Assignment 2

Pritam Kumar
E9 241: Digital Image Processing

September 22, 2025

1 Spatial Filtering and Binarisation

For this question, I applied Gaussian and Average Filters.

$$B_f(x,y) = \frac{1}{K}, \quad x, y \in \{-(m-1)/2 \cdots (m-1)/2\}$$

$$B_f(x,y) = \frac{1}{m^2} \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & m \end{bmatrix}}_{m\times m}, \quad \text{where } B_f(x,y) \text{ is an } m \times m \text{ average (uniform) filter.}$$

The images below are the results of filtering the noisy moon image with different filter sizes. I performed the convolution on the noisy moon image using the library cv2. Before applying the convolution, I converted the image into grayscale image.

1.1 Analysis of class FilteringAndBinarization(m)

Purpose: The `FilteringAndBinarization(m)` class provides image filtering (smoothing) and binarization (thresholding) utilities for grayscale or RGB images.

Methods:

- **box_filter():** Returns an $m \times m$ averaging filter (used for smoothing).
- **convolve(image, kernel):** Converts input to grayscale if needed, then applies convolution using OpenCV (`cv2.filter2D`).
  Time Complexity: $\mathcal{O}(N \times M \times k^2)$ for an $N \times M$ image and $k \times k$ kernel (but OpenCV is optimized).
  Space Complexity: $\mathcal{O}(N \times M)$ for the output image.
- **histogram_compute(img):** Computes the normalized histogram of pixel intensities.
  Time Complexity: $\mathcal{O}(N \times M)$ for flattening and histogram calculation.
  Space Complexity: $\mathcal{O}(256)$ for histogram bins.

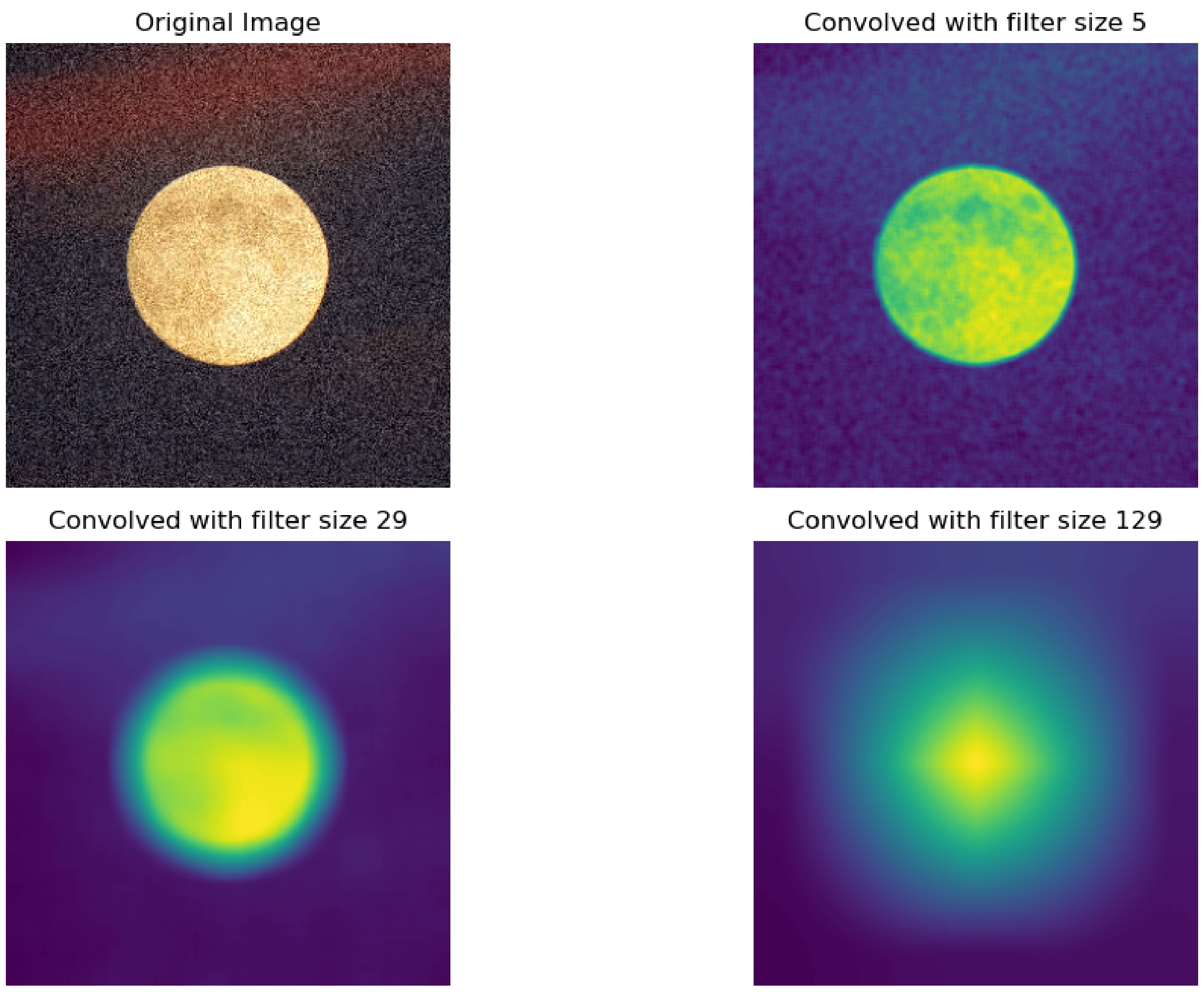
Original Image
Convolved with filter size 5
Convolved with filter size 29
Convolved with filter size 129

Figure 1: The first image is the original image.

