Lab 1 – Introduction to Image Processing & Computer Vision

- Overview: DIP vs Computer Vision.
- Loading & displaying images/videos (OpenCV in Python).
- Color spaces (RGB, Grayscale, HSV).
- Basic intensity transformations (brightness, contrast, gamma correction).
- Convolution
- Noise removal with mean, median, Gaussian filters.

DATASET: *https://www.kaggle.com/datasets/saadahmedbaust/testing-images*

Lab 1 – Introduction to Image Processing & Computer Vision

Part 1 – Introduction: DIP vs Computer Vision

What is Digital Image Processing (DIP)?
- Definition: Digital Image Processing involves manipulating images at the pixel level to enhance quality or extract basic features.
- Goal: Improve or prepare images for further analysis.
- Examples:
  - Noise removal using Gaussian filter
  - Image sharpening using Laplacian filter
  - Morphological operations (erosion, dilation)
  - Histogram equalization for contrast enhancement

What is Computer Vision (CV)?
- Definition: Computer Vision focuses on enabling machines to interpret and understand the content in images or videos.
- Goal: Extract meaningful information for decision-making.
- Examples:
  - Face recognition in your phone's camera
  - Self-driving car detecting pedestrians
  - OCR (reading text from images)
  - Detecting tumors from medical scans

Key Differences:
DIP: Pixel-level processing, enhancement & transformations, uses filters and morphology.
CV: Semantic-level understanding, object detection & recognition, uses AI/ML models.

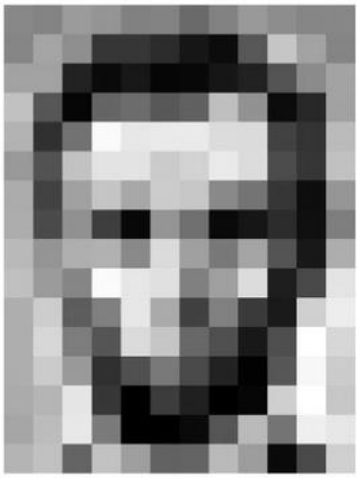
Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU

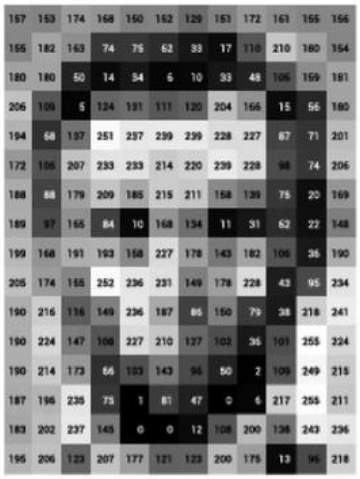
Part 2 – Lab Topics

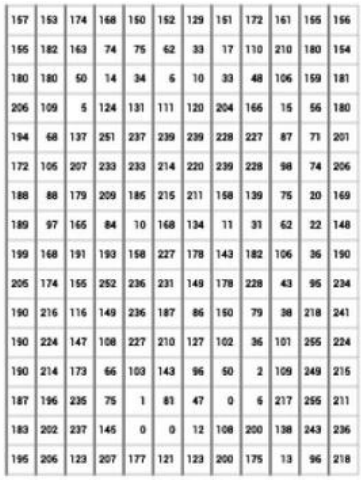
0. Basics of Digital Images

An image is a 2D array (matrix) of pixels.
Pixel values:
- Grayscale: 0 (black) to 255 (white)
- Color: 3 values (R, G, B), each 0–255
Image resolution: Width × Height in pixels.

Format	Full Form	Simple Working
BMP	Bitmap Image File	Stores raw pixel data in a grid (uncompressed). Large file size but simple structure.
JPEG / JPG	Joint Photographic Experts Group	Uses **lossy compression** to reduce file size by removing some details. Best for photos.
PNG	Portable Network Graphics	Uses **lossless compression** (no data loss) and supports **transparency**. Best for web graphics, logos.
GIF	Graphics Interchange Format	Uses **indexed colors (256 max)** and supports **simple animations**. Good for short moving images.
TIFF	Tagged Image File Format	Stores high-quality images with multiple compression options (lossy or lossless). Used in printing and scanning.







