Student: Aditya Sandhu

University: Colorado State University Global

Course: 25FC - CSC515 - 1 [Module 4 – Image Formation]

Critical Thinking Assignment [OpenCV Gaussian-then-Laplacian application to noisy image]

Instructor: Dr Dong Nguyen

Date – 09/30/2025

GIT LINKS

Document Link – [25FC-CSC515-1/MODULE4/csc515-1-module4-critical-thinking-aditya-sandhu.docx at main · 65AR645ASAN/25FC-CSC515-1](https://github.com/65AR645ASAN/25FC-CSC515-1/blob/main/MODULE4/csc515-1-module4-critical-thinking-aditya-sandhu.docx)

Python File – [25FC-CSC515-1/MODULE4/ csc515-1-module4-critical-thinking-aditya-sandhu00.py at main · 65AR645ASAN/25FC-CSC515-1](https://github.com/65AR645ASAN/25FC-CSC515-1/blob/main/MODULE4/%20csc515-1-module4-critical-thinking-aditya-sandhu00.py)

Image filtering is a fundamental process in computer vision that allows us to enhance images, reduce noise, and detect important features such as edges. Three filtering techniques Gaussian, Laplacian, and Gaussian with Laplacian are applied to a noisy image. The Gaussian filter is a smoothing operator that reduces random variations in pixel intensity by performing a weighted average of neighboring pixels. The Laplacian filter, in contrast, is a second-order derivative operator that highlights rapid changes in intensity, making it useful for edge detection but highly sensitive to noise. Combining the two, Gaussian with Laplacian, allows noise reduction before applying edge detection, producing sharper and more reliable feature extraction. To analyze these effects, we apply each filter using three different kernel sizes (3×3, 5×5, and 7×7) with a fixed Gaussian sigma, and present the results side-by-side for direct visual comparison. This experiment provides insight into how filter choice and kernel size impact both noise suppression and edge preservation. These are two critical factors in computer vision tasks.

Figure 1 - Image Processing with OpenCV Filters

A screenshot of a computer program

AI-generated content may be incorrect.

import cv2, imports the OpenCV library for image processing tasks. import numpy as np, Imports NumPy (aliased as np) for numerical operations and import matplotlib.pyplot as plt, Imports Matplotlib's plotting module (aliased as plt) for visualization. image\_file = r"C:\Users\Aditya\Desktop\CSUDOCS\CSC515\25FC-CSC515-1\MODULE4\Mod4CT2.jpg", defines the raw string path to the image file, specifying a Windows file location. The r prefix ensures backslashes are treated literally. original\_image = cv2.imread(image\_file, cv2.IMREAD\_COLOR) loads the image from the specified path in color mode. The if original\_image is None, the block checks if the image failed to load (e.g., due to an incorrect path or missing file). If true, it raises a FileNotFoundError with a message indicating the issue and the file location.

grayscale\_image = cv2.cvtColor(original\_image, cv2.COLOR\_BGR2GRAY)

This code, converts the loaded color image (in BGR format, OpenCV's default) to grayscale. This prepares the image for filter application, as many image processing techniques work better or are designed for single-channel (grayscale) images.

gaussian\_sigma = 1.0 sets the standard deviation (sigma) value for Gaussian operations, controlling the amount of blurring. A value of 1.0 is a moderate choice for noise reduction.  kernel\_sizes = [3, 5, 7], defines a list of kernel sizes (3x3, 5x5, 7x7) to test the filters across different scales. Larger kernels typically increase the smoothing or edge detection area. And, filtered\_images = {}: Initializes an empty dictionary to store the filtered images, with kernel sizes as keys and tuples of filtered results as values. This structure will hold the output of the upcoming filter operations.

**Figure 2 - Image Filter Application and Visualization**

A screenshot of a computer program

AI-generated content may be incorrect.

