BORING MATHS
BUT
INTERSTING VISUALIZATIONS

CONVOLUTION

Mathematical Definition

For an image I and kernel K:

$$(I * K)(x, y) = \sum_i \sum_j I(x + i, y + j) \cdot K(i, j)$$

If kernel size is 3×3, then $i, j \in \{-1, 0, 1\}$.

ChatGPT
definition

FOR US... , WE JUST UTILIZE

Make a kernel (basically a matrix) and convolute. Usse kya hoga ??

- Blur
- Sharpen
- Extract edges
- Detect a direction
- Smooth noise
- Highlight textures

https://www.youtube.com/shorts/4xWpQe3G9qI

AFTER YOU EXPERIMENT

Two major properties:

- Sum of weights
  - Blur kernels → usually sum = 1
  - Sharpen kernels → sum > 1 or sum < 0
  - Edge detectors → sum = 0 (zero-sum kernels)
- Symmetry
  - Gaussian blur → symmetric
  - Sobel → directional

So, basically, look at the neighbouring pixel values, and replace the value with weighted sum. Thats convolution. Depending on different kernels, you get different effects.

If anyone does DSA, you will find Sliding Window

BLURRING

AVERAGE BLUR

Kernel :

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

All 9 pixels get equal weight i.e. equal smoothing.

Effect:

- Reduces noise
- Makes image look soft
- Preserves no details

```
avg_blur_cv = cv2.blur(gray, (3,3))
```

```
avg_blur_cv_rgb = cv2.blur(img_rgb, (3,3))
```

GAUSSIAN BLUR

Kernel :

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

Center weight highest i.e. neighboring weights decrease smoothly.

Effect:

- ur eyes naturally blur like Gaussian
- Reduces noise but preserves edges better than average
- Smooth fall-off from center → more natural

```
gaussian_cv = cv2.GaussianBlur(gray, (3,3), 0)
```

GRADIENTS

Differentiation

A gradient tells us how fast the pixel intensity is changing and in which direction.

- If intensity changes slowly → gradient is small
- If intensity changes quickly → gradient is large
- At edges → gradient is maximum

Gradient ≈ intensity(x+1) – intensity(x)

In 2D, we measure change in:

- x direction → Gx
- y direction → Gy

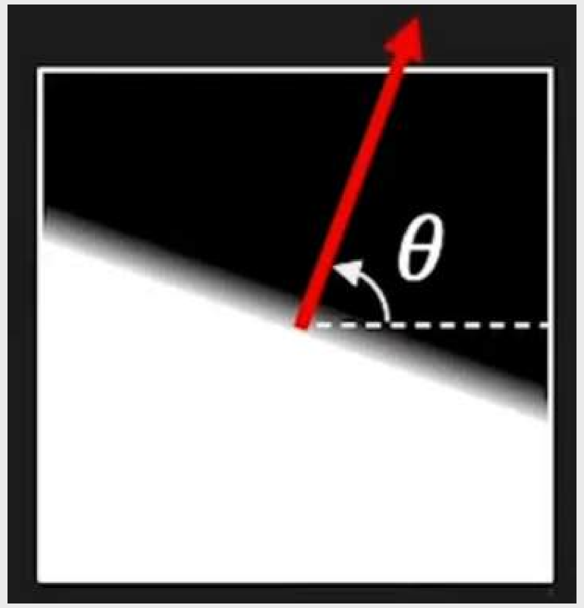
We combine them (like vectors) to get:

- Magnitude: how strong is the edge
- Direction: which way the edge is pointing

Good thing is we can implement it as Convolution with different operators (a part of the study video)

$$S = \|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

$$\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} \Big/ \frac{\partial I}{\partial x}\right)$$



EDGE DETECTION

First, **SOBEL** :

- Define the kernels
- Take convolutions with both
- Find the magnitude
- Either Threshold it or just return the same

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

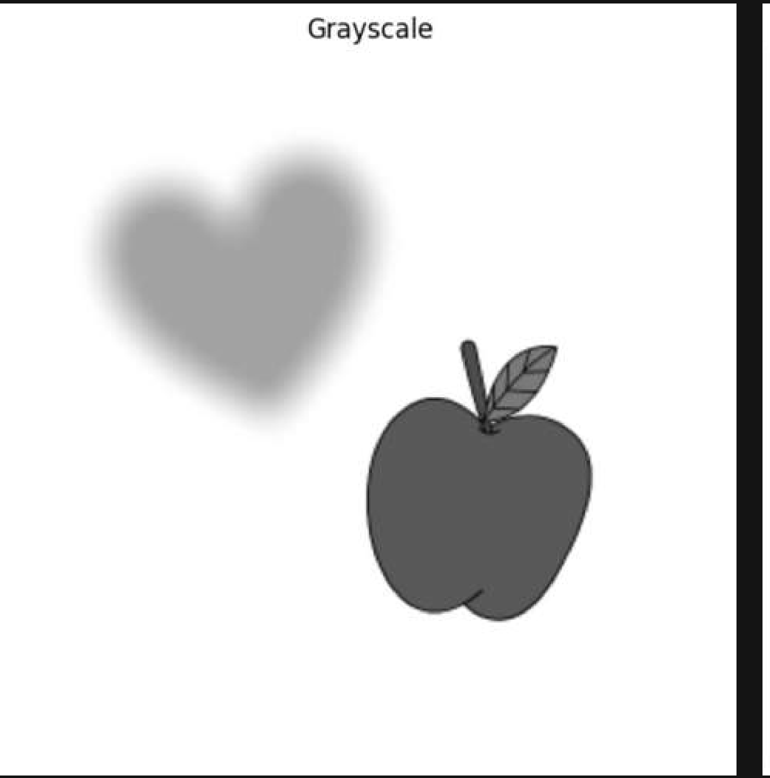
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

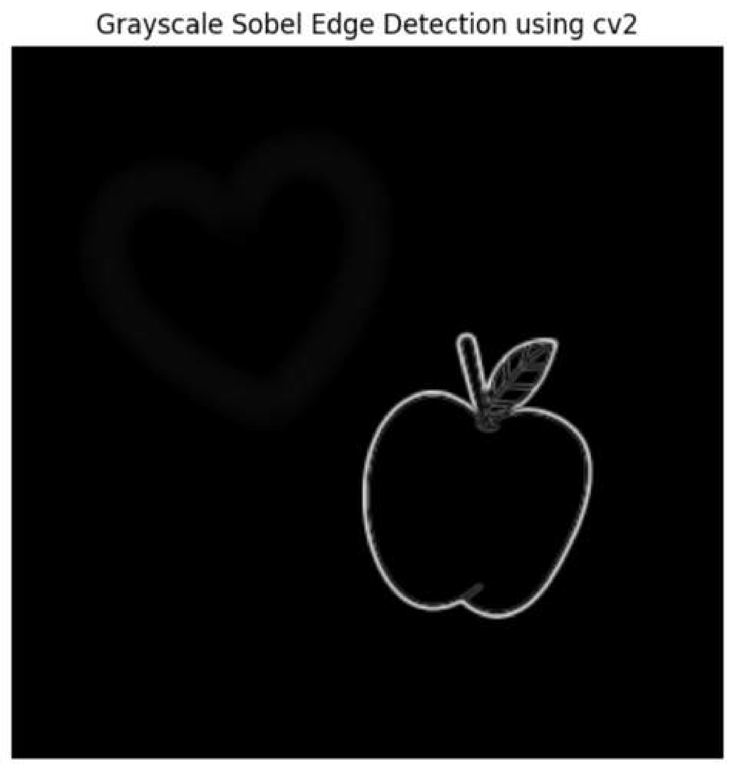
$$G = \sqrt{G_x^2 + G_y^2}$$

To Threshold :

- Binary Thresholding
- In 3 categories (Hysteresis Based)

```
gx_cv = cv2.Sobel(gray, cv2.CV_32F, 1, 0)
gy_cv = cv2.Sobel(gray, cv2.CV_32F, 0, 1)
mag_cv = cv2.magnitude(gx_cv, gy_cv)
plt.figure(figsize=(6,6))
plt.imshow(mag_cv, cmap='gray')
plt.title("Grayscale Sobel Edge Detection using cv2")
plt.axis("off")
plt.show()
```





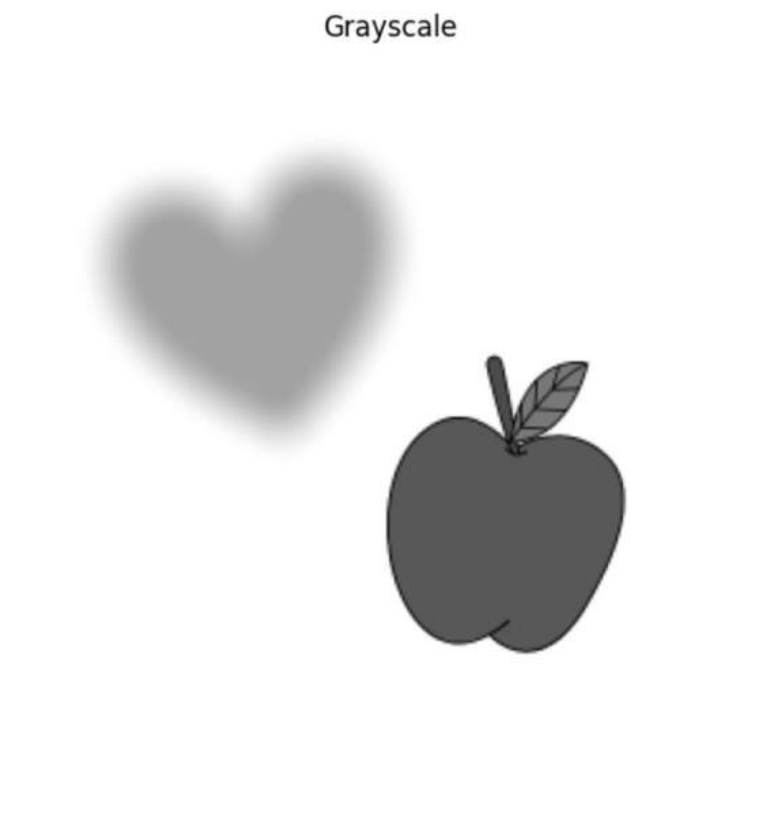
CANNY EDGE DETECT

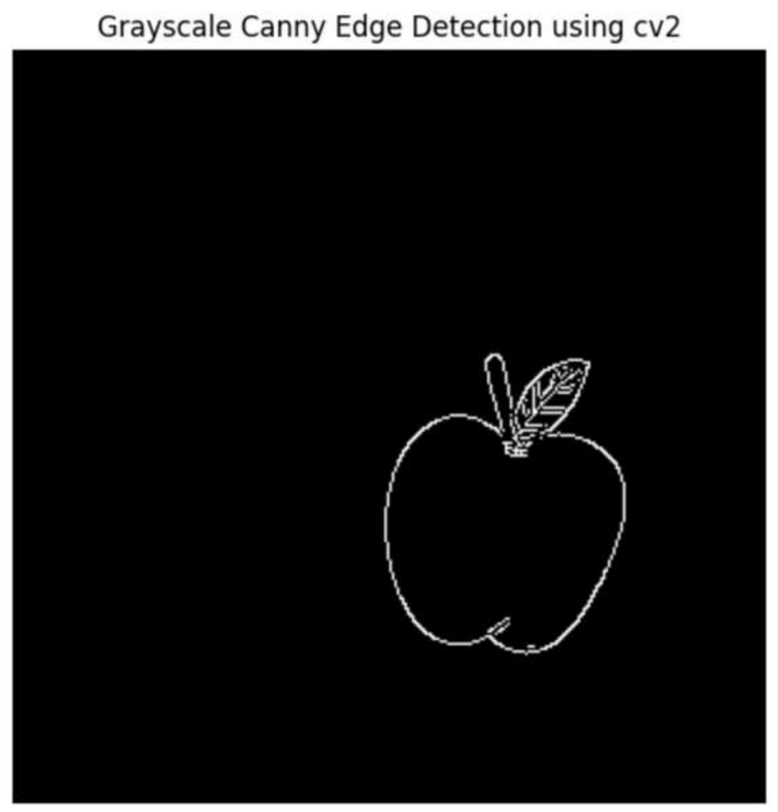
Canny =

1. Gaussian blur
2. Sobel gradients
3. Laplacian
4. Non-max suppression
5. Double threshold
6. Edge tracking by hysteresis

This is the gold-standard classical edge detector.
In more detail : https://www.youtube.com/watch?v=hUC1uoigH6s

```
canny = cv2.Canny(gray, 100, 200)
plt.figure(figsize=(6,6))
plt.imshow(canny, cmap='gray')
plt.title("Grayscale Canny Edge Detection using cv2")
plt.axis("off")
plt.show()
```





Scan me

SHARPENING

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

1	1	1
1	-8	1
1	1	1

LAPLACIAN SHARPENING

The Laplacian is a second-order derivative operator.

- First-order derivatives (Sobel, Prewitt) detect edges.
- Second-order derivatives detect regions where edges change quickly i.e., edges even more strongly.

In images, the Laplacian highlights points and thin edges, and makes edges look crisper.

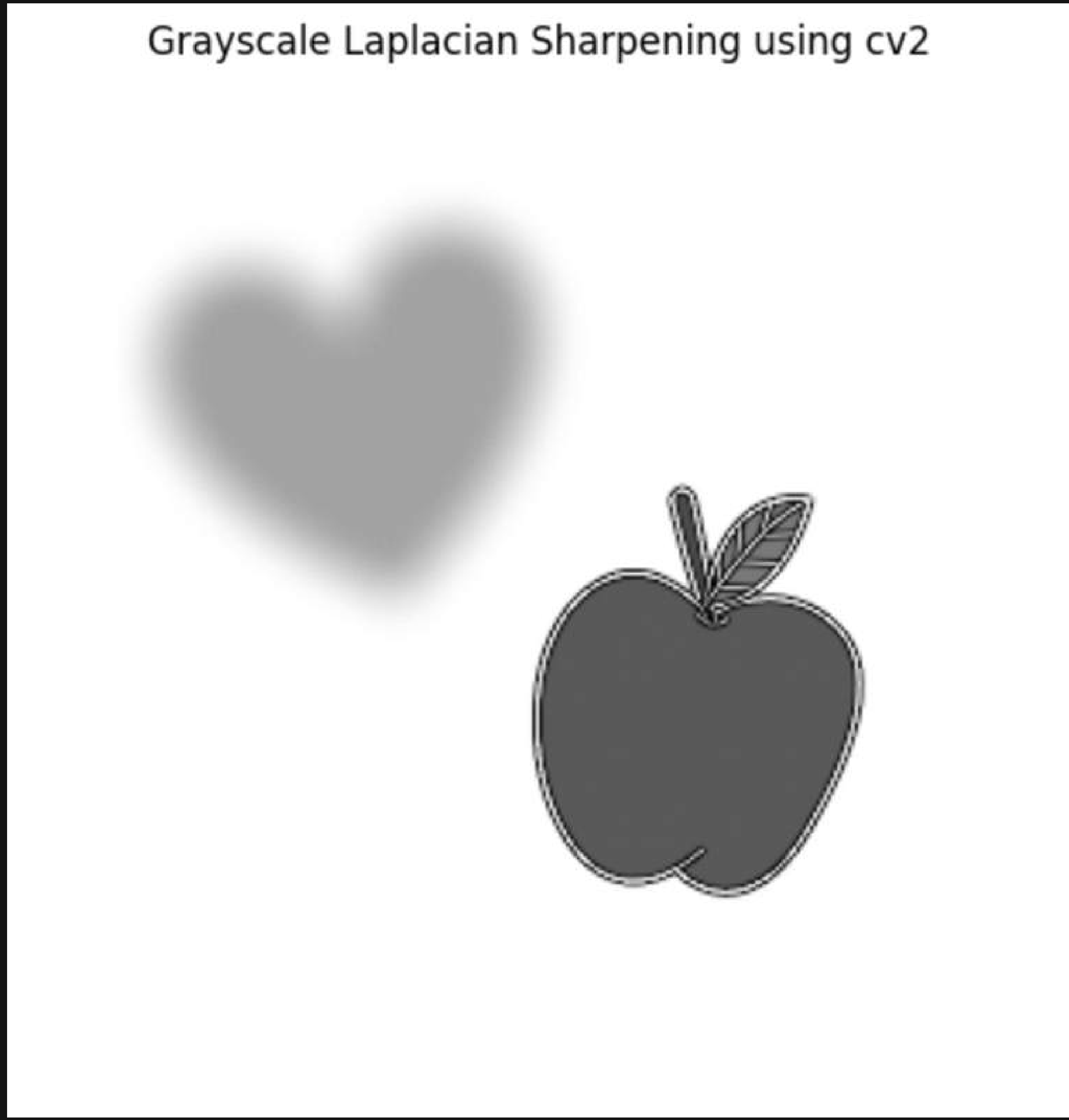
The Laplacian gives you an image that:

- is mostly zero in smooth areas
- becomes positive/negative around edges
- is extremely good at isolating fine details

So, we take Sharpened = Original - Laplacian
Or, Sharpened = Original - a * Laplacian

WHY NOT ADD ? JUST A SIGN DIFFERENCE

```
lap_cv = cv2.Laplacian(gray, cv2.CV_32F)
sharp_cv = np.clip(gray + lap_cv, 0, 255).astype(np.uint8)
plt.figure(figsize=(6,6))
plt.imshow(sharp_cv, cmap='gray')
plt.title("Grayscale Laplacian Sharpening using cv2")
plt.axis("off")
plt.show()
```



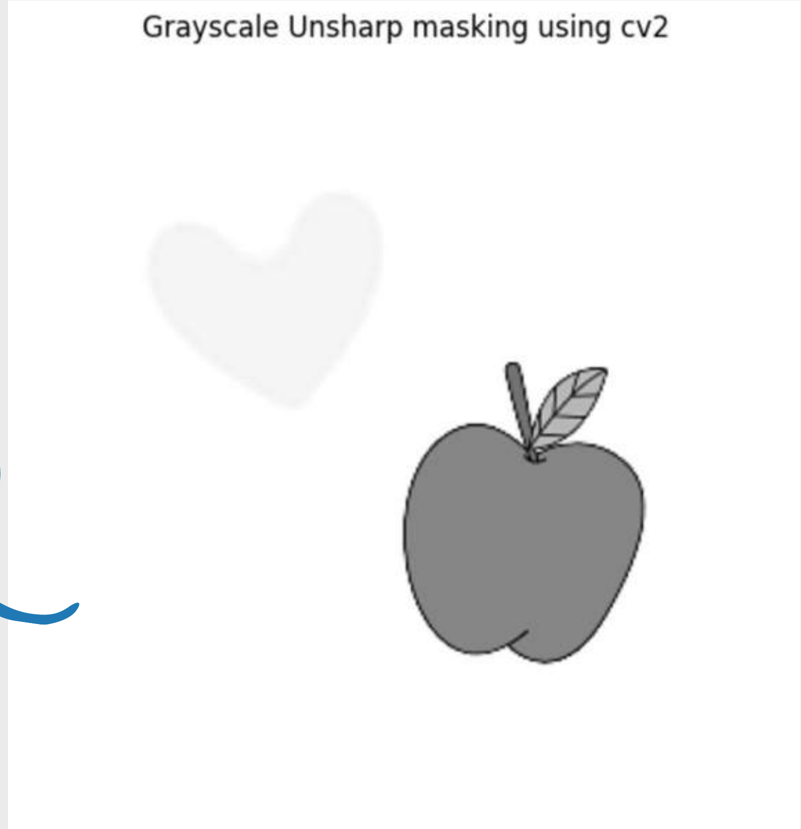
UNSHARP MASKING

Steps:

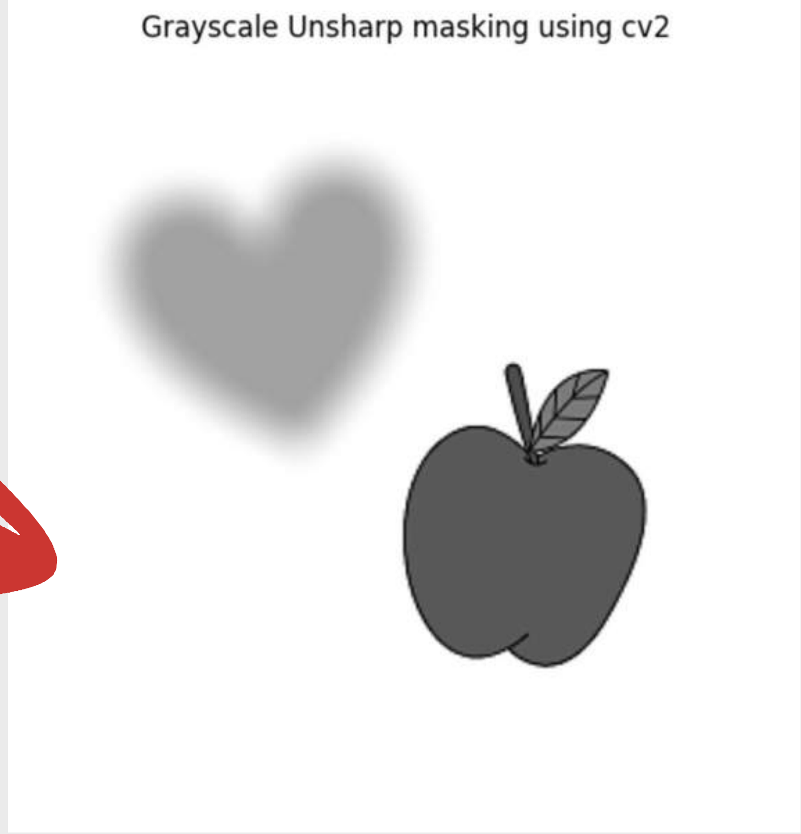
1. Blur the image → B
2. Compute mask → M= I – B
3. Add mask back to original

$I_{sharp} = I + kM$, where k controls sharpness

k = 2




```
blur = cv2.GaussianBlur(gray, (5,5), 0)
unsharp_cv = cv2.addWeighted(gray, 1.5, blur, -0.5, 0)
plt.figure(figsize=(6,6))
plt.imshow(unsharp_cv, cmap='gray')
plt.title("Grayscale Unsharp masking using cv2")
plt.axis("off")
plt.show()
```



KEY POINTS

Convolution ≈ Filtering in Frequency Domain

- Gaussian Blur must attenuate high frequencies (think about Magnitude Spectrum)
- Sharpening must amplify the higher frequencies

It implies that :

- FFT of Gaussian kernel → circular low-pass
- FFT of sharpen kernel → ring-shaped high-pass

So, Kernels = Masks in Frequency Domain

BYEEE :))