SOS 2024
IMAGE PROCESSING

Mentor:Amruta
Mentee: Ayush Kumar Kamal

MIDTERM REPORT

Introduction

Images:- images are visual representations of external objects around us.Images hold a wealth of information, open to interpretation from various perspectives

It can be produced through via

- Analog means
- Digital means

Advantage of one over the other:- It boils down to cost, speed, and quality.

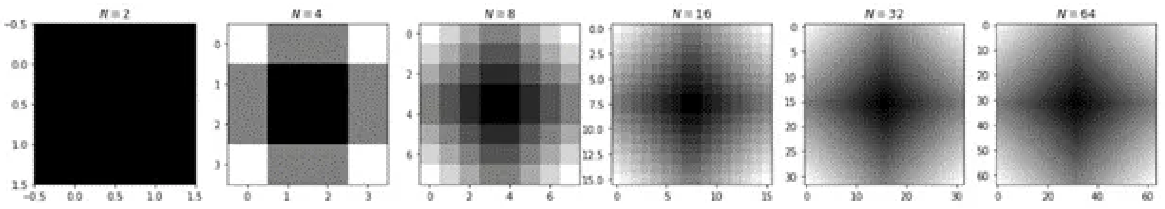
- Digital images are way cheaper and faster to produce than analog ones

Sampling and Quantization

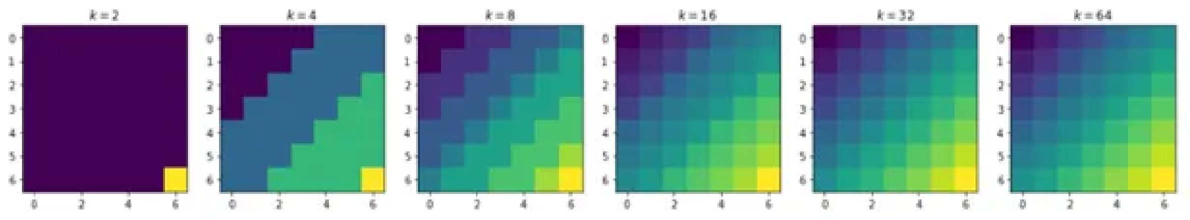
Images that we encounter and interact with in real life are mostly analog (continuous sensed data) in nature.
Analog signals can't be stored directly as it will require an infinite amount of bits(storage) to represent and such is not of any digital computer capability.

Techniques that will be used for digitalizing such images — **sampling** and **quantization.**

Sampling involves the reduction of a continuous-time signal to a discrete-time signal.Meaning it takes the value of the image at regular spatial intervals.



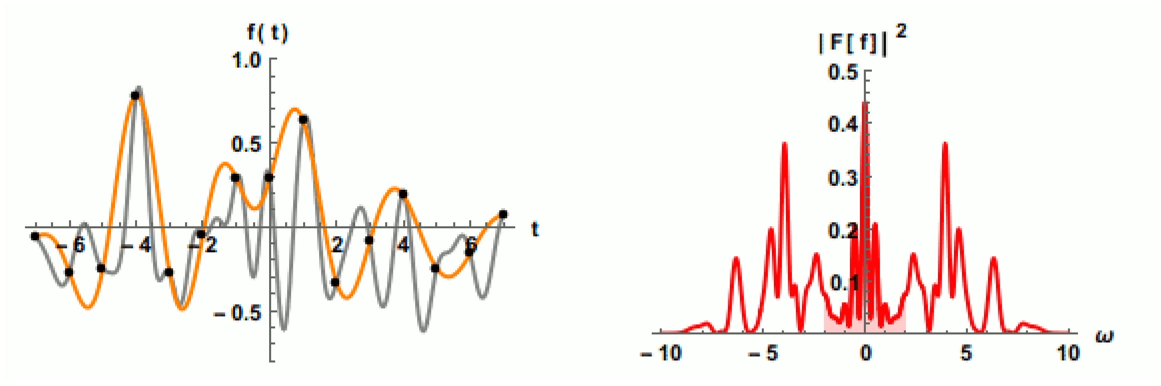
Quantization on the other hand discretizes the intensity values of an analog image. Below we show how quantization is being done with a sample analog image.



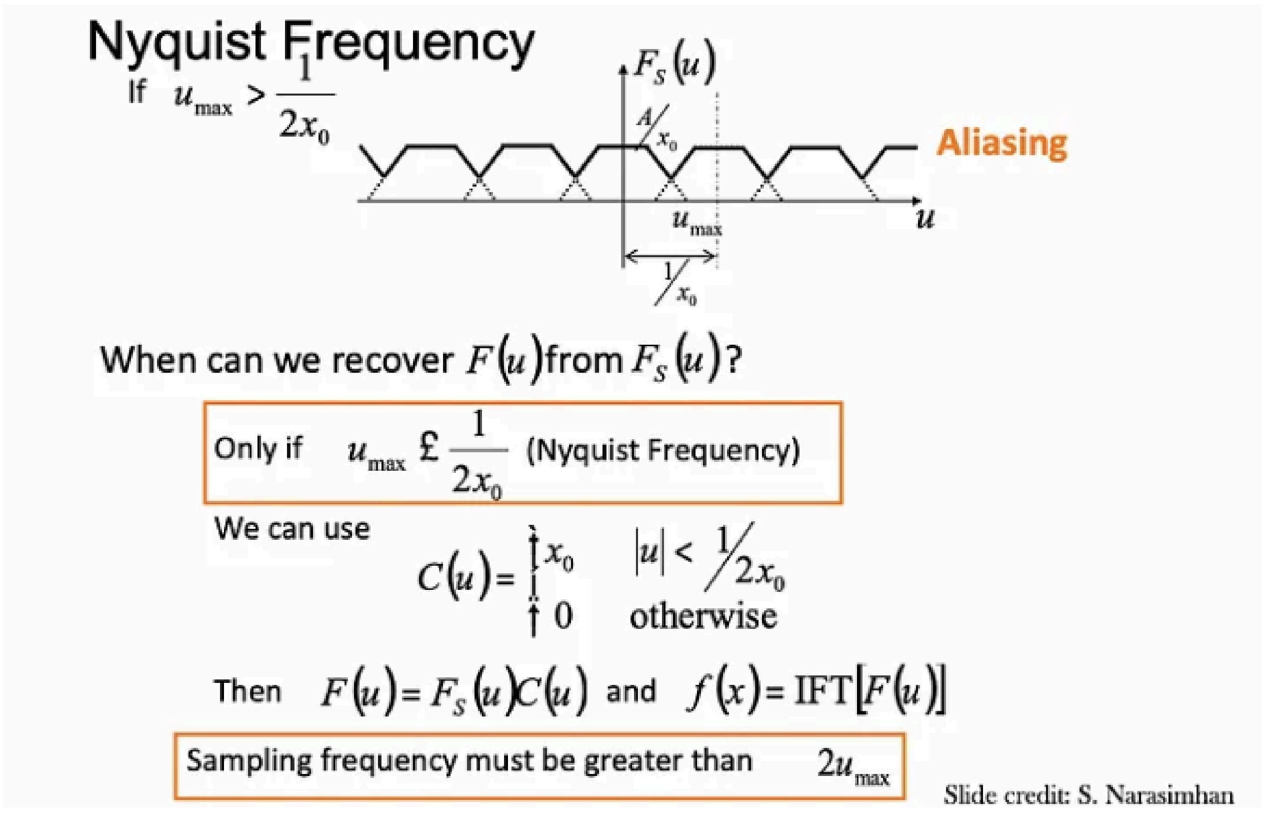
Note: The number of bits k used to represent an intensity value is known as its **bit depth**.

