IMAGE BASICS AND FFTS

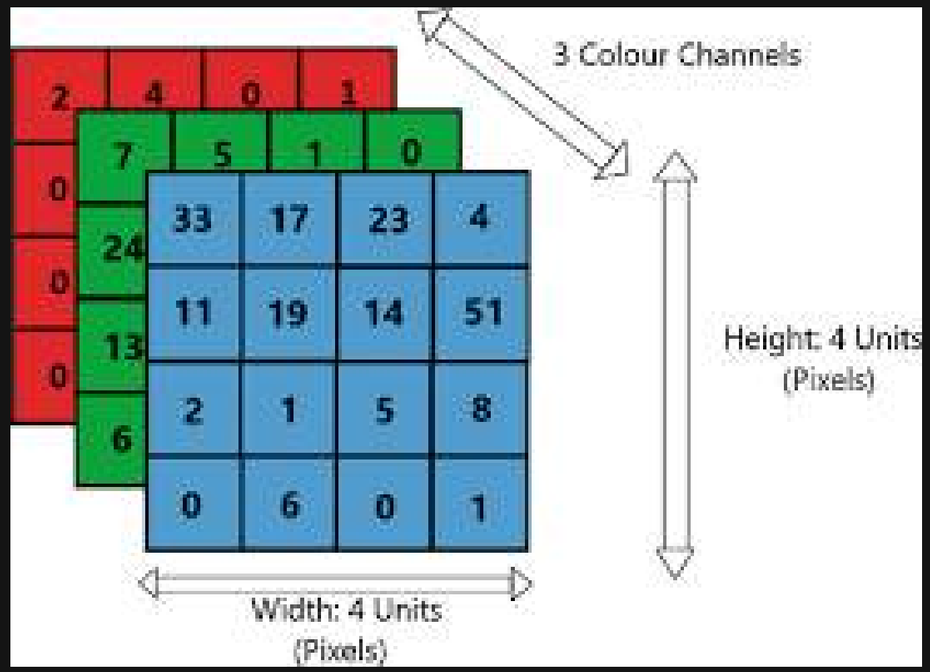
WHAT IS IMAGE

An **image** is a visual representation of something.
It can show real things, like people or objects, or it can be made-up, like drawings or graphics.

THIS IS **JUNK**

IMAGES = MATRICES

- A color image = H × W × 3 matrix
- Each pixel = (R, G, B)
- Each value = 0 to 255 (uint8)
- For math operations: convert to float32



CODE - LOAD IMAGE

LIBRARIES :

- opencv-python
- numpy
- matplotlib

Use Jupyter Notebook
Best - Google Colab

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img_bgr = cv2.imread("image.png")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(6,6))
plt.imshow(img_rgb)
plt.title("Original Image (RGB)")
plt.axis("off")
plt.show()
```

KEY IDEA

Different libraries use different channel orders.

Always convert BGR → RGB for correct colors.

GRAYSCALE

- Many CV pipelines use grayscale for simplicity
- FFT visualizations are easier in grayscale
- Edge detection often begins with grayscale

Average Method:

$$I_{GRAY} = \frac{R + G + B}{3}$$

Weighted Method:

$$I_{GRAY} = [0.3(R) + 0.59(G) + 0.11(B)]$$

Gray = 0.299R+0.587G+0.114B

```python
# Convert to Grayscale
gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)

plt.figure(figsize=(6,6))
plt.imshow(gray, cmap='gray')
plt.title("Grayscale")
plt.axis("off")
plt.show()
```

RED		
56	53	54
63	62	64
52	57	61

GREEN		
61	58	57
69	68	67
59	64	68

BLUE		
54	51	50
59	58	58
43	48	52

GRAY		
59	56	55
66	65	65
55	60	64

FREQUENCY DOMAIN

Fourier Series : https://www.youtube.com/watch?v=r6sGWTCMz2k

Fourier Tranform : https://www.youtube.com/watch?v=spUNpyF58BY

DFT/FFT : https://www.youtube.com/watch?v=QmgJmh2I3Fw

Convolution : https://www.youtube.com/watch?v=KuXjwB4LzSA

Additionals :

Does math in front of you : https://www.youtube.com/watch?v=2oEt5lbsyhM
More into FFT algo : https://www.youtube.com/watch?v=h7apO7q16V0

KEY POINTS

Fourier Transform (FT)
Breaks an image into a sum of sinusoids of varying frequencies.
Produces:

- Magnitude → how strong a frequency is
- Phase → where that frequency occurs

Phase preserves structure; magnitude preserves texture.

DFT (Discrete Fourier Transform)

- Mathematical definition
- $O(N^2)$ complexity
- Very slow for large images

FFT (Fast Fourier Transform)

- Optimized algorithm
- O(N log N)

Low Frequency =

- Smooth regions
- Gradients
- Background
- Soft color transitions

High Frequency =

- Edges
- Textures
- Noise
- Fine details

CODE - FFTs & CONV

```python
# Convert to float for math
gray_float = gray.astype(np.float32)

# FFT
f = np.fft.fft2(gray_float)

# Shift low frequencies to center
f_shift = np.fft.fftshift(f)
```

The low frequencies naturally appear at the corners.
So shift them to the center for visualization.

```python
f_lpf = f_shift * mask_lpf
```

CONVOLUTION

Scan karlo !!

https://forms.gle/EsEbrNRYtBh8rT4B7

MAGNITUDE & PHASE

Magnitude Spectrum

Shows where most energy is located.
Bright center = strong low frequencies
Patterns in edges = textures & structure

PHASE > MAGNITUDE
Phase alone can reconstruct useful image structures.

$$M_{ag} = \log(|F_{(u,v)}| + 1)$$

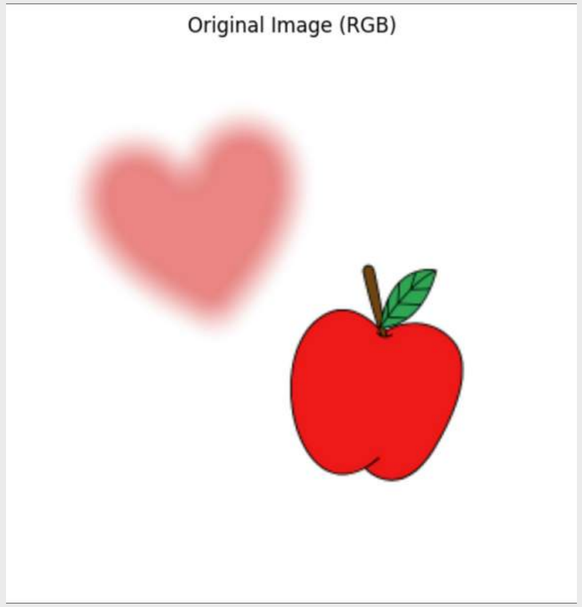
```
# Magnitude Spectrum
magnitude = 20 * np.log(np.abs(f_shift) + 1)
```

```
# Phase Spectrum
phase = np.angle(f_shift)
```

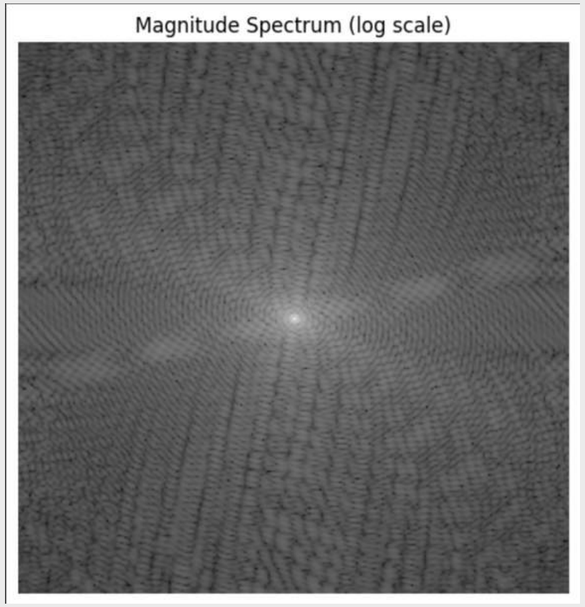
Phase Spectrum

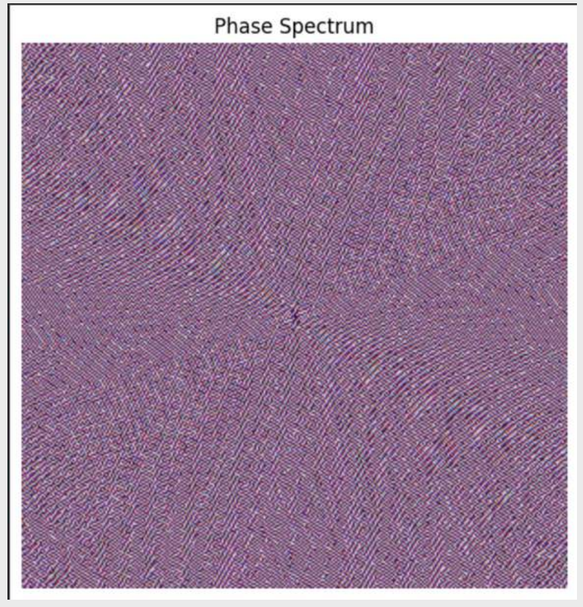
Contains spatial structure
Edges, outlines, shapes

AN EXAMPLE


Original Image (RGB)


Grayscale


Magnitude Spectrum (log scale)


Phase Spectrum

FILTERS

Low Pass Filter:

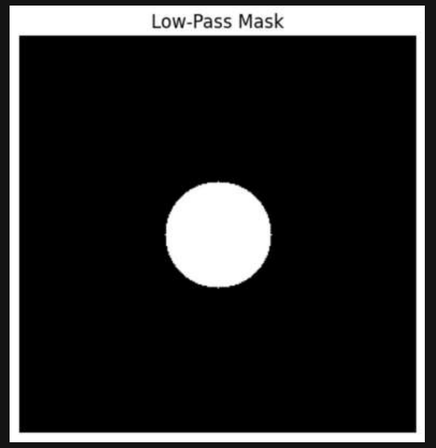
Keeps low-frequency components, removes high-frequency.

Effect:

- Blurring
- Smooths details
- Removes noise

Mask Look:

- White center
- Black outer region



Result:

Image becomes smooth & blurry.

High Pass Filter:

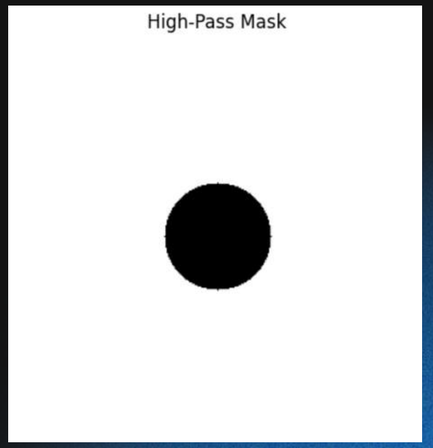
Keeps high-frequency components, removes low-frequency.

Effect:

- Sharpening
- Edge enhancement
- Detail extraction

Mask Look:

- Black center
- White outer region



Result:

Image highlights edges & details.

USING MASKS

CREATING MASKS :

```
# Low Pass Mask
rows, cols = gray.shape
crow, ccol = rows//2, cols//2

# Create LPF mask
radius = 40  # tune radius for blur level

mask_lpf = np.zeros((rows, cols), np.uint8)
cv2.circle(mask_lpf, (ccol, crow), radius, 1, -1)
```

Low-pass mask:

- 1 inside circle
- 0 outside circle

High-pass mask:

```
mask_hpf = 1 - mask_lpf
```

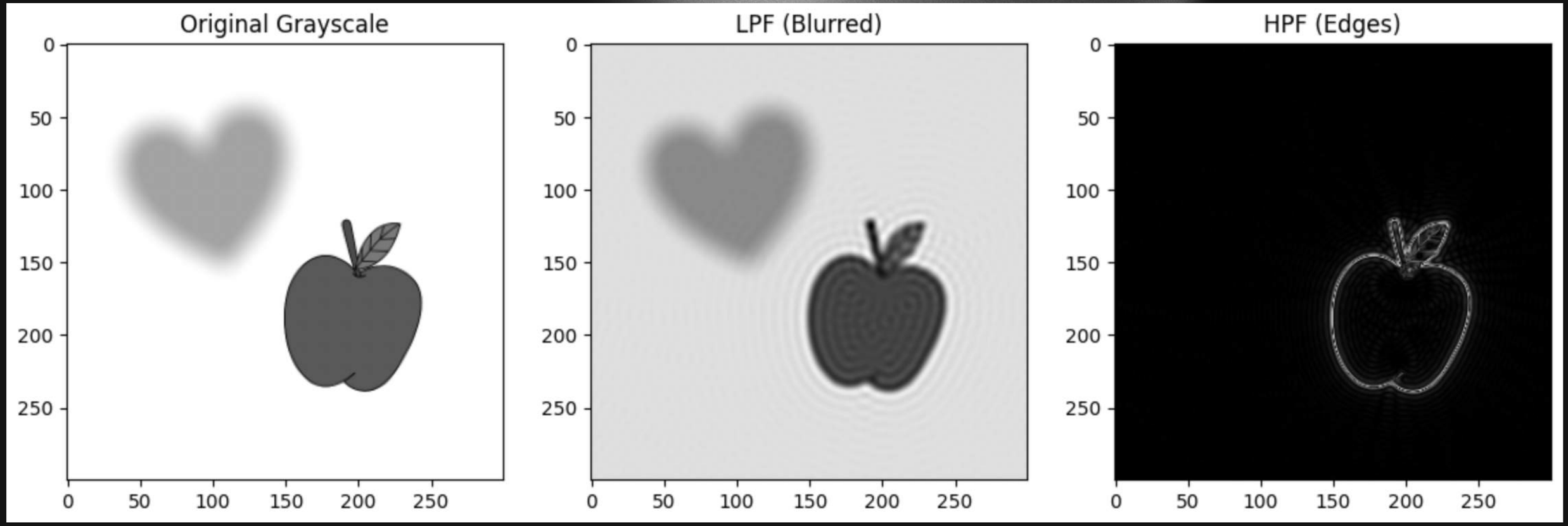
APPLYING MASKS :

- Compute FFT
- Shift low frequencies
- Apply mask
- Inverse shift
- Inverse FFT
- Take absolute value

```
f_lpf = f_shift * mask_lpf
f_hpf = f_shift * mask_hpf
```

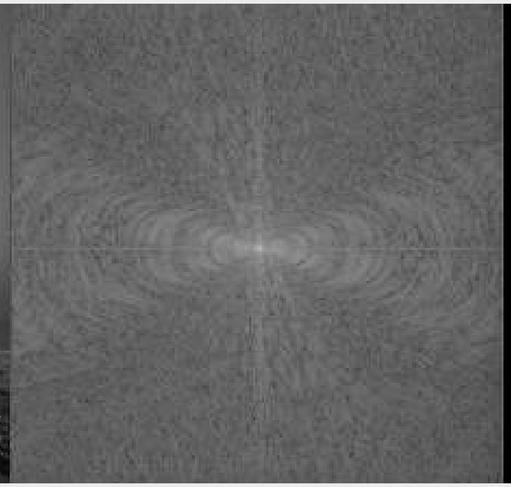
```
# Inverse FFT for Low pass
ishift_lpf = np.fft.ifftshift(f_lpf)
img_back_lpf = np.fft.ifft2(ishift_lpf)
img_back_lpf = np.abs(img_back_lpf)
# Inverse FFT for High pass
ishift_hpf = np.fft.ifftshift(f_hpf)
img_back_hpf = np.fft.ifft2(ishift_hpf)
img_back_hpf = np.abs(img_back_hpf)
```

RESULTS

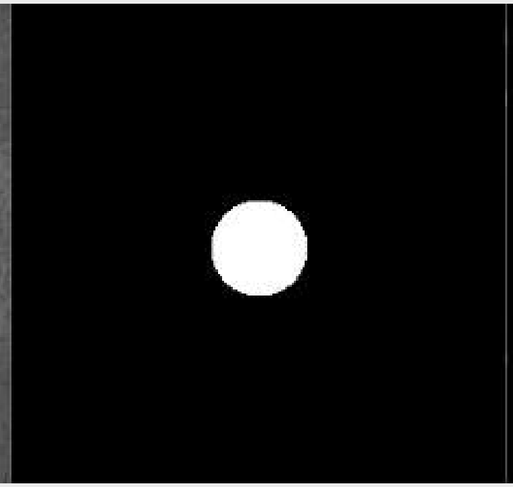

Original Grayscale
LPF (Blurred)
HPF (Edges)
0
50
100
150
200
250

ANOTHER EXAMPLE

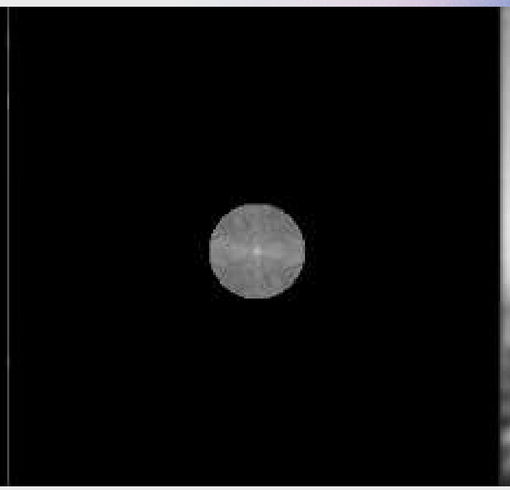
IMAGE



MAGNITUDE SPECTRUM

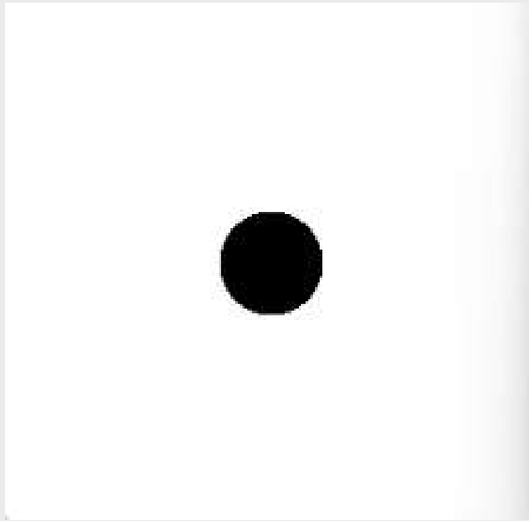


LPF

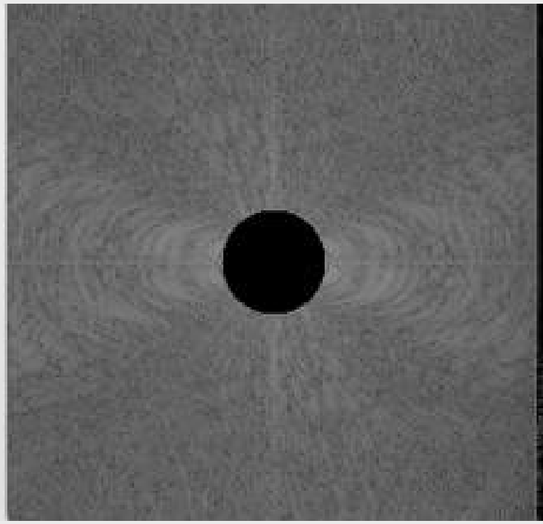


LPF OUTPUT

INVERSED LPF OUTPUT



HPF



HPF OUTPUT

INVERSED HPF OUTPUT

IMPORTANT STUFF

The key to understanding CV and developing an intuition is to experiment with images until you get the perfect result. Don’t just read, do your own stuff.

IT MAY BE HARD, BUT YOU CAN BE HARDER

BYEEE :))