1. 1-bit (Monochrome)

- Only **2 colors**: black and white.
- Each pixel is 0 or 1.

Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU

2. 8-bit grayscale

- 256 shades of gray (2^8).
- Common in black-and-white photography.

3. 8-bit color (per channel) = 24-bit color

- 8 bits per color channel: **Red, Green, Blue** (RGB).
- Total colors = $2^8 \times 2^8 \times 2^8$ = **16.7 million colors**.
- Standard for most color images.

4. Higher bit depths

- **10-bit**, **12-bit**, **16-bit** per channel used in professional imaging.
- 16-bit per channel = 48-bit color (281 trillion colors).
- Used for high dynamic range (HDR) and precise editing.

1. Loading & Displaying Images

Theory:
- cv2.imread() loads images.
- cv2.imshow() displays images.
Note: OpenCV reads in BGR format.

Code Example:

```
import cv2
import matplotlib.pyplot as plt

# Read images
img_color = cv2.imread('/kaggle/input/testing-images/lena.png')
img_gray = cv2.imread('/kaggle/input/testing-images/lena.png', cv2.IMREAD_GRAYSCALE)

# Convert BGR (OpenCV default) to RGB for correct color display
img_color_rgb = cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)

# Display images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Color Image')
plt.imshow(img_color_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Grayscale Image')
plt.imshow(img_gray, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

****Print image as matrix**

2. Color Spaces (RGB, Grayscale, HSV)

Theory:
- Color Space: Representation of colors numerically.
- RGB: Red, Green, Blue channels.

Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU

- Grayscale: Single channel (0–255).
- HSV: Hue, Saturation, Value – better for color filtering.

Blue Channel (B): This channel highlights the amount of blue in the image. Areas with more blue will appear brighter, while other areas will be darker or gray.

Green Channel (G): Similar to the blue channel, this displays the green intensity. It plays a important role in natural images, particularly landscapes.

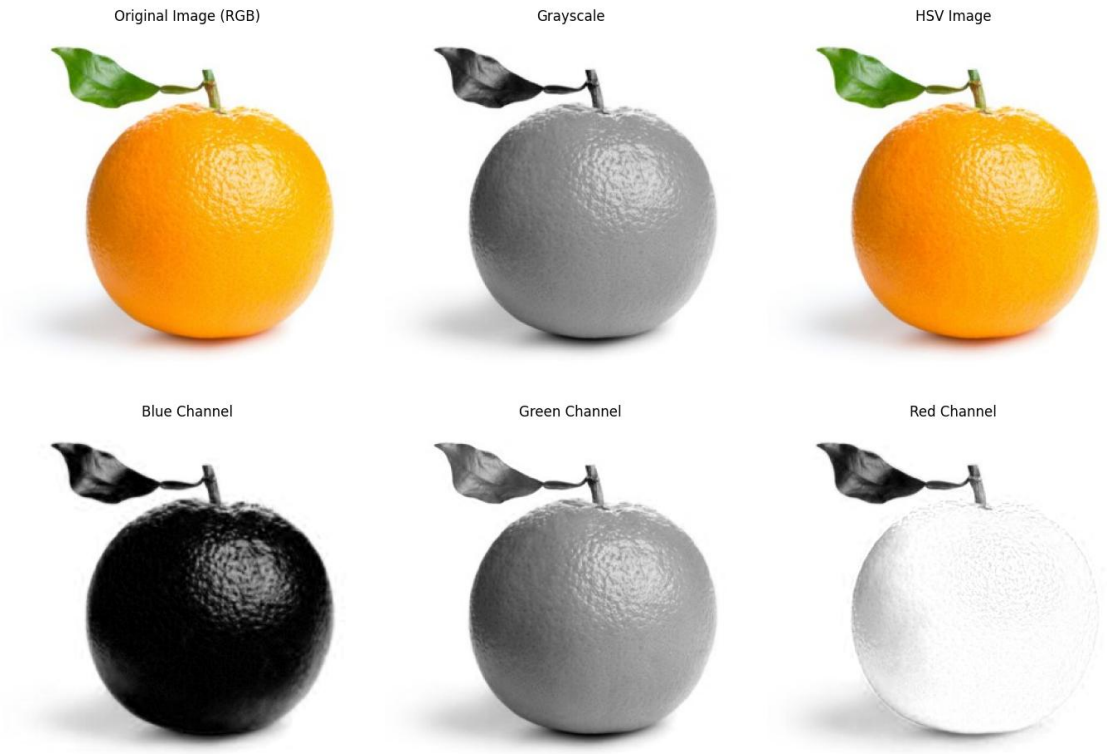
Red Channel (R): This channel displays red intensity which influences the warmth of the image, highlighting reds, oranges and skin tones.

Code Example:

```python
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = '/kaggle/input/testing-images/orange.jpg'  # Update this path if needed
image_bgr = cv2.imread(image_path)

# Check if the image loaded correctly
if image_bgr is None:
    raise FileNotFoundError(f"Image not found at: {image_path}")

# Convert to other color spaces
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
image_gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
image_hsv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2HSV)

# Split channels
B, G, R = cv2.split(image_bgr)

# Display all images using matplotlib
plt.figure(figsize=(14, 10))

# Original Image (RGB)
plt.subplot(2, 3, 1)
plt.imshow(image_rgb)
plt.title('Original Image (RGB)')
plt.axis('off')
```

```python
# Grayscale Image
plt.subplot(2, 3, 2)
plt.imshow(image_gray, cmap='gray')
plt.title('Grayscale')
plt.axis('off')

# HSV Image (just show as RGB for viewing)
plt.subplot(2, 3, 3)
plt.imshow(cv2.cvtColor(image_hsv, cv2.COLOR_HSV2RGB))
plt.title('HSV Image')
plt.axis('off')

# B Channel
plt.subplot(2, 3, 4)
plt.imshow(B, cmap='gray')
plt.title('Blue Channel')
plt.axis('off')

# G Channel
plt.subplot(2, 3, 5)
plt.imshow(G, cmap='gray')
plt.title('Green Channel')
plt.axis('off')

# R Channel
plt.subplot(2, 3, 6)
plt.imshow(R, cmap='gray')
plt.title('Red Channel')
plt.axis('off')

plt.tight_layout()
plt.show()
```


Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU



3. Basic Intensity Transformations (Brightness, Contrast, Gamma Correction)

Theory:

- Brightness: Add/subtract a constant to pixels.
- Contrast: Multiply pixel values by a factor.
- Gamma: Non-linear brightness adjustment.

What is an Intensity Transformation?

An intensity transformation changes the pixel values of an image to enhance or modify its appearance.

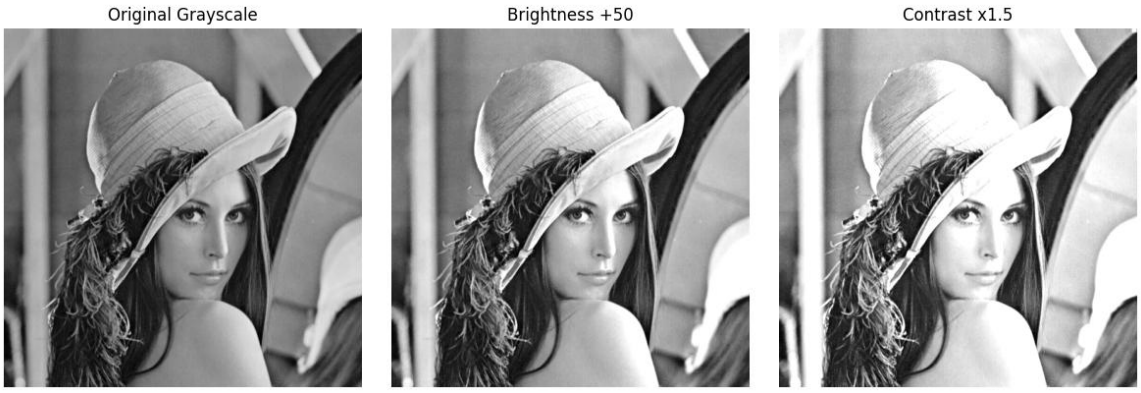
- Input pixel value: r (0–255 for grayscale)
- Output pixel value: s (after transformation)
- Transformation: s=T(r)s = T(r)s=T(r)

Why use them?

- Improve visibility of features
- Adjust brightness & contrast
- Highlight certain intensity ranges
- Prepare for further processing

Code Example:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image in grayscale
gray_img = cv2.imread('/kaggle/input/testing-images/lena.png', cv2.IMREAD_GRAYSCALE)

# Brightness adjustment
brightness = 50
bright_img = cv2.add(gray_img, brightness)  # Adds 50 to each pixel, saturates at 255

# Contrast adjustment
contrast = 1.5
contrast_img = cv2.multiply(gray_img, contrast)  # Prevent overflow
contrast_img = np.clip(contrast_img, 0, 255)
```



