IMAGE PROCESSING IN FREQUENCY DOMAIN

1. Write a program to perform Fourier Transform and inverse Fourier Transform of a given image
2. Perform blur detection using Fourier Transform.
3. Apply frequency domain low pass and high pass filter.

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
```

Fourier Transform and inverse Fourier Transform

In this practical exercise, we performed Fourier Transform and inverse Fourier Transform of a given image using the following formulas:

1. Fourier Transform: The Fourier Transform of an image $F(u, v)$ is calculated as:
$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$
2. Inverse Fourier Transform: The inverse Fourier Transform of the transformed image $f(x, y)$ is given by:
$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \cdot e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

By performing these transforms, we can analyze the frequency components of the image and reconstruct the original image from its frequency domain representation.

Blur Detection using Fourier Transform

In this practical exercise, we performed blur detection using Fourier Transform in CVIP. The key steps involved are as follows:

1. Compute the Fourier Transform of the image to obtain its frequency domain representation:
$$F(u,v) = \mathcal{F}\{f(x,y)\}$$
2. Calculate the Power Spectrum of the transformed image to analyze the frequency content:
$$P(u,v) = |F(u,v)|^2$$
3. Apply a blur detection metric to identify the presence of blur in the image. One common metric is the variance of the Power Spectrum:
$$\text{Blur Metric} = \frac{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [(u - M/2)^2 + (v - N/2)^2] \cdot P(u,v)}{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} P(u,v)}$$

By analyzing the blur metric, we can determine the level of blur present in the image. Higher values indicate more blur, while lower values suggest sharper image content. This technique leverages the frequency domain information provided by the Fourier Transform to assess the image's sharpness and clarity.

Frequency Domain Low-Pass and High-Pass Filters

In this practical exercise, we implemented frequency domain low-pass and high-pass filters in CVIP using the Fourier Transform. The key steps involved are as follows:

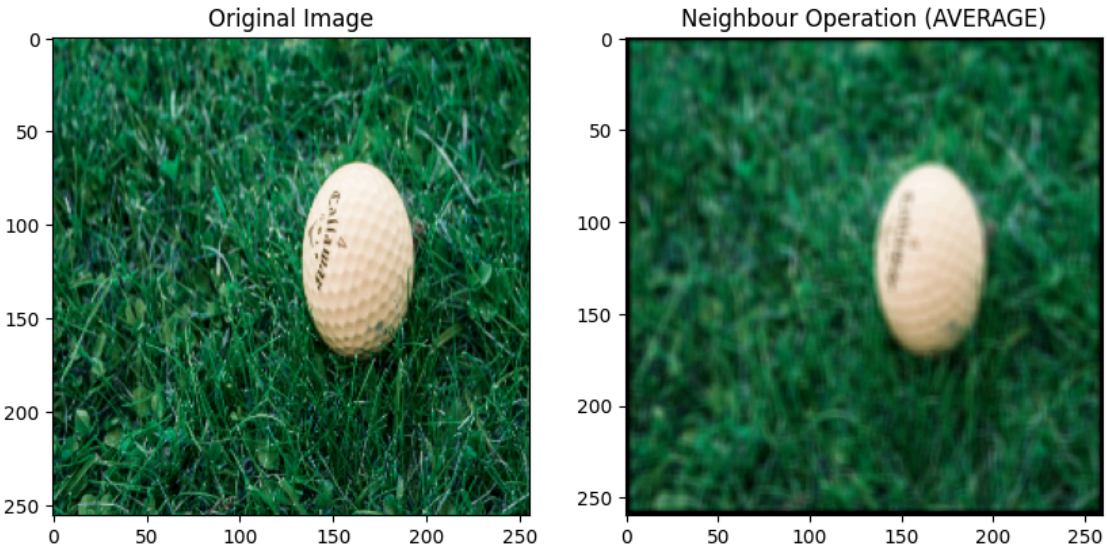
1. Frequency Domain Low-Pass Filter:
    - Compute the Fourier Transform of the image to obtain its frequency domain representation: $F(u,v) = \mathcal{F}\{f(x,y)\}$
    - Apply a low-pass filter mask $H(u,v)$ to attenuate high-frequency components: $$G(u,v) = H(u,v) \cdot F(u,v)$$
    - Perform the inverse Fourier Transform to obtain the filtered image: $g(x,y) = \mathcal{F}^{-1}\{G(u,v)\}$
2. Frequency Domain High-Pass Filter:
    - Compute the Fourier Transform of the image to obtain its frequency domain representation: $F(u,v) = \mathcal{F}\{f(x,y)\}$
    - Apply a high-pass filter mask $H(u,v)$ to attenuate low-frequency components: $$G(u,v) = H(u,v) \cdot F(u,v)$$
    - Perform the inverse Fourier Transform to obtain the filtered image: $g(x,y) = \mathcal{F}^{-1}\{G(u,v)\}$

The filter masks $H(u, v)$ can be designed using various techniques such as ideal, Butterworth, or Gaussian filters based on the desired frequency response. By applying these filters in the frequency domain, we can selectively enhance or suppress specific frequency components in the image to achieve desired filtering effects.

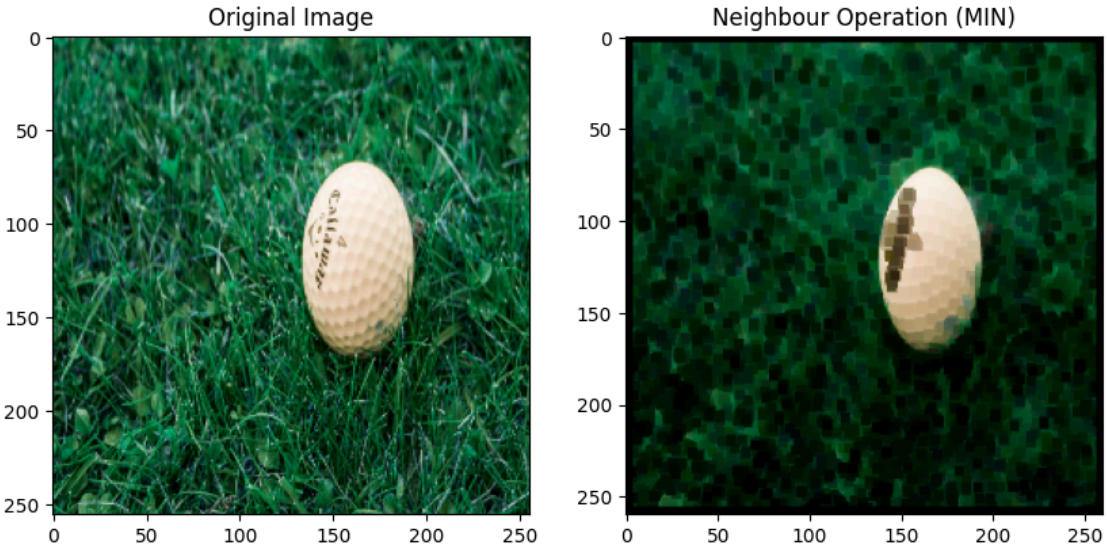
```
img = cv2.resize(cv2.imread('/content/drive/MyDrive/Cvip_Lab/images/golf.png', cv2.IMREAD_COLOR), (256,256))


# Make black border
image = cv2.copyMakeBorder(img, 2, 2, 2, 2, cv2.BORDER_CONSTANT, None)

print('New dimensions', image.shape[0], 'X', image.shape[1])
cv2_imshow(image)
```

```
New dimensions 260 X 260
```

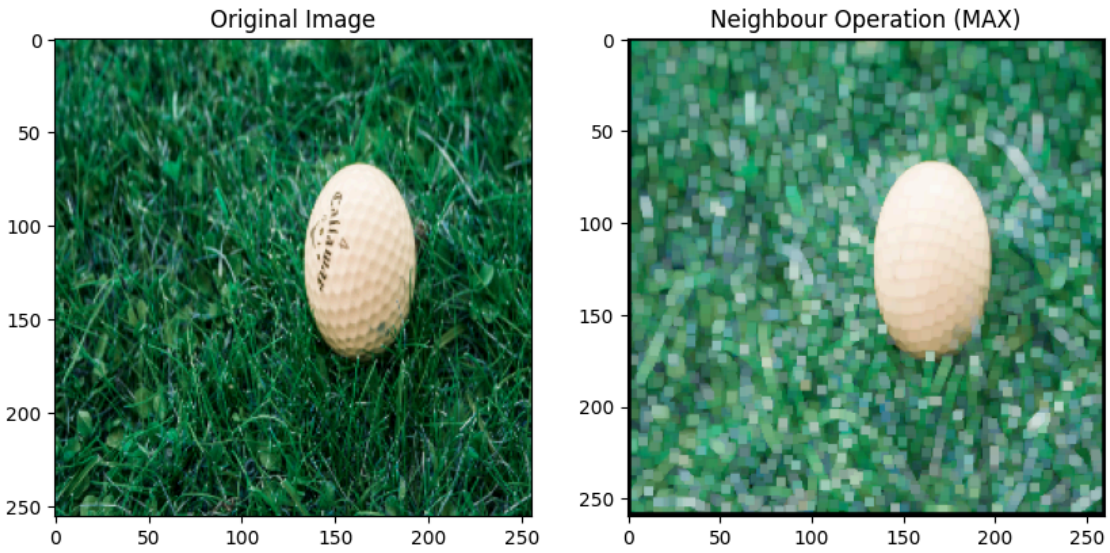
```
# Creating a copy for traversing original values and updating in new matrix
img2avg = image.copy()
for i in range(2,258):
    for j in range(2,258):