- **otsu_binarization(norm_histogram, threshold):** Calculates within-class variance for a given threshold using Otsu's method.
  Time Complexity: $\mathcal{O}(256)$ (since histogram bins are fixed).
  Space Complexity: $\mathcal{O}(1)$ (scalar values).

- **otsu_threshold(img):** Finds the optimal threshold by minimizing within-class variance.
  Time Complexity: $\mathcal{O}(256)$ (calls `otsu_binarization` for each possible threshold).
  Space Complexity: $\mathcal{O}(256)$ for storing variances.

Specifications:

- Handles both PIL images and NumPy arrays (RGB or grayscale).
- Uses OpenCV for fast convolution.
- Implements Otsu's method for automatic threshold selection.

- Designed for image preprocessing in segmentation and analysis tasks.

Typical Workflow:

`Filter image` → `Compute histogram` → `Find Otsu threshold` → `Binarize image`

1.2 Analysis of Output

- **Small Filter Size** (5×5)**:**
  - Performs mild smoothing, reducing noise while preserving most image details.
  - Object edges remain sharp and well-defined.
  - Good for removing small-scale noise without significant blurring.
- **Medium Filter Size** (29×29)**:**
  - Produces stronger smoothing, effectively reducing noise and local variations.
  - Edges begin to blur, and fine textures are lost.
  - Suitable for applications where denoising is more important than edge sharpness.
- **Large Filter Size** (129×129)**:**
  - Performs heavy smoothing, leading to a very blurred image.
  - Most fine details and textures are lost; the object shape becomes very diffused.
  - Acts as a very strong low-pass filter, preserving only large-scale intensity changes.
- **General Observation:**
  - Increasing filter size reduces noise but also causes loss of detail and edge sharpness.
  - There is a trade-off between denoising and detail preservation; an optimal filter size must be chosen based on the application.

Now, I will attach the graphs fig. 2 of within-class variances for various filter sizes and the optimal threshold for them. We can see that

Table 1: Within-Class Variance and Otsu Threshold for Different Filter Sizes

Filter Size $(m \times m)$	**Within-Class Variance**	**Otsu Threshold**
5×5	169.62	127
29×29	287.54	121
129×129	270.51	91

The minimum within-class variance (σ_w^{2*})is **169.62** for the filter size 5×5, indicating that this filter size achieves the best separation between foreground and background classes. The corresponding Otsu's threshold is **127**, which effectively binarizes the image by minimizing intra-class intensity variations while preserving important image details and edges. σ_w^{2*} increases for some m and then starts decreasing. But the optimal threshold decreases monotonically.

The graph fig. 3 shows the experimental result of m_s and corresponding σ_w^{2*} and t^*