*# Loop through each kernel size and apply filters  
for* size *in* kernel\_sizes:  
 *# Apply Gaussian blur for smoothing* smoothed\_image = cv2.GaussianBlur(grayscale\_image, (size, size), gaussian\_sigma)  
 *# Apply Laplacian for edge enhancement (convert to abs for display)* edge\_image = cv2.Laplacian(grayscale\_image, cv2.CV\_64F, ksize=size)  
 edge\_image = cv2.convertScaleAbs(edge\_image)  
 *# Apply Gaussian blur followed by Laplacian for noise-reduced edges* smoothed\_edge\_image = cv2.Laplacian(smoothed\_image, cv2.CV\_64F, ksize=size)  
 smoothed\_edge\_image = cv2.convertScaleAbs(smoothed\_edge\_image)  
 *# Store the results as a tuple* filtered\_images[size] = (smoothed\_image, edge\_image, smoothed\_edge\_image)

The for loop iterates over each value in the kernel\_sizes list (previously defined as [3, 5, 7]). The variable size takes on each kernel dimension (3, 5, 7) one at a time. The loop applies the following filter operations for each kernel size. The smoothed\_image applies the Gaussian blur to the grayscale\_image to reduce noise. cv2.GaussianBlur() is an open CV function that smooths the image by averaging pixel values using a Gaussian kernel.

The kernel size is set to (size, size), e.g., (3, 3), (5, 5), or (7, 7), based on the current loop iteration. gaussian\_sigma (previously set to 1.0) defines the standard deviation of the Gaussian distribution, controlling the extent of smoothing. The result, smoothed\_image, is a new image with reduced noise, stored for later use.

# Apply Laplacian for edge enhancement (convert to abs for display)

edge\_image = cv2.Laplacian(grayscale\_image, cv2.CV\_64F, ksize=size)

edge\_image = cv2.convertScaleAbs(edge\_image)

This applies a two-step process, first using the previously computed smoothed\_image (from Gaussian blur), then applying the Laplacian. cv2.Laplacian(smoothed\_image, cv2.CV\_64F, ksize=size) detects edges in the smoothed image, reducing the impact of noise compared to the direct Laplacian on the original. cv2.convertScaleAbs(smoothed\_edge\_image) again converts the result to absolute values for display purposes. The result, smoothed\_edge\_image, provides edge detection with less noise interference due to the initial smoothing.

# Store the results as a tuple

filtered\_images[size] = (smoothed\_image, edge\_image, smoothed\_edge\_image)

This line stores the three processed images (smoothed\_image, edge\_image, smoothed\_edge\_image) as a tuple, using the current size (e.g., 3, 5, or 7) as the key in the filtered\_images dictionary. After the loop completes, filtered\_images will contain entries like {3: (smoothed\_3x3, edge\_3x3, smoothed\_edge\_3x3), 5: (...), 7: (...)}, allowing access to results by kernel size for later visualization or analysis.

*# Create a 3x3 subplot grid for visualization*figure, subplots = plt.subplots(3, 3, figsize=(12, 12))  
filter\_types = ["Gaussian Smoothing", "Laplacian Edges", "Smoothed + Edges"]  
  
*# Populate the subplots with images  
for* row, size *in* enumerate(kernel\_sizes):  
 *for* col, type\_name *in* enumerate(filter\_types):  
 subplots[row, col].imshow(filtered\_images[size][col], cmap="gray")  
 subplots[row, col].set\_title(f"{type\_name} ({size}x{size})")  
 subplots[row, col].axis("off")  
  
*# Adjust layout and display the plot*plt.tight\_layout()  
plt.show()

This code segment creates a 3x3 subplot grid using Matplotlib to visualize the results of applying three different image filters (Gaussian Smoothing, Laplacian Edges, and Smoothed + Edges) across three kernel sizes (3x3, 5x5, 7x7) stored in the filtered\_images dictionary. The plt.subplots(3, 3, figsize=(12, 12)) function initializes the grid with a 12x12 inch figure, and filter\_types defines the labels for the columns. A nested loop iterates over kernel\_sizes (rows) and filter\_types (columns), using imshow to display each filtered image in grayscale from filtered\_images[size][col], adding a title that combines the filter type and kernel size, and turning off axes with axis("off") for clarity. Finally, plt.tight\_layout() adjusts the spacing, and plt.show() renders the plot, allowing visual comparison of how each filter and kernel size affects the noisy image.

Figure 3 - Visualization of Filter Effects on a Noisy Image

A collage of images of a person

AI-generated content may be incorrect.

The output image displays a 3x3 grid of subplots generated by the script, comparing the effects of three filