Indian Institute of Science

Electronics And Communication Engineering Department

Digital Image Processing

Assignment 4

Ansh Bansal

24250

Date: November 8, 2025

Contents

1 Image Downsampling

1.1 Downsample the city.png image for various factors

Objective

The objective of this experiment is to downsample the grayscale image `city.png` by factors of 2, 4, and 5 without using any library functions such as `cv2.resize()` or `skimage.resize()`. The aim is to observe the visual effects of direct pixel decimation on image quality and analyze the resulting loss of detail and aliasing.

Implementation Summary

The downsampling process was performed by directly selecting every n^{th} pixel in both the horizontal and vertical directions, where n is the downsampling factor. This effectively reduces the spatial resolution of the image by a factor of n in each dimension.

Mathematically, this operation can be expressed as:

$$I_{\text{down}}(x, y) = I(x \cdot n, y \cdot n)$$

where I_{down} is the downsampled image, and I is the original image.

The following Python code snippet demonstrates the implementation:

```
kernel_size = 5
sigma = 2

img_name = "city"
img = load_image(img_name)
titles, imgs = ["Original City Image"], [img]
for factor in [2, 4, 5]:
    imgs.append(img[::factor, ::factor])
    titles.append(f"{img_name} downsampled by factor of {factor}")
stack_images(imgs, "Downsample", titles)
```

This approach simply takes every n^{th} pixel, reducing the image resolution without applying any anti-aliasing filter.

Results and Visualization

Figure 1.1 shows the original city image alongside its downsampled versions obtained by factors of 2, 4, and 5.

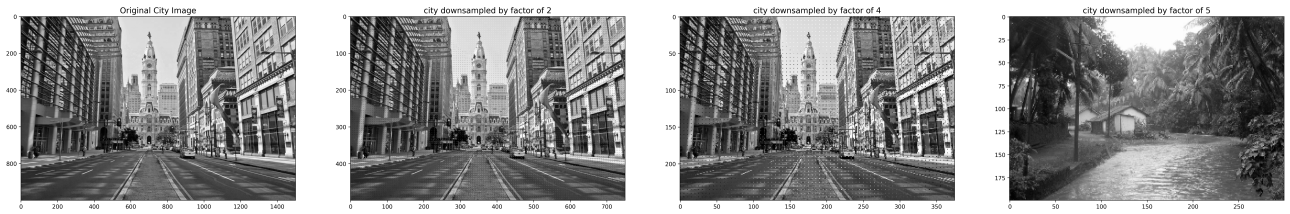

Figure 1.1: Original image and its downsampled versions by factors of 2, 4, and 5.

Inference

As the downsampling factor increases, the image resolution decreases significantly. The following observations can be made:

- For a downsampling factor of 2, the image retains most of its sharpness and structural details.
- For factors 4 and 5, a strong degradation in visual quality is observed, with fine textures and edges becoming indistinct.
- The higher downsampling factors introduce noticeable **aliasing artifacts**, such as jagged edges and moiré patterns.

These effects occur because the image was directly subsampled without any low-pass (anti-aliasing) filtering. According to the Nyquist sampling theorem, high-frequency components should be removed prior to sampling to avoid spectral overlap. The absence of such filtering causes high-frequency details to fold into the lower frequencies, producing visually unpleasant artifacts.

This issue is addressed in the subsequent experiment by introducing Gaussian low-pass filtering before downsampling.

1.2 Comparision With the Sckit-image

Objective

The objective of this experiment is to examine the effect of applying a **Gaussian low-pass filter (LPF)** before downsampling an image. The aim is to reduce aliasing artifacts by suppressing high-frequency components of the image prior to sampling. A spatial-domain Gaussian filter with a kernel size of 5×5 and $\sigma = 2$ is used. The results are compared with the output from a standard library function (`scikit-image.resize`) which internally applies anti-aliasing.

Implementation Summary

The image `city.png` was first converted to grayscale and then blurred using a 2D Gaussian kernel defined as:

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

The kernel was normalized such that $\sum_{x,y} G(x, y) = 1$. The filtering was implemented in the spatial domain using OpenCV's `filter2D()` function. For each downsampling factor $f \in \{2, 4, 5\}$, the blurred image was reduced in size by selecting every f^{th} pixel in both spatial directions. The following procedure was applied:

1. Apply Gaussian blur to the input image using kernel size 5 and $\sigma = 2$.
2. Downsample by taking every f^{th} pixel in both x and y directions.
3. Generate a reference downsampled image using `scikit-image.resize()` with `anti_aliasing=True`.
4. Compute Mean Squared Error (MSE) between the custom and library downsampled images:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (I_1(i) - I_2(i))^2$$

Results and Visualization

Figure 1.2 shows the blurred image and its downsampled versions for factors 2, 4, and 5. Visual comparison with the unfiltered downsampling results reveals a significant reduction in aliasing artifacts.

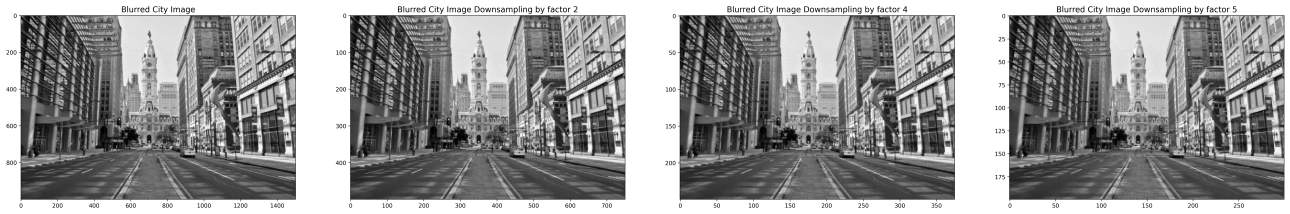

Figure 1.2: Blurred City Image downsampled by factors of 2, 4, and 5.

The difference images and MSE values comparing the custom downsampled results with the `scikit-image` anti-aliased resize function are shown in Figure 1.3.

