If you want, next I can:

- Explain **convolution step-by-step numerically**
- Compare **mean vs median** (with table)
- Explain **which filter to use for which noise**
- Give **short exam answers + numericals**

Just tell me what you want next.

write python code to apply min max, median, average and weighted averge filters on a image. i have not done ay works on it before and i dont have much theory knowledge of it too. so first write code, and then give me a description of it all.

Understood. I will treat you as a **complete beginner** to neighbourhood filtering and proceed in the most **hands-on, lab-friendly** way.

I will do this in **two clear phases**:

1. **First: Python code only** (simple, readable, runnable)
2. **Second: Conceptual explanation** (slow, structured, exam-oriented)

PHASE 1 — PYTHON CODE (RUN THIS FIRST)

Requirements

Install once if not installed:

bash

```bash
pip install opencv-python numpy matplotlib
```

Copy code

---

1. Python Code: Min, Max, Median, Average & Weighted Average Filters

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

# ----------------------------
# Load image (grayscale)
# ----------------------------
image = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)

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

# ----------------------------
# 1. Min Filter (Erosion)
# ----------------------------
kernel = np.ones((3, 3), np.uint8)
min_filtered = cv2.erode(image, kernel)

# ----------------------------
# 2. Max Filter (Dilation)
# ----------------------------
max_filtered = cv2.dilate(image, kernel)

# ----------------------------
# 3. Median Filter
# ----------------------------
median_filtered = cv2.medianBlur(image, 3)

# ----------------------------
# 4. Average (Mean) Filter
# ----------------------------
average_filtered = cv2.blur(image, (3, 3))

# ----------------------------
# 5. Weighted Average Filter (Gaussian)
# ----------------------------
weighted_average_filtered = cv2.GaussianBlur(image, (3, 3), 0)

# ----------------------------
# Display results
# ----------------------------
```

```
titles = [
    "Original Image",
    "Min Filter",
    "Max Filter",
    "Median Filter",
    "Average Filter",
    "Weighted Average Filter"
]

images = [
    image,
    min_filtered,
    max_filtered,
    median_filtered,
    average_filtered,
    weighted_average_filtered
]

plt.figure(figsize=(12, 8))

for i in range(len(images)):
    plt.subplot(2, 3, i + 1)
    plt.imshow(images[i], cmap="gray")
    plt.title(titles[i])
    plt.axis("off")

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

📌 **Important**

- Place `image.jpg` in the same folder as this script
- Use any grayscale image (or convert color → grayscale)

PHASE 2 — EXPLANATION (FROM ZERO THEORY)

1. What Is Happening Conceptually?

All these filters are **neighbourhood operations**:

- A **3×3 window** moves over the image
- For every pixel:
  - Look at surrounding pixels

- Compute a new value
- Replace the center pixel

---

2. Min Filter (Minimum Filter)

What it does

- Takes the **minimum value** in the neighborhood
- Dark pixels dominate

Copy code

Effect

- Removes **white noise**
- Shrinks bright regions

Code logic

python

```python
cv2.erode(image, kernel)
```

Copy code

Example

csharp

```csharp
[120 130 140]
[125 255 135] → min = 120
[122 128 138]
```

Copy code

---

3. Max Filter (Maximum Filter)

What it does

- Takes the **maximum value** in the neighborhood
- Bright pixels dominate

Effect

- Removes **black noise**
- Expands bright regions

Code logic

python

```python
cv2.dilate(image, kernel)
```

Copy code

---

4. Median Filter (MOST IMPORTANT)

What it does

- Sorts neighborhood values
- Picks the **middle value**

Why it is powerful

- Removes **salt-and-pepper noise**
- Preserves edges

Code logic

python

```python
cv2.medianBlur(image, 3)
```

Copy code

Example

ini

```ini
[10, 12, 14,
 15, 255, 16,
 17, 18, 19]

Median = 15
```

Copy code

---

5. Average (Mean) Filter

What it does

- Computes **simple average** of neighbors

Effect

- Smooths image
- Blurs edges
- Reduces Gaussian noise

Kernel

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

Code logic

python

```python
cv2.blur(image, (3, 3))
```

Copy code

---

6. Weighted Average Filter (Gaussian Filter)

What it does

- Like average filter
- **Center pixels get more weight**

Why better than mean

- Less blurring
- More natural smoothing

Kernel example

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

Code logic

python

```python
cv2.GaussianBlur(image, (3, 3), 0)
```

Copy code

---

7. Comparison Table (Very Exam-Friendly)

Filter	Type	Removes Noise	Edge Preservation
Min	Non-linear	White noise	Poor
Max	Non-linear	Black noise	Poor

Filter	Type	Removes Noise	Edge Preservation
Median	Non-linear	Salt & pepper	Excellent
Average	Linear	Gaussian	Poor
Weighted Avg	Linear	Gaussian	Better

---

8. One-Line Definitions (Memorize)

- **Min filter**: replaces pixel with minimum of neighbors
- **Max filter**: replaces pixel with maximum of neighbors
- **Median filter**: replaces pixel with median of neighbors
- **Average filter**: replaces pixel with mean of neighbors
- **Weighted average filter**: weighted mean giving more importance to center pixels

---

9. What You Should Do Next (Recommended)

1. Try **different kernel sizes**: 5×5
2. Try **adding noise** and removing it
3. Write **short theory answers** for exams
4. Implement **manual convolution** (next level)

If you want next, I can:

- Add **salt & pepper noise + remove it**
- Explain **convolution numerically**
- Write **lab record–ready experiment format**
- Convert this into **pure NumPy (no OpenCV)**

Just tell me what you want next.

```
PS C:\Users\aayus\Desktop\Study
Materials\CV\Experiments> python -u
"c:\Users\aayus\Desktop\Study
```