        # for a 5x5 window sliding
        vector_sum = 0
        n=0
        # Fetch all neighbours for middle element of a 5x5 matrix
        for k in range(i-2,i+3):
            for l in range(j-2,j+3):
                n+=1
                #Calculate sum of
                vector_sum+= image[k,l].astype(int)
        #Finding average of neigbours, excluding centre element
        vector_mean = (vector_sum - image[i,j]) / (n-1)
        img2avg[i,j] = vector_mean

cv2.imwrite('average_neighbor_operation.jpg', img2avg)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(img2avg)
plt.title('Neighbour Operation (AVERAGE)')

plt.show()
```



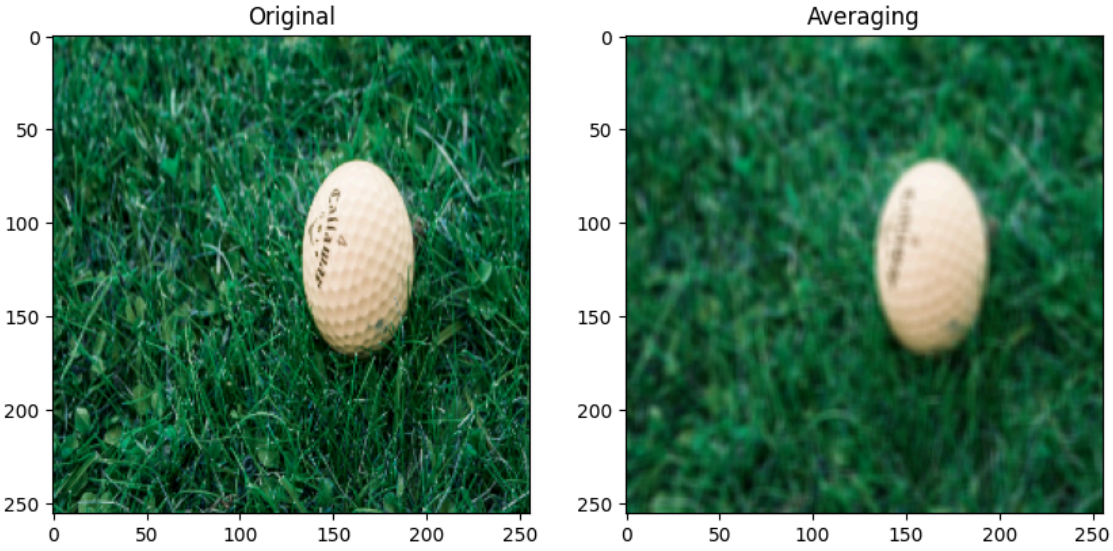

```
# Creating a copy for traversing original values and updating in new matrix
img2min = image.copy()
# Traverse image portions alone
for i in range(2,258):
    for j in range(2,258):
        # for a 5x5 window sliding
        neighbors = []
        # Fetch all neighbours for middle element of a 5x5 matrix
        for k in range(i-2,i+3):
            for l in range(j-2,j+3):
                neighbors.append(image[k,l])
        minimum = np.amin(neighbors, axis = 0)
        #print(minimum)
        img2min[i,j] = minimum

cv2.imwrite('minimum_neighbor_operation.jpg', img2min)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(img2min)
plt.title('Neighbour Operation (MIN)')