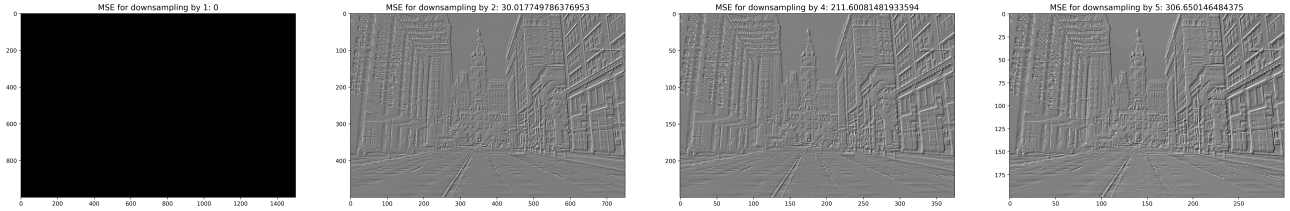

Figure 1.3: Difference images between custom Gaussian-prefiltered downsampling and scikit-image resize outputs. Lower MSE values indicate higher similarity.

Typical MSE values observed were:

Downsampling Factor	MSE
2	30.02
4	211.61
5	306.65

Inference

The application of Gaussian low-pass filtering before downsampling significantly reduces aliasing. Fine textures and sharp edges, which previously produced jagged or moiré patterns in the directly downsampled images, are now smoother and more visually consistent. This occurs because the Gaussian filter suppresses high-frequency components that cannot be represented after subsampling.

Moreover, the results obtained from the custom implementation closely match those from the `scikit-image` function with anti-aliasing enabled, indicating that a properly chosen Gaussian kernel can effectively approximate standard library-based anti-aliasing methods. The small MSE values confirm this quantitative similarity.

Conclusion

Gaussian pre-filtering serves as an effective method to prevent aliasing in image downsampling tasks. The choice of kernel size and σ directly controls the amount of smoothing, with ($k = 5, \sigma = 2$) providing a good trade-off between preserving edge sharpness and eliminating high-frequency distortions.

1.3 Finding Optimal Kernel Size and Variance

Objective

The objective of this experiment is to determine the optimal Gaussian filter parameters — the **kernel size** (k) and the **standard deviation** (σ) — that minimize the **Mean Squared Error (MSE)** between the custom Gaussian-blurred and downsampled image and the output obtained using the `scikit-image.resize` function with anti-aliasing enabled. The experiment focuses on a downsampling factor of 5, where the impact of aliasing is most significant.

Implementation Summary

A continuous optimization approach was adopted to find the pair (k, σ) that minimizes the MSE between the two images. The following procedure was followed:

1. The input image `city.png` was converted to grayscale and normalized.
2. A Gaussian kernel was generated for given parameters (k, σ) and applied to the image using spatial filtering with `cv2.filter2D`.
3. The resulting blurred image was downsampled by a factor of 5 using pixel decimation.
4. The same image was downsampled using the library function `resize()` from `scikit-image` with `anti_aliasing=True`.
5. The mean squared error between the two downsampled outputs was computed as:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (I_1(i) - I_2(i))^2$$

6. The optimal parameters were estimated using the `scipy.optimize.minimize` function with the L-BFGS-B algorithm and appropriate bounds for $k \in [3, 101]$ and $\sigma \in [0.1, 10]$.

Results and Visualization

The optimization converged successfully to the following values:

Parameter	Optimal Value
Kernel size (k)	15
Sigma (σ)	5.0
Minimum MSE	80.552

Figure 1.4 compares the custom Gaussian-blurred image, the `scikit-image` blurred output, and their difference map.

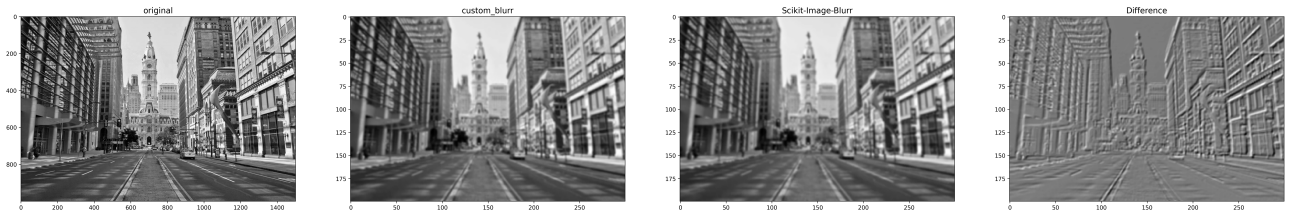

Figure 1.4: Comparison between original image, custom Gaussian blur, `scikit-image` blur, and their difference.

Inference

The optimal parameters ($k = 15, \sigma = 5$) produce the closest match between the custom downsampling and the anti-aliased library downsampling. A larger kernel and moderate σ ensure sufficient smoothing to suppress high-frequency components before sampling, thus reducing aliasing. The difference image shows only minor edge variations, confirming a close similarity in the frequency response of the two filters. The relatively low MSE value (80.55) further validates the accuracy of the optimized parameters.

This experiment demonstrates that carefully tuning the Gaussian filter parameters allows the manual downsampling process to emulate the results of standard anti-aliased resizing functions effectively.

2 Edge Detection

2.1 Detect Edges Using Gradient based method and Laplacian of Gaussian Operator

Objective

The objective of this experiment is to perform and analyze edge detection on a given set of grayscale images — `Checkerboard.png`, `Coins.png`, `MainBuilding.png`, and `flowers.png`. Two different approaches were implemented and compared:

1. **Gradient-based method** using the Sobel operator.
2. **Laplacian of Gaussian (LoG)** method for combined smoothing and edge detection.

The goal is to understand how gradient-based and second-derivative-based techniques differ in edge representation, noise sensitivity, and feature localization.

Implementation Summary

The experiment was conducted using custom implementations of Sobel and Laplacian of Gaussian operators. For each image, edge detection was performed under the following conditions:

- Without noise (clean image).
- With Gaussian smoothing and/or noise addition (tested separately).

The Sobel operator computes intensity gradients in both x and y directions, which are then combined to obtain the gradient magnitude:

$$G_x = I * S_x, \qquad G_y = I * S_y, \qquad G = \sqrt{G_x^2 + G_y^2}$$

where S_x and S_y are Sobel masks and $*$ denotes convolution.