Nyquist Theorem

The Nyquist theorem is a theorem that focuses on signal regeneration via signal re-sampling frequency. The theorem basically states that, in order to recover the information in a signal, such a signal has to be sampled at a frequency which is twice the frequency of operation for such a signal. For instance, if the operating frequency for signal A is 30kHz; to recover such a signal, we have to sample at nothing less than 60kHz. This is the Nyquist theorem.



Nyquist Frequency


$F_S(u)$
A/x_0
Aliasing
$u_{\max}$
u
$1/x_0$

Nyquist Frequency

If $u_{\max} > \frac{1}{2x_0}$

When can we recover $F(u)$ from $F_S(u)$?

Only if $u_{\max} \pounds \frac{1}{2x_0}$ (Nyquist Frequency)

We can use

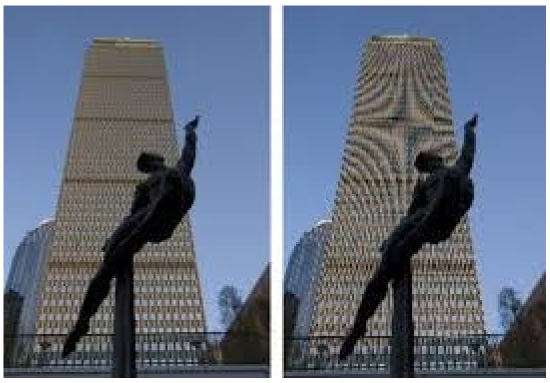
$$C(u) = \begin{cases} x_0 & |u| < 1/2x_0 \\ 0 & \text{otherwise} \end{cases}$$

Then $F(u) = F_S(u)C(u)$ and $f(x) = \mathrm{IFT}[F(u)]$

Sampling frequency must be greater than $2u_{\max}$

Slide credit: S. Narasimhan

When we sample such a message signal at a frequency less than Nyquist frequency, then we will be having a situation called **aliasing**.
Aliasing is a distortion effect in a reconstructed signal which produces an entirely different signal different from the signal to be sampled.

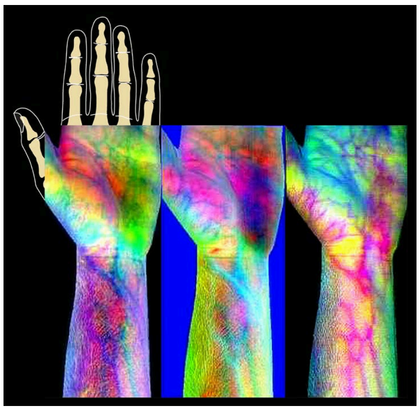


Effect of aliasing in the image

Digital images taxonomy:

1. Colour images: Multiple colour channels
2. Grey-scale images : Images having shades of grey as their only colour are grayscale images

3. Binary images: Having only two pixel values(0 or 1)
4. Multispectral images: images that capture image data ranging across the electromagnetic spectrum within some specific wavelength.


Multispectral image

Grey-scale image

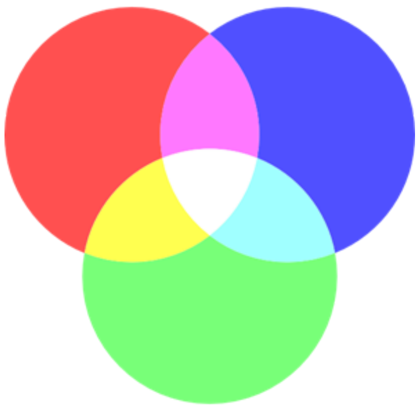
Colour Image

Binary Image

Image Colour Space

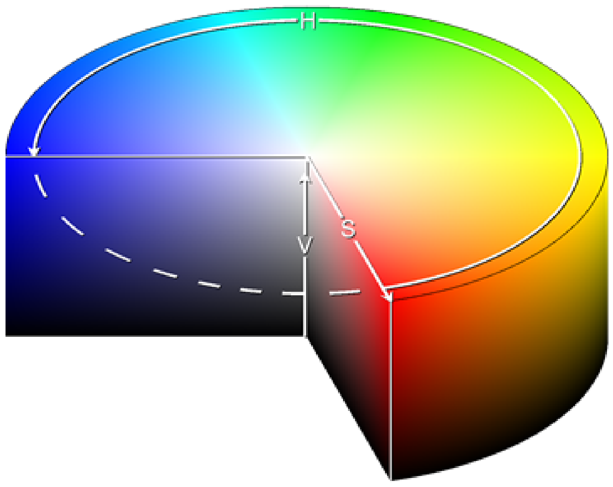
A ‘colour space’ describes the range of colours that are available within any system, such as a computer, an image file, and a printed photograph. It is a standard way of holding these colours, so that when this colour standard is known, other systems can reproduce the correct colours.
RGB and HSV colour spaces

RGB (Red, Green, Blue)



The RGB (red, green , blue) colour space relates very closely to the way we perceive colour with the r, g and b receptors in our retinas. RGB uses additive colour mixing and is the basic colour model used in television or any other medium that projects colour with light**.**

HSV (Hue, Saturation, Value)



HUE

Hue is the colour portion of the model, expressed as a number from 0 to 360 degrees:

- Red falls between 0 and 60 degrees.
- Yellow falls between 61 and 120 degrees.
- Green falls between 121 and 180 degrees.
- Cyan falls between 181 and 240 degrees.
- Blue falls between 241 and 300 degrees.
- Magenta falls between 301 and 360 degrees.

SATURATION

Saturation describes the amount of gray in a particular colour, from 0 to 100 percent. Reducing this component toward zero introduces more gray and produces a faded effect. Sometimes, saturation appears as a range from 0 to 1, where 0 is gray, and 1 is a primary colour.

VALUE (OR BRIGHTNESS)

Value works in conjunction with saturation and describes the brightness or intensity of the colour, from 0 to 100 percent, where 0 is completely black, and 100 is the brightest and reveals the most colour.

