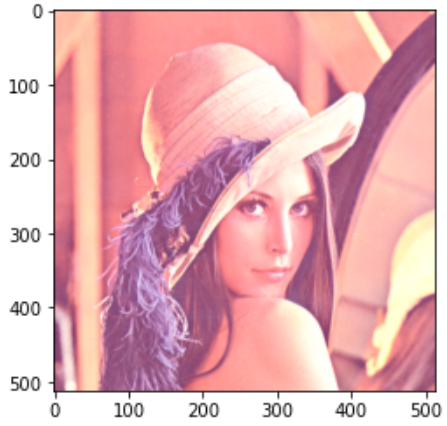
BRIGHTNESS

BETA = BRIGHTNESS_VALUE (negative and positive values)

In [18]:
```
# brightness adjustment using cv2.convertScaleAbs() function.
# The beta Value will indicate if the image will be brighter or darker.
def bright(img, brightness_value):
    image_to_bright = cv2.convertScaleAbs(img, beta=brightness_value)
    return image_to_bright
```

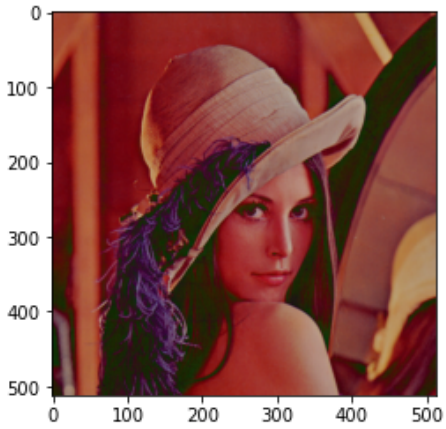
In [19]:
```
# positive beta value (brighter image).
bright_test1 = bright(image, 60)
plt.imshow(bright_test1)
```

Out[19]: <matplotlib.image.AxesImage at 0x2f4551f92e0>



In [20]:
```
# negative beta value (darker image).
bright_test2 = bright(image, -60)
plt.imshow(bright_test2)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x2f455257a00>
```

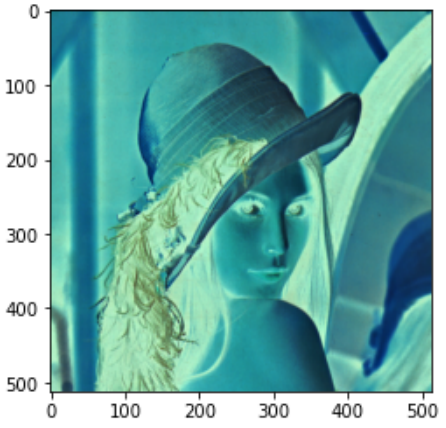


INVERT FILTER

In [21]:
```
# invert filter using cv2.bitwise_not().
# This invert filter means inverting the pixel values by subtracting the
# pixel by 255 or use the bitwsie function.
def invert(img):
    convert_to_invertFilter = cv2.bitwise_not(img)
    return convert_to_invertFilter
```

In [22]:
```
#Calling the function to invert the image color.
invert_test = invert(image)
plt.imshow(invert_test)
```

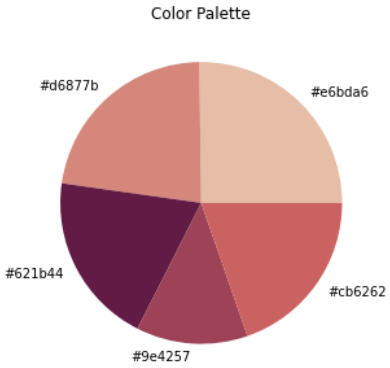
Out[22]:
```
<matplotlib.image.AxesImage at 0x2f4552b6fa0>
```

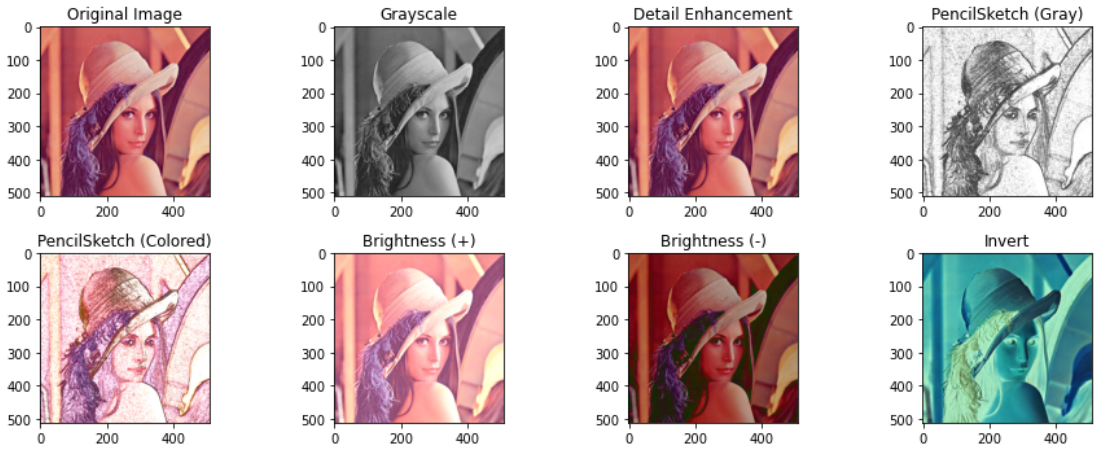


SUMMARY

SUMMARY OF TESTINGS:

In [23]:
```
#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Brig

#The plots created earlier are added inside the list
images = [image, grayscale_test, enhance_test1, pensketch_test1, pensketch_test2, bright_test1, bright_test2, invert_test]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

#Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```



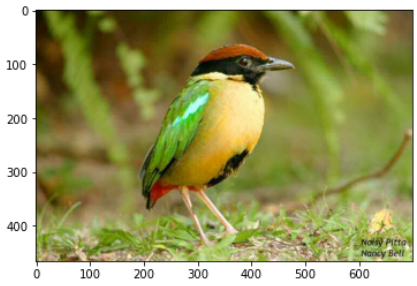


TESTING OTHER IMAGE

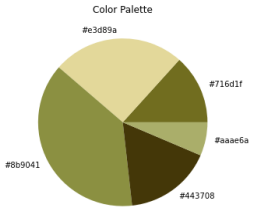
TEST 1

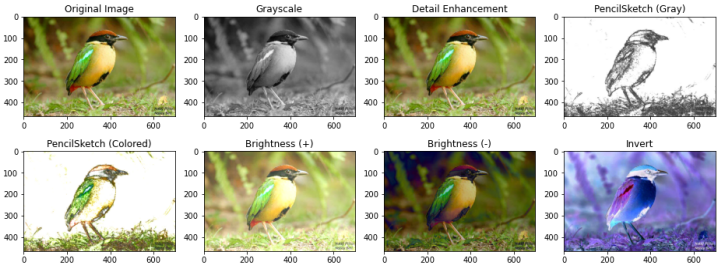
In [24]:
```
#Importing the first image that will be tested
new_img = cv2.imread('bird.jpeg')

#changing the color from bgr to rgb
image = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[24]: <matplotlib.image.AxesImage at 0x2f455526310>



In [25]:
```
#Calling the functions that created in the previous cells. Values of some parameters can be changed based on your own desires.
#Filters range can be changed here.
grayscale_img = grayscale(image)
enhance_img = enhance(image,10,0.12)
pensketch_gray = pen_sketch_gray(image,90,0.07,0.1)
pensketch_colored = pen_sketch_colored(image,60,0.07,0.1)
bright_posimg = bright(image, 60)
bright_negimg = bright(image, -60)
invert_img = invert(image)

#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Brig

#The plots created earlier are added inside the list
images = [image, grayscale_img, enhance_img, pensketch_gray, pensketch_colored, bright_posimg,bright_negimg, invert_img]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

#Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```



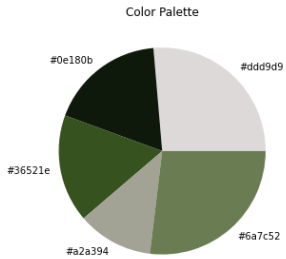


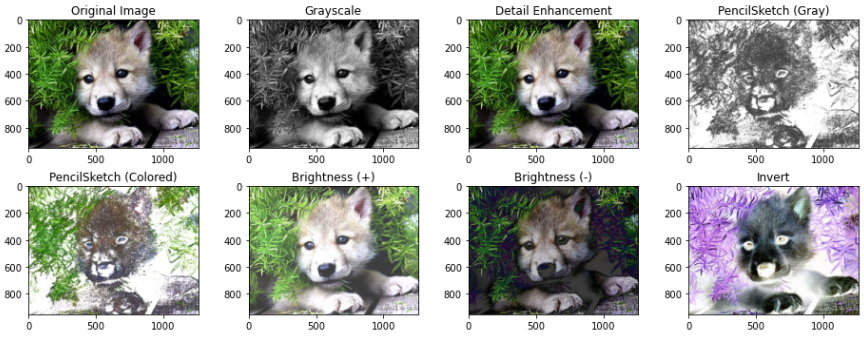
TEST 2

In [26]:
```
#Importing the second image that will be tested
new_img = cv2.imread('wolf.jpg')

#changing the color from bgr to rgb
image = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[26]: <matplotlib.image.AxesImage at 0x2f45558acd0>

In [27]:
```
#Calling the functions that created in the previous cells. Values of some parameters can be changed based on your own desires.
#Filters range can be changed here.
grayscale_img = grayscale(image)
enhance_img = enhance(image,10,0.12)
pensketch_gray = pen_sketch_gray(image,90,0.07,0.1)
pensketch_colored = pen_sketch_colored(image,60,0.07,0.1)
bright_posimg = bright(image, 60)
bright_negimg = bright(image, -60)
invert_img = invert(image)

#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Brig

#The plots created earlier are added inside the list
images = [image, grayscale_img, enhance_img, pensketch_gray, pensketch_colored, bright_posimg,bright_negimg, invert_img]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

    #Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```



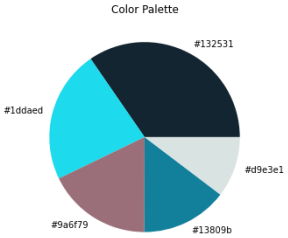


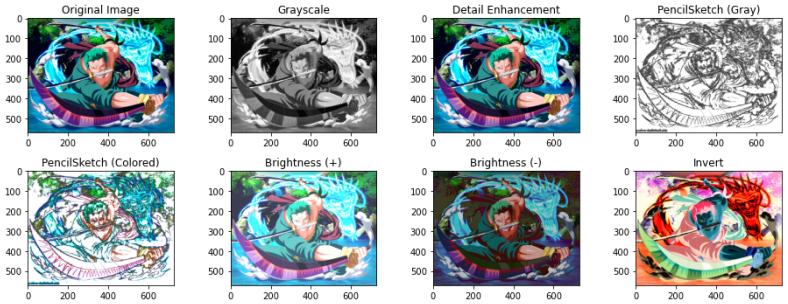
TEST 3

In [28]:
```
#Importing the first image that will be tested
new_img = cv2.imread('zoro.jpg')

#changing the color from bgr to rgb
image = cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[28]:
```
<matplotlib.image.AxesImage at 0x2f4585e2910>
```

In [29]:
```
#Calling the functions that created in the previous cells. Values of some parameters can be changed based on your own desires.
#Filters range can be changed here.
grayscale_img = grayscale(image)
enhance_img = enhance(image,10,0.12)
pensketch_gray = pen_sketch_gray(image,90,0.07,0.1)
pensketch_colored = pen_sketch_colored(image,60,0.07,0.1)
bright_posimg = bright(image, 60)
bright_negimg = bright(image, -60)
invert_img = invert(image)

#Calling the color pallette generator
color_palette(image)

#Titles for each plot
titles = ['Original Image',"Grayscale","Detail Enhancement","PencilSketch (Gray)","PencilSketch (Colored)","Brightness (+)","Brig

#The plots created earlier are added inside the list
images = [image, grayscale_img, enhance_img, pensketch_gray, pensketch_colored, bright_posimg,bright_negimg, invert_img]

#Specifying the figure size
plt.figure(figsize=(13,5))

#Loop for calling all the plots in the list
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])

    #Used to automaticallly adjust the subplots
plt.tight_layout()

#Showing all the plots
plt.show()
```





Reference

Majumder, P. (2021, July 14). *Creating Instagram like Filters With Opencv! Analytics Vidhya.* Retrieved May 26, 2022, from https://www.analyticsvidhya.com/blog/2021/07/an-interesting-opencv-application-creating-filters-like-instagram-and-picsart/