The Laplacian of Gaussian (LoG) operator detects edges by finding zero-crossings in the second derivative of a Gaussian-smoothed image. The LoG kernel is defined as:

$$\nabla^2 G(x, y) = \frac{(x^2 + y^2 - 2\sigma^2)}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The implementation code is shown below:

```
img_names = ["Checkerboard", "Coins", "flowers", "MainBuilding"]

def edge_detection(image, type, sigma=1.0):
    if type == "gradient":
        sobel_x = np.array([[ -1, 0, 1],
                            [ -2, 0, 2],
                            [ -1, 0, 1]])
        sobel_y = np.array([[ 1,  2,  1],
                            [ 0,  0,  0],
                            [-1, -2, -1]])
        grad_x = cv2.filter2D(image, -1, sobel_x)
        grad_y = cv2.filter2D(image, -1, sobel_y)
        mag = np.sqrt(grad_x**2 + grad_y**2)
    else:
        size = int(2 * np.ceil(3 * sigma) + 1)
        ax = np.linspace(-(size // 2), size // 2, size)
        xx, yy = np.meshgrid(ax, ax)
        term = (xx**2 + yy**2)
        kernel = ((term - 2 * sigma**2) / sigma**4) * np.exp(-term / (2 * sigma**2))
        kernel /= (2 * np.pi * sigma**2)
        kernel -= np.mean(kernel)
        mag = cv2.filter2D(image.astype(float), -1, kernel)
    res = ((mag - np.min(mag)) / (np.max(mag) - np.min(mag))) * 255
    return res

def run(add_noise=False, smooth=False):
    titles = ['Original', 'Sobel', 'LoG']
    for name in img_names:
        imgs = []
        img = load_image(name)
        if add_noise:
            img = add_gaussian_noise(img, 0, 25)
        if smooth:
            img = gaussian_smooth(img, 7, 3)
        imgs.append(img)
        imgs.append(edge_detection(img, "gradient"))
        imgs.append(edge_detection(img, "Laplace"))
        stack_images(imgs, name + "_edge_detected", titles)
```

The function `run()` was used to execute edge detection for all four images, and the results were saved for visualization.

Results and Visualization

Figures 2.1 to 2.4 illustrate the edge maps obtained for all images using both Sobel and Laplacian of Gaussian methods.

Inference

The following key observations were made:

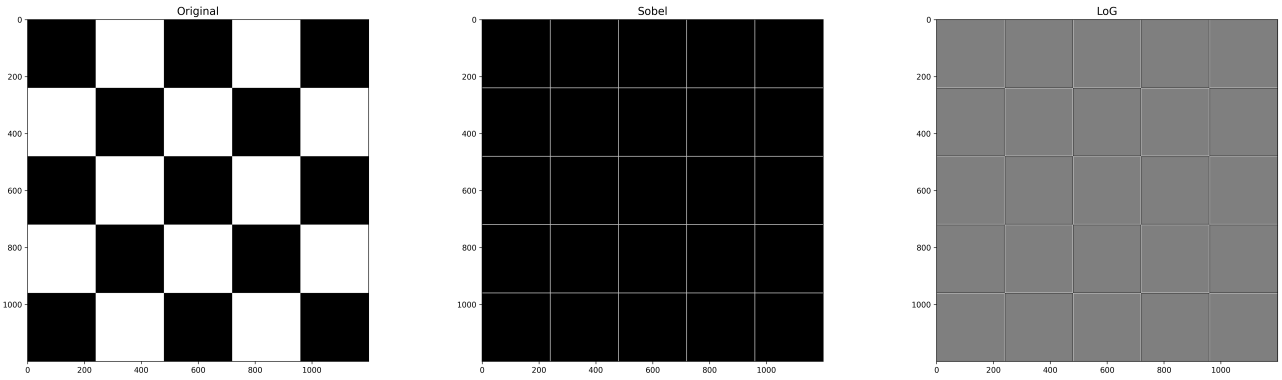

Figure 2.1: Edge detection results for `Checkerboard.png`.

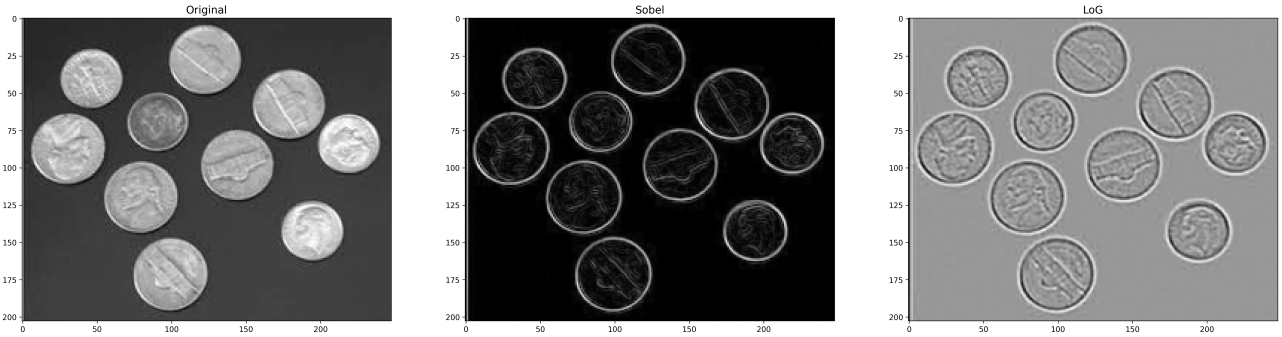

Figure 2.2: Edge detection results for `Coins.png`.

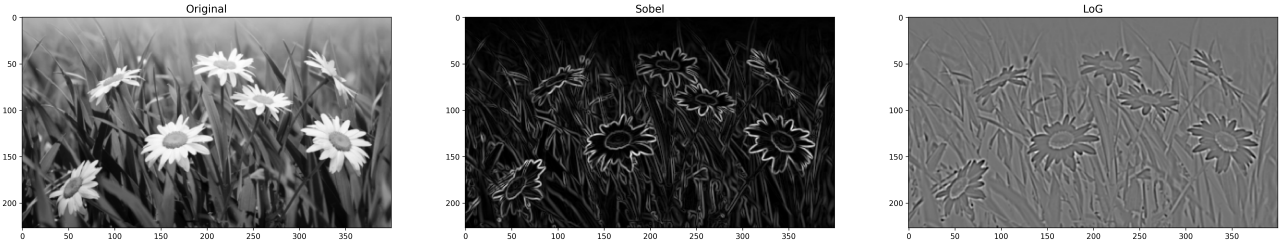

Figure 2.3: Edge detection results for `flowers.png`.

