Image Transforms in the Frequency Domain

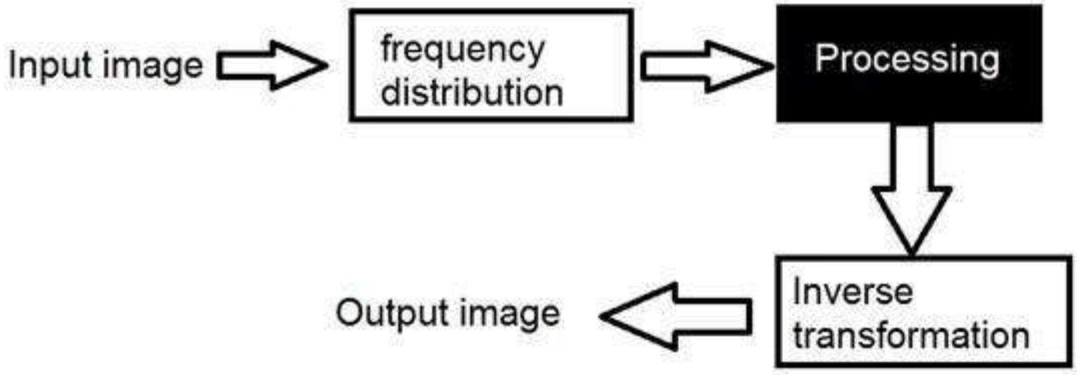
- Image processing often requires converting an image from the spatial domain to the frequency domain to perform various operations such as filtering, compression, and enhancement. This is achieved using different frequency domain transforms. Below is an introduction to some key concepts and techniques related to image transforms

1. Introduction to Frequency Domain Transforms

- **Spatial Domain vs. Frequency Domain:**
  - **Spatial Domain:** Images are represented directly by their pixel values (e.g., intensity values at each coordinate).
  - **Frequency Domain:** Images are represented by their frequency components (sinusoidal waves of varying frequency and amplitude). In this domain, an image is described in terms of its frequencies, which represent how the pixel intensity values change across space.

Frequency Domain

- We first transform the image to its frequency distribution. Then our black box system perform what ever processing it has to performed, and the output of the black box in this case is not an image, but a transformation. After performing inverse transformation, it is converted into an image which is then viewed in spatial domain.



Transformation

- A signal can be converted from time domain into frequency domain using mathematical operators called transforms. There are many kind of transformation that does this. Some of them are given below.
- Fourier Series
- Fourier transformation
- Laplace transform
- Z transform