***** Reduce brightness and contract**

Negative Transformation

Theory:

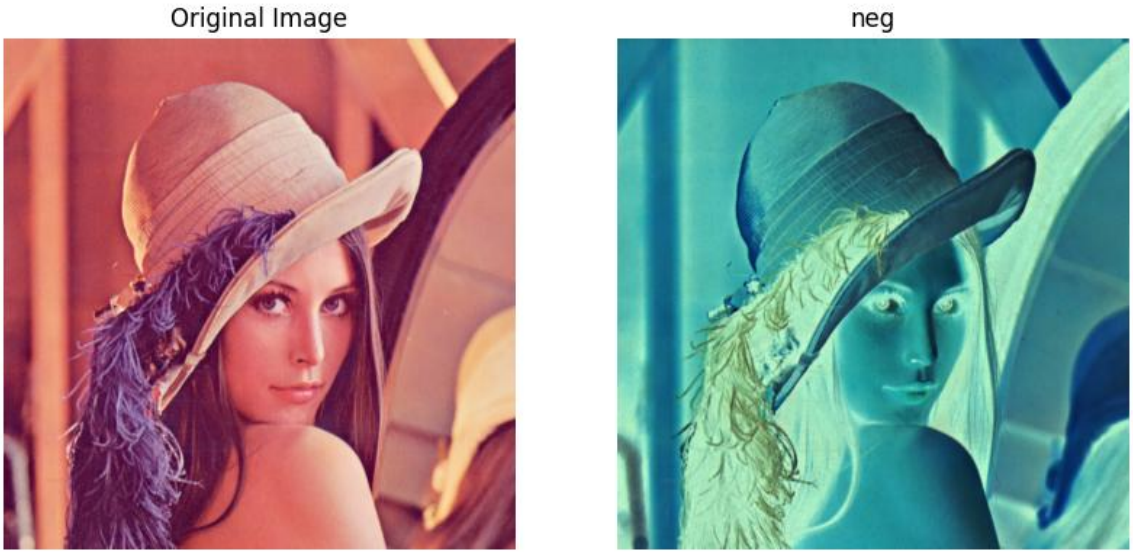
- Dark pixels → light pixels, and vice versa
- Useful for medical images, x-rays, etc.

Formula:

$$s = 255 - r$$

Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU

```python
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('/kaggle/input/testing-images/lena.png')
negative = 255 - img


# Display side by side
plt.figure(figsize=(10,5))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(negative, cv2.COLOR_BGR2RGB))
plt.title("neg")
plt.axis('off')

plt.show()
```



Power-Law (Gamma) Transformation

Theory:

- Controls brightness by raising pixel values to a power (γ\gammaγ)
- γ<1\gamma < 1γ<1 → brightens
- γ>1\gamma > 1γ>1 → darkens

Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU

```
gamma = 0.5
normalized = img / 255.0
gamma_corrected = np.power(normalized, gamma)
gamma_corrected = np.uint8(gamma_corrected * 255)

# Display side by side
plt.figure(figsize=(10,5))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(gamma_corrected, cv2.COLOR_BGR2RGB))
plt.title(f'Gamma Corrected (γ={gamma})')
plt.axis('off')

plt.show()
```

4. Convolution

Theory:

1. What is Convolution?

Convolution is a mathematical operation where we apply a **filter/kernel** to an image to extract certain features or modify it.

- The kernel is usually a **small matrix** (e.g., 3×3, 5×5).
- We slide this kernel over the image, multiply overlapping values, sum them up, and store the result in the output image.