plt.show()
```



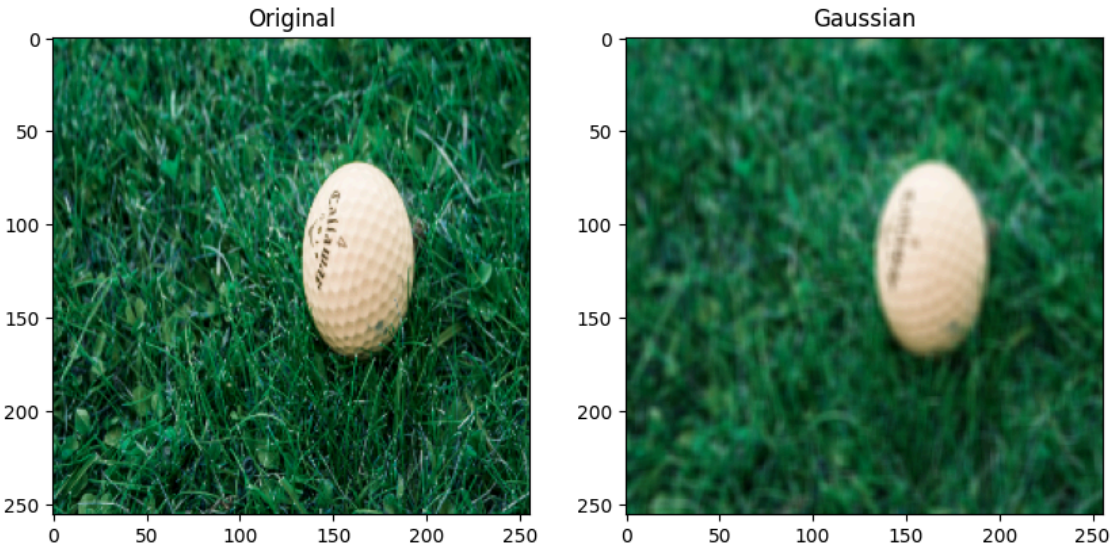
```
# Creating a copy for traversing original values and updating in new matrix
img2max = image.copy()
# Traverse image portions alone
for i in range(2,258):
    for j in range(2,258):
        # for a 5x5 window sliding
        neighbors = []
        # Fetch all neighbours for middle element of a 5x5 matrix
        for k in range(i-2,i+3):
            for l in range(j-2,j+3):
                neighbors.append(image[k,l])
        maximum = np.amax(neighbors, axis = 0)
        #print(maximum)
        img2max[i,j] = maximum

cv2.imwrite('minimum_neighbor_operation.jpg', img2max)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(img2max)
plt.title('Neighbour Operation (MAX)')

plt.show()
```



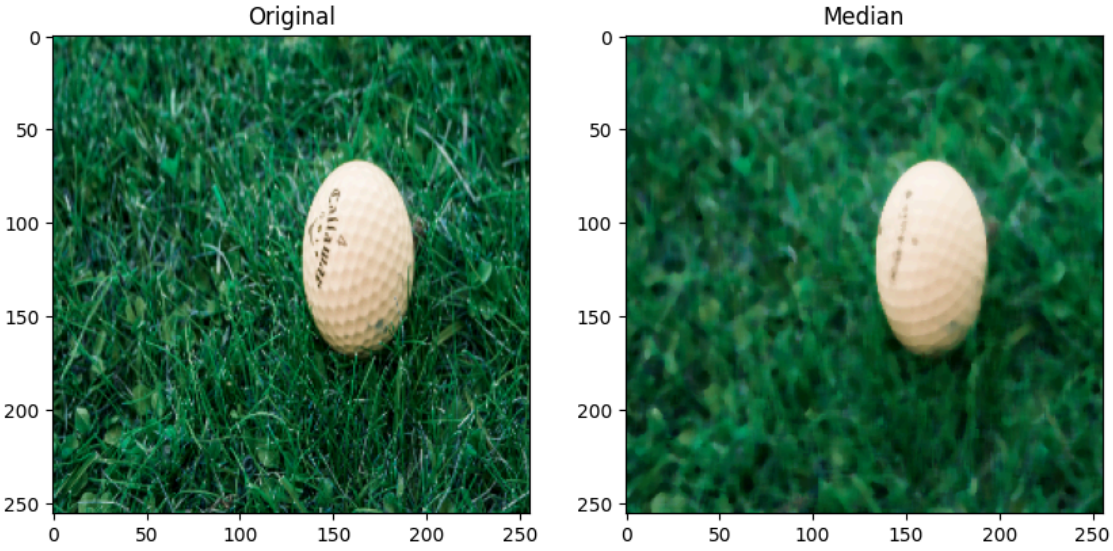

```
IMG = cv2.resize(cv2.imread('/content/drive/MyDrive/Cvip_Lab/images/golf.png', cv2.IMREAD_COLOR), (256,256))

blur1 = cv2.blur(IMG,(5,5))

plt.figure(figsize=(10, 5))

plt.subplot(1,2,1)
plt.imshow(IMG)
plt.title('Original')

plt.subplot(1,2,2)
plt.imshow(blur1)
plt.title('Averaging')

plt.show()
```



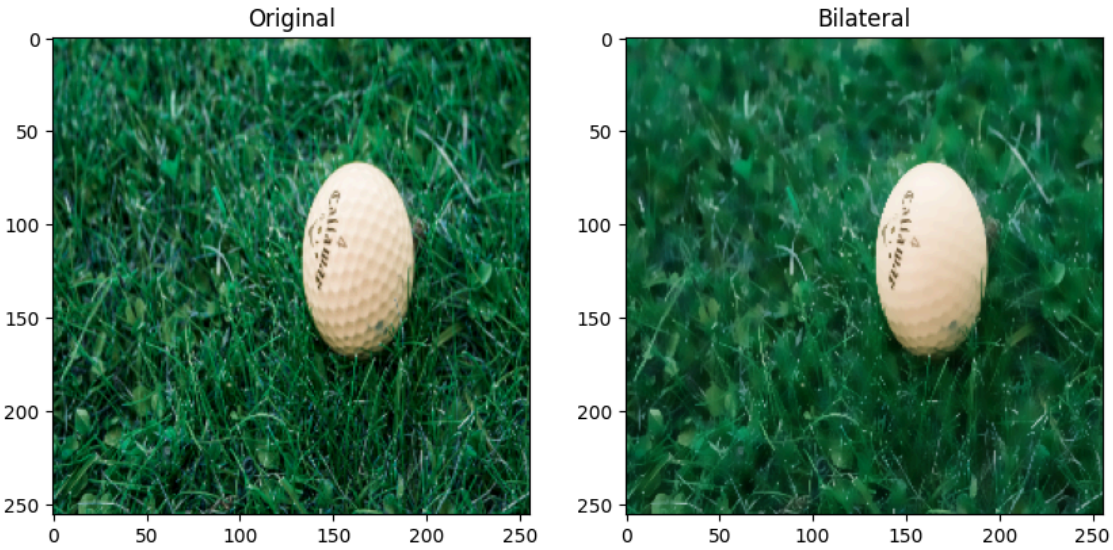

```
blur2 = cv2.blur(IMG,(5,5),0)

plt.figure(figsize=(10, 5))

plt.subplot(1,2,1)
plt.imshow(IMG)
plt.title('Original')

plt.subplot(1,2,2)
plt.imshow(blur2)
plt.title('Gaussian')

plt.show()
```



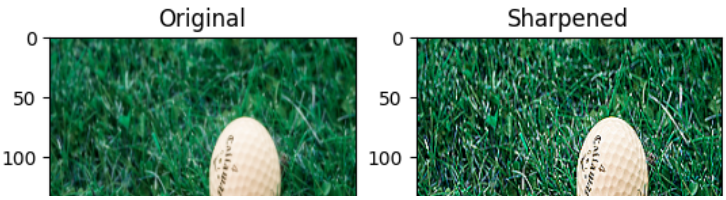

```
blur3 = cv2.medianBlur(IMG,5)

plt.figure(figsize=(10, 5))

plt.subplot(1,2,1)
plt.imshow(IMG)
plt.title('Original')

plt.subplot(1,2,2)
plt.imshow(blur3)
plt.title('Median')

plt.show()
```




```
blur4 = cv2.bilateralFilter(IMG,9,75,75)

plt.figure(figsize=(10, 5))

plt.subplot(1,2,1)
plt.imshow(IMG)
plt.title('Original')

plt.subplot(1,2,2)
plt.imshow(blur4)
plt.title('Bilateral')

plt.show()
```




```
kernel = np.array([[0, -1, 0],
[-1, 5, -1],
[0, -1, 0]])
sharpened = cv2.filter2D(IMG, -1, kernel)

plt.subplot(1,2,1)
plt.imshow(IMG)
plt.title('Original')

plt.subplot(1,2,2)
plt.imshow(sharpened)
plt.title('Sharpened')

plt.show()
```




```
from scipy.ndimage.filters import median_filter

def unsharp(image, sigma, strength):

    # Median filtering
    image_mf = median_filter(image, sigma)

    # Calculate the Laplacian
    lap = cv2.Laplacian(image_mf,cv2.CV_64F)

    # Calculate the sharpened image
    sharp = image-strength*lap

    return sharp

sharp1 = np.zeros_like(img)
for i in range(3):
```