**Manual:**

1. Applies the 2D Fourier Transform to the grayscale image. Then converts the image from the spatial domain to the frequency domain.

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

1. The output of fft2 places the zero-frequency component at the corners of the image, here we shift it to the center.

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

1. It’s an important step before filtering, because it helps to visualize the frequency content of the image.

magnitude\_spectrum = 20 \* np.log(np.abs(fshift))

1. Create a copy of the image to avoid altering the original image.

Then we modify specific frequency components by setting them to 1 (this is for the vertical axis).

image\_gray\_fft2 = fshift.copy()  
image\_gray\_fft2[:286, fshift.shape[1]//2] = 1  
image\_gray\_fft2[-286:, fshift.shape[1]//2] = 1

1. We undo the shift, apply the Inverse Fourier Transform to return to the spatial domain, and then take the absolute value to get a real filtered image.

# 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)

1. We used the same technique as before, except this time we will do it for the RGB image, we used the same filter also. But we used the for loop so that it can get into each channel of the RGB

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))))

1. In step 6, we did it for each channel individually, now we will combine what we have done so that we create the final filtered image.

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)

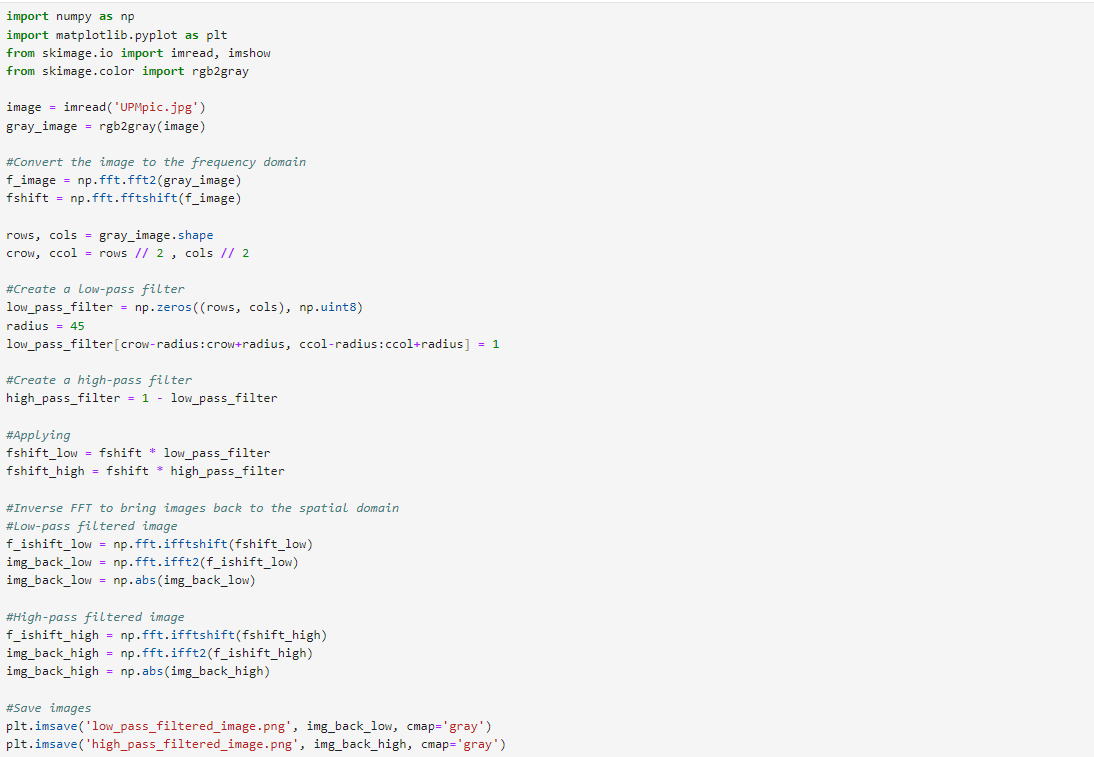
**Assignment:**



The Histogram Equalization using exposure.equalize\_hist, improves the contrast of the image by spreading out pixel intensities, making it easier to see details in both bright and dark areas.

The Gamma Correction using exposure.adjust\_gamma, is applied with a gamma value of 0.6. This adjusts the brightness based on the gamma value.

And If the gamma value is less than 1, it will make the image brighter.



A screenshot of a computer

Description automatically generated

This code enhances an image by using frequency domain filtering.

First, the image is converted to grayscale and then transformed to the frequency domain using FFT.

Two filters are created:

A low-pass filter that keeps low-frequency details and removes high-frequency details.

A high-pass filter that does the opposite, keeping edges and removing smooth areas.

After applying each filter, the code converts the images back to the spatial domain and saves them.

Finally, it displays the original, low-pass filtered, and high-pass filtered images for comparison.

More:

We converted the image from spatial domain to the frequency domain, using (FFT).

Then shift the zero-frequency to the center.

Then we get the rows and columns in the image to calculate the center for the frequency domain image.

Then within the specified radius, is set to 1. It will allows low frequency to get through while block high frequency.

High frequency is basically the opposite of low frequency.

Then we multiply the frequency domain image by the low-pass and high-pass.

Then we take the inverse FFT to convert back the spatial domain and return it to its original position.

Then we saved the images and plot them