Frequency components

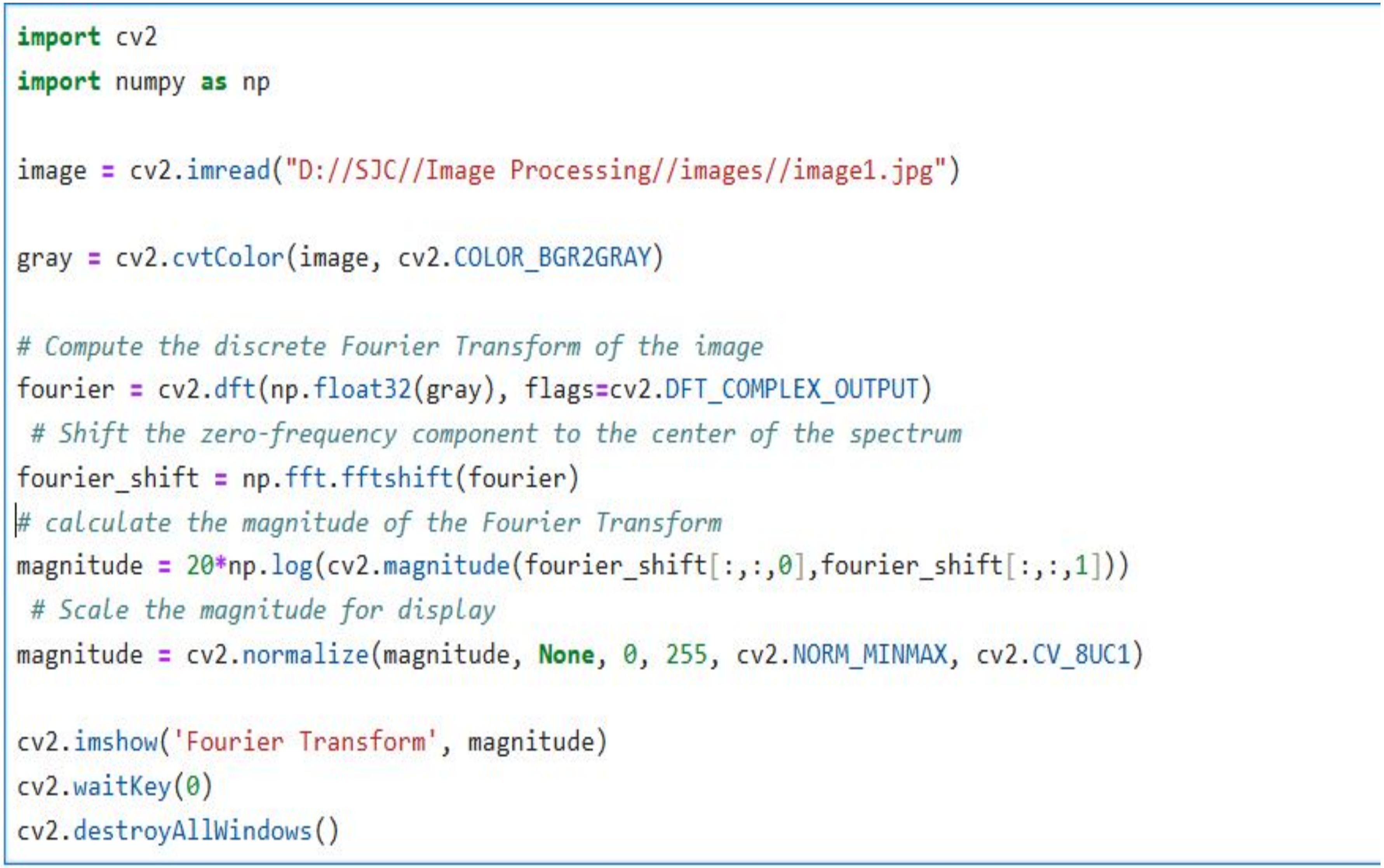
- Any image in spatial domain can be represented in a frequency domain. But what do this frequencies actually mean.

Frequency components into two major components.

- **High frequency components**
- High frequency components correspond to edges in an image.
- **Low frequency components**
- Low frequency components in an image correspond to smooth regions.

Steps to find the Fourier Transform of an image using OpenCV

- **Step 1:** Load the image using the **cv2.imread()** function. This function takes in the path to the image file as an argument and returns the image as a NumPy array.
- **Step 2:** Convert the image to grayscale using the **cv2.cvtColor()** function. This is optional, but it is generally easier to work with grayscale images when performing image processing tasks.
- **Step 3:** Use the **cv2.dft()** function to compute the **discrete Fourier Transform** of the image. This function takes in the image as an argument and returns the Fourier Transform as a NumPy array.
- **Step 4:** Shift the zero-frequency component of the Fourier Transform to the center of the array using the **numpy.fft.fftshift()** function. This step is necessary because the cv2.dft() function returns the Fourier Transform with the zero-frequency component at the top-left corner of the array.
- **Step 5:** Compute the magnitude of the Fourier Transform using the **numpy.abs()** function. This step is optional, but it is generally easier to visualize the frequency content of an image by looking at the magnitude of the Fourier Transform rather than the complex values.
- **Step 6:** Scale the magnitude of the Fourier Transform using the **cv2.normalize()** function. This step is also optional, but it can be useful for improving the contrast of the resulting image.
- **Step 7:** Use the **cv2.imshow()** function to display the magnitude of the Fourier Transform.



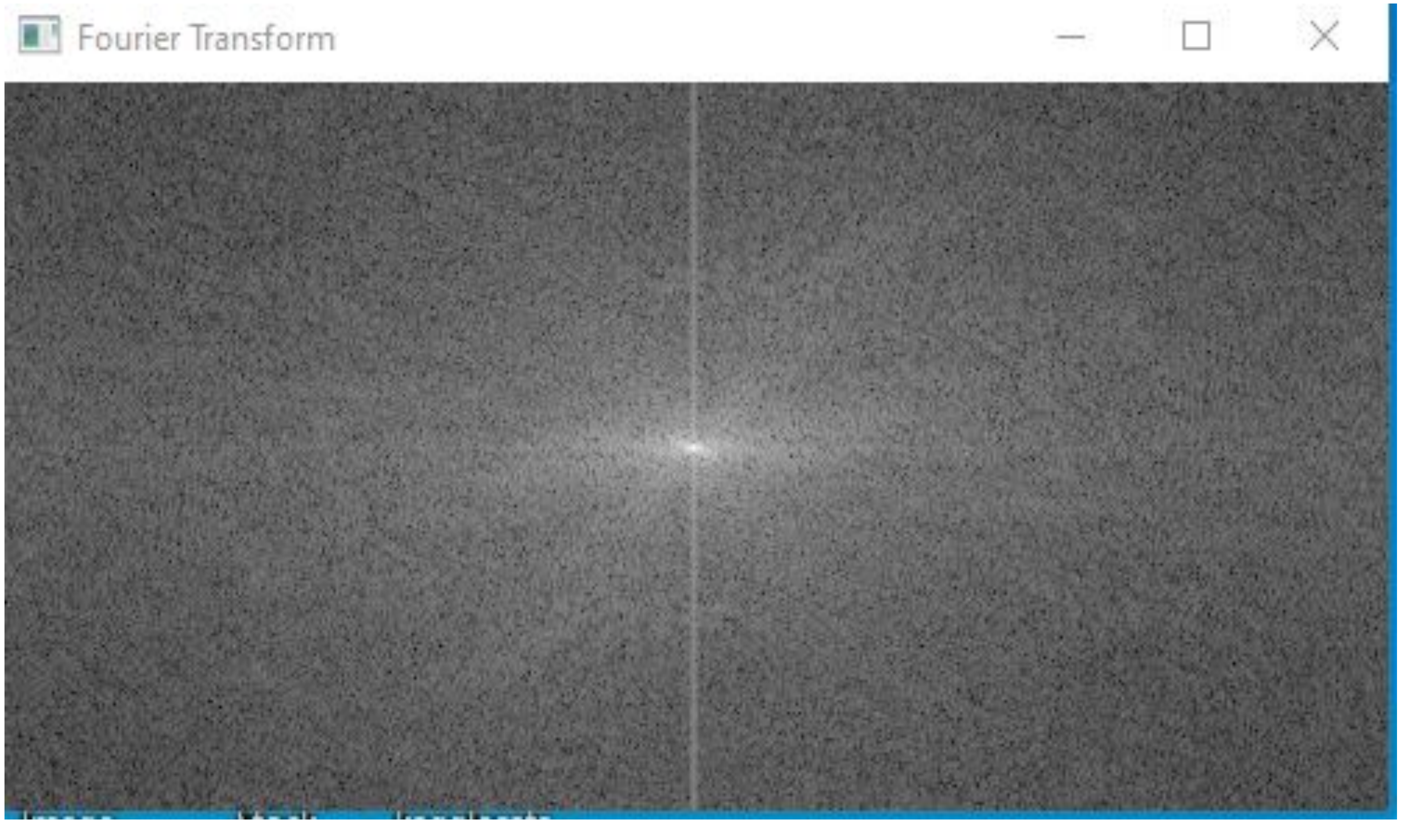
```
import cv2
import numpy as np

image = cv2.imread("D://SJC//Image Processing//images//image1.jpg")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Compute the discrete Fourier Transform of the image
fourier = cv2.dft(np.float32(gray), flags=cv2.DFT_COMPLEX_OUTPUT)
 # Shift the zero-frequency component to the center of the spectrum
fourier_shift = np.fft.fftshift(fourier)
# calculate the magnitude of the Fourier Transform
magnitude = 20*np.log(cv2.magnitude(fourier_shift[:,:,0],fourier_shift[:,:,1]))
 # Scale the magnitude for display
magnitude = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8UC1)

cv2.imshow('Fourier Transform', magnitude)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


Fourier Transform

Why Use Frequency Domain Transforms?

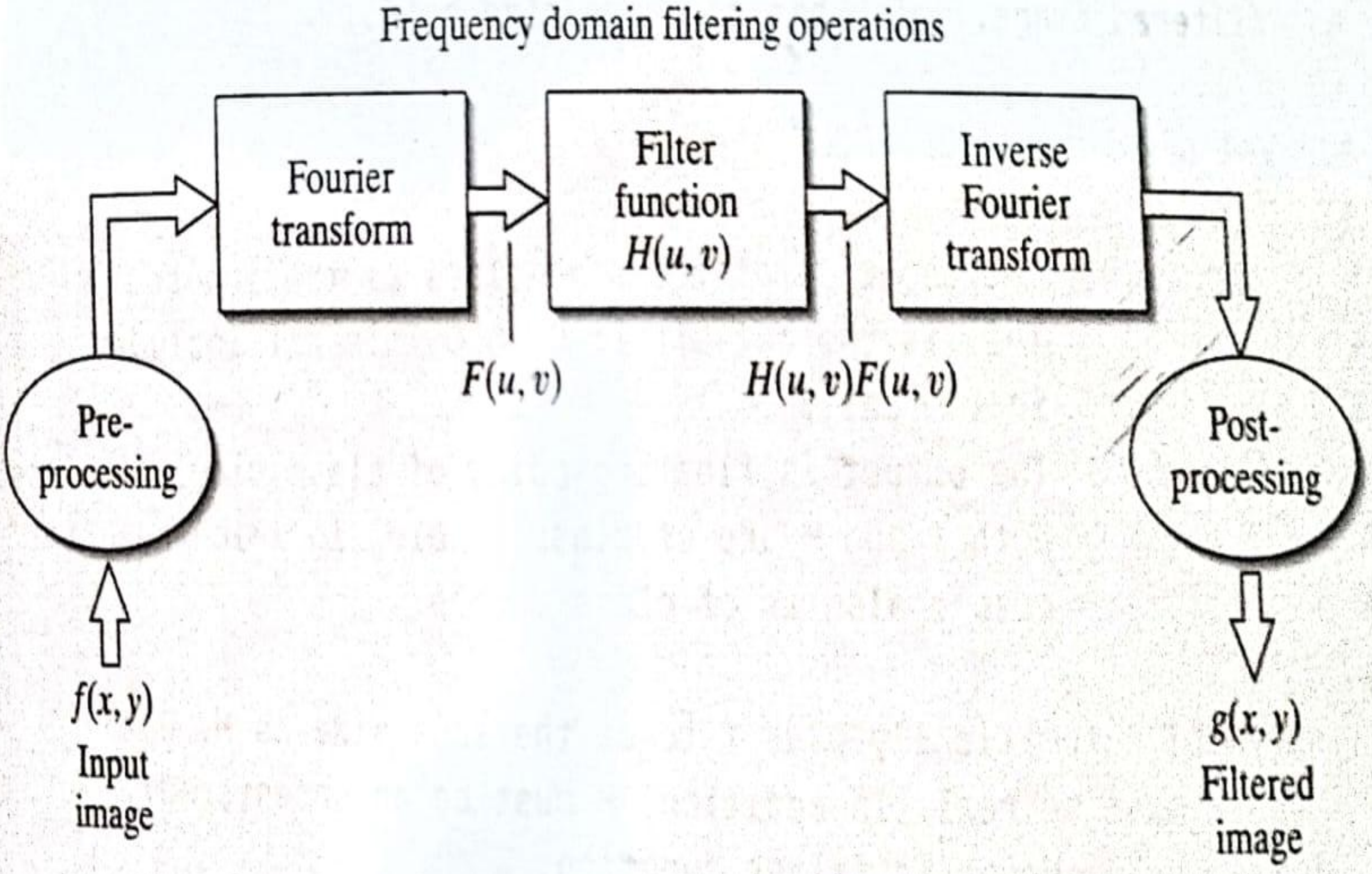
- Frequency domain representations are useful for certain operations like smoothing, edge detection, image compression, and noise filtering. They allow us to separate and manipulate different frequency components in an image.

Key Frequency Domain Transforms

- **Fourier Transform (FT):** Transforms an image into the frequency domain, separating the image into different frequency components.
- **Discrete Cosine Transform (DCT):** Used for image compression, especially in JPEG encoding.
- **Discrete Wavelet Transform (DWT):** Allows multi-resolution analysis, useful in compression and noise removal.