2. Why do we use Convolution?

- **Blurring** (reduce noise)

- **Sharpening** (highlight edges/details)
- **Edge detection** (find boundaries)
- **Feature extraction** (for Computer Vision)

Code Example:

```
import numpy as np
kernel = np.ones((3,3), np.float32) / 9
convolved = cv2.filter2D(img, -1, kernel)

# Show original and convolved side by side
plt.figure(figsize=(10,5))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(convolved, cv2.COLOR_BGR2RGB))
plt.title('Convolved Image')
plt.axis('off')

plt.show()
```

***** DO IT MANUALLY using matrix manipulation**

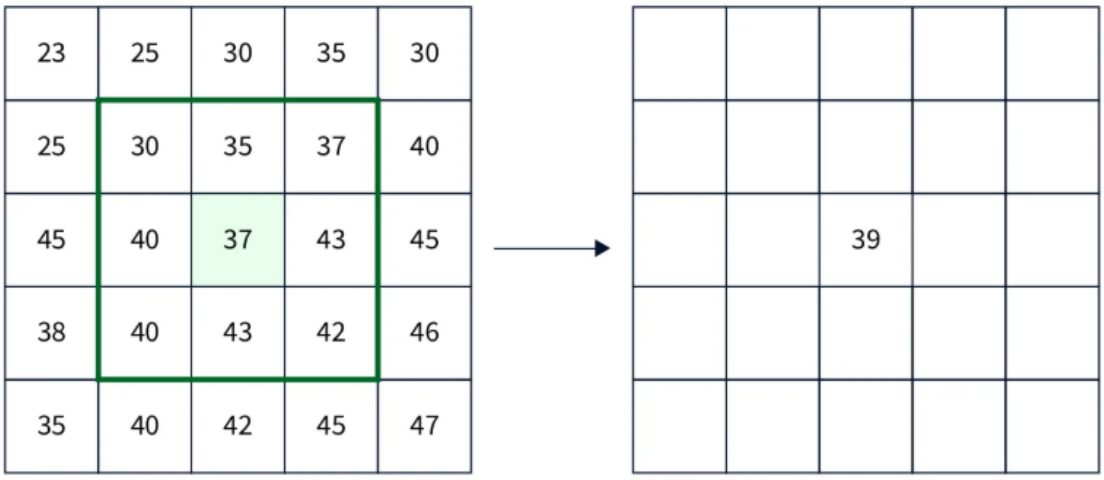
5. Noise Removal (Mean, Median, Gaussian Filters)

Theory:

- Noise: Unwanted variations in pixel values.
- Filters:
  - Mean: Averages neighborhood.
  - Median: Picks median value.
  - Gaussian: Weighted average.

Mean Filter

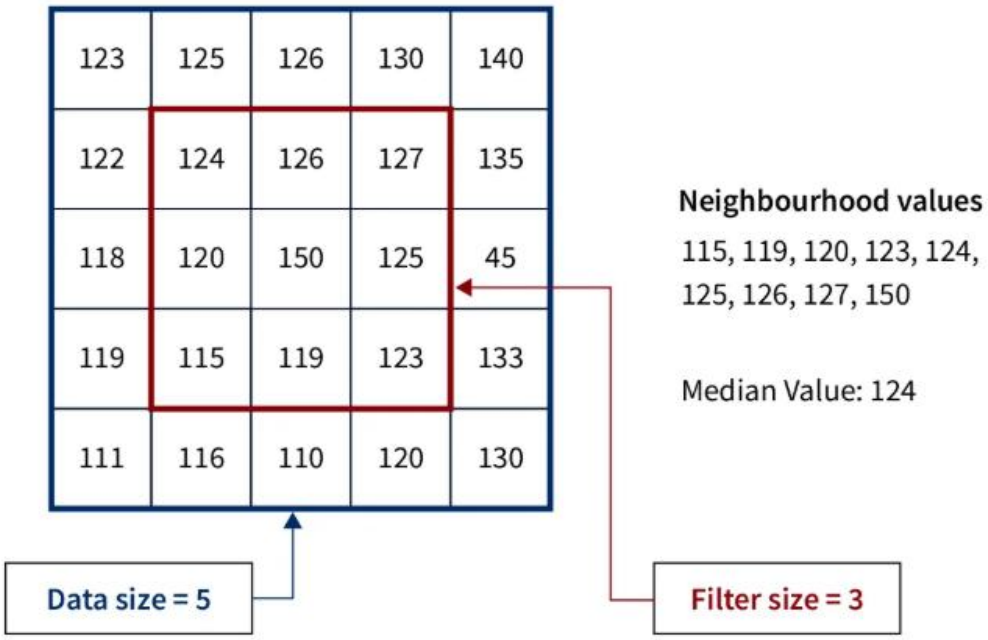
The mean filter is a type of **linear smoothing filter** that replaces each pixel in the image with the average of its neighboring pixels. The size of the neighboring pixels is defined by the filter kernel or mask. The larger the kernel, the stronger the smoothing effect. Mean filters are easy to implement and are commonly used in low-level image processing tasks such as noise reduction and edge detection. However, they tend to blur edges and details in the image, leading to loss of image quality.