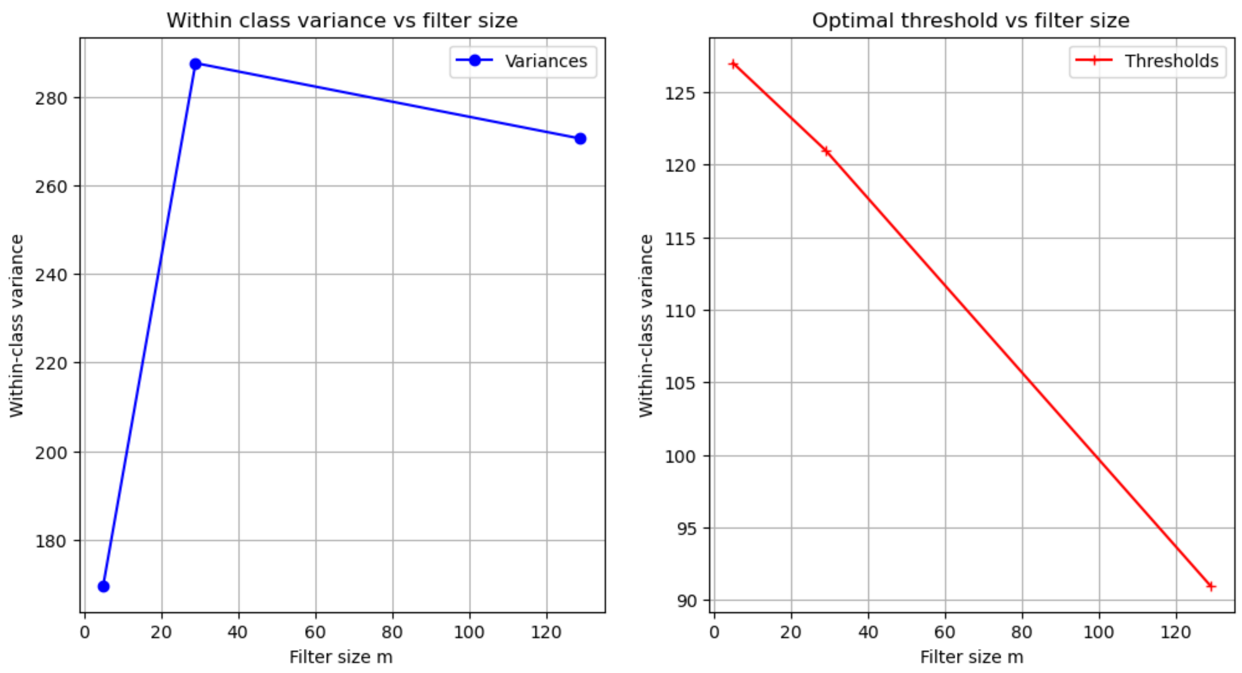

Figure 2: The left graph shows the within-class variance, and the right graph shows the optimal threshold corresponding to each m

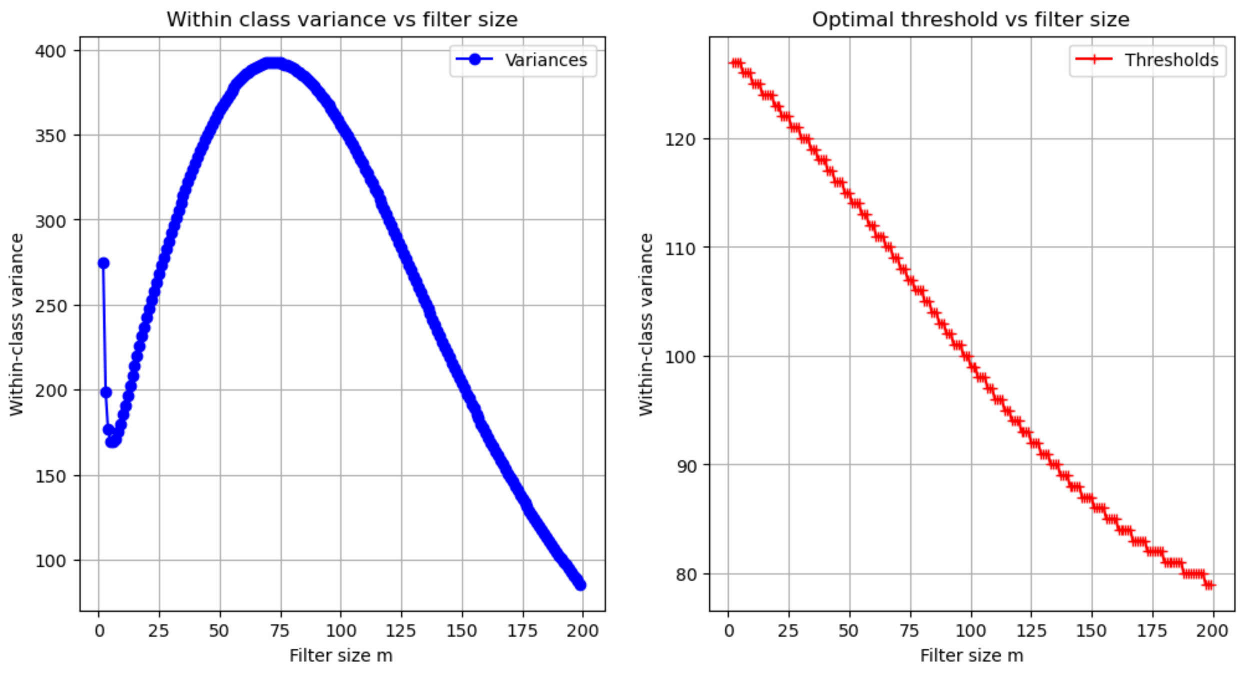

Figure 3: Graph of different m and corresponding σ_w^{2*} and t^*

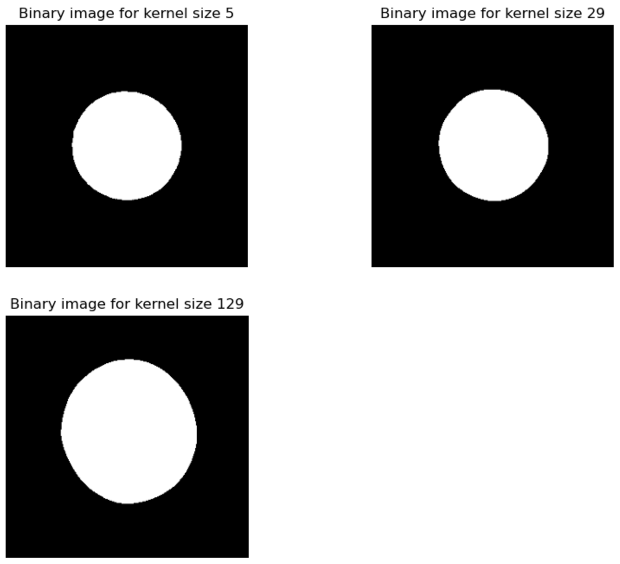
Binary image for kernel size 5
Binary image for kernel size 29
Binary image for kernel size 129