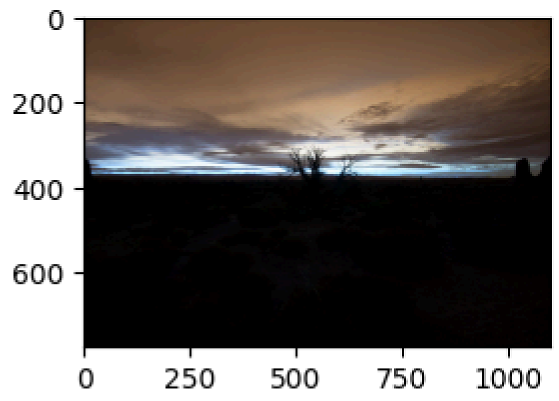
Intensity Transformation

Intensity transformation in image processing refers to the manipulation of the intensity values of an image's pixels to enhance or modify the image's appearance. This process involves applying a mathematical function to each pixel's intensity value (brightness) to achieve the desired effect.

```
[12]: import cv2
      import matplotlib.pyplot as plt
      import matplotlib.image as img
      from PIL import Image
      import numpy as np
```

```
[26]: image_unex = cv2.imread('under_exposedImage.jpg')
      plt.figure(figsize = (3,3))
      plt.imshow(image_unex)
```

```
[26]: <matplotlib.image.AxesImage at 0x1c6e26d68a0>
```

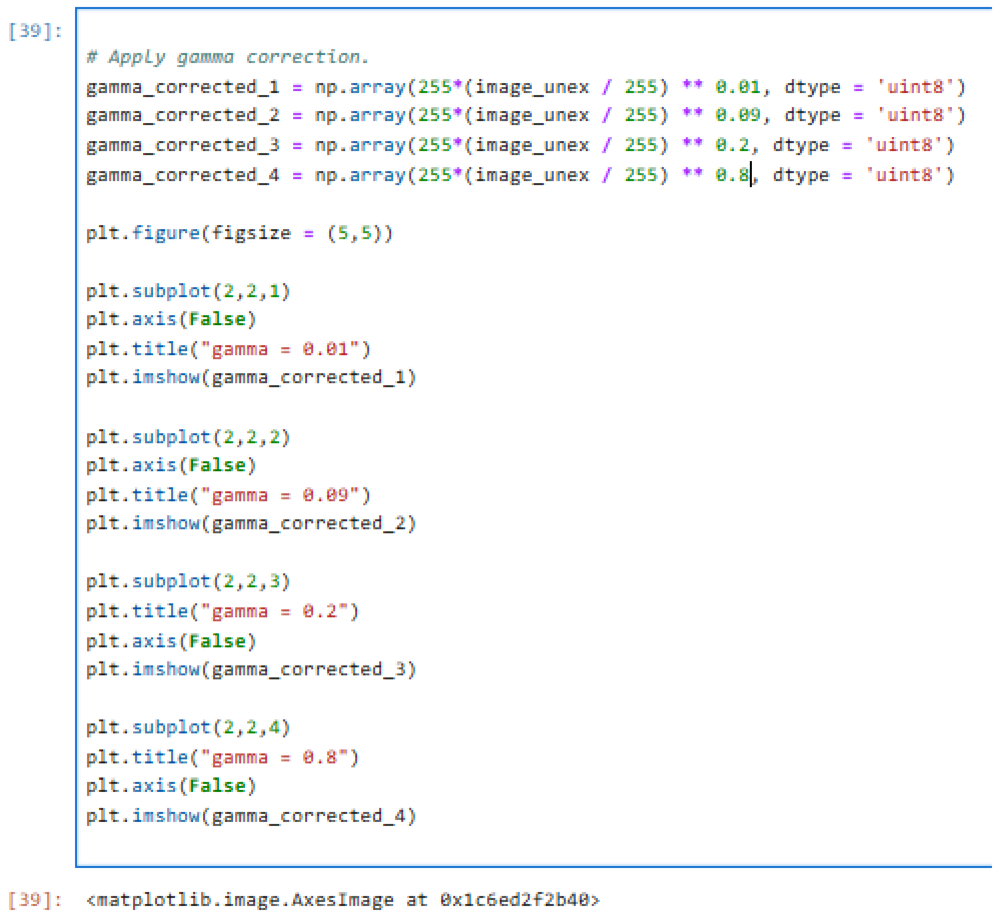


Power-Law(Gamma) Transformation

The general form of Power law (Gamma) transformation function is

$$s = c*r^{(gamma)}$$

Where, ‘s’ and ‘r’ are the output and input pixel values, respectively and ‘c’ and gamma are the positive constants.



```
[39]:
# Apply gamma correction.
gamma_corrected_1 = np.array(255*(image_unex / 255) ** 0.01, dtype = 'uint8')
gamma_corrected_2 = np.array(255*(image_unex / 255) ** 0.09, dtype = 'uint8')
gamma_corrected_3 = np.array(255*(image_unex / 255) ** 0.2, dtype = 'uint8')
gamma_corrected_4 = np.array(255*(image_unex / 255) ** 0.8, dtype = 'uint8')

plt.figure(figsize = (5,5))

plt.subplot(2,2,1)
plt.axis(False)
plt.title("gamma = 0.01")
plt.imshow(gamma_corrected_1)

plt.subplot(2,2,2)
plt.axis(False)
plt.title("gamma = 0.09")
plt.imshow(gamma_corrected_2)

plt.subplot(2,2,3)
plt.title("gamma = 0.2")
plt.axis(False)
plt.imshow(gamma_corrected_3)

plt.subplot(2,2,4)
plt.title("gamma = 0.8")
plt.axis(False)
plt.imshow(gamma_corrected_4)

[39]: <matplotlib.image.AxesImage at 0x1c6ed2f2b40>
```

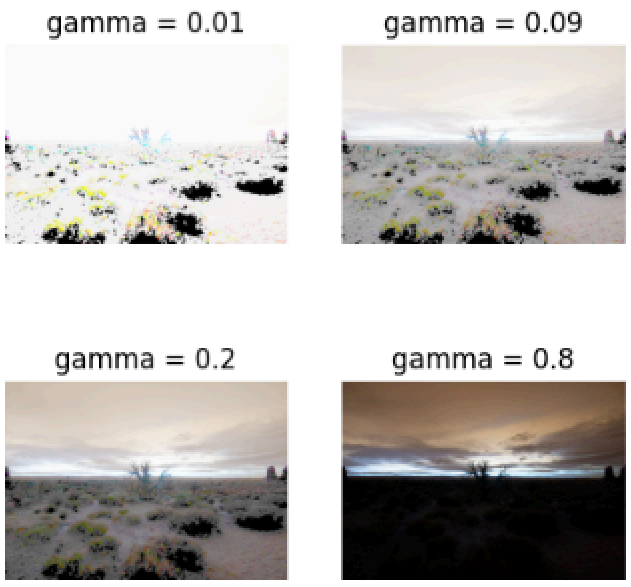


Image Histogram

Image Histogram is a typical histogram representing distribution of pixel values(intensity level) within an image.

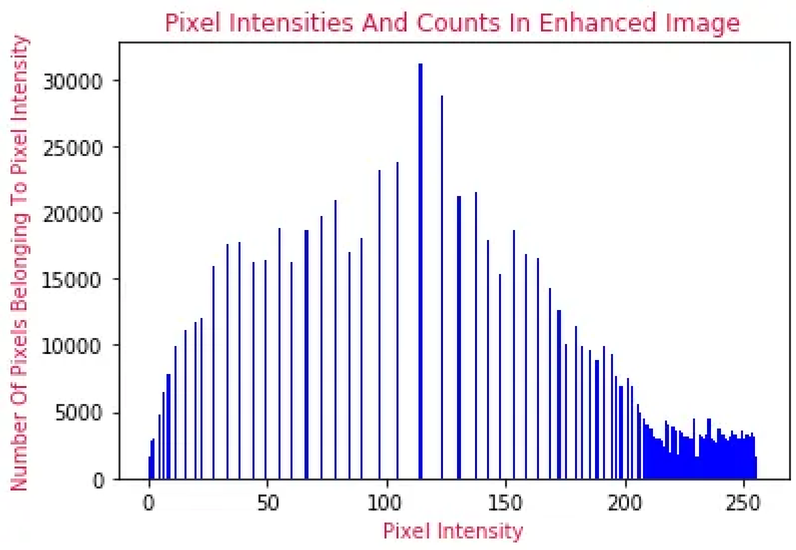
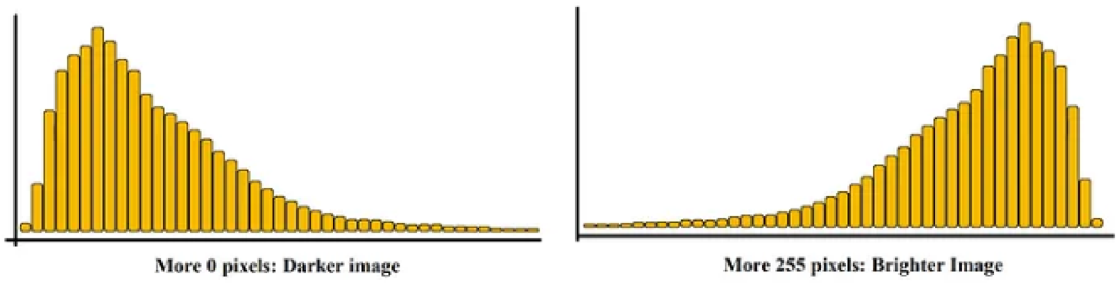


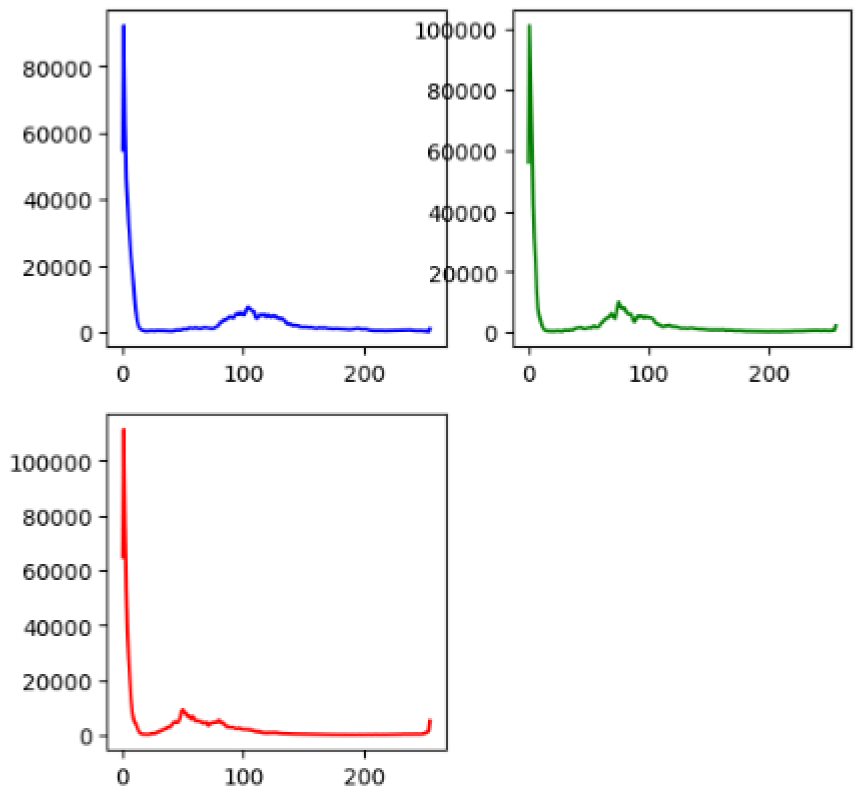
Image Histogram



Obtaining Image histograms of a colour image for each of its colour channels namely Red, Blue, Green using ***cv2.calcHist*** function in open cv module.

[17]:
```
#
B_histo = cv2.calcHist([image_unex],[0], None, [256], [0,256])
G_histo = cv2.calcHist([image_unex],[1], None, [256], [0,256])
R_histo = cv2.calcHist([image_unex],[2], None, [256], [0,256])

plt.figure(figsize = (6,6))
plt.subplot(2, 2, 1)
plt.plot(B_histo, 'b')
plt.subplot(2, 2, 2)
plt.plot(G_histo, 'g')
plt.subplot(2, 2, 3)
plt.plot(R_histo, 'r')
```

[17]:
```
[<matplotlib.lines.Line2D at 0x1c6e8d62ea0>]
```

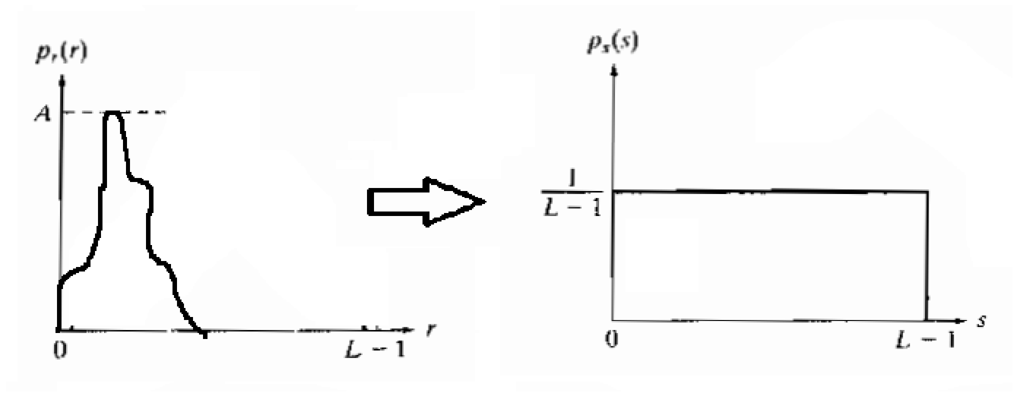


Histogram Equalization

Consider an underexposed or overexposed image whose pixel values are confined to some specific range of values only.
For eg: the brighter image will have all pixels confined to high values.

But a good image will have pixels from all regions of the image. So you need to stretch this histogram to either end. This normally improves the contrast of the image.

The goal is to achieve a uniform pdf distribution as shown below:



$$s = T(r) = (L-1)\int_0^r Pr(w)\ dw$$

Source: Author

Now, differentiation of s with respect to r is:

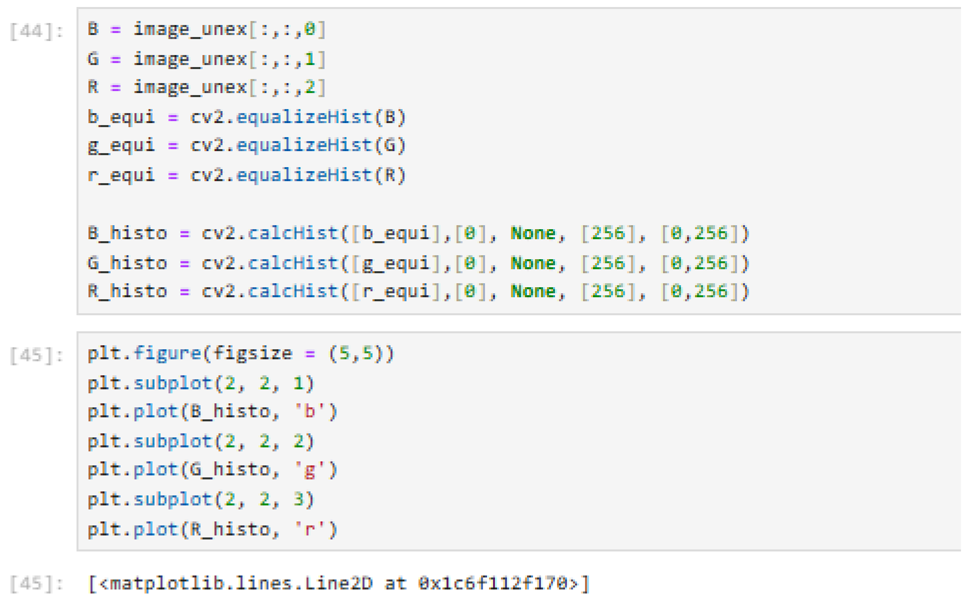
$$\begin{aligned}\frac{ds}{dr} &= \frac{d}{dr}T(r)\\ &= \frac{d}{dr}(L-1)\int_0^r Pr(w)\ dw\\ &= (L-1)Pr(r)\end{aligned}$$

Relation between Pr(r) and Ps(s) can be achieved as:

$$\begin{aligned}Ps(s) &= Pr(r)\left|\frac{ds}{dr}\right|\\ &= Pr(r)\left|\frac{1}{(L-1)Pr(r)}\right|\\ &= \frac{1}{(L-1)};0 \leq L-1\end{aligned}$$

So, as can be seen, Ps(s) is a normalised distribution. We can say that equalisation of the histogram can be achieved by an assumed transfer function.

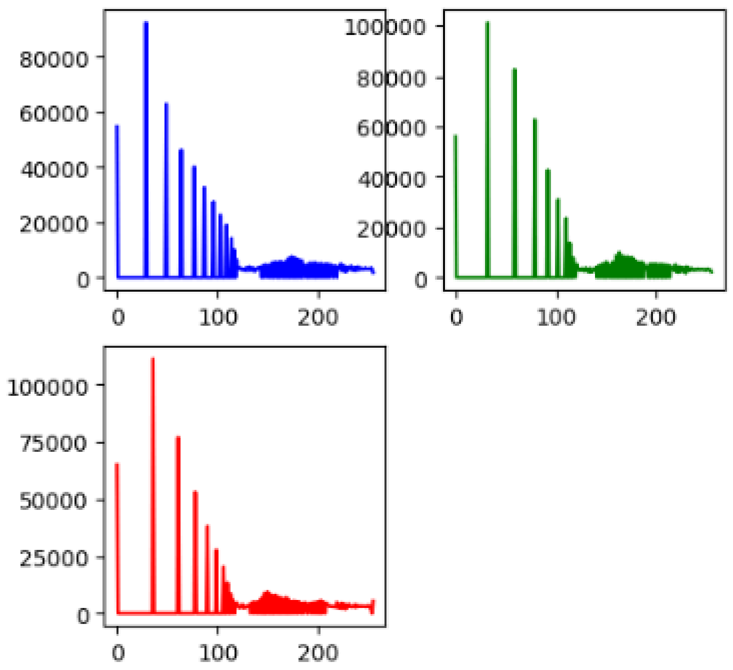
In python histogram equalisation of an underexposed or overexposed image can be obtained using ***cv2.equalizeHist .***



```
[44]: B = image_unex[:,:,0]
      G = image_unex[:,:,1]
      R = image_unex[:,:,2]
      b_equi = cv2.equalizeHist(B)
      g_equi = cv2.equalizeHist(G)
      r_equi = cv2.equalizeHist(R)

      B_histo = cv2.calcHist([b_equi],[0], None, [256], [0,256])
      G_histo = cv2.calcHist([g_equi],[0], None, [256], [0,256])
      R_histo = cv2.calcHist([r_equi],[0], None, [256], [0,256])
```

```
[45]: plt.figure(figsize = (5,5))
      plt.subplot(2, 2, 1)
      plt.plot(B_histo, 'b')
      plt.subplot(2, 2, 2)
      plt.plot(G_histo, 'g')
      plt.subplot(2, 2, 3)
      plt.plot(R_histo, 'r')
```

```
[45]: [<matplotlib.lines.Line2D at 0x1c6f112f170>]
```



<u>The reason for not having flat histogram(which is desired) after histogram equalisation is:-</u>

The transformation function used in histogram equalisation is based on the cumulative distribution function (CDF) of the image's histogram. This mapping ensures that the output histogram matches the desired distribution as closely as possible, but due to the finite and discrete nature of pixel values, this mapping cannot achieve a perfectly uniform distribution.

<u>Spatial & Frequency Filtering</u>

Spatial Filters

These are of two type:-

1. Linear Spatial Filter

2. Non-linear Spatial Filter

***Smoothing Spatial Filter*:** Smoothing filter is used for blurring and noise reduction in the image. Blurring is pre-processing steps for removal of small details and Noise Reduction is accomplished by blurring.

Its Types are :-

1. Linear Filter (Mean Filter)

2. Order Statistics (Non-linear) filter

Mean Filter:

Linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. The idea is replacing the value of every pixel in an image by the average of the grey levels in the neighborhood define by the filter mask.

Types of Mean filter:

- **(i) Averaging filter:** It is used in reduction of the detail in image. All coefficients are equal.
- **(ii) Weighted averaging filter:** In this, pixels are multiplied by different coefficients. Center pixel is multiplied by a higher value than average filter.

Order Statistics Filter:

It is based on the ordering the pixels contained in the image area encompassed by the filter. It replaces the value of the center pixel with the value determined by the ranking result. Edges are better preserved in this filtering.

Types of Order statistics filter:

- **(i) Minimum filter:** 0th percentile filter is the minimum filter. The value of the center is replaced by the smallest value in the window.
- **(ii) Maximum filter:** 100th percentile filter is the maximum filter. The value of the center is replaced by the largest value in the window.
- **(iii) Median filter:** Each pixel in the image is considered. First neighboring pixels are sorted and original values of the pixel is replaced by the median of the list.

***Sharpening Spatial Filter*:** It is also known as derivative filter. The purpose of the sharpening spatial filter is just the opposite of the smoothing spatial filter. Its main focus in on the removal of blurring and highlight the edges. It is based on the first and second order derivative.

First order derivative:

- Must be zero in flat segments.
- Must be non zero at the onset of a grey level step.
- Must be non zero along ramps.

First order derivative in 1-D is given by:

```
f' = f(x+1) - f(x)
```

Second order derivative:

- Must be zero in flat areas.
- Must be zero at the onset and end of a ramp.
- Must be zero along ramps.

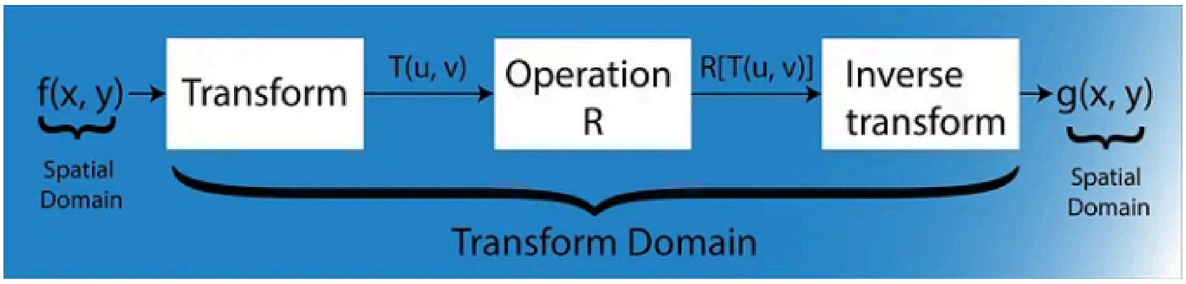
Second order derivative in 1-D is given by:

```
f'' = f(x+1) + f(x-1) - 2f(x)
```

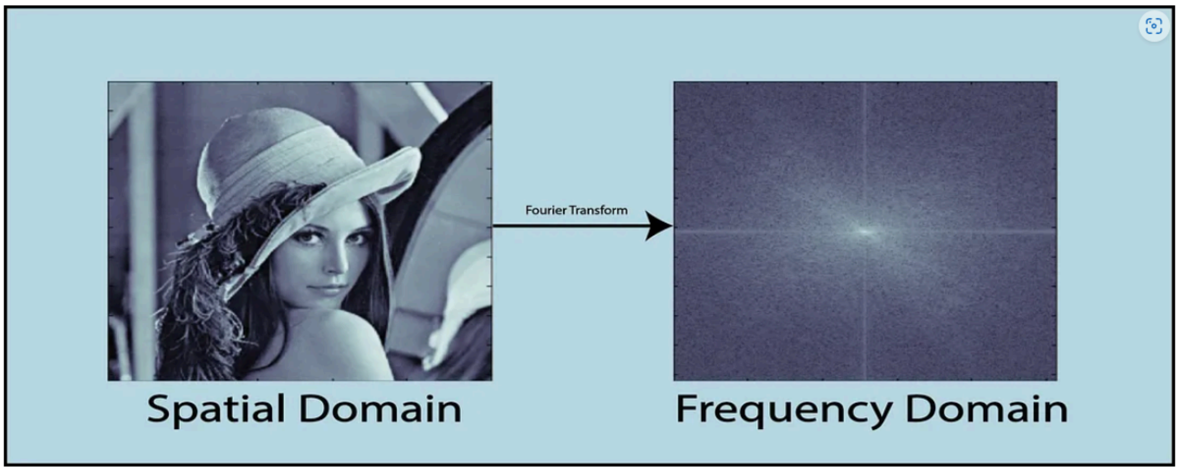
Need for a domain other than spatial domain:-

Many times, image processing tasks are best performed in a domain other than the spatial domain. Moreover, it is easy to detect some features in a particular domain,i.e., a new information can be obtained in other domains.

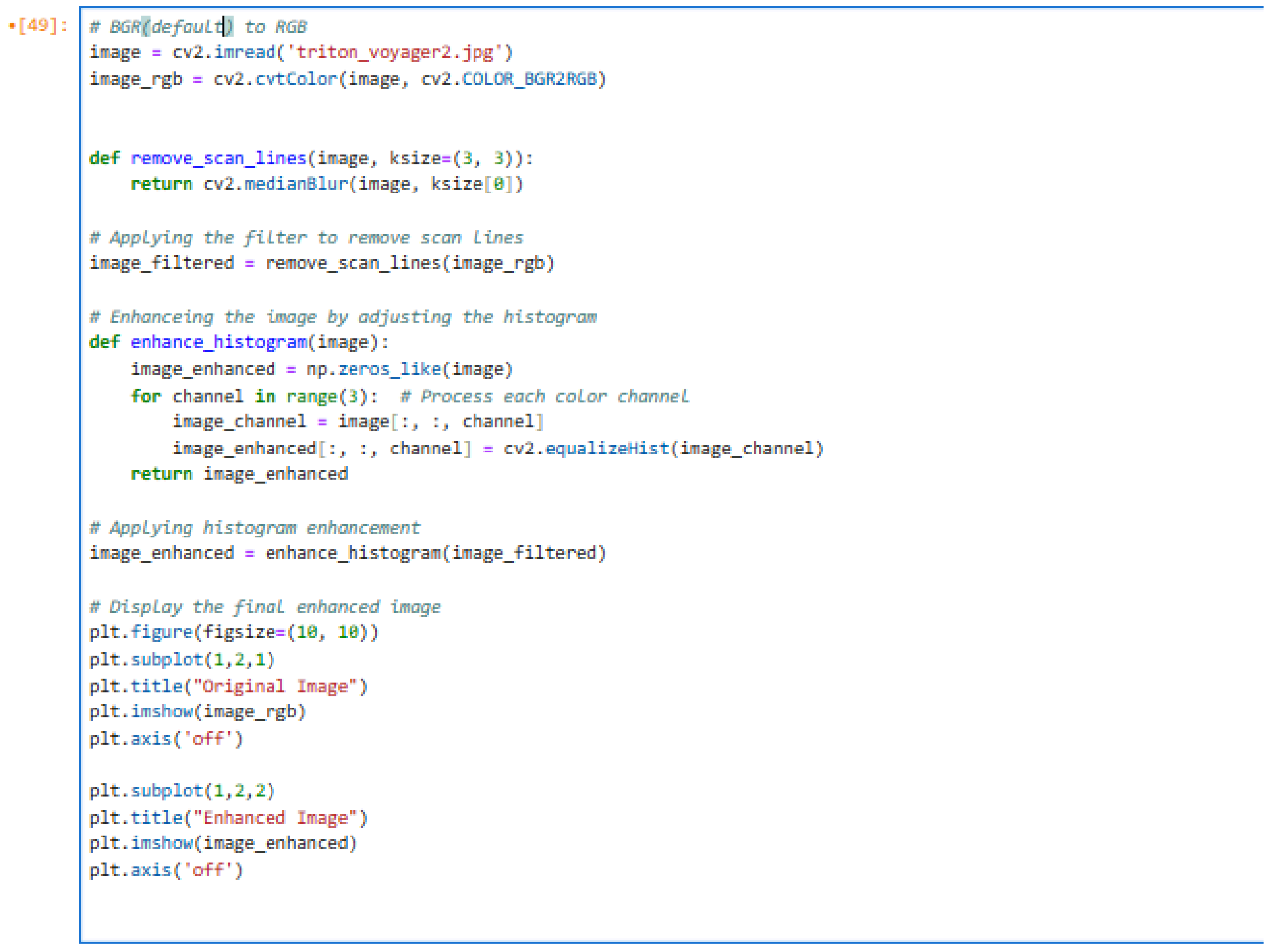
Image Transformation mainly follows three steps-



> ***Step-1.*** *Transform the image.*
>
> ***Step-2.*** *Carry the task(s) in the* transformed domain.
>
> ***Step-3.*** *Apply* inverse transform *to return to the spatial domain.*

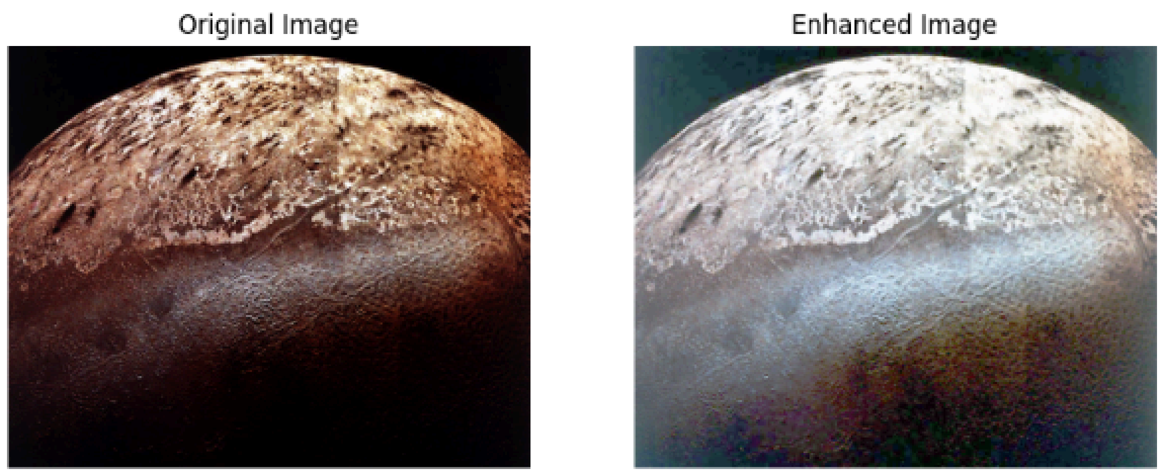


Histogram Manipulation(Simply applying histogram equalization for each color channels) for removing the scan lines



```
[49]:
# BGR(default) to RGB
image = cv2.imread('triton_voyager2.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


def remove_scan_lines(image, ksize=(3, 3)):
    return cv2.medianBlur(image, ksize[0])

# Applying the filter to remove scan lines
image_filtered = remove_scan_lines(image_rgb)

# Enhanceing the image by adjusting the histogram
def enhance_histogram(image):
    image_enhanced = np.zeros_like(image)
    for channel in range(3):  # Process each color channel
        image_channel = image[:, :, channel]
        image_enhanced[:, :, channel] = cv2.equalizeHist(image_channel)
    return image_enhanced

# Applying histogram enhancement
image_enhanced = enhance_histogram(image_filtered)

# Display the final enhanced image
plt.figure(figsize=(10, 10))
plt.subplot(1,2,1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1,2,2)
plt.title("Enhanced Image")
plt.imshow(image_enhanced)
plt.axis('off')
```

```
[49]: (-0.5, 679.5, 550.5, -0.5)
```

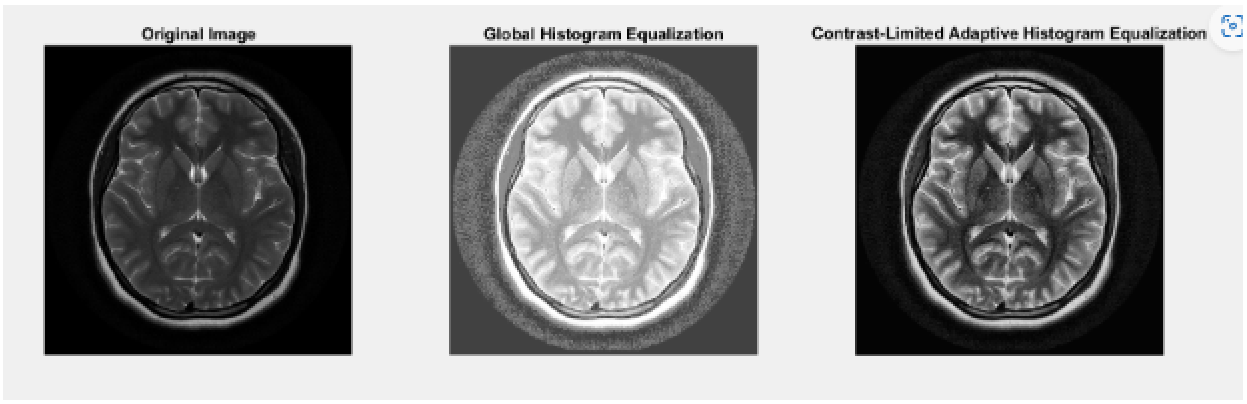


Conclusion: Does not seem to work properly as image quality appears to reduced instead of increasing!