Prepared by Saad Ahmed & Modified by Md. Manirujjaman
Lecturer, Dept of CSE, BAUST & Lecturer, Dept of CSE, DIU

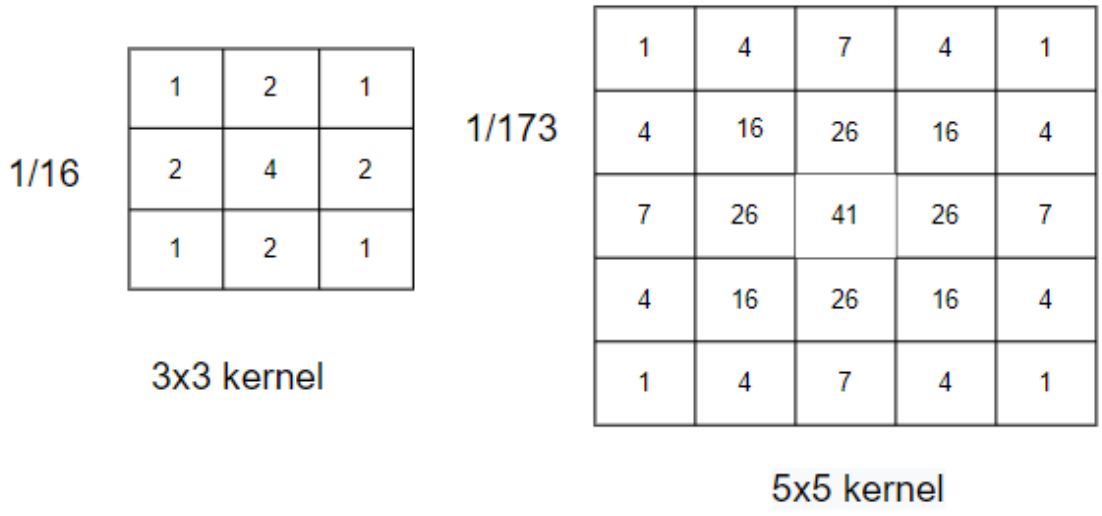
Median Filter

The median filter is a type of **nonlinear smoothing filter** that replaces each pixel in the image with the median value of its neighboring pixels. The size of the neighborhood is also defined by the filter kernel. Unlike the mean filter, the median filter does not blur edges and details in the image. Instead, it preserves them while removing the noise. Median filters are commonly used in image processing tasks that involve removing salt and pepper noise from the image.



Gaussian Filter…

This filter is a 2-D convolutional operator. It use to blur images. Also, it removes details and noises. Gaussian filter is similar to mean filter. But main difference is, Gaussian filter use a kernel. That kernel has a shape of gaussian hump. Gaussian kernel weights pixels at its center much more strongly than its boundaries. There are different gaussian kernels. Based on the kernel size, output image will be different.



Code Example:

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image_path = '/kaggle/input/testing-images/noise.png'  # Change this path as needed
bgr_img = cv2.imread(image_path)

if bgr_img is None:
    raise FileNotFoundError("Image not found. Check the file path.")

# Convert BGR to RGB for proper display
rgb_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)

# Apply different blurs
mean_blur = cv2.blur(rgb_img, (5, 5))
median_blur = cv2.medianBlur(rgb_img, 5)
gaussian_blur = cv2.GaussianBlur(rgb_img, (5, 5), 0)
```