Figure 4: Binary images of filtered images with different filter(kernel) sizes

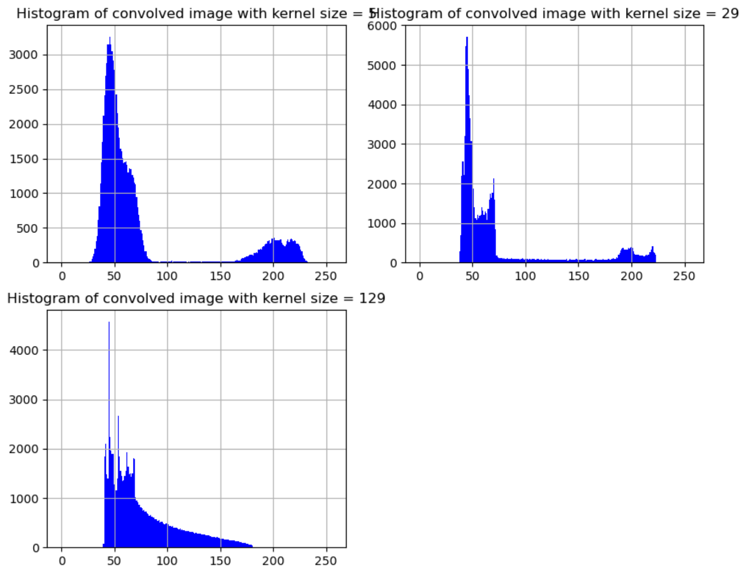

Figure 5: Histogram plots of different filtered images

The σ_w^{2*} decreases for $m \in \{2, 3, 4, 5\}$ and then increases for $m \in \{6, 7... \cdots, 66\}$ and then continously decreases for $m \in \{67, \cdots 200\}$.

2 Scaling and Rotation with Interpolation:

Bilinear Interpolation: Bilinear interpolation is used to compute an output pixel value based on a weighted average of its four nearest pixels. It is mainly used when scaling up an image, rotating it, or performing 3D rendering.

For upsampling and rotation, we can use a method called *Nearest Neighbour Interpolation*, but it tends to produce undesirable artifacts, such as severe distortion of straight edges. For this reason, we don't use this method very frequently. Instead, we turn to *Bilinear Interpolation*, which gives better results at the cost of a little computational overhead. Here is the approach for bilinear interpolation

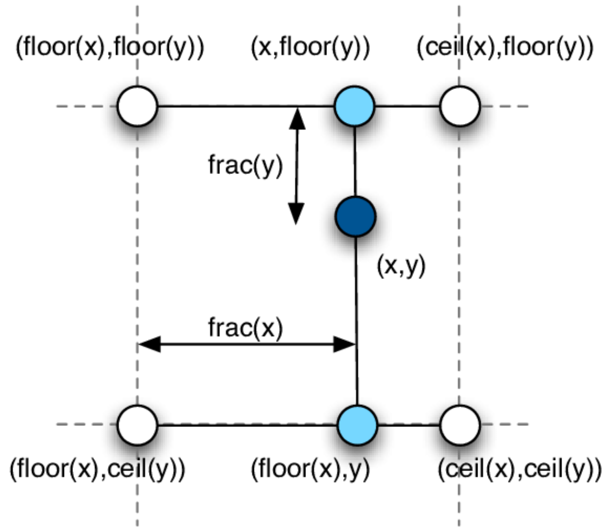

Figure 6: The figure showing the four nearest neighbours, using which we can estimate the intensity at the desired location

Assume the four neighboring pixels are (x_0, y_0), (x_0, y_1), (x_1, y_0), and(x_1, y_1) with corresponding intensities I_a, I_b, I_c, and I_d. Let

$$dx = x - \lfloor x \rfloor, \qquad dy = y - \lfloor y \rfloor$$

denote the fractional parts of x and y. Then the intensity at (x, y) using bilinear interpolation is given by eq. (1)

$$I(x, y) = I_a(1 - dx)(1 - dy) + I_b(1 - dx)dy + I_c dx(1 - dy) + I_d dxdy. \tag{1}$$

Note that each weight in eq. (1) is proportional to the distance from the *opposite* pixel location. For example, the coefficient of I_a is $(1-dx)(1-dy)$, which assigns a higher weight to I_a when (x, y) is closer to (x_0, y_0) and a lower weight as (x, y) moves away. Similarly, the other coefficients $(1 - dx)dy$, $dx(1 - dy)$, and $dxdy$ assign weights to I_b, I_c, and I_d based on their relative proximity to (x, y). Thus, bilinear interpolation performs a weighted average where weights decrease linearly with distance from each neighboring pixel.
Here is the result of upsampling [7] and then rotation on the flower image.