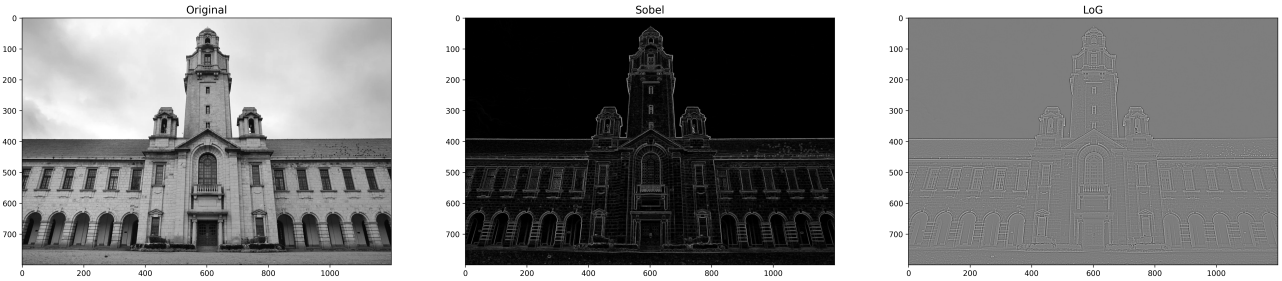

Figure 2.4: Edge detection results for `MainBuilding.png`.

- The **Sobel operator** (gradient-based) effectively detects prominent edges and object boundaries but is sensitive to noise and may produce double edges along intensity transitions.
- The **Laplacian of Gaussian (LoG)** operator provides smoother and more continuous edges due to the initial Gaussian smoothing step. However, it may slightly blur fine edge details.

- For noisy images, LoG performs better because of its inherent smoothing property, while Sobel tends to amplify noise gradients.
- Overall, the two operators complement each other: Sobel offers sharper edge localization, while LoG offers better noise suppression and global edge continuity.

In conclusion, the Laplacian of Gaussian provides a more robust edge representation for real-world images where noise and illumination variations are present, whereas gradient-based methods like Sobel are better suited for synthetic images such as the checkerboard pattern with distinct edge transitions.

2.2 Comparing Edge Detection after adding Gaussian Noise and Gaussian Noise

2.2.1 After adding Only Noise

Objective

The objective of this experiment is to evaluate the robustness of different edge detection methods — specifically the **Sobel operator** and the **Laplacian of Gaussian (LoG)** — under noisy conditions. Four grayscale images (`Checkerboard.png`, `Coins.png`, `flowers.png`, and `MainBuilding.png`) were used. The aim is to observe how the presence of Gaussian noise affects the quality, continuity, and accuracy of detected edges.

Implementation Summary

To analyze the effect of noise, zero-mean Gaussian noise with a standard deviation of 25 was added to each image before performing edge detection. The following steps were followed:

1. Load each image in grayscale and normalize pixel values.
2. Add Gaussian noise defined by:

$$I_{noisy}(x, y) = I(x, y) + \mathcal{N}(0, \sigma^2)$$

   where $\sigma = 25$.
3. Apply the Sobel operator to detect gradient-based edges.
4. Apply the Laplacian of Gaussian operator for smoothed second-derivative edge detection.
5. Normalize the resulting edge maps for visualization.

The implementation was based on the following code:

```
print("Edge Detection for Noisy Images")
run(add_noise = True, smooth = False)
```

Here, the function `run()` executes the edge detection pipeline for all images in the dataset, generating Sobel and LoG edge maps for each noisy image.

Results and Visualization

The results for noisy images are shown in Figures 2.5 to 2.8. Each figure displays the original noisy image, the Sobel edge map, and the Laplacian of Gaussian (LoG) edge map.

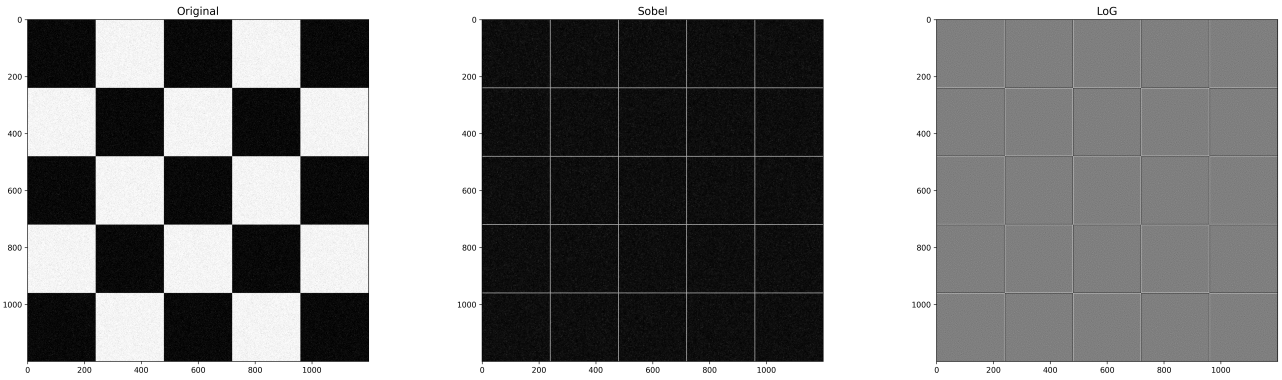

Figure 2.5: Edge detection for noisy `Checkerboard.png`.