2. Image Representations in Discrete Fourier Transform (DFT)

- The **Discrete Fourier Transform (DFT)** converts an image from the spatial domain to the frequency domain by representing it as a sum of sinusoidal waves with varying frequencies. The DFT is given by the formula:
- The **Magnitude Spectrum** of the DFT shows the strength of different frequencies in the image, while the **Phase Spectrum** shows the spatial information.
- Typically, low frequencies are in the center, and high frequencies are at the edges, unless shifted for visualization.


Frequency domain filtering operations
Fourier transform
Filter function $H(u,v)$
Inverse Fourier transform
$F(u,v)$
$H(u,v)F(u,v)$
Pre-processing
Post-processing
$f(x,y)$
Input image
$g(x,y)$
Filtered image

Discrete Fourier Transform (DFT)

For a 2D signal f(x,y)f(x, y)f(x,y) of size M×NM \times NM×N, the 2D DFT F(u,v)F(u, v)F(u,v) is defined as:

$$X(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot \exp\left(-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

where $f(x,y)$ is the image function in the spatial domain, and $X(u,v)$ is the frequency representation in the DFT domain.

Where:
f(x,y)f(x, y)f(x,y) is the input 2D signal (e.g., an image).
F(u,v)F(u, v)F(u,v) is the output frequency spectrum.
M and N are the dimensions of the signal (height and width).
u and v are the spatial frequency indices.j is the imaginary unit.

Inverse **Discrete Fourier Transform (DFT)**

The inverse 2D Discrete Fourier Transform (IDFT) is given by:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \cdot e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

Applications:

- **Image Compression**: Reducing the size of images by transforming them into the frequency domain and removing less significant frequencies.
- **Image Filtering**: Modifying specific frequency components (e.g., high-pass or low-pass filters).
- **Signal Analysis**: Identifying and analyzing frequency components of signals.

3. Discrete Cosine Transform (DCT)

- The **Discrete Cosine Transform (DCT)** is widely used in image compression techniques, especially JPEG encoding. It represents an image in terms of cosine functions with different frequencies.
- DCT helps separate an image into parts of differing importance based on frequency. High-frequency components usually carry details (such as edges), while low-frequency components carry smoother areas..
- The **Discrete Cosine Transform (DCT)** is a mathematical transformation similar to the Fourier Transform, but it uses only real numbers instead of complex exponentials. It is widely used in signal processing, especially in image compression algorithms such as JPEG, as well as in video compression (like MPEG).

Discrete Cosine Transform (DCT)

The formula for the 2D DCT is:

$$X(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot \cos\left[\frac{\pi(2x+1)u}{2M}\right] \cdot \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

Where:
f(x,y) is the input 2D signal (like an image).
F(u,v) is the 2D DCT transformed output.
M and N are the dimensions of the signal.

Applications:

- **Image Compression (JPEG)**: The JPEG image compression standard uses the 2D DCT to convert image blocks into the frequency domain, and then it quantizes and compresses the coefficients.
- **Video Compression (MPEG)**: Similar to JPEG, but applied to video frames.
- **Audio Compression (MP3, AAC)**: Used in audio compression to remove redundancies.
- **Signal Processing**: In general, the DCT is used in various applications that require efficient frequency domain analysis.

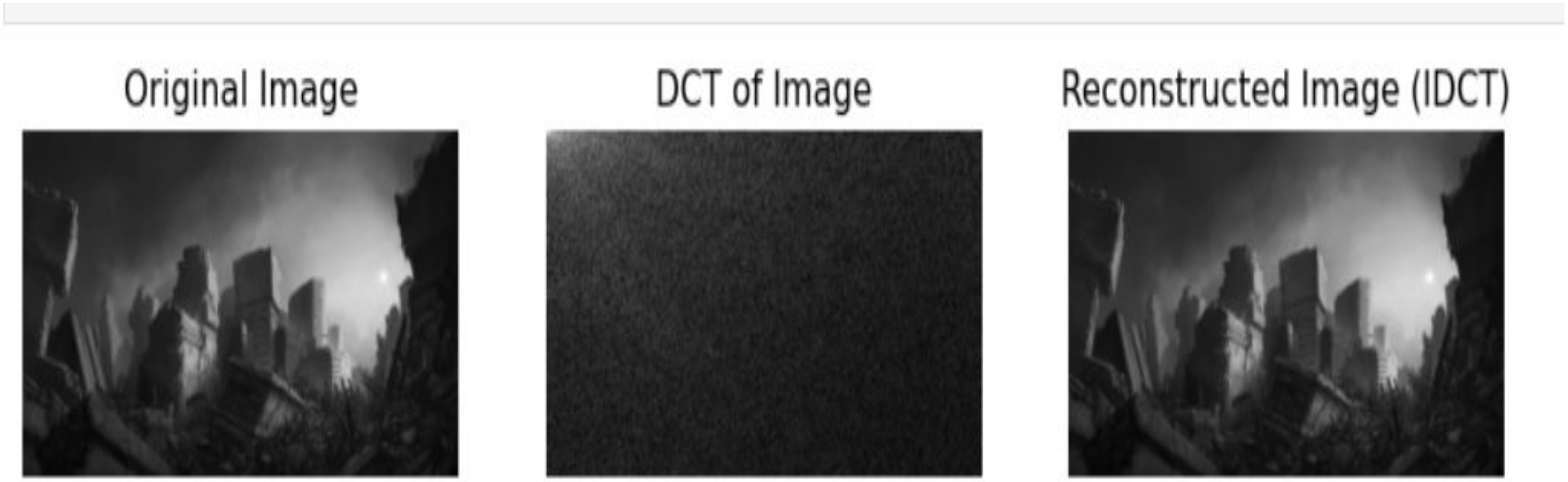
Breakdown of the Code:

- **Load Image:**
  - We read an image using OpenCV in grayscale mode (cv2.IMREAD_GRAYSCALE).
- **Apply 2D DCT:**
  - OpenCV provides the function cv2.dct(), which performs the Discrete Cosine Transform on the image (or any 2D array). Note that the image must be in the float32 format for DCT.
- **Apply Inverse DCT:**
  - We use cv2.idct() to apply the Inverse DCT to reconstruct the image from its DCT coefficients.
- **Visualization:**
  - We display three images:
    - The original image.
    - The DCT transformed image (using a logarithmic scale to better visualize the low-frequency components).
    - The reconstructed image after applying IDCT to the DCT image.

```
image = cv2.imread("D://SJC//Image Processing//images//image1.jpg", cv2.IMREAD_GRAYSCALE)  # Read the image in grayscale
image_float = np.float32(image)                   # OpenCV DCT requires the input to be in float32 and preferably normalized.
dct_image = cv2.dct(image_float)                 # Apply 2D DCT
idct_image = cv2.idct(dct_image)         # Step 3: Apply Inverse DCT to reconstruct the image

plt.figure(figsize=(10, 5))              # Step 4: Display the original, DCT transformed, and reconstructed images
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.imshow(np.log(np.abs(dct_image) + 1), cmap='gray')  # Apply log for better visibility of low frequencies
plt.title('DCT of Image')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.imshow(idct_image, cmap='gray')
plt.title('Reconstructed Image (IDCT)')
plt.axis('off')
plt.show()
```

DCT output-



5. Image Smoothing and Sharpening Using Frequency Domain Filters

- In the frequency domain, **filters** are used to enhance or suppress certain frequency components of an image, which can smooth or sharpen the image.

Smoothing (Low-Pass Filtering)

- **Smoothing filters** reduce high-frequency noise and fine details, effectively blurring the image. In the frequency domain, low-pass filters allow low frequencies to pass and suppress high frequencies.
- **Types of Low-Pass Filters:**
  - **Ideal Low-Pass Filter (ILPF):** A sharp cutoff filter that passes all frequencies below a certain threshold and rejects all higher frequencies. However, it has a non-smooth transition, leading to artifacts like ringing.
  - **Butterworth Low-Pass Filter:** A smoother version of the ideal filter with a more gradual transition, resulting in less ringing.
  - **Gaussian Low-Pass Filter:** The smoothest filter, with a Gaussian function applied to gradually reduce high-frequency components.

Sharpening (High-Pass Filtering)

- **Sharpening filters** emphasize high-frequency components (edges and fine details) while suppressing low-frequency components.
- **Types of High-Pass Filters:**
  - **Ideal High-Pass Filter (IHPF):** This filter removes low-frequency components and keeps high-frequency components, which enhances the sharpness of edges. Similar to low-pass filters, the sharp cutoff can cause ringing.
  - **Butterworth High-Pass Filter:** A smoother high-pass filter that reduces ringing by applying a gradual cutoff.
  - **Gaussian High-Pass Filter:** A high-pass filter that is smooth and helps retain edges without sharp artifacts.

6. Filter Design:

- **Ideal Filters:**
  - An ideal filter is theoretical, providing a perfect cutoff between low and high frequencies. However, due to its non-smooth nature, it leads to artifacts in the image when used in practice.
- **Butterworth Filters:**
  - These filters provide a smooth cutoff between the low and high frequencies. They are more practical than ideal filters and are less likely to cause ringing.
- **Gaussian Filters:**
  - These filters provide a smooth transition between the frequencies and are especially useful in reducing noise with minimal distortion in the image.

1. Spatial Domain Filters

- In the **spatial domain**, filters work directly on pixel values, modifying them based on their neighbors. This is often achieved through convolution, where the image is convolved with a kernel (a small matrix). The most common spatial filters are **low-pass** (smoothing) and **high-pass** (sharpening) filters.

Low-Pass Filters (Smoothing Filters):

- **Purpose:** These filters remove high-frequency components (such as noise and fine details) and emphasize low-frequency components (such as smooth areas). They are often used for noise reduction, image smoothing, and blurring.
- **Example Filters:**
  - **Box Filter (Mean Filter):** A simple filter that averages the pixel values within a kernel window. It is easy to compute but can blur edges.
  - **Gaussian Filter:** This filter applies a Gaussian function to weight pixels based on their distance from the center, providing a smooth and natural blur effect.
  - **Median Filter:** A non-linear filter that replaces each pixel value with the median value of its neighbors, which is effective in removing salt-and-pepper noise.

High-Pass Filters (Sharpening Filters):

- **Purpose:** These filters enhance high-frequency components (such as edges and fine details) and suppress low-frequency components. They are useful for image sharpening and edge detection.
- **Example Filters:**
  - **Laplacian Filter:** Highlights areas of rapid intensity change, which are typically edges in an image.
  - **Sobel Filter:** A gradient filter that detects edges by computing the gradient of the image intensity.
  - **Unsharp Masking:** A technique that enhances edges by subtracting a blurred version of the image from the original image.

2. Frequency Domain Filters

- In the **frequency domain**, images are transformed using techniques such as the **Discrete Fourier Transform (DFT)**, **Discrete Cosine Transform (DCT)**, or **Wavelet Transform (DWT)**. Once in the frequency domain, filters can be designed to manipulate the image’s frequency components, such as suppressing high frequencies (low-pass filtering) or emphasizing high frequencies (high-pass filtering).

Low-Pass Filters:

- **Purpose:** These filters allow low-frequency components to pass through while attenuating high-frequency components, which are typically associated with fine details and noise.
- **Types of Low-Pass Filters:**
  - **Ideal Low-Pass Filter (ILPF):** A filter that perfectly passes frequencies below a certain cutoff and blocks all higher frequencies. While this gives a perfect cutoff, it causes significant ringing artifacts in the spatial domain.
  - **Butterworth Low-Pass Filter:** A smoother version of the ideal filter. It has a gradual transition between passed and blocked frequencies and doesn't introduce ringing artifacts.
  - **Gaussian Low-Pass Filter:** A filter that uses a Gaussian function to smoothly attenuate high-frequency components. It is the most natural and widely used filter for blurring and smoothing in the frequency domain.
- **Applications:** Noise reduction, image smoothing, and blur

- **High-Pass Filters:**
- **Purpose:** High-pass filters allow high-frequency components to pass while blocking low-frequency components. They emphasize edges and fine details, making them ideal for image sharpening or edge detection.
- **Types of High-Pass Filters:**
  - **Ideal High-Pass Filter (IHPF):** This filter allows high frequencies to pass through while blocking low frequencies. It has a sharp cutoff, similar to the ideal low-pass filter, but in the opposite direction. It often introduces artifacts.
  - **Butterworth High-Pass Filter:** Similar to the Butterworth low-pass filter, this filter has a smoother transition and reduces ringing artifacts.
  - **Gaussian High-Pass Filter:** A smoother high-pass filter that is ideal for edge enhancement without sharp cutoff artifacts.
- **Applications:** Image sharpening, edge detection, and emphasizing fine details.

Band-Pass and Band-Stop Filters:

- **Band-Pass Filter:** A filter that allows frequencies within a certain range (band) to pass through and attenuates frequencies outside that range. This can be used for extracting certain frequency bands from an image.
- **Band-Stop Filter:** A filter that attenuates a certain range of frequencies while allowing others to pass. It is used to suppress certain frequencies, such as periodic noise.

3. **Filter Design Techniques**

- Designing filters in image processing often involves selecting the appropriate type of filter (e.g., low-pass, high-pass) and determining its properties (e.g., cutoff frequency, filter size). There are various techniques to design these filters:

1. Ideal Filter Design:

- **Ideal Low-Pass Filter (ILPF):** Involves setting a cutoff frequency D0D_0D0, where all frequencies less than D0D_0D0 pass through and all others are blocked. In practice, this filter is not used due to its sharp cutoff, which leads to significant ringing artifacts in the spatial domain.
- **Ideal High-Pass Filter (IHPF):** Similarly, this filter passes high frequencies above a certain cutoff and blocks low frequencies. It is rarely used due to similar artifacts.

2. Butterworth Filter Design:

- The **Butterworth filter** has a more gradual cutoff compared to the ideal filter. It is designed to minimize distortion and ringing artifacts, making it more practical for real-world applications.

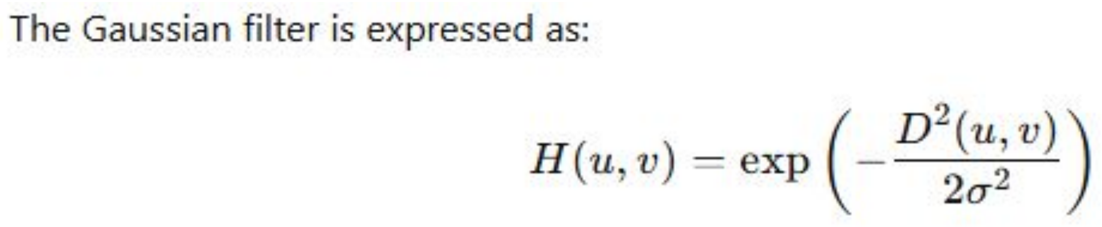
- The transfer function for a Butterworth filter is given by:

$$H(u,v) = \frac{1}{1 + (\frac{D(u,v)}{D_0})^{2n}}$$

- \
frequency domain, D0 is the cutoff frequency, and n is the order of the filter. Higher values of n make the transition sharper.

3. Gaussian Filter Design:

- The **Gaussian filter** is designed using the Gaussian function, which provides a smooth transition between passed and blocked frequencies, making it ideal for many image processing applications.



The Gaussian filter is expressed as:

$$H(u,v) = \exp\left(-\frac{D^2(u,v)}{2\sigma^2}\right)$$

4. Practical Considerations in Filter Design

- **Filter Size:** The size of the filter (e.g., 3x3, 5x5) affects both performance and computational complexity. Larger filters can smooth more effectively but increase computational cost.
- **Cutoff Frequency:** The cutoff frequency (for low-pass or high-pass filters) determines which frequencies are preserved or attenuated. This is a critical parameter when designing a filter for a specific task, such as noise reduction or edge enhancement.
- **Edge Effects:** Filters can introduce artifacts along image borders, especially when using larger kernels. Techniques such as zero-padding or wrapping may be used to mitigate these effects.
- **Computational Efficiency:** Frequency domain filtering can often be more efficient than spatial domain filtering, especially for large images, because of the use of the Fast Fourier Transform (FFT).

4. Discrete Wavelet Transform (DWT)

- **Wavelet Transforms** decompose an image into multi-resolution representations. Unlike the Fourier transform, which only provides frequency components, wavelets allow both frequency and spatial localization.
- The **Discrete Wavelet Transform (DWT)** decomposes an image into a set of coefficients that represent different frequency bands (low, high) at multiple resolutions.
- The most common wavelets used are **Haar wavelets** and **Daubechies wavelets**. These can capture details like edges and textures better than DFT, and are especially useful in image compression (e.g., JPEG2000).