Figure [8] is the result of rotation and then upsampling on the flower image

Later, we computed the difference between the two final images. And the minimum of the difference is 0.00, and the maximum of the difference is 255.00. Also, I will attach the resulting image 9 we got.

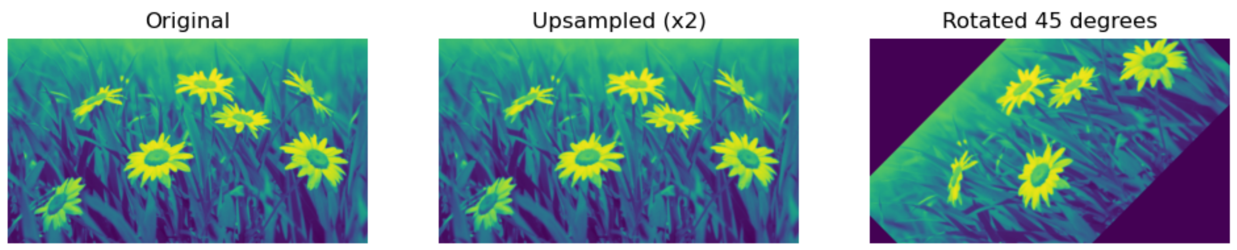

Figure 7: The original image first upsampled ($\times 2$) and rotated by 45^o counter-clockwise

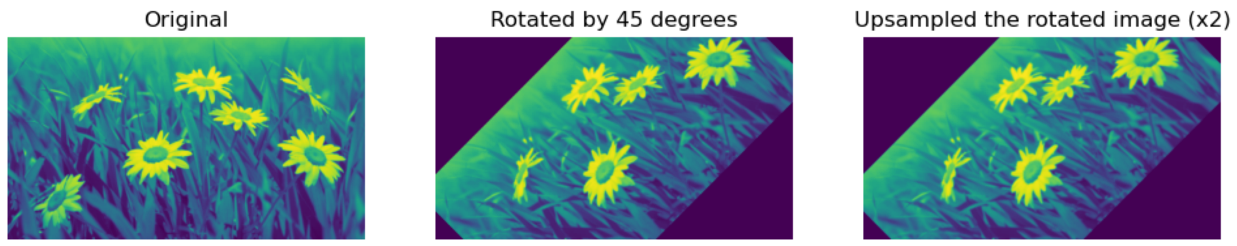

Figure 8: The original image first rotated by 45^o counter-clockwise and then upsampled $\times 2$

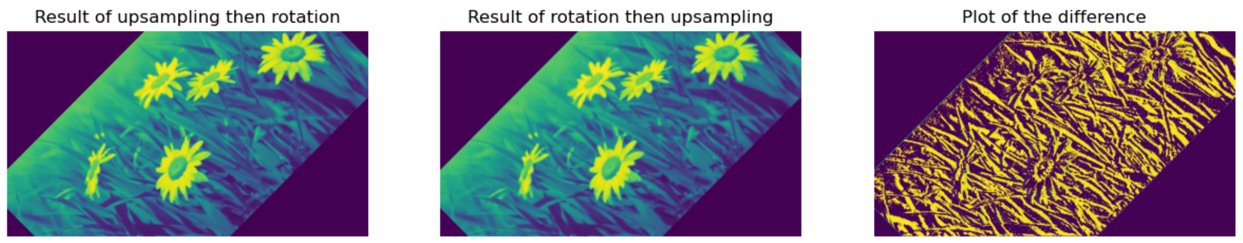
Result of upsampling then rotation
Result of rotation then upsampling
Plot of the difference

Figure 9: The left image shows upsample $\rightarrow$ rotation. The middile image shows rotation $\rightarrow$ upsample and the right image shows the difference of the two.

2.1 Analysis of the results

(a) Upsample and Rotation

- **Upsample ($\times 2$):** Enlarges the image while keeping full rectangular coverage. The resolution is effectively increased.
- **Rotate by 45°** Now the enlarged image is rotated, but the black corners occupy relatively less of the frame, since the image is bigger before rotation. Interpolation smoothness looks better, too.

(b) Rotation and Upsampling

- **Rotate by 45°:** Produces a diamond-shaped image inside a rectangular frame (black corners due to rotation).
- **Upsample ($\times 2$):** Magnifies the rotated image. The black corners also get enlarged, and interpolation artifacts may be more noticeable.

(c) **Order of operations matters:**

- Rotate $\rightarrow$ Upsample: magnifies the rotated frame (including empty black regions). Artifacts become more visible.
- Upsample $\rightarrow$ Rotate: preserves more detail from the original image, and the black borders are proportionally smaller.

The operations are not commutative. The output depends on the order, so the results are **NOT Identical**.