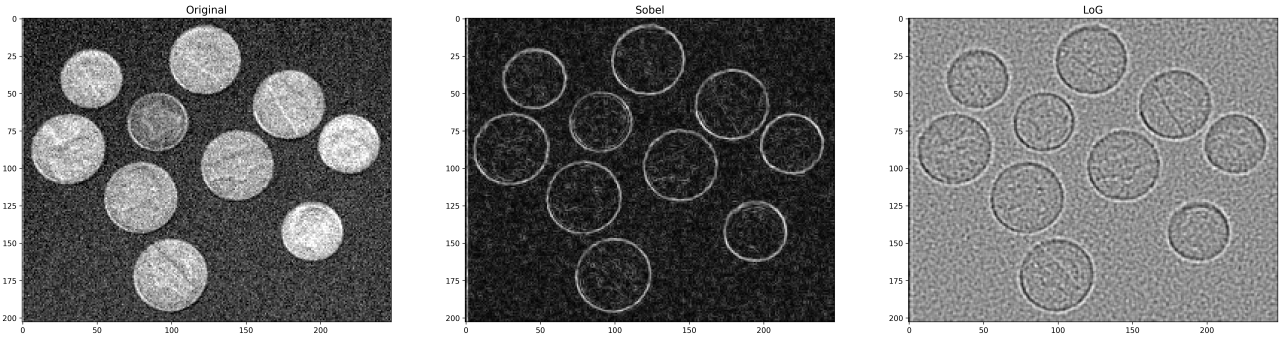

Figure 2.6: Edge detection for noisy `Coins.png`.

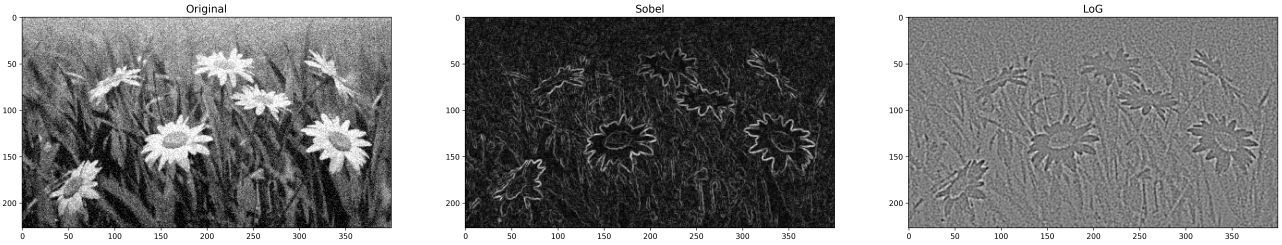

Figure 2.7: Edge detection for noisy `flowers.png`.

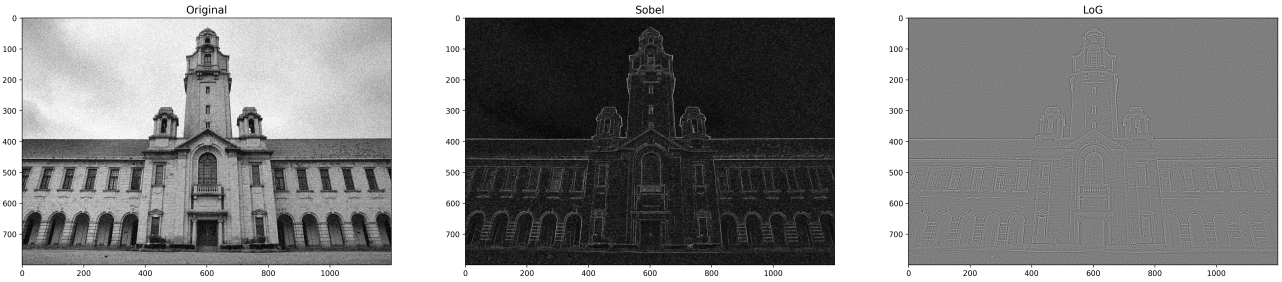

Figure 2.8: Edge detection for noisy `MainBuilding.png`.

Inference

From the experimental results, the following observations can be made:

- The **Sobel operator** is highly sensitive to noise. Random intensity variations due to Gaussian noise produce spurious edges, particularly in smooth regions. The resulting edge maps appear cluttered and contain false detections.
- The **Laplacian of Gaussian (LoG)** operator demonstrates better robustness to noise.

The Gaussian smoothing stage effectively attenuates high-frequency noise, resulting in cleaner and more continuous edges.

- In images such as `Coins.png` and `MainBuilding.png`, the LoG output preserves the major structural edges while suppressing minor texture noise.
- For synthetic images like `Checkerboard.png`, both operators perform reasonably well, though Sobel still shows minor random edge responses.
- In natural images (`flowers.png`), where texture is abundant, LoG provides a visually more coherent edge representation.

In conclusion, the Laplacian of Gaussian operator is more effective than the Sobel operator for edge detection in noisy conditions because it inherently combines smoothing with edge detection. This experiment demonstrates the importance of pre-smoothing for robust edge extraction in real-world scenarios.

2.2.2 After Only Smoothing

Objective

The objective of this experiment is to investigate the effect of **Gaussian smoothing** on edge detection performance using the **Sobel** and **Laplacian of Gaussian (LoG)** operators. The goal is to understand how pre-smoothing influences the sharpness, continuity, and accuracy of detected edges, particularly in images with fine textures or varying intensity gradients.

Implementation Summary

In this experiment, all input grayscale images — `Checkerboard.png`, `Coins.png`, `flowers.png`, and `MainBuilding.png` — were first smoothed using a Gaussian filter before applying edge detection. The Gaussian smoothing step is defined as:

$$I_s(x, y) = I(x, y) * G(x, y; \sigma)$$

where $G(x, y; \sigma)$ is a 2D Gaussian kernel of size 7×7 and $\sigma = 3$.

The Sobel operator was then applied to the smoothed images to compute gradient-based edges, while the Laplacian of Gaussian (LoG) operator was used to extract zero-crossing edges.

The experiment was implemented using the following function call:

```
print("Edge Detection for Smooth Images")
run(add_noise = False, smooth = True)
```

Here, the parameter `smooth=True` ensures that each image is convolved with a Gaussian kernel prior to edge detection. The same `run()` function was used to generate edge maps for both Sobel and LoG methods.

Results and Visualization

The results of edge detection on Gaussian-smoothed images are shown in Figures 2.9 to 2.12. Each figure presents the original smoothed image, Sobel edge map, and LoG edge map for visual comparison.

