### **Assignment Number: 02**

### **Name:** Mihir Unmesh Patil

### **Roll No:** TYCOC213

### **Batch:** C/C-3

### **Title:** Apply any 4 mask processing techniques on sample image

### **Aim:**

### The aim of this project is to demonstrate the concept and implementation of four mask processing techniques—Weighted Average Filter, Custom Gaussian Filter, Median Filter, and Min Filter—using Python and OpenCV. These techniques will be applied to a sample grayscale image to perform smoothing, noise reduction, and feature enhancement, highlighting the role of mask-based operations in image processing.

### **Objectives:**

### **Weighted Average Filter:** To smooth the image by applying a weighted averaging mask, reducing noise and fine details.

### **Custom Gaussian Filter:** To apply a custom Gaussian mask for enhanced smoothing, preserving image structure while reducing high-frequency noise.

### **Median Filter:** To reduce salt-and-pepper noise in the image using a median mask, maintaining edge integrity.

### **Min Filter:** To apply a minimum filter mask to darken the image and emphasize low-intensity regions. These objectives are aimed at understanding how different mask processing techniques manipulate pixel neighborhoods to achieve specific image enhancements.

### 

### 

### **Theory:**

### Mask processing techniques involve applying kernels or masks to an image to transform pixel values based on their local neighborhoods. These methods are widely used in image processing for tasks like noise reduction, smoothing, and feature extraction. Below is an explanation of the four selected techniques:

### **Weighted Average Filter:**

### The Weighted Average Filter uses a mask where pixel values are weighted based on their proximity to the center pixel, typically giving higher weight to the center. In this case, a 3x3 kernel ([[1, 2, 1], [2, 4, 2], [1, 2, 1]] / 16) is applied, normalizing the weights to sum to 1. This smooths the image by averaging neighboring pixels, reducing noise and sharp transitions. **Process:**

### Define the weighted kernel.

### Convolve it with the image using cv2.filter2D().

### Replace each pixel with the weighted average of its neighborhood.

### **Custom Gaussian Filter:**

### The Custom Gaussian Filter uses a larger 5x5 Gaussian kernel ([[1, 4, 7, 4, 1], [4, 16, 26, 16, 4], ...] / 273) based on the Gaussian distribution. This provides stronger smoothing than the weighted average filter by assigning weights that decrease with distance from the center, effectively blurring the image while preserving its overall structure. **Process:**

### Define the Gaussian kernel and normalize it (sum = 1).

### Convolve it with the image using cv2.filter2D().

### Display the smoothed result.

### **Median Filter:**

### The Median Filter is a non-linear technique that replaces each pixel with the median value of its neighborhood (e.g., a 5x5 window). This is particularly effective for removing salt-and-pepper noise while preserving edges, unlike linear filters that blur them. **Process:**

### Slide a 5x5 mask over the image.

### Compute the median of the pixel values in the mask.

### Replace the central pixel with the median using cv2.medianBlur().

### **Min Filter:**

### The Min Filter replaces each pixel with the minimum value in its neighborhood (e.g., a 3x3 window). This darkens the image and emphasizes low-intensity regions, often used to suppress bright outliers or highlight dark features. **Process**

### Apply a 3x3 mask using minimum\_filter from scipy.ndimage.

### Replace each pixel with the minimum value in its neighborhood.

### Display the darkened result.

### These techniques demonstrate how masks can be tailored to achieve specific effects, from smoothing to noise removal and intensity adjustment.

### **Methodology:**

### The implementation of the four mask processing techniques was done using Python, OpenCV, and additional libraries like NumPy and SciPy, following these steps:

### **Loading the Image:**

### The input image (girl.jpg) is read in grayscale mode using cv2.imread() to simplify mask application.

### The original image is displayed with matplotlib.pyplot.imshow().

### **Applying Weighted Average Filter:**

### A 3x3 weighted kernel ([[1, 2, 1], [2, 4, 2], [1, 2, 1]] / 16) is defined.

### The kernel is convolved with the image using cv2.filter2D().

### The result is displayed, and matrices before and after are printed for comparison.

### 

### **Applying Custom Gaussian Filter:**

### A 5x5 Gaussian kernel ([[1, 4, 7, 4, 1], ...] / 273) is defined and normalized.

### The kernel is applied using cv2.filter2D().

### The smoothed image is displayed with matrix outputs.

### **Applying Median Filter:**

### A 5x5 median filter is applied using cv2.medianBlur().

### The noise-reduced image is displayed with matrix comparisons.

### **Applying Min Filter:**

### A 3x3 minimum filter is applied using minimum\_filter() from scipy.ndimage. The darkened image is displayed with matrix outputs.

### **Code Implementation:** [**Assignment\_02\_IVP**](https://colab.research.google.com/drive/1K7LDHi8-kkyflMhGH4b14QQbfVirPmdm?usp=sharing)

### The code is implemented in a Jupyter notebook environment, with functions for each filter and a main apply\_filter() function to execute the user’s choice.

### **Conclusion:**

### This project successfully demonstrates the application of four mask processing techniques—Weighted Average Filter, Custom Gaussian Filter, Median Filter, and Min Filter—using OpenCV and Python on a sample grayscale image. The Weighted Average Filter provides basic smoothing, the Custom Gaussian Filter offers enhanced blurring, the Median Filter effectively removes noise while preserving edges, and the Min Filter darkens the image to highlight low-intensity features. These techniques illustrate the power of mask-based operations in manipulating pixel neighborhoods for various image processing tasks. Such methods are invaluable in applications like photography, medical imaging, and computer vision, where noise reduction and feature enhancement are critical.

### 