3 Image Sharpening Concept:

For this question, I have used the concept of *unsharp masking* to sharpen the image. To acheive the desired result, we subtract the unsharp (blurred) version of the image. This process is called *unsharp masking*. The steps, we followed are the following

1. Blur the original image
2. Subtract the blurred image from the original (the resulting difference is called the mask.)
3. Add the mask to the original.

$$g_{\text{mask}}(x,y) = f(x,y) - \overline{f}(x,y), \tag{2}$$

where $\overline{f}(x,y)$ denotes the blurred (or smoothed) version of $f(x,y)$.

$$g(x,y) = f(x,y) + kg_{mask}(x,y) \tag{3}$$

$$g(x,y) = f(x,y) + k[f(x,y) - \overline{f}(x,y)]$$

$$g(x,y) = (1+k)f(x,y) - k\overline{f}(x,y) \tag{4}$$

If $k = 1$, then we call it unsharp masking, and if $k > 1$, then high-boost filtering. For $k < 1$, it de-emphasizes the contribution of the unsharp mask. The figure [10] shows how we get a sharpened image using unsharp masking. For, blurring the image, I have a Gaussian filter with different σ

3.1 Analysis of results

- For $p = 0.5$, Edges and fine details are enhanced, but not aggressively. A good balance between clarity and naturalness.
- High-frequency components are fully added back. Edges are crisp, but risk of haloing or noise amplification increases.

3.2 Conclusion

1. With $p = 0$, we don't see any change, or we see the original image.
2. Higher p values can introduce negative pixel values, especially near low-intensity regions.
3. These may cause dark halos around edges, which can be visually objectionable or misleading in diagnostic imaging.

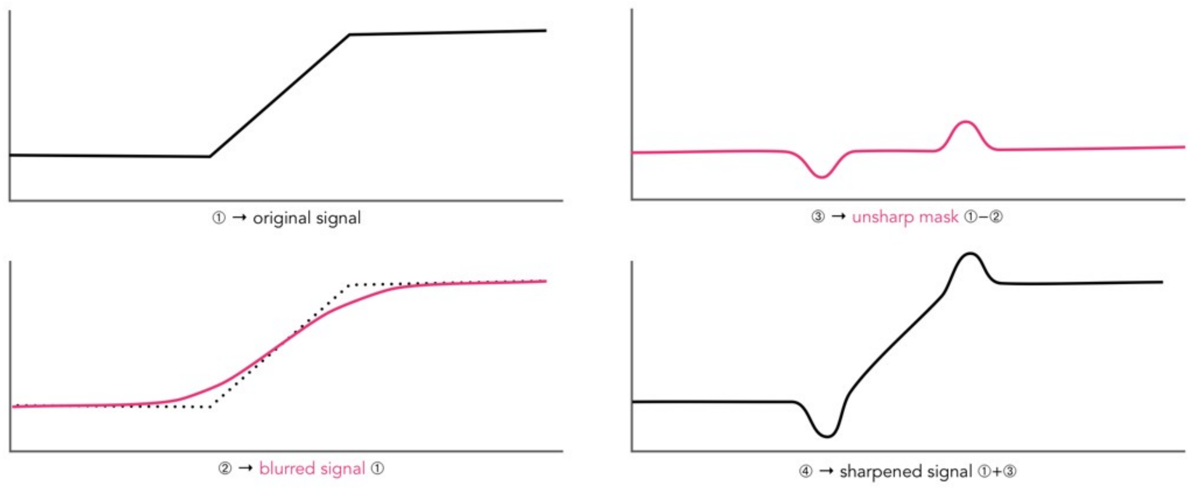

Figure 10: 1D-Illustration of the mechanics of unsharp masking.

I will attach a few results of sharpening.

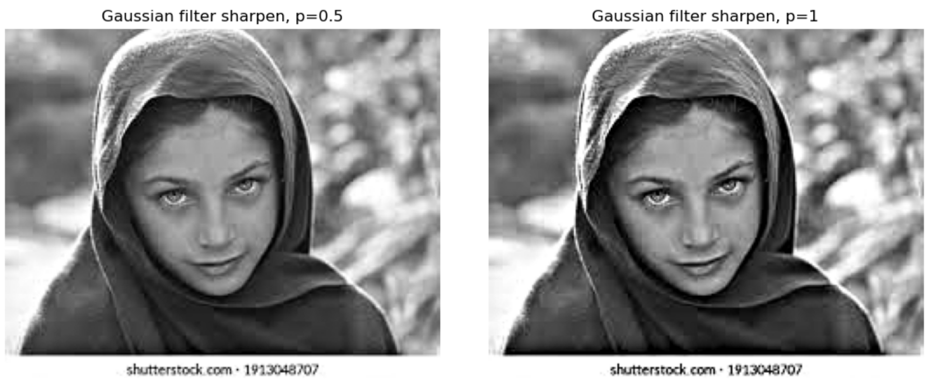
Gaussian filter sharpen, p=0.5
shutterstock.com · 1913048707
Gaussian filter sharpen, p=1
shutterstock.com · 1913048707