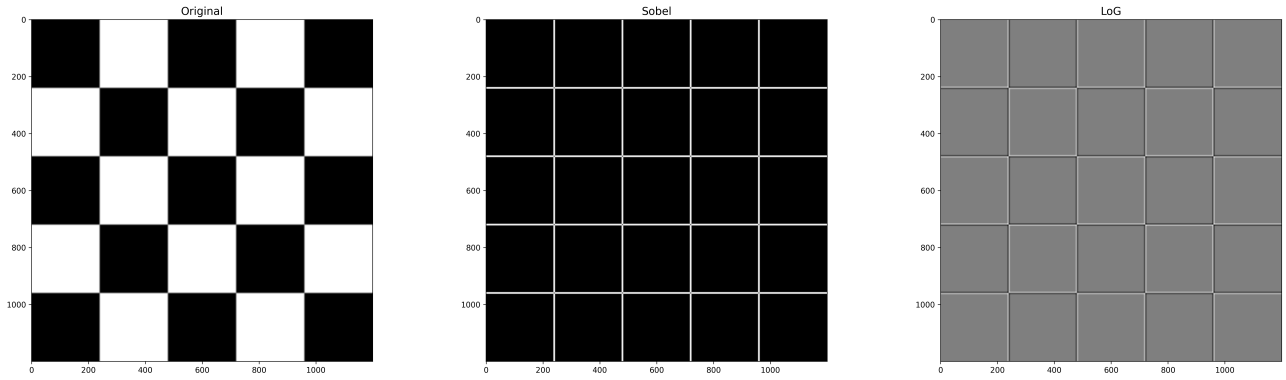

Figure 2.9: Edge detection results for smoothed `Checkerboard.png`.

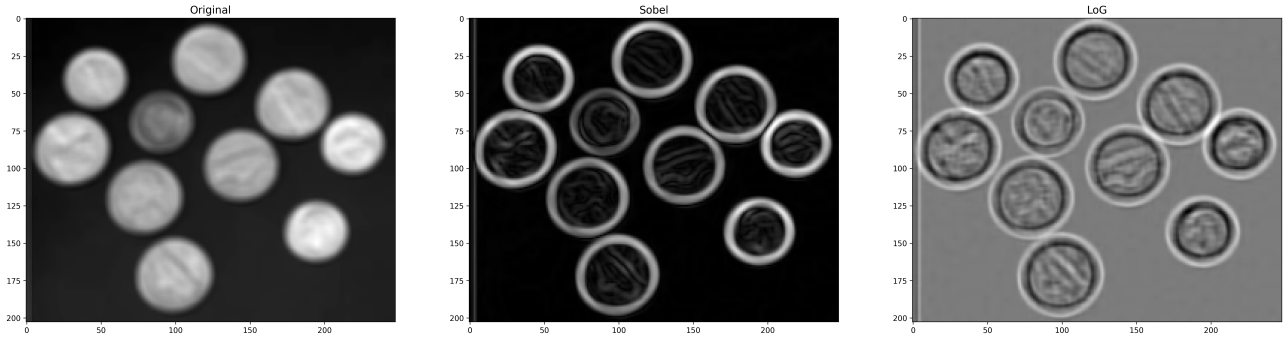

Figure 2.10: Edge detection results for smoothed `Coins.png`.

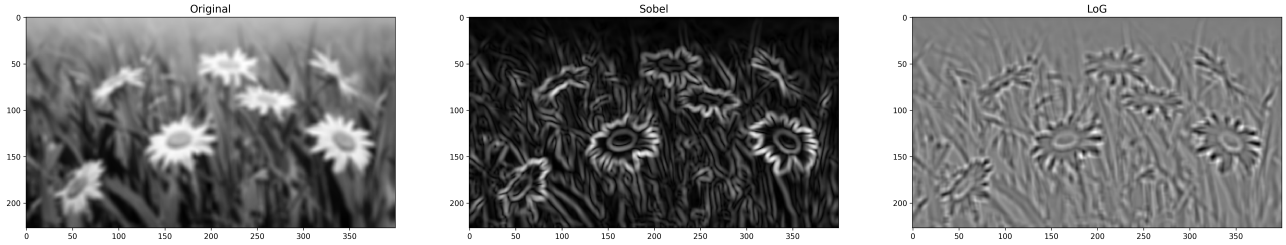

Figure 2.11: Edge detection results for smoothed `flowers.png`.

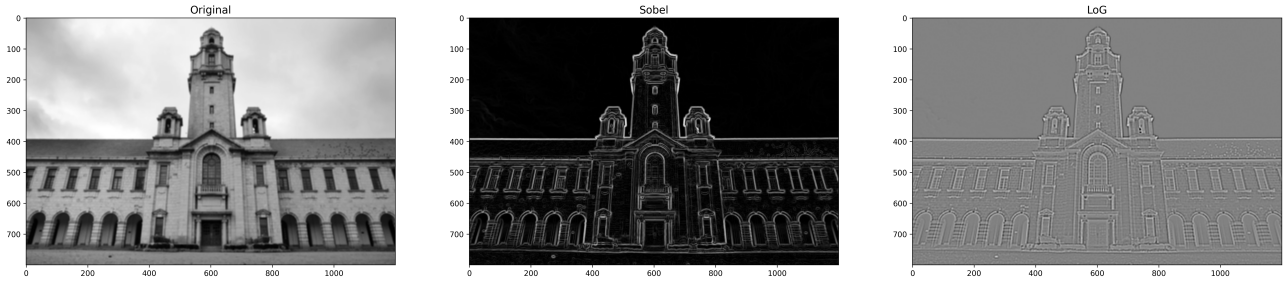

Figure 2.12: Edge detection results for smoothed `MainBuilding.png`.

Inference

From the results, the following observations were made:

- Gaussian smoothing significantly reduces high-frequency noise and texture details, leading to smoother and more coherent edge maps.
- In the `Coins.png` image, the circular boundaries are detected more clearly and without the noisy fluctuations observed in the unsmoothed version.

- The **Sobel operator** produces well-defined and localized edges after smoothing, although some weaker details are suppressed due to blurring.
- The **Laplacian of Gaussian (LoG)** maintains good edge localization and produces smooth, continuous contours, especially around large-scale features such as building boundaries.
- In highly textured images such as `flowers.png`, smoothing removes small details, leaving only dominant object outlines, improving visual clarity.

Overall, Gaussian smoothing before edge detection improves edge quality by suppressing noise and reducing unwanted fine texture responses. However, it introduces a trade-off: some finer edges are blurred or completely lost. Therefore, the choice of σ in Gaussian smoothing must balance noise suppression and edge sharpness.

2.2.3 Adding both Noise and Smoothing

Objective

The objective of this experiment is to study the combined effect of **Gaussian noise** and **Gaussian smoothing** on edge detection performance using two techniques — **Sobel** and **Laplacian of Gaussian (LoG)**. This case aims to simulate real-world conditions where images are both noisy and pre-processed with smoothing filters to reduce noise before applying edge detection.

Implementation Summary

In this setup, Gaussian noise was first added to each image to simulate a noisy environment, followed by Gaussian smoothing to reduce noise. The smoothed-noisy images were then processed using Sobel and LoG operators for edge detection.

The following steps were performed:

1. Gaussian noise was added to each image with $\mu = 0$ and $\sigma = 25$.
2. Gaussian smoothing was applied with a kernel size of 7×7 and standard deviation $\sigma = 3$.
3. Sobel operator was used to compute gradient-based edges.
4. Laplacian of Gaussian (LoG) was used for smoothed second-derivative edge detection.

The implementation was executed using:

```
print("Edge Detection for Noisy and Smooth Images")
run(add_noise = True, smooth = True)
```

Here, the argument `add_noise=True` adds Gaussian noise, and `smooth=True` applies Gaussian blurring prior to edge detection.

Results and Visualization

Figures 2.13 to 2.16 present the results of edge detection on images with both noise and smoothing applied. Each figure shows the original (noisy + smoothed) image, followed by the Sobel and LoG edge maps.

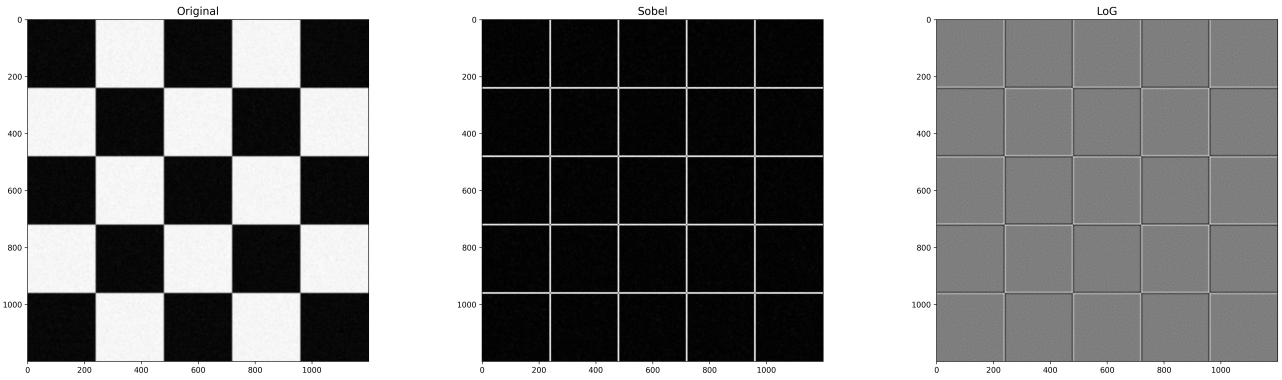

Figure 2.13: Edge detection results for noisy and smoothed `Checkerboard.png`.

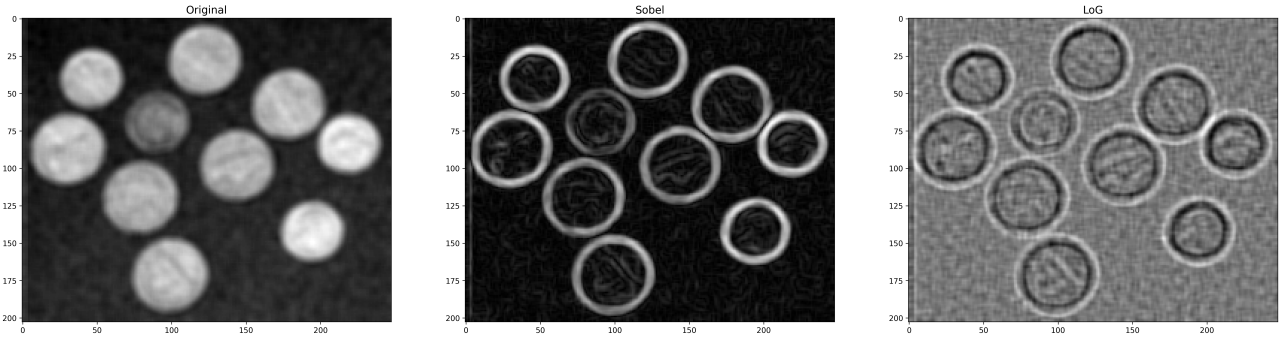

Figure 2.14: Edge detection results for noisy and smoothed `Coins.png`.

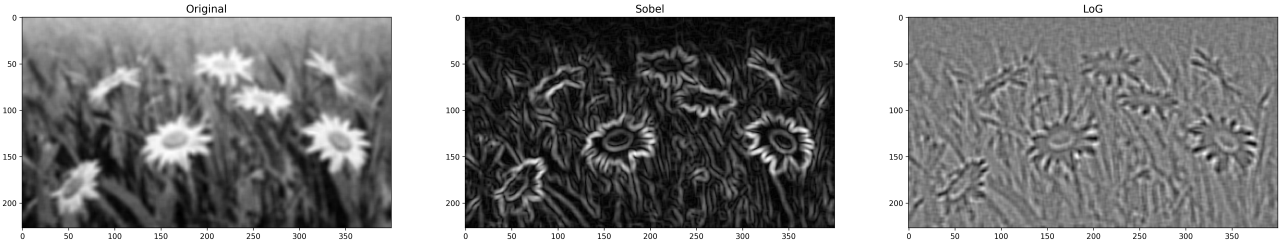

Figure 2.15: Edge detection results for noisy and smoothed `flowers.png`.