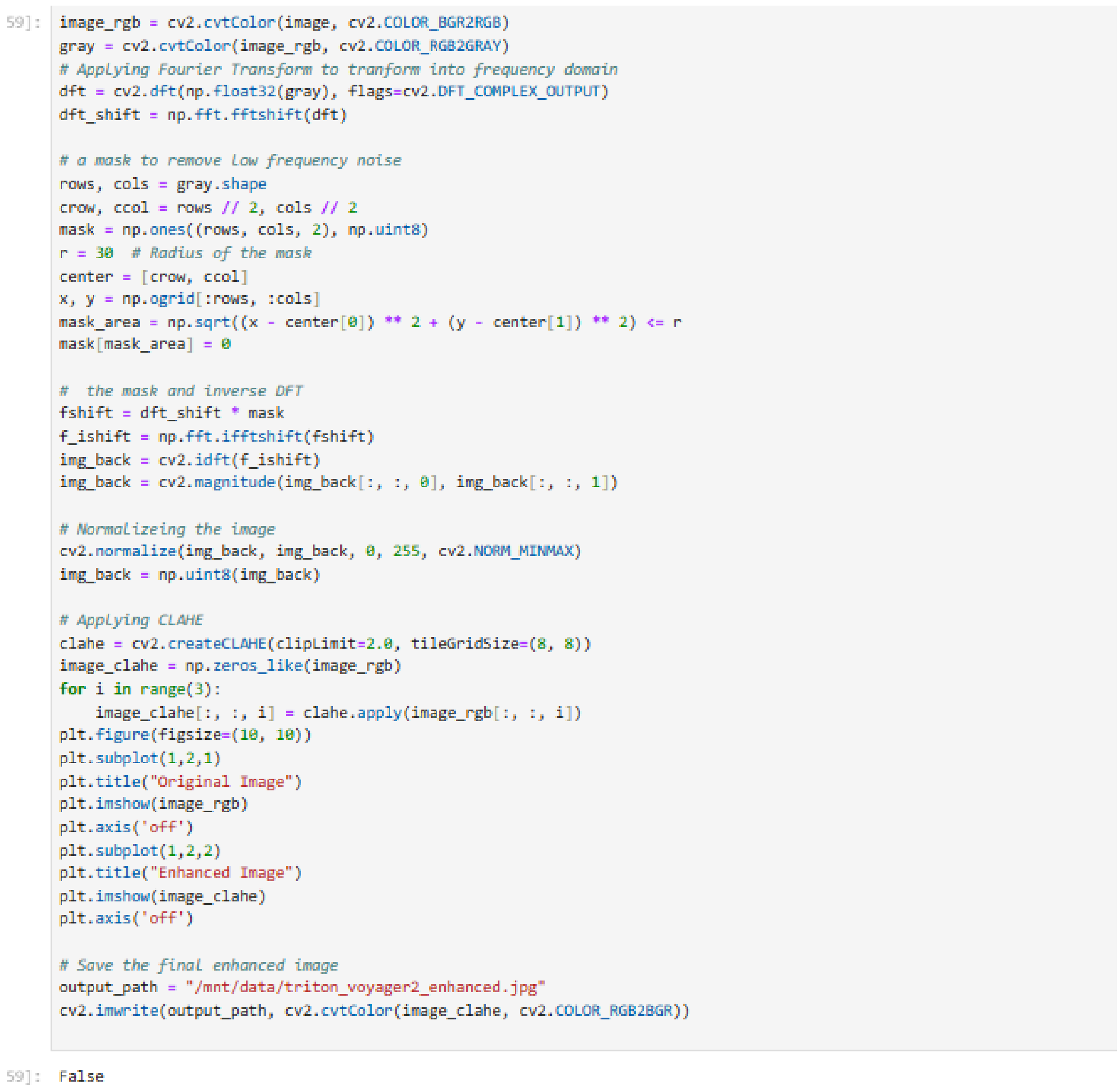
<u>Contrast-Limited Adaptive Histogram Equalization (CLAHE)</u>

Contrast-Limited Adaptive Histogram Equalization (CLAHE) can be divided into two parts

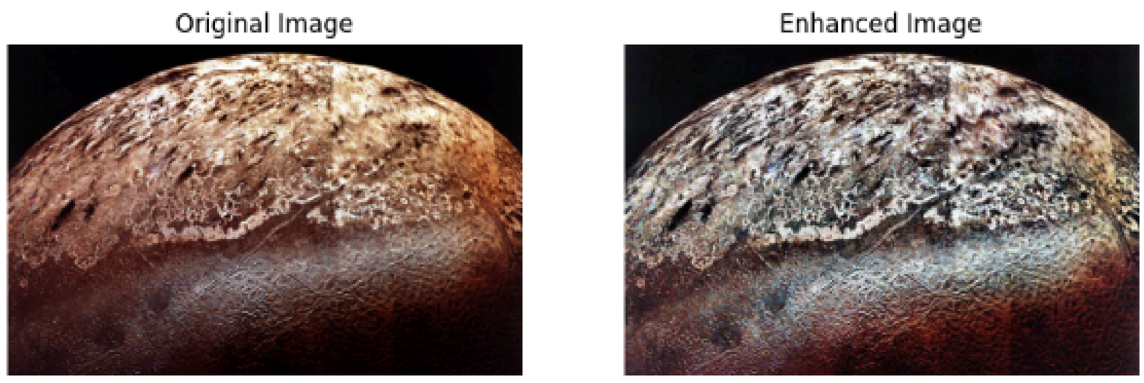
- Adaptive Histogram Equalization and
- Contrast-Limiter


Original Image
Global Histogram Equalization
Contrast-Limited Adaptive Histogram Equalization

Rather we can transform image first to frequency domain and then applying histogram equalization(its advanced form CLAHE)



```
59]: image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
     gray = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
     # Applying Fourier Transform to tranform into frequency domain
     dft = cv2.dft(np.float32(gray), flags=cv2.DFT_COMPLEX_OUTPUT)
     dft_shift = np.fft.fftshift(dft)

     # a mask to remove low frequency noise
     rows, cols = gray.shape
     crow, ccol = rows // 2, cols // 2
     mask = np.ones((rows, cols, 2), np.uint8)
     r = 30  # Radius of the mask
     center = [crow, ccol]
     x, y = np.ogrid[:rows, :cols]
     mask_area = np.sqrt((x - center[0]) ** 2 + (y - center[1]) ** 2) <= r
     mask[mask_area] = 0

     #  the mask and inverse DFT
     fshift = dft_shift * mask
     f_ishift = np.fft.ifftshift(fshift)
     img_back = cv2.idft(f_ishift)
     img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

     # Normalizeing the image
     cv2.normalize(img_back, img_back, 0, 255, cv2.NORM_MINMAX)
     img_back = np.uint8(img_back)

     # Applying CLAHE
     clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
     image_clahe = np.zeros_like(image_rgb)
     for i in range(3):
         image_clahe[:, :, i] = clahe.apply(image_rgb[:, :, i])
     plt.figure(figsize=(10, 10))
     plt.subplot(1,2,1)
     plt.title("Original Image")
     plt.imshow(image_rgb)
     plt.axis('off')
     plt.subplot(1,2,2)
     plt.title("Enhanced Image")
     plt.imshow(image_clahe)
     plt.axis('off')

     # Save the final enhanced image
     output_path = "/mnt/data/triton_voyager2_enhanced.jpg"
     cv2.imwrite(output_path, cv2.cvtColor(image_clahe, cv2.COLOR_RGB2BGR))

59]: False
```



ALL THE CODE CAN BE FOUND HERE:-
https://github.com/kamal-ayush1920/SOS-2024_IMAGE-PROCESSING.git