Figure 11: Girl image before and after sharpening

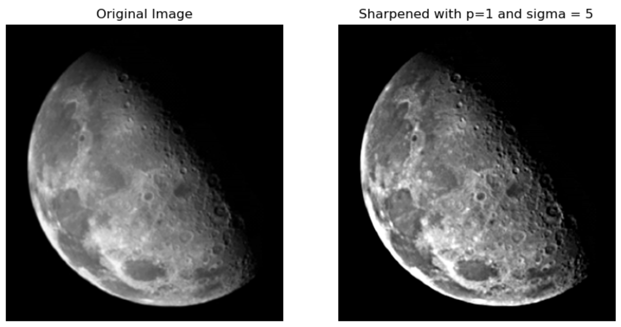

Figure 12: Earth image before and after sharpening

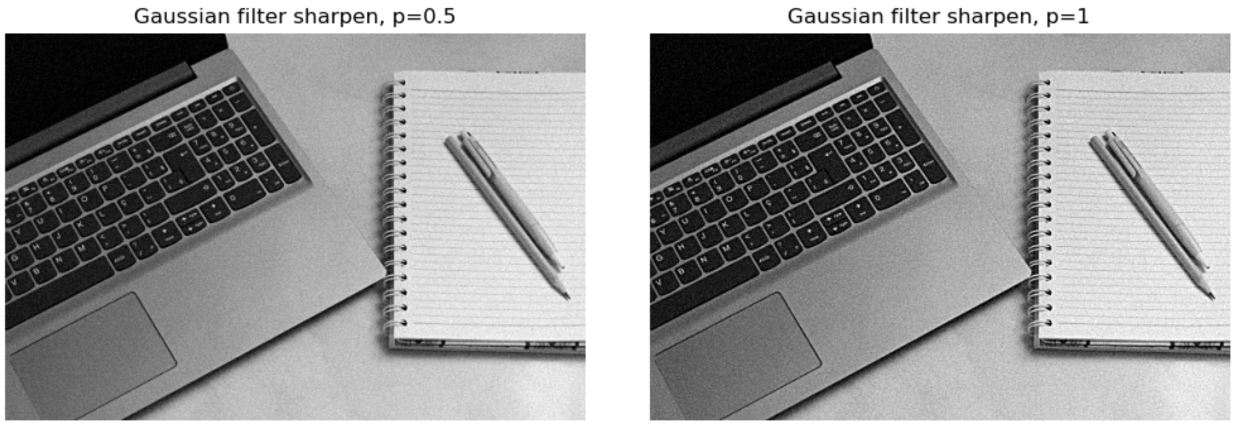

Figure 13: Sharpening with p=0.5 and p=1

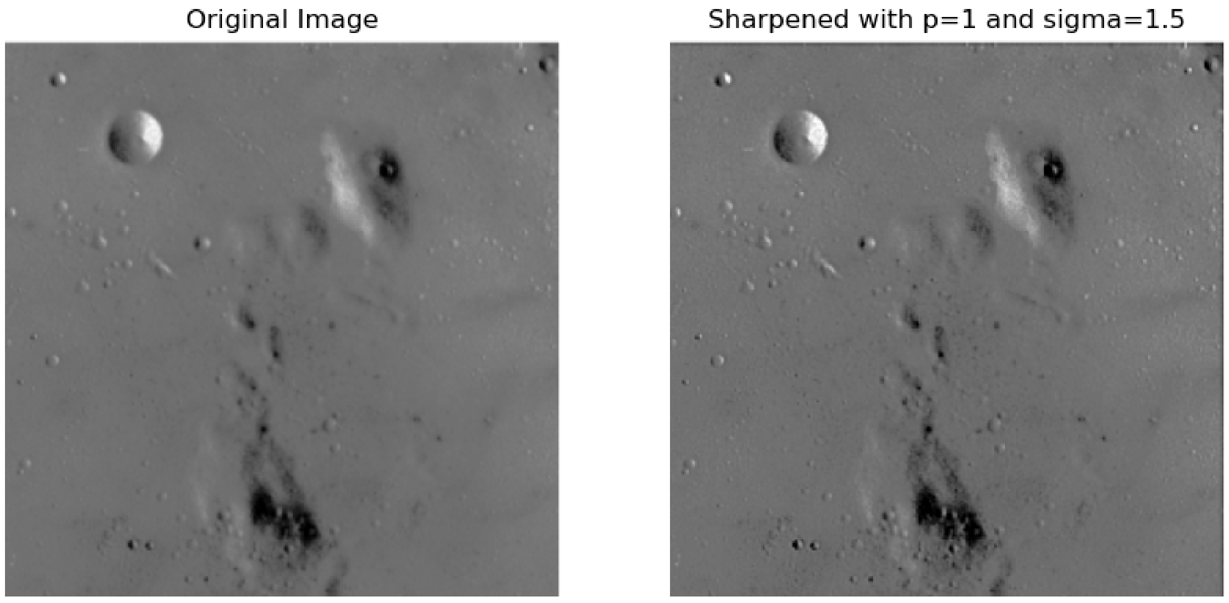

Figure 14: Earth image before and after applying sharpening

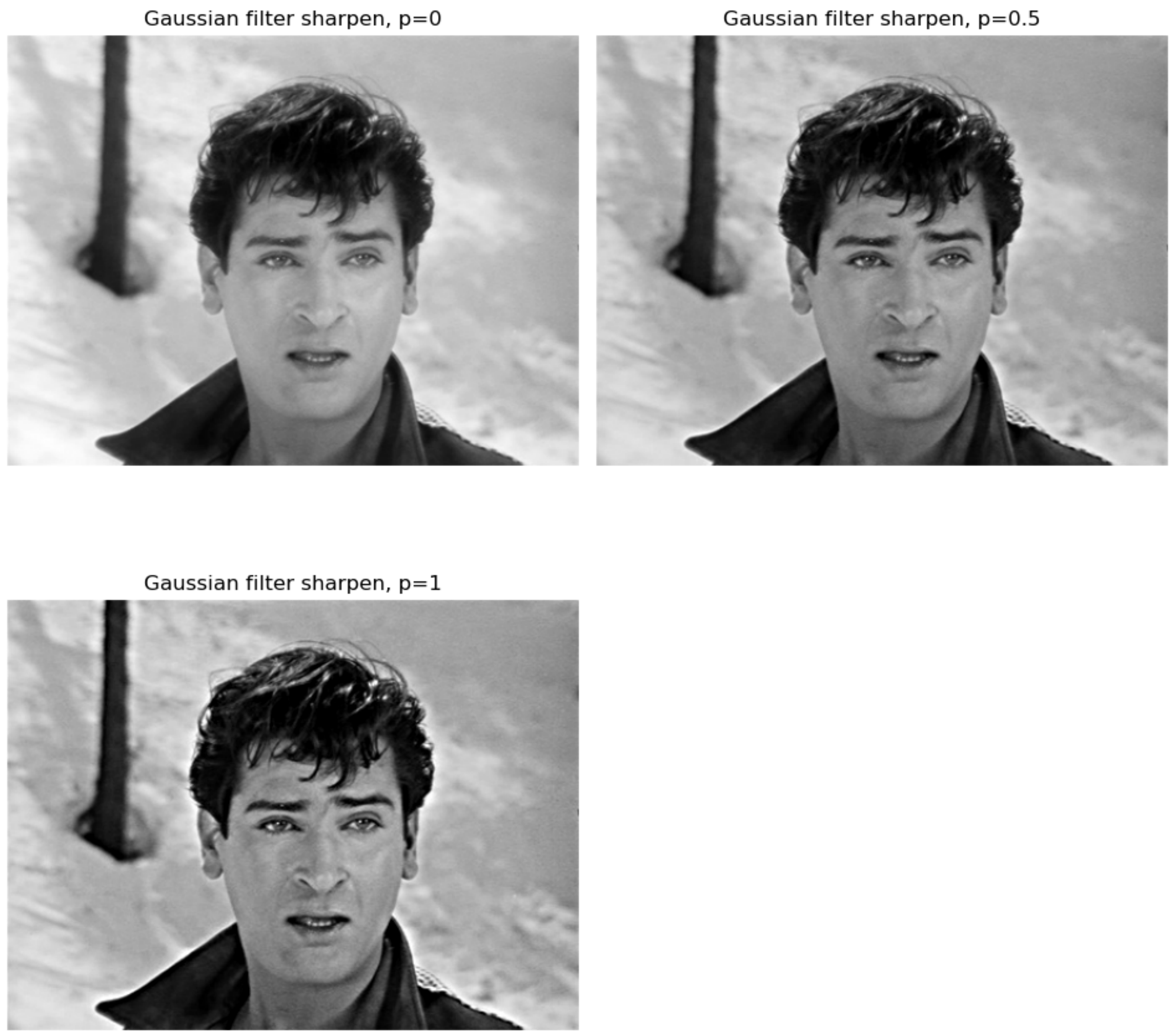
Gaussian filter sharpen, p=0
Gaussian filter sharpen, p=0.5
Gaussian filter sharpen, p=1

Figure 15: Image of an Indian actor before and after applying sharpening

4 Conclusion

1. **Spatial Filtering and Binarization** Through box blurring and Otsu's thresholding, we observed how varying filter sizes influence image smoothness and segmentation quality. The optimal filter size minimized within-class variance σ_w^2, highlighting the trade-off between noise suppression and edge preservation in preprocessing pipelines.

2. **Interpolation and Geometric Transformations** The comparison between scaling-then-rotation versus rotation-then-scaling revealed that interpolation order significantly affects pixel distribution and visual fidelity. The difference image underscored how transformation sequencing can introduce subtle artifacts, which are critical in precision-sensitive domains like medical imaging.

3. **Sharpening with Adjustable Intensity:** The `sharpenAdjust(img, p)` function

demonstrated how controlled enhancement via parameter p can improve edge clarity while managing halo artifacts. Testing across multiple images confirmed that moderate sharpening (e.g., $p = 0.5$) offers a balanced improvement without compromising visual integrity.