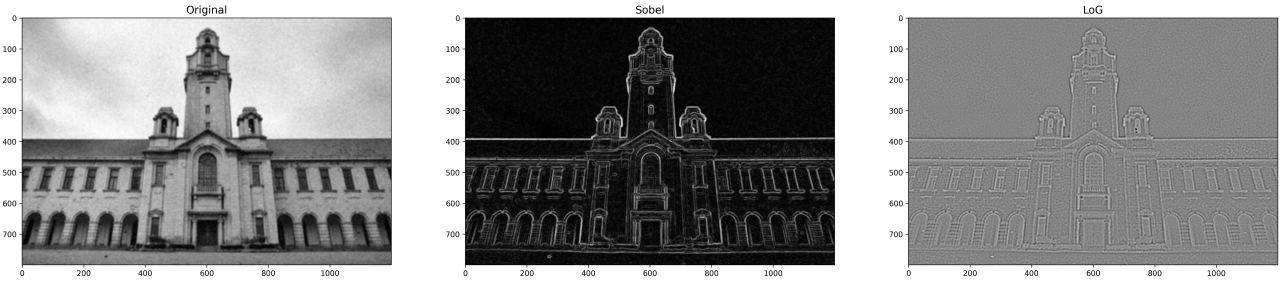

Figure 2.16: Edge detection results for noisy and smoothed `MainBuilding.png`.

Inference

The following key inferences were made based on the experimental results:

- When both noise and smoothing are present, the Gaussian smoothing effectively reduces high-frequency noise, producing cleaner edge maps.
- The **Sobel operator** performs significantly better than in the noisy-only case, as smoothing suppresses noise-induced false edges.

- The **Laplacian of Gaussian (LoG)** method continues to show strong resilience to noise, producing smooth, continuous edge contours.
- In `Coins.png` and `MainBuilding.png`, the combination of noise reduction and edge enhancement preserves structural features while minimizing spurious edges.
- However, some fine edges are lost due to smoothing, especially in `flowers.png`, where delicate texture edges are suppressed.

In conclusion, applying Gaussian smoothing before edge detection is an effective strategy for noisy images. It enhances the robustness of both Sobel and LoG operators by suppressing random fluctuations while maintaining meaningful edge structures. Among the two, LoG still provides superior visual continuity and robustness to noise.

2.3 Comparative Discussion on Edge Detection Performance

This section presents a qualitative comparison of edge detection performance across different experimental conditions. The analysis considers three key comparisons:

1. Clean vs. noisy images,
2. Gradient-based (Sobel) vs. Laplacian of Gaussian (LoG) edge detectors,
3. With and without Gaussian smoothing.

1. Clean vs. Noisy Images:

- In clean images, both Sobel and LoG operators produce well-defined and sharp edges. Structural boundaries such as the coin edges in `Coins.png` and building contours in `MainBuilding.png` are clearly visible without significant distortion.
- When Gaussian noise is introduced, edge quality degrades noticeably. The **Sobel operator**, which depends directly on image gradients, becomes highly sensitive to noise and detects false edges across smooth regions, especially visible in `flowers.png`.
- In contrast, the **LoG operator** exhibits greater robustness under noise. Although minor fluctuations appear, the overall edges remain smoother and more consistent.
- In synthetic patterns like `Checkerboard.png`, both operators still perform adequately under noise due to strong contrast transitions. However, the Sobel operator shows slight random pixel-level variations, while LoG preserves clean boundary outlines.

2. Gradient-based (Sobel) vs. Laplacian of Gaussian (LoG) Edge Detectors:

- The **Sobel operator** computes the first derivative of the image, highlighting areas of rapid intensity change. It tends to emphasize fine texture details and is effective for high-contrast boundaries, such as object edges in `Coins.png`.
- However, its reliance on local gradients makes it more prone to false detections in noisy regions. Without prior smoothing, Sobel results appear fragmented and cluttered.
- The **Laplacian of Gaussian (LoG)** operator combines Gaussian smoothing with second-derivative edge detection. This approach enhances structural edges while suppressing noise.

- In all experiments, LoG consistently produced smoother and more continuous edge maps. In `MainBuilding.png`, LoG successfully delineates architectural edges while maintaining uniform intensity, outperforming Sobel in stability and edge continuity.
- For natural images such as `flowers.png`, LoG effectively captures the overall flower outlines but slightly loses the finer petal details compared to Sobel.

3. With and Without Gaussian Smoothing:

- Applying Gaussian smoothing before edge detection significantly improves robustness to noise by attenuating high-frequency variations. In `Coins.png` and `MainBuilding.png`, this results in well-defined boundaries with fewer spurious responses.
- For clean images, smoothing slightly blurs fine details but helps in producing more continuous and visually appealing edges.
- In noisy images, the difference is more pronounced — smoothing suppresses random noise edges and enhances major contours. For instance, in `flowers.png`, smoothed edges capture the dominant shapes of flowers rather than numerous false lines seen in the noisy case.
- Between detectors, Sobel benefits more visibly from smoothing than LoG, as LoG inherently includes a smoothing step through its Gaussian component.
- For highly structured images like `Checkerboard.png`, smoothing has minimal visual impact since the underlying pattern already contains sharp, high-contrast edges.

Overall Comparison:

- **Clean Images:** Both Sobel and LoG detect clear, distinct edges.
- **Noisy Images:** Sobel produces false edges, while LoG maintains structural integrity.
- **Smoothed Images:** Edges become smoother and less fragmented; minor details are lost but overall clarity improves.
- **Noisy + Smoothed Images:** LoG provides the best balance between noise suppression and edge preservation. Sobel performance improves significantly with pre-smoothing.

Conclusion: Gaussian smoothing plays a crucial role in improving the reliability of edge detection under noise. The Laplacian of Gaussian (LoG) consistently outperforms Sobel in stability and continuity, especially in noisy or natural images. However, Sobel remains a good choice for clean, high-contrast images where fine detail retention is desired. Thus, a combination of smoothing and LoG-based detection provides the most visually robust results across diverse image types.