

**Department of Artificial Intelligence**

**College of Computer Science and Information Technology**

1. **Objectives**
2. Understanding Image Enhancements and and Frequency Domain

**Due Date: Tuesday October 15, 2024 @ 11:59 PM**

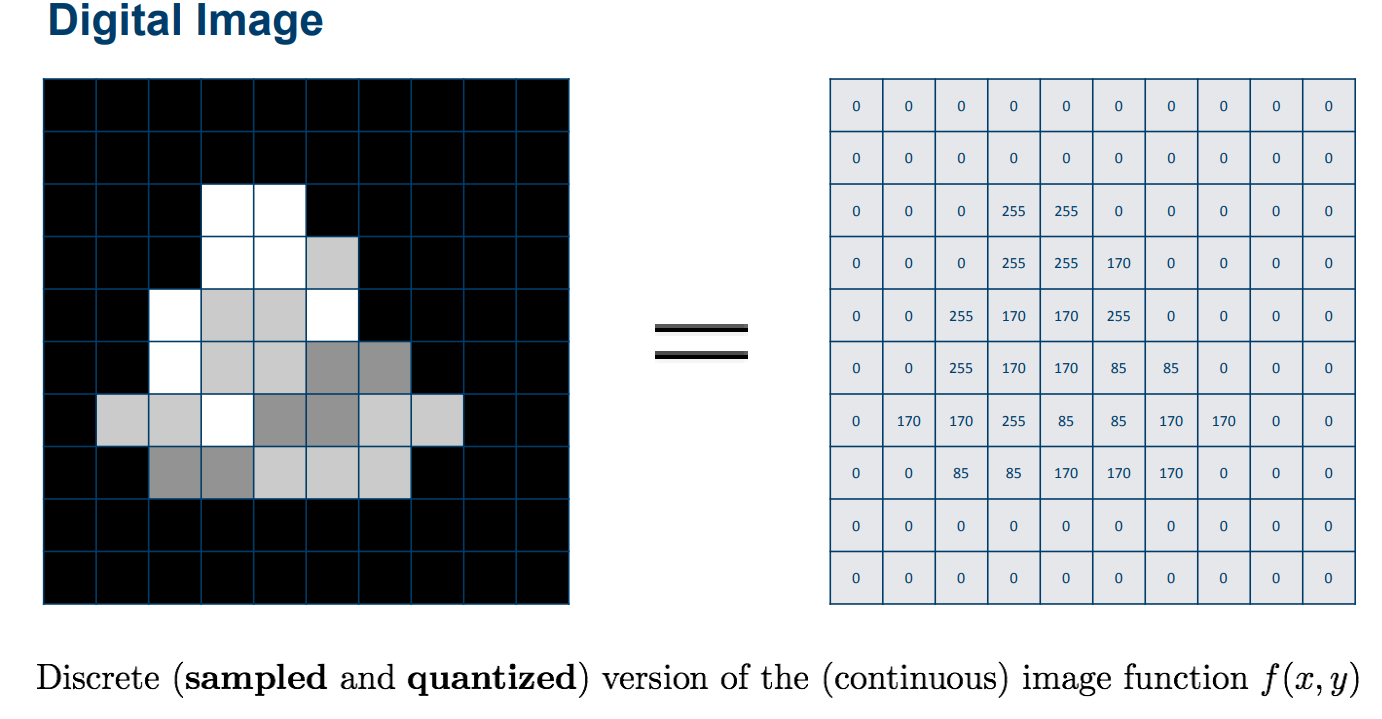
**Late Submissions:**

* Q: Can I skip the lab and submit the solution?
  + You will receive a mark of **zero** if you do not attend the lab, even if you complete the exercise. Attending the labs is compulsory for evaluation. If you have a justified excuse, you may receive a partial mark depending on the circumstances. See the next question for information on late submissions.
* **Q:** If I submit it at 12:00am, you’ll still mark it, right?
  + **A:** 11:59pm and earlier is on time. Anything after 11:59pm is late. Anything late will **NOT** be probably marked. If I find you have a legitimate cause, you will be graded according to the following rules (24 hours after deadline 🡪 assignment is marked out of 75% only, 48 hours after deadline 🡪 assignment is marked out of 50% only, 72 hours after deadline 🡪 assignment is marked out of 25% only).

1. **Introduction**

Image is 2D signals where f(x,y) gives intensity at positions (x,y).

****

****

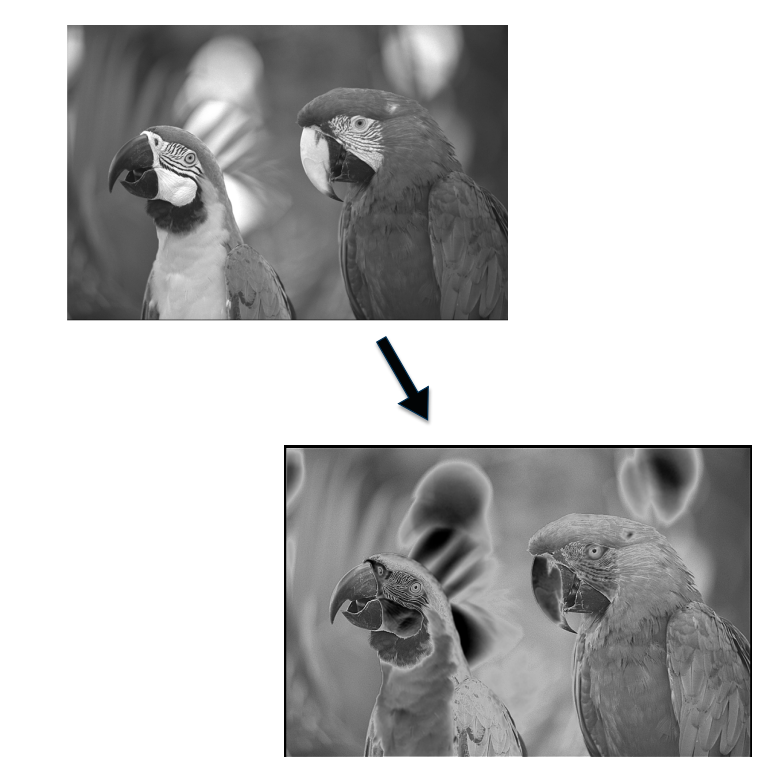
An image can be represented in two domains: the spatial domain and the frequency domain. The spatial domain is the most common representation of an image, with pixel values representing brightness or color at each point in the image. The frequency domain, on the other hand, represents the image as a collection of sinusoidal waves of varying frequencies and amplitudes.

Image Processing

• Point operators

• Image filtering in spatial domain

• Image filtering in frequency domain

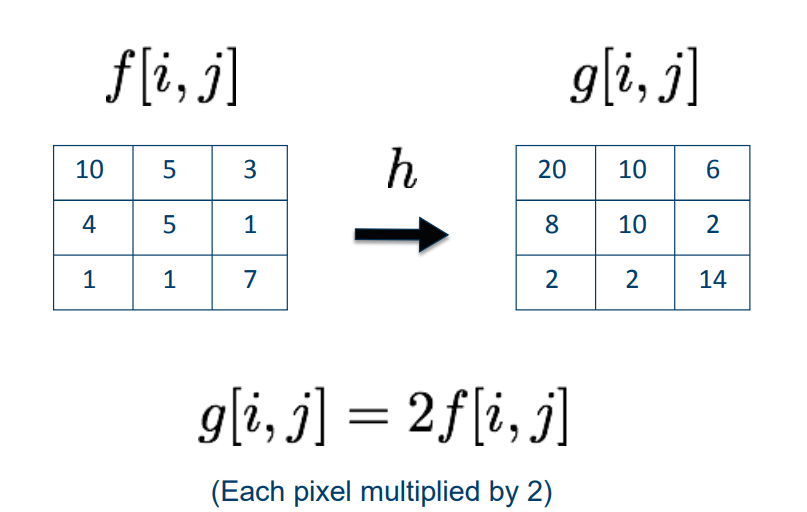


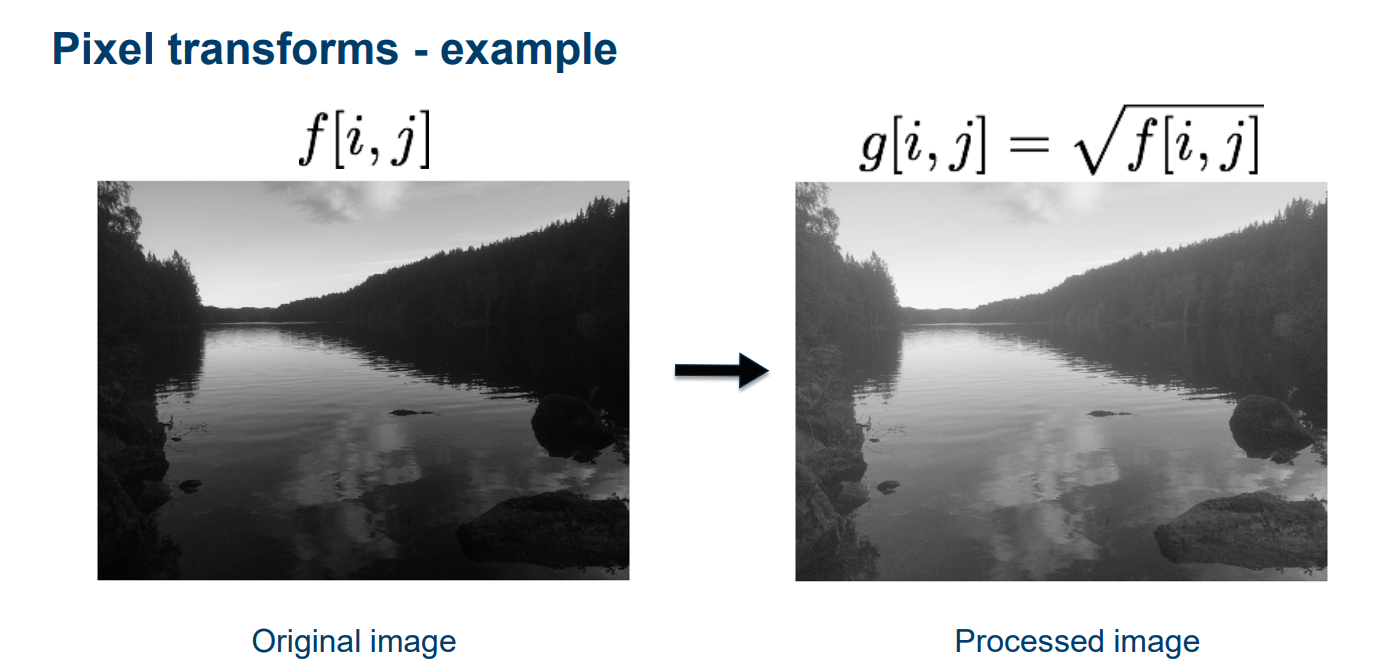
**Point Operator**

Point operations modify a given pixel based on its value and possibly its position. These operations can be represented by a mapping or transfer function.

Example:

* Pixel transforms: Brightness adjustment and Contrast adjustment
* Color transforms
* Histogram equalization





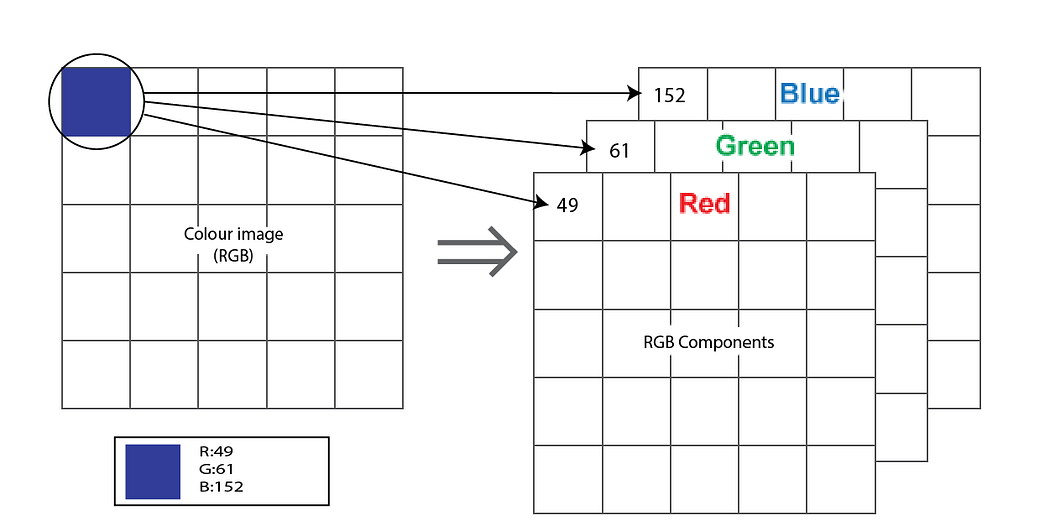
**Spatial Domain**

An image can be represented in the form of a 2D matrix where each element of the matrix represents pixel intensity. This state of 2D matrices that depict the intensity distribution of an image is called Spatial Domain. It can be represented as shown A blue background with a blue background with a blue background with a blue background with a blue and white background with a black and white image and a black and white image with a white and black image

Description automatically generated with medium confidence

Illustration of Spatial Domain

For the RGB image, the spatial domain is represented as a 3D vector of 2D matrices. Each 2D matrix contains the intensities for a single color as shown below-



Spatial domain for color image(RGB)

Each pixel intensity is represented as I(x,y) where x,y is the co-ordinate of the pixel in the 2D matrix. Different operations are carried out in this value. For example- operation T(say, addition of 5 to all the pixel) is carried out in I(x,y) which means that each pixel value is increased by 5. This can be written as-

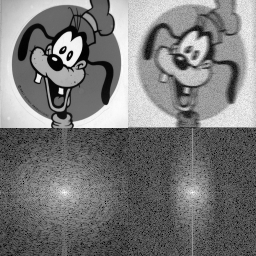
I’(x,y) = T[I(x,y)]

where, I’(x,y) is the new intensity after adding 5 to I(x,y).

**Frequency Domain**

In frequency-domain methods are based on Fourier Transform of an image. Roughly, the term frequency in an image tells about the rate of change of pixel values.

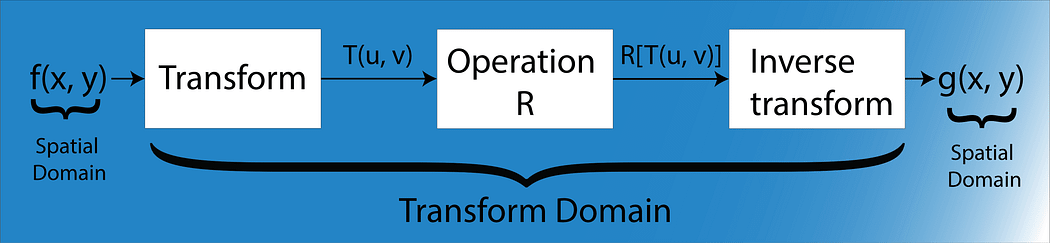
Below diagram depicts the conversion of image from spatial domain to frequency domain using Fourier Transformation-



Source: [www.cs.unm.edu](https://www.cs.unm.edu/)

Why we need 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**



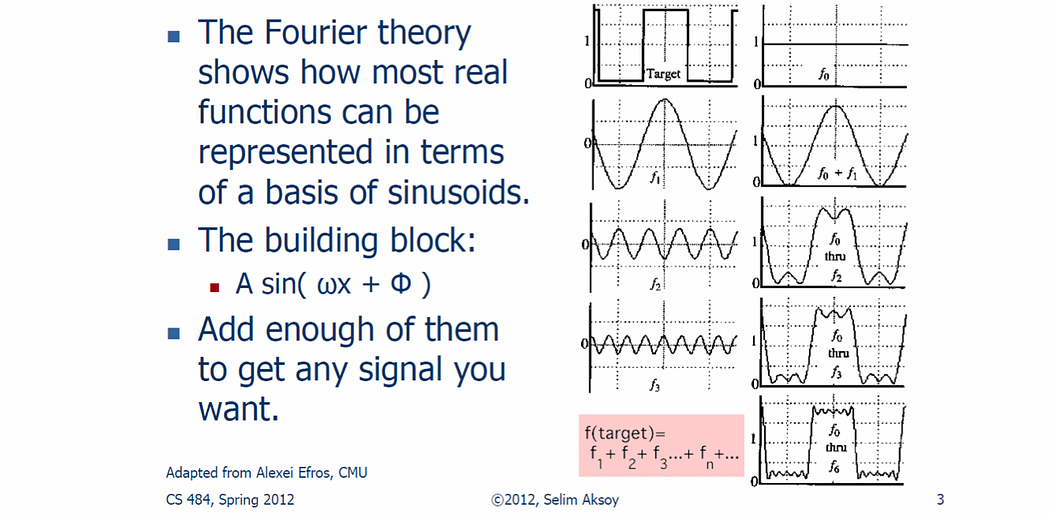
***Step-1.****Transform the image.*

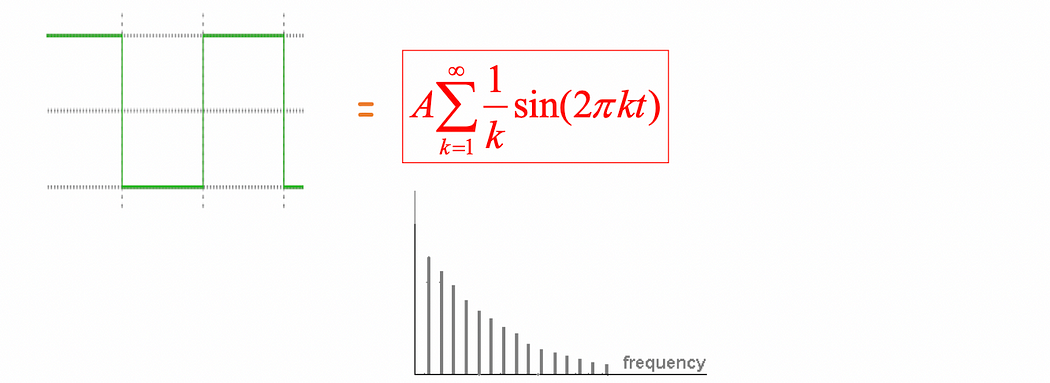
***Step-2.****Carry the task(s) in the*transformed domain*.*

***Step-3.****Apply*inverse transform*to return to the spatial domain.*

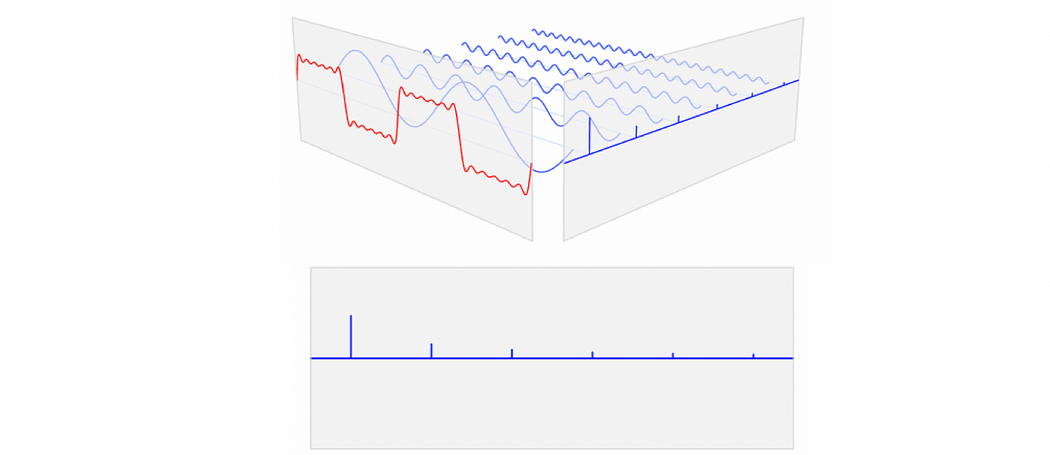
**Fourier theory.**

Any function that periodically repeats itself can be expressed as a sum of sines and cosines of different frequencies each multiplied by a different coefficient.

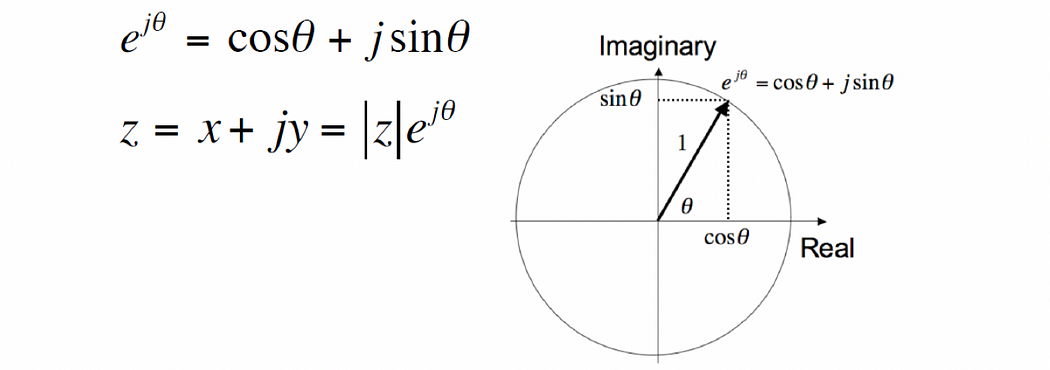




Frequency spectra



***Basics: Euler’s Formula***



**Fourier Transform**

**Fourier Transform** is a powerful tool used in image processing to analyze an image’s frequency components. Essentially, it helps us identify intricate patterns and structures in the image’s frequency domain that are difficult to see in the spatial domain. Think of it as putting on glasses that allow you to see things in greater detail and clarity.

One common application of the Fourier Transform is noise removal from images. Noise typically appears in the high-frequency components of an image, which can be easily identified and isolated through Fourier Transform. By analyzing the frequency components of an image, we can remove noise or unwanted features and enhance important features. To achieve this, we *convert the image from the spatial domain to the frequency domain* using **Fourier Transform**, filter out the noise, and then use the **Inverse Fourier Transform** *to convert the image back to the spatial domain.*

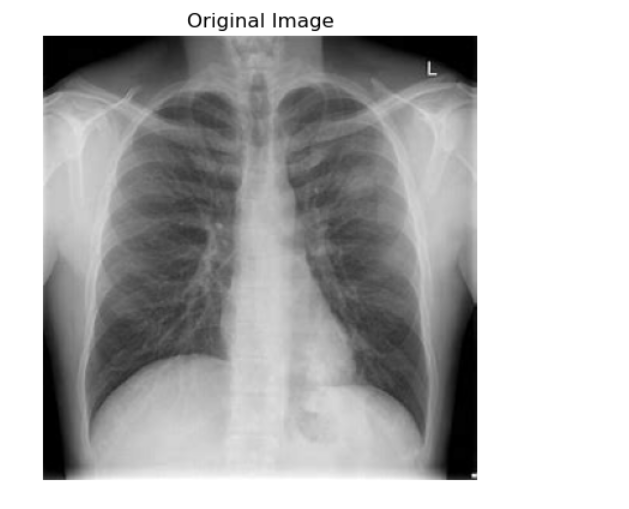
Now that we understand the potential of using Fourier Transform in image processing, let’s dive into a step-by-step process of how it can be used to improve the quality of an image and correct any distortions or noise.

**Step 1: Import the necessary libraries**

import numpy as np  
import matplotlib.pyplot as plt  
from skimage.color import rgb2gray  
from skimage.io import imread, imshow

**Step 2: Load and display the image**

image = imread('lunar\_orbiter.jpg')  
plt.imshow(image)  
plt.title('Original Image')  
plt.axis('off')  
plt.show()



We can apply Fourier Transform in this image to remove the scan marks and improve its quality.

**Step 3: Convert the image to grayscale and compute the 2D Fourier Transform**

Converting the image to grayscale simplifies the analysis and reduces computational complexity. The 2D Fourier Transform is computed to obtain the frequency domain representation of the image.

gray\_image = rgb2gray(image)  
f\_image = np.fft.fft2(gray\_image)

**Step 4: Shift the DC or zero-frequency components to the center and compute the magnitude spectrum**

An image’s **frequency components**provide insight into the spatial patterns and repetitive structures within the image:

* **Low-frequency components** correspond to smooth and gradual changes, while
* **high-frequency components** represent abrupt changes and fine details.
* The **DC component,** or**zero frequency**, represents the average brightness of the image.

By understanding the importance of frequency components, we can use FT effectively in image processing to manipulate images and uncover hidden information.

After calculating the Fourier Transform, we need to shift the zero-frequency components (i.e., the DC component) to the center of the spectrum to better visualize the low and high frequencies.

We can use the np.fft.fftshift() function for this purpose:

fshift = np.fft.fftshift(f\_image)

**Step 5: Visualize the magnitude spectrum**

To visualize the magnitude spectrum, we can calculate the magnitude of the complex numbers obtained from the Fourier Transform and plot them:

magnitude\_spectrum = 20 \* np.log(np.abs(fshift))  
plt.imshow(magnitude\_spectrum, cmap='gray')  
plt.title('Magnitude Spectrum')  
plt.axis('off')  
plt.show()

A grey square with a white light

Description automatically generated

The **magnitude spectrum** is a visual representation of the frequency components of an image. It allows us to see the various patterns and details within an image and separates them into three main frequency components mentioned earlier:

**1. DC component:** This is the zero-frequency component, which corresponds to the average brightness or luminosity of the image. It is usually located at the center of the magnitude spectrum.

**2. Low frequencies**: These components represent the smooth and gradual changes in the image, such as large uniform areas, soft gradients, or slow transitions in brightness. They are generally found closer to the center of the magnitude spectrum, surrounding the DC component. Low frequencies contain more image information than higher ones, as their magnitude is higher.

**3. High frequencies:** These components capture the abrupt changes, fine details, edges, textures, or noise within the image. They are typically found towards the outer edges of the magnitude spectrum, with smaller magnitudes compared to low frequencies.

Analyzing the magnitude spectrum helps us identify areas that may require enhancement or filtering to improve image quality. In the example, we can see two dominating patterns, vertical and horizontal lines, that are visible in the spectrum. These patterns typically stem from the regular structures present in the background of the image (such as the very evident horizontal scan marks in the image).

This visualization of the magnitude spectrum is important for understanding the distribution of frequency components that we can later use to design filters that target specific regions of the frequency domain to enhance or remove noise from our original image (such as the scan marks).

**Step 6: Apply filters to the frequency domain representation of the image**

Applying filters to the frequency domain representation allows us to remove unwanted frequency components (e.g., noise like the scan marks in the image) or emphasize specific frequencies. In this example, we apply vertical filters to remove horizontal lines from the image.

It is important to note that we should **avoid filtering the DC component**, as it represents the average brightness of the image, and modifying it can affect the overall luminosity. This is why we excluded 286 (or fshift.shape[0]//2 -1) in the code below.

Since we shifted the DC to the center, we can use fshift.shape[1]//2 to locate the center of the frequency domain representation (magnitude spectrum) along the horizontal axis. The fshift.shape[1] returns the number of columns in the frequency domain array, and dividing it by 2 (//2) gives us the middle column.

We apply vertical filters to target the horizontal lines in the frequency domain representation. To do this, we set the values of the magnitude spectrum in the vertical direction around the center (both above and below) to 1. This operation effectively filters out the frequency components responsible for the horizontal lines in the image. As a result, when we perform the Inverse Fourier Transform, the horizontal lines are suppressed, and the image appears cleaner.

image\_gray\_fft2 = fshift.copy()  
image\_gray\_fft2[:286, fshift.shape[1]//2] = 1  
image\_gray\_fft2[-286:, fshift.shape[1]//2] = 1  
  
plt.figure(figsize=(7,7))  
plt.imshow(np.log(abs(image\_gray\_fft2)), cmap='gray');

A close-up of a graph

Description automatically generated

In this example, I’ve created a vertical filter in the magnitude spectrum. You can also include horizontal filter, excluding the DC component, to see how it affects the output. (Spoiler alert: It did not help that much. In some cases, you may want to preserve some patterns in the magnitude spectrum (such as the horizontal filter) to maintain the overall image structure, as removing them could lead to *unwanted artifacts or loss of important details or may not be necessary at all*.

For example, when processing an image with a striped pattern, removing horizontal lines might compromise the pattern’s visual integrity and add unwanted vertical line artifacts. It is crucial to analyze the image and its frequency components to make an informed decision on whether to filter these lines or not, based on the specific requirements of your image processing task and the desired outcome.

**Step 7: Use the inverse Fourier Transform to obtain the filtered grayscale image and visualize it**

After applying the desired filters, we can use the Inverse Fourier Transform to convert the filtered frequency domain representation back to the spatial domain:

# Use Inverse Fourier Transform  
inv\_fshift = np.fft.ifftshift(image\_gray\_fft2)  
filtered\_gray\_image = np.fft.ifft2(inv\_fshift)  
filtered\_gray\_image = np.abs(filtered\_gray\_image)  
  
# Plot the original and fourier-transformed grayscale image  
fig, ax = plt.subplots(1, 2, figsize=(14, 7))  
ax[0].imshow(image)  
ax[0].set\_title('Original Image')  
ax[0].axis('off')  
  
ax[1].imshow(filtered\_gray\_image, cmap='gray')  
ax[1].set\_title('Fourier Transformed Grayscale Image')  
ax[1].axis('off')  
  
plt.show()

A close-up of x-ray images

Description automatically generated

**Step 8: Apply the Fourier Transform to each channel of the RGB image and use the same filters as Step 6**

Now that we have seen how to apply the Fourier Transform to a grayscale image, let’s take a look at how we can apply it to a color image. We can use the same technique we used earlier, but this time we apply it to each channel of the RGB image.

First, we iterate over each channel of the image and apply the Fourier Transform to it using the np.fft.fft2 function. We then shift the zero-frequency components to the center using np.fft.fftshift, which is necessary for the filtering step:

transformed\_channels = []  
for i in range(3):  
 rgb\_fft = np.fft.fftshift(np.fft.fft2((image[:, :, i])))  
 rgb\_fft2 = rgb\_fft.copy()  
   
 # Use the same filters as the grayscale image, just change the variables  
 rgb\_fft2[:286, rgb\_fft.shape[1]//2] = 1  
 rgb\_fft2[-286:, rgb\_fft.shape[1]//2] = 1  
 transformed\_channels.append(abs(np.fft.ifft2(np.fft.ifftshift(rgb\_fft2))))

Here, we use the same vertical filter as before, but this time we apply it to each channel separately.

**Step 9: Combine the filtered channels and clip the values to the valid range**

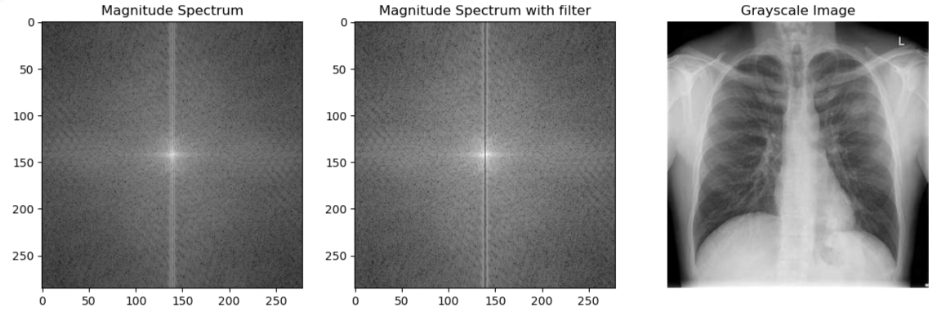
Now that we have filtered each channel of the RGB image, we need to combine them to create the final filtered image. We use the np.dstack function to stack the filtered channels along the depth axis, creating a new 3D array. We then clip the pixel values of this new image to the valid range of 0-255 using np.clip, and convert the pixel values to unsigned 8-bit integers using the astype method.

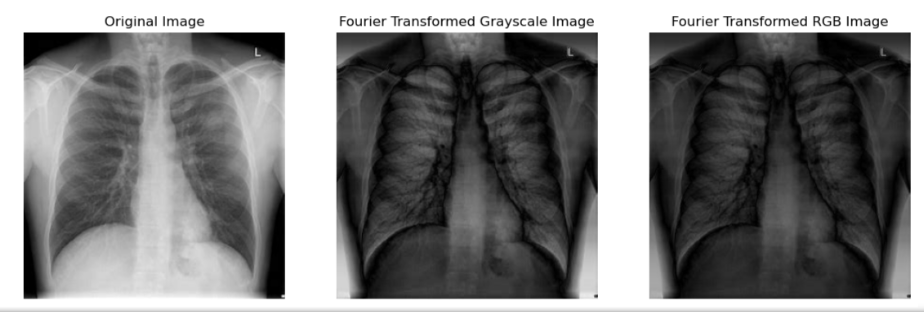
filtered\_rgb\_image = np.dstack([transformed\_channels[0], transformed\_channels[1], transformed\_channels[2]])  
filtered\_rgb\_image = np.clip(filtered\_rgb\_image, 0, 255).astype(np.uint8)

**Step 10: Visualize the results using the plot\_fourier\_transformer function**

Finally, we can visualize the filtering process results using the plot\_fourier\_transformer function. This function takes three arguments: the original image, the filtered grayscale image, and the filtered RGB image.

def plot\_fourier\_transformer(image, filtered\_gray\_image, filtered\_rgb\_image):  
   
 if len(image.shape) == 2:  
 image = np.stack((image, image, image), axis=-1)  
  
 # Convert the image to grayscale  
 gray\_image = rgb2gray(image[:,:,:3])  
   
 # Calculate the 2D Fourier transform and shift the zero-frequency components to the center  
 f\_image = np.fft.fft2(gray\_image)  
 fshift = np.fft.fftshift(f\_image)  
 magnitude\_spectrum = 20 \* np.log(np.abs(fshift))  
  
 # Plot the images  
 fig, ax = plt.subplots(2, 3, figsize=(14, 10))  
  
 ax[0, 0].imshow(magnitude\_spectrum, cmap='gray')  
 ax[0, 0].set\_title('Magnitude Spectrum')  
  
 ax[0, 1].imshow(magnitude\_spectrum, cmap='gray')  
 ax[0, 1].imshow(np.log(abs(image\_gray\_fft2)), cmap='gray')  
 ax[0, 1].set\_title('Magnitude Spectrum with filter')  
  
 ax[0, 2].imshow(gray\_image, cmap='gray')  
 ax[0, 2].set\_title('Grayscale Image')  
 ax[0, 2].set\_axis\_off()  
  
 ax[1, 0].imshow(image)  
 ax[1, 0].set\_title('Original Image')  
 ax[1, 0].set\_axis\_off()  
  
 ax[1, 1].imshow(filtered\_gray\_image, cmap='gray')  
 ax[1, 1].set\_title('Fourier Transformed Grayscale Image')  
 ax[1, 1].set\_axis\_off()  
  
 ax[1, 2].imshow(filtered\_rgb\_image)  
 ax[1, 2].set\_title('Fourier Transformed RGB Image')  
 ax[1, 2].set\_axis\_off()  
  
 # Save the Fourier-transformed RGB image  
 plt.savefig('filtered\_rgb\_image.png', dpi=300)  
 plt.show()  
  
plot\_fourier\_transformer(image, filtered\_gray\_image, filtered\_rgb\_image)

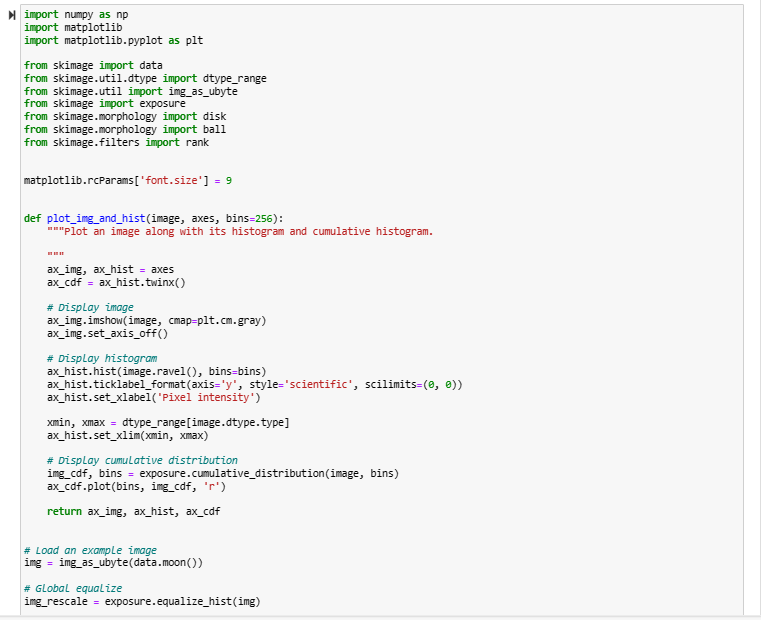




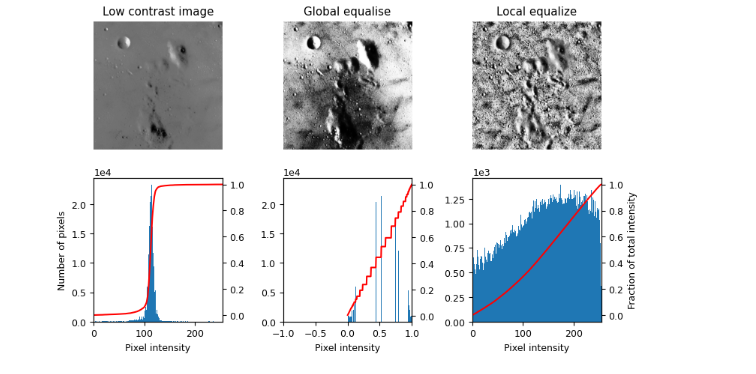
This function plots six images in a 2x3 grid. The top row contains the magnitude spectrum of the grayscale image, the magnitude spectrum of the grayscale image with the filter applied, and the grayscale image itself. The bottom row contains the original image, the filtered grayscale image, and the filtered RGB image.

The filtered image have vertical lines removed and have a smoother appearance, with less high-frequency noise.

Equalization







# References

<https://python.plainenglish.io/introduction-to-image-processing-with-python-18fec2d8dff8>