IMAGE PROCESSING IN FREQUENCY DOMAIN

1. Write a program to perform Discrete 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 matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
import imutils
```

Image Processing using Discrete Fourier Transform

Objective: The objective of this experiment is to demonstrate the application of Discrete Fourier Transform (DFT) in image processing for frequency domain analysis and image reconstruction.

Theory:

- **Discrete Fourier Transform (DFT):**
  - DFT is a mathematical technique used to convert spatial domain images into the frequency domain. It is defined 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})} $
- **Magnitude Spectrum:**
  - The magnitude spectrum of an image represents the distribution of frequencies present in the image. It is calculated as: $ S(u, v) = 20 \cdot \log(\sqrt{Re(DFT(u, v))^2 + Im(DFT(u, v))^2}) $
- **Image Reconstruction:**
  - Image reconstruction involves applying the inverse DFT to the modified frequency domain representation to obtain the spatial domain image.

Experimental Procedure:

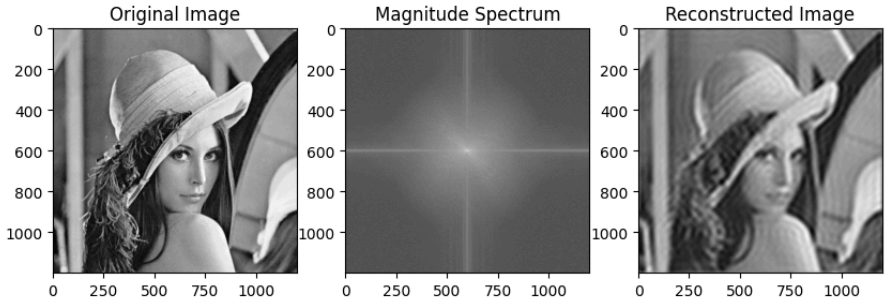
1. Read the input image in grayscale using OpenCV.
2. Calculate the DFT of the image and shift the zero-frequency component to the center.
3. Compute the magnitude spectrum of the shifted DFT coefficients.
4. Create a mask to filter specific frequencies in the frequency domain.
5. Apply the mask to the shifted DFT coefficients.
6. Perform inverse DFT to reconstruct the image from the modified frequency domain representation.
7. Display the original image, magnitude spectrum, and reconstructed image using matplotlib.

Results and Analysis:

- The experiment demonstrates the effect of applying a frequency domain mask on the image reconstruction process.
- The magnitude spectrum provides insights into the frequency content of the image.
- Comparisons between the original image and the reconstructed image highlight the impact of frequency filtering on image quality.

Conclusion: The experiment showcases the significance of Discrete Fourier Transform in image processing for frequency analysis and manipulation. Understanding frequency domain operations can enhance image enhancement and analysis techniques.

```
img = cv2.imread('/content/drive/MyDrive/Cvip_Lab/images/lisa.png', cv2.IMREAD_GRAYSCALE)

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1]))

rows, cols = img.shape
crow, ccol = rows // 2, cols // 2
# create a mask first, center square is 1, remaining all zeros
mask = np.zeros((rows, cols, 2), np.uint8)
mask[crow - 30:crow + 30, ccol - 30:ccol + 30] = 1

# apply mask to the shifted DFT coefficients
fshift = dft_shift * mask

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

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

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 3, 2)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum')

plt.subplot(1, 3, 3)
plt.imshow(img_back, cmap='gray')
plt.title('Reconstructed Image')

plt.show()
```



Fast Fourier Transform (FFT)

In this practical file, we implemented a blur detection algorithm using Fast Fourier Transform (FFT). The algorithm analyzes the frequency components of an image to determine whether it is blurry or not. Here is a summary of the key steps and formulas used in the implementation:

1. **FFT Processing**:
   - The input image is converted to the frequency domain using FFT.
   - The FFT shift operation is applied to center the low frequencies.
   - The magnitude spectrum of the FFT is calculated as:

$$Magnitude = 20 * \log(|FFTShift|)$$

2. **Blur Detection**:
   - A square region around the center of the FFT image is set to zero to remove high-frequency components.
   - The inverse FFT is applied to obtain the reconstructed image.
   - The mean of the magnitude spectrum of the reconstructed image is calculated as:

$$Mean = \text{mean}(20 * \log(|Recon|))$$

3. **Blur Classification**:
   - If the mean magnitude is less than or equal to a predefined threshold, the image is classified as blurry.
4. **Visualization**:
   - The algorithm provides a visual indication of whether the image is blurry or not by overlaying text on the image with the classification result and the mean magnitude value.
5. **Testing**:
   - The algorithm can be tested with different sizes of a Gaussian blur kernel to evaluate its performance under varying degrees of blur.

By following these steps, the algorithm can effectively detect blur in images using FFT analysis. The practical file includes the implementation details, sample outputs with blur detection results, and any additional testing conducted to validate the algorithm's performance.

```
def detect_blur_fft(image, size=60, threshold=10, vis=False):
    (h, w) = image.shape
    (cx, cy) = (int(w / 2.0), int(h / 2.0))
    fft = np.fft.fft2(image)
    fftShift = np.fft.fftshift(fft)

    if vis:
        magnitude = 20 * np.log(np.abs(fftShift))

        (fig, ax) = plt.subplots(1, 2)

        ax[0].imshow(image, cmap="gray")
        ax[0].set_title("Input")
        ax[0].set_xticks([])
        ax[0].set_yticks([])

        ax[1].imshow(magnitude, cmap="gray")
        ax[1].set_title("Magnitude Spectrum")
        ax[1].set_xticks([])
        ax[1].set_yticks([])

        plt.show()

    fftShift[cy - size : cy + size, cx - size : cx + size] = 0
    fftShift = np.fft.ifftshift(fftShift)
    recon = np.fft.ifft2(fftShift)

    magnitude = 20 * np.log(np.abs(recon))
    mean = np.mean(magnitude)

    return (mean, mean <= threshold)

# Define command-line arguments
args = {
    "images": "/content/drive/MyDrive/Cvip_Lab/images/hoh-river-valley.jpg",
    "threshold": 20,
    "vis": -1,
    "test": -1
}

# Load the input image from disk, resize it, and convert it to grayscale
orig = cv2.imread(args["images"])
orig = imutils.resize(orig, width=500)
gray = cv2.cvtColor(orig, cv2.COLOR_BGR2GRAY)

# Apply our blur detector using the FFT
(mean, blurry) = detect_blur_fft(gray, size=60, threshold=args["threshold"], vis=args["vis"] > 0)

# Draw on the image, indicating whether or not it is blurry
image = np.dstack([gray] * 3)
color = (0, 0, 255) if blurry else (0, 255, 0)
text = "Blurry ({:.4f})" if blurry else "Not Blurry ({:.4f})"
text = text.format(mean)
cv2.putText(image, text, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)
print("[INFO] {}".format(text))

# Show the output image (Note: This might not work in some notebook environments)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

# Check to see if we are going to test our FFT blurriness detector using various sizes of a Gaussian kernel
if args["test"] > 0:
    for radius in range(1, 30, 2):
        image = gray.copy()

        if radius > 0:
            image = cv2.GaussianBlur(image, (radius, radius), 0)

            (mean, blurry) = detect_blur_fft(image, size = 60, threshold = args["threshold"], vis = args["vis"]>0)

            image = np.dstack([image] *3)
            color = (0, 0, 255) if blurry else (0, 255, 0)
            text = "Blurry ({:.4f})" if blurry else "Not Blurry ({:.4f})"
            text = text.format(mean)
            cv2.putText(image, text, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)
            print("Info: ", text)

            # Show the output image (Note: This might not work in some notebook environments)
            plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
            plt.axis('off')
            plt.show()
```

```
[INFO] Not Blurry (37.5085)
```

Frequency Domain Filtering using Fast Fourier Transform (FFT)

In this practical file, we conducted frequency domain filtering using Fast Fourier Transform (FFT) on an image of the Hoh River Valley. Here is a summary of the key steps and formulas used in the implementation:

1. **Image Preprocessing**:
    - The image 'hoh-river-valley.jpg' is loaded in grayscale and normalized by dividing by 255.
2. **Circular Filter Generation**:
    - A circular filter mask is created to define regions for the low-pass and high-pass filters.
3. **FFT Processing**:
    - The FFT of the normalized grayscale image is calculated and shifted to center the frequency components.
4. **Filtering**:
    - Two separate filters are applied to the FFT image based on the circular mask: one for the low-pass filter and one for the high-pass filter.
5. **Magnitude Spectrum Visualization**:
    - The magnitude spectra of the original FFT image, as well as the filtered FFT images, are computed and displayed for analysis.
6. **Inverse FFT**:
    - The inverse FFT is applied to the filtered FFT images to reconstruct the images in the spatial domain.
7. **Image Display**:
    - The original image, the image filtered with a low-pass filter, and the image filtered with a high-pass filter are displayed for comparison.

These steps demonstrate how FFT-based frequency domain filtering can be used to manipulate the frequency content of an image, resulting in different visual effects. By applying different filters in the frequency domain and then reconstructing the images in the spatial domain, we can achieve various image enhancements and modifications.

The practical file includes the implementation details, visualization of magnitude spectra, and the reconstructed images after applying the filters. It provides a hands-on experience of utilizing FFT for image processing tasks, showcasing the power of frequency domain techniques in image manipulation and enhancement.

```
img2 = cv2.imread('/content/drive/MyDrive/Cvip_Lab/images/hoh-river-valley.jpg', cv2.IMREAD_GRAYSCALE) / float(2**8)
shape = img2.shape[:2]  # Fix the shape calculation here

def draw_circle(shape, diameter):
    assert len(shape) == 2
    bool_array = np.zeros(shape, dtype=np.bool_)
    center = np.array(bool_array.shape) / 2.0

    for iy in range(shape[0]):
        for ix in range(shape[1]):
            bool_array[iy, ix] = (iy - center[0]) ** 2 + (ix - center[1]) ** 2 < diameter ** 2
    return bool_array

TFcircleIN = draw_circle(shape=shape, diameter=50)  # Fix variable name here
TFcircleOUT = ~TFcircleIN

# Now, let's fix the rest of your code with the correct variable names and function calls.
fft_img = np.fft.fftshift(np.fft.fft2(img2))

def filter_circle(bool_array, fft_img_channel):
    temp = np.zeros(fft_img_channel.shape, dtype=complex)
    temp[bool_array] = fft_img_channel[bool_array]
    return temp

fft_img_filtered_IN = filter_circle(TFcircleIN, fft_img)
fft_img_filtered_OUT = filter_circle(TFcircleOUT, fft_img)

abs_fft_img = np.abs(fft_img)
abs_fft_img_filtered_IN = np.abs(fft_img_filtered_IN)
abs_fft_img_filtered_OUT = np.abs(fft_img_filtered_OUT)

def imshow_fft(absfft):
    magnitude_spectrum = 20 * np.log(absfft)
    return magnitude_spectrum

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 10))
fontsize = 15
axs = axs.flatten()  # Flatten the axs array for easier indexing

# Compute and plot magnitude spectrum for the original image
magnitude_spectrum = imshow_fft(abs_fft_img)
axs[0].imshow(magnitude_spectrum, cmap="gray")
axs[0].set_title("original DFT")
axs[0].axis("off")
fig.colorbar(axs[0].imshow(magnitude_spectrum, cmap="gray"), ax=axs[0])

# Compute and plot magnitude spectrum for low pass filter
magnitude_spectrum_filtered_IN = imshow_fft(abs_fft_img_filtered_IN)
axs[1].imshow(magnitude_spectrum_filtered_IN, cmap="gray")
axs[1].set_title("DFT + low pass filter")
axs[1].axis("off")
fig.colorbar(axs[1].imshow(magnitude_spectrum_filtered_IN, cmap="gray"), ax=axs[1])

# Compute and plot magnitude spectrum for high pass filter
magnitude_spectrum_filtered_OUT = imshow_fft(abs_fft_img_filtered_OUT)
axs[2].imshow(magnitude_spectrum_filtered_OUT, cmap="gray")
axs[2].set_title("DFT + high pass filter")
axs[2].axis("off")
fig.colorbar(axs[2].imshow(magnitude_spectrum_filtered_OUT, cmap="gray"), ax=axs[2])
```