Image and Video Processing
Project 1: Spatial Filtering, Processing

Table of Contents

Dependencies

```
import cv2
import numpy as np
import random
from matplotlib import pyplot as plt
```

Colour Spaces

The first two tasks of this project required choosing two colourful images, a bright one and a pale one. The following two images were chosen:

```
BGRImageBirds = cv2.imread("images project1/birds.jpg")
BGRImageStone = cv2.imread("images project1/stone.jpg")
```

The first task asks to transform the images from RGB to HSV. The latter is an alternative representation of the RGB model, and it stands for Hue, Saturation, and Value (or brightness).
Unlike the RGB model, the HSV colour space is more similar to how humans perceive colours, in terms of their saturation and brightness levels, and the values are calculated in degrees.
As the images were read using the imread() function, they are stored in the BGR format. Then to convert from BGR to HSV we can use the "OpenCV-Python" library. Through this, the function cvtColor() is called, in order to convert the images from one colour space to another.

```
HSVImageBirds = cv2.cvtColor(BGRImageBirds, cv2.COLOR_BGR2HSV)
```

Figures 1 and 2 displays the results of the conversion to the HSV colour space.

Figure 1: HSV Birds

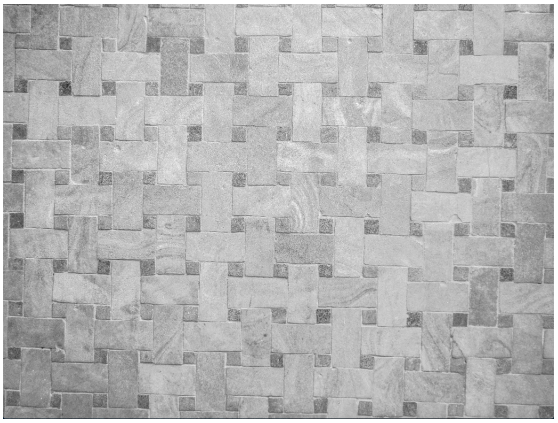
Figure 2: HSV Stones

Another color model that this report will look at is the HSI color model. This is closer to the way humans interpret colour and, moreover, it allows the separation between the color and the gray-scale information. The letter "I" stands for intensity, which refers to the gray levels, thus this models allows the intensity to be easily measurable and interpretable. The letters, H and S, on the other hand, stand for Hue (angle measure on the color wheel) and Saturation (measure of the colour's purity).
The formula to transform RGB images to an intensity "I" in the HSI space, is the average of the RGB values:

$$I = \frac{1}{3}(R + G + B)$$

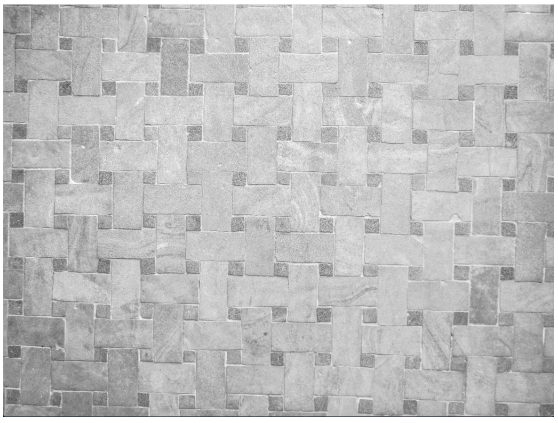
Before applying this formula the RGB values of the image have to be normalised to the range [0, 1].

```
BGRImageBirds = np.float32(BGRImageBirds)/255
RImage = BGRImageBirds[:,:,2]
GImage = BGRImageBirds[:,:,1]
BImage = BGRImageBirds[:,:,0]
HSIImageBirds = np.divide(RImage+GImage+BImage, 3)
```



As previously mentioned, in the HSV colour space, V stands for Value. To connect it to the RGB model, the value would be the maximum value between the red, blue and green colours.

```
HsvVBirds = np.maximum(np.maximum(RImage, GImage), BImage)
```



Pointwise Transforms, Histogram Equalisation

Moving on to the Pointwise Transforms and Histogram Equalisation, the following two images were used:

Using the *matplotlib.pyplot* library, the function hist() takes in the image, the number of bins and the range. Thus, the number of bins will be 256, for each intensity level, and will range from [0, 256]. In order for the image to be processed it has to be in the form of a 1-D array, instead of its original form, which is a 2-D array (matrix). This is possible by applying the numpy.ravel() method which will return a 1-D array containing all the elements of the input image.

```
def imageHistogram(img):
    histogram = plt.hist(img.ravel(), 256, [0.0, 256.0])
    plt.show()
```

To produce the photographic negative of a picture, the intensity levels of that image have to be reversed. This can be done by using an intensity transformation function.

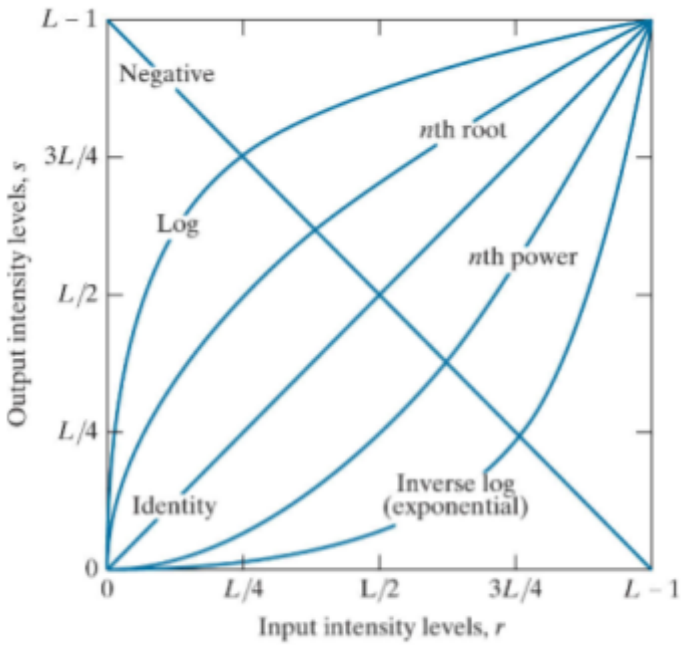


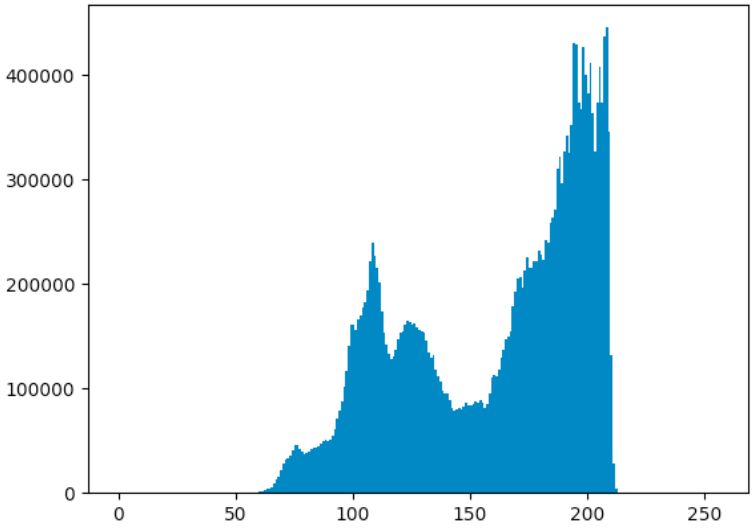
Figure 3: Basic intensity transformation functions

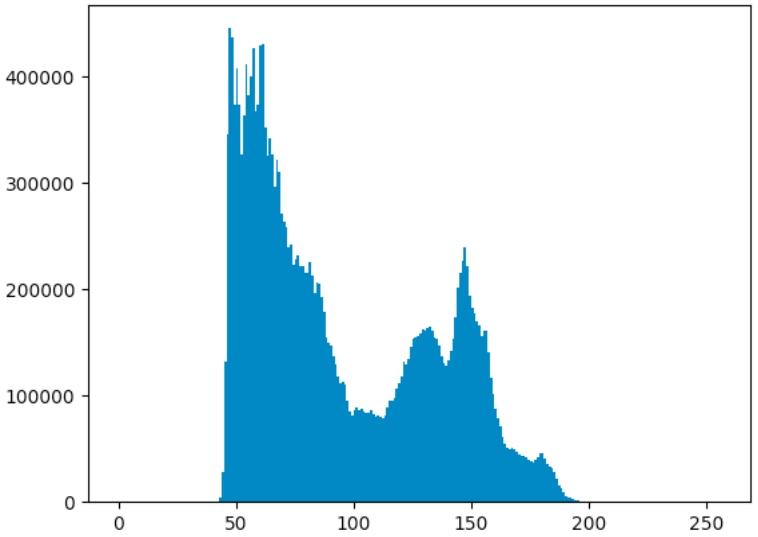
The intensity levels of the original image are in the range [0, L-1] where L is 256. Hence the negative transformation function is given by the following function, where r is the initial itensity level and s is the final intensity level :

$$s = L - 1 - r$$

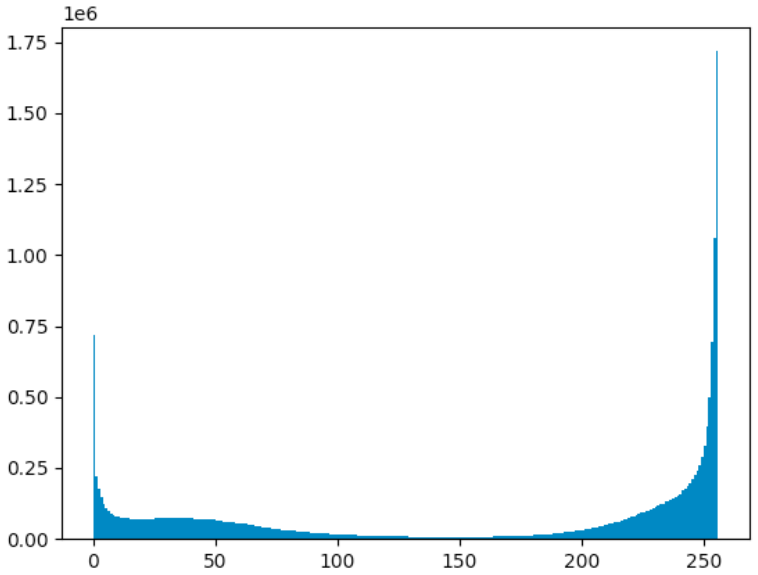
```
s = 255 - imageFog
```

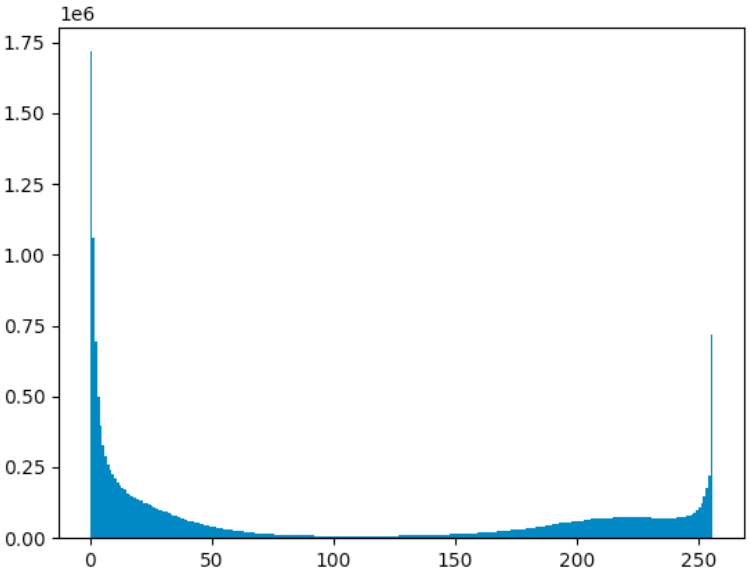
The figures below show the original image and its histogram, as well as the negative transform of the original image and its corresponding histogram.
The shape of the histograms are related to the appearance of the image. In the original image, the most prevalent feature is the fog, giving the image a lighter gray (almost white) look. For this reason, as it can be observed by its histogram, the most populated bins are towards the higher end of the scale (on the right). In fact, it is important to recall that as intensity levels approach 255, they approach the lighter end of the scale, while as they approach 0, they approach the darker end. For this reason, when the negative transformation is applied, and the image is considerably darker, the histogram is reversed, and the most populated bins are closer to 0.





The second image used for this task, is a high contrast image. In fact, the components of its histogram cover a wider range of the intensity scale, and only few bins are much more populated than others, making the distributions of the pixels almost uniform. Similarly to the example above, the most populated bins of the histogram corresponding to the original image are closer to the light end of the intensity scale. However, in this case the bins on the dark end of the scale are also very populated, considering the dark colors given by the shadows in the image. Also in this case the histogram of the negative transform of this image the same as the histogram of the original image, but reversed, with the most populated bins on the darker end of the scale.



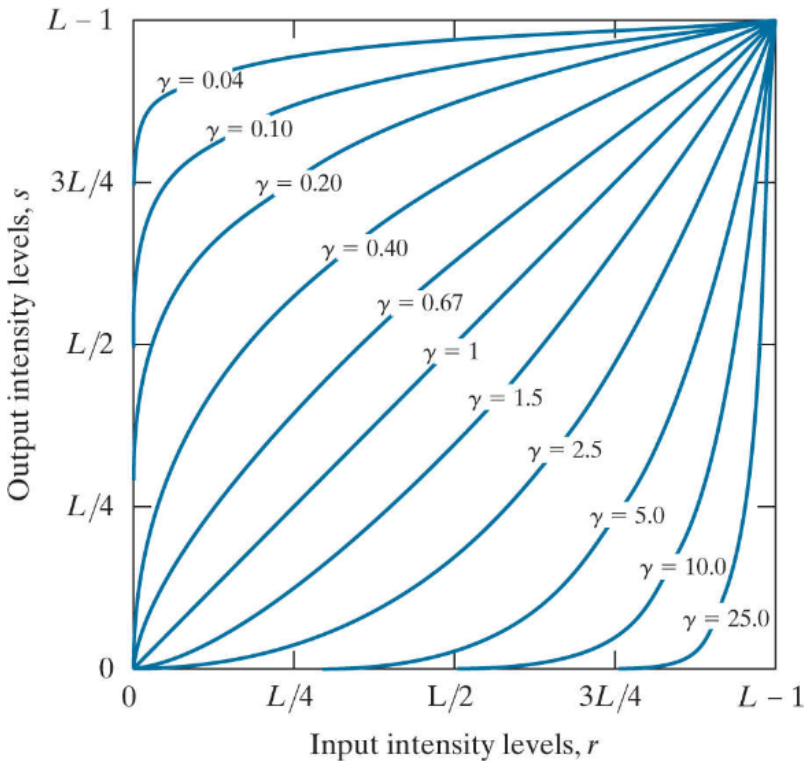


Another transformation is the *Power-Law (Gamma) transformation* and it has the following form:

$$s = cr^{\gamma}$$

Where c (in this case c = 1) and γ and positive constants

As it can be seen from the figure below. (which plots *s* after gamma transformations), the effect of $\gamma < 1$ is the opposite of when $\gamma > 1$. This is because, as gamma increases it results in an increase in contrast due to the compression of intensity levels, and vice versa when gamma decreases, this time due to the expansion of intensity levels.



The gamma (γ) values corresponding to the following Figures 4-7 are 0.1, 0.5, 1.5 and 2.0 respectively.

```
gamma = [0.1,0.5,1.5,2.0]
for x in gamma:
   correctedImage = np.array(255*(imageFog/255) ** x, dtype='uint8')
```

Figure 4: 0.1

Figure 5: 0.5

Figure 6: 0.1

Figure 7: 0.5

(Appendix A: Another example of contrast enhancement using power-law intensity transformation)

Special Effects

For this next task, an image was warped to the polar coordinate system. The transformation from cartesian to polar coordinates is given by:

$$r = \sqrt{x^2 + y^2} \quad \text{and} \quad \theta = tan^{-1}(\frac{y}{x})$$

Which allows us to define any point (x, y) as (r, θ). Since, the polar coordinate system uses the unit circle, the features that were straight (in the cartesian system) are now circular, and the features that were once more circular, now appear straight
The image used for this task (Figure 8) contains stong straight features, and as it can be observed after its conversion to polar coordinates, its edges are now circular.

```
W = img.shape[0]
H = img.shape[1]
cent_x = W/2
cent_y = H/2
maxRadius = np.sqrt(W**2 + H**2)/2
img2 = cv2.linearPolar(img, (cent_x, cent_y), maxRadius, cv2.WARP_FILL_OUTLIERS)
```

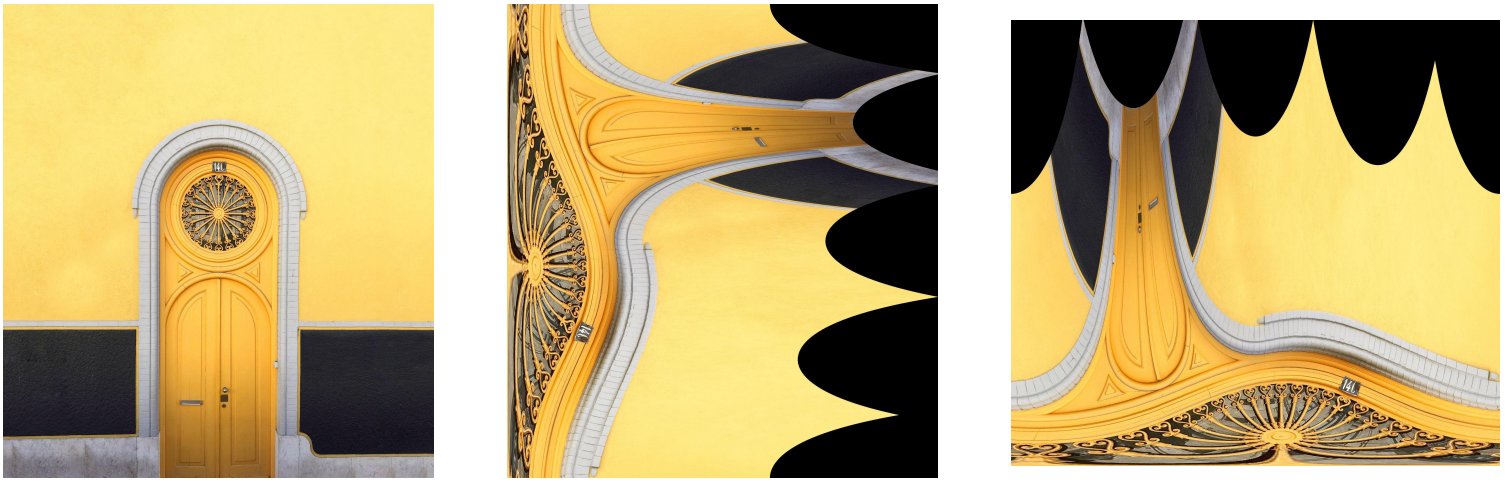


Figure 8: From Cartesian to polar Coordinate System

The next task asked to Cartoonify an image, and this was done using edge detection and color quantization.
In cartoons the edges of the objects in the image are darker. To recreate this, the Adaptive threshold technique is used. This allows for all the pixels whose values are above a chosen threshold to stand out, while leaving the rest in the background. Before applying this technique, the input image needs to be converted to gray scale:

```
grayImage = cv2.cvtColor(imgBeforeCartoon, cv2.COLOR_BGR2GRAY)
```

Moving further, using the adaptiveThreshold() function from the OpenCV library:

```
edges = cv2.adaptiveThreshold(grayImage, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 9, 5)
```

The last parameter in this method is the threshold value, and only the pixels with a value higher than this threshold are considered. Figure 9 shows the results of using this technique with a threshold value of 8, while Figure 10 for a value of 5.

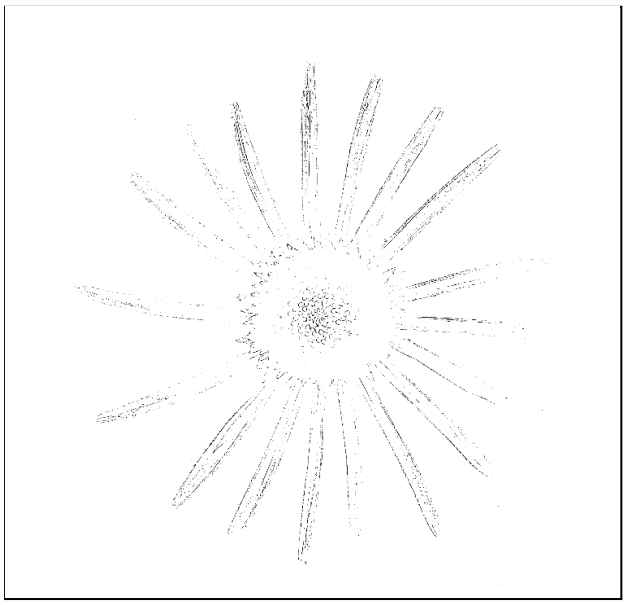


Figure 9

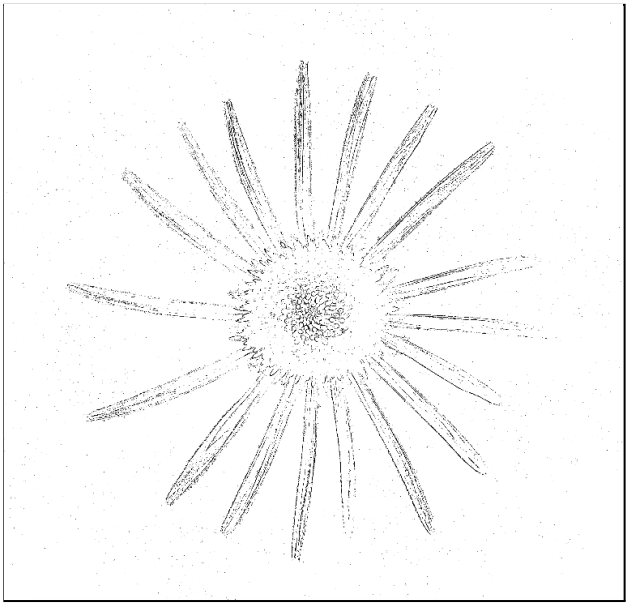
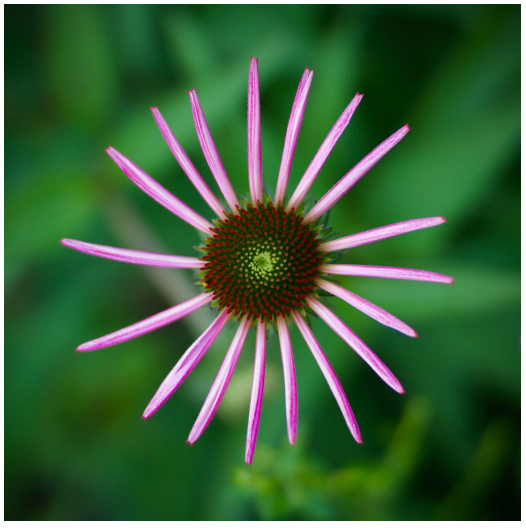


Figure 10

Color quantization, on the other hand is used to reduce the number of colors displayed without affecting its quality. This uses K-means clustering to group the pixels in k clusters/groups. In order to perform this technique, the image has to be reshaped. Its original format (which can be seen with image.shape) consists of the number or rows, columns and its channel, while when resized the rows and colum will be combined to a single column of pixels.

```
reshaped = np.float32(imgBeforeCartoon).reshape((-1, 3))
```

The next step is to choose the number of clusters wanted, thus K. The more groups the more colours will be represented, more specifically, for this task K = 10, so 10 centers containing RGB values. The output image is than reshaped to the shape of the input image, but now only containing 10 unique colors.
Lastly, the image is smoothed out by applying the medianBlur() function, and the edge detection and the color quantization are combined to make the final product.



Frequency Domain Properties

For this task a vertical translation (Figure 11) was applied to the image below:

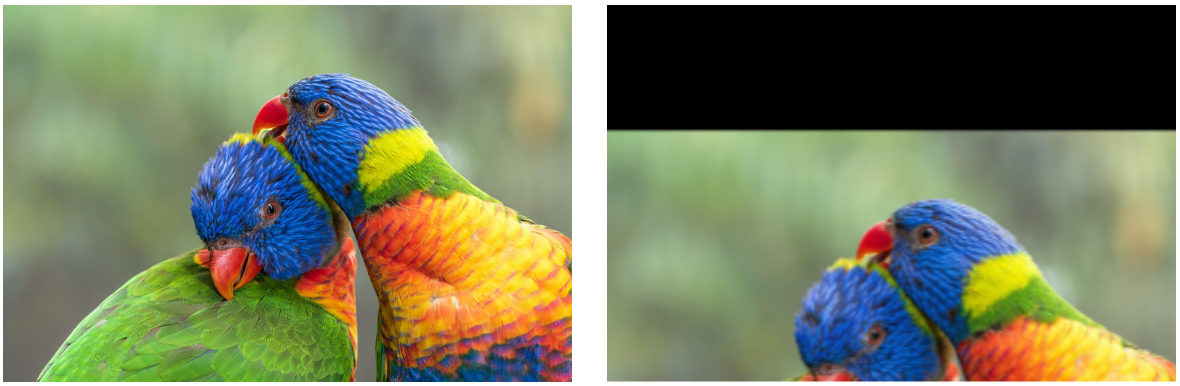


Figure 11 : Image vertically translated

This was accomplished by creating a translation matrix, that denotes the shifts along the x- (tx) and the y-axis (ty).

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

For the purpose of this report, only a vertical translation of ⅓ of the image size was applied to the original image. The final translation was brought out using the cv2.warpAffine() function.

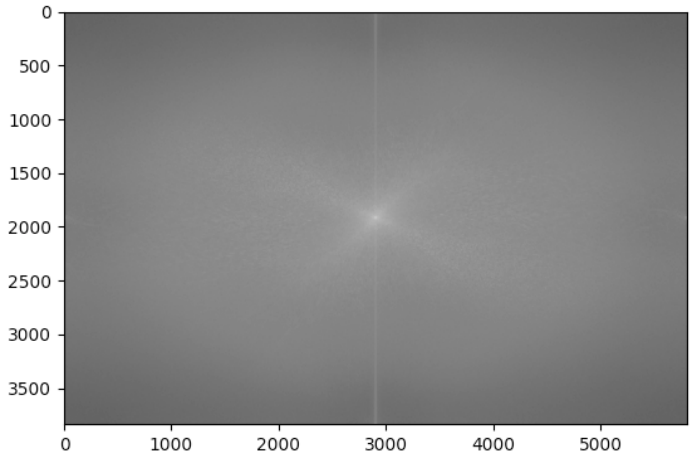
```
height, width = img.shape[:2]
translationMat = np.float32([[1,0,0],[0, 1, height/3]])
translation = cv2.warpAffine(img, translationMat, (width, height))
```

To find the frequency domain of the translated image, we need to look at the 2D Discrete Fourier Transform (DFT), and the Fast Fourier Transfer (FFT) to calculate it. The input image will be the signal, and it is samples in two direction, X and Y. Thus, the frequency representation of the image can be calculated by taking the fourier transform in both directions.

```
def fourierTransform(imgInput):
    f = np.fft.fft2(imgInput)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20*np.log(np.abs(fshift))

    plt.imshow(magnitude_spectrum, cmap = 'gray')
    plt.show()
```

The numpy library has an FFT package that is used to find the Fourier transform. The np.fft.fft2() function takes in the grayscale input image, outputting the frequency transform as a complex array. Next, using the np.fft.fftshift() function we the zero frequency component to the center. To plot the result the abs() function is user, as the output from the FFT will be an array of complex numbers. The magnitude spectrum can then be calculated. The magnitude (or absolute value) of the complex value, represents the amplitude of a part of a complex sinusoid with that frequency.



The average of the image's values is the "dot" in the middle of the magnitude spectrum, this is because the RGB values are between 0-255, thus the average is non-zero.

Periodic Noise Removal

For the last task, *Salt and Pepper* noise was applied to an image. To do so, 10000 random pixels were chosen using the randInt() function and given the either the value 255 (white) or 0 (black).

```
def add_noise(img):
    r, c = img.shape
    for i in range(10001):
        y_white = random.randint(0, r-1)
        x_white = random.randint(0, c-1)
        img[y_white][x_white] = 255

        y_black = random.randint(0, r-1)
        x_black = random.randint(0, c-1)
        img[y_black][x_black] = 0
    return img
```

To denoise the image, the cv2.fastNlMeansDenoising() function was used:

```
denoised = cv2.fastNlMeansDenoising(noisyImage, None, 20, 20)
```

References

https://www.had2know.org/technology/hsv-rgb-conversion-formula-calculator.html
https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.hist.html
https://matplotlib.org/3.5.0/tutorials/introductory/images.html#sphx-glr-tutorials-introductory-images-py
https://www.geeksforgeeks.org/python-intensity-transformation-operations-on-images/
https://socratic.org/trigonometry/the-polar-system/polar-coordinates
https://pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-adaptivethreshold/
https://www.geeksforgeeks.org/image-translation-using-opencv-python/
https://thepythoncodingbook.com/2021/08/30/2d-fourier-transform-in-python-and-fourier-synthesis-of-images/

Appendix

A)

B) Another example of a cartoonified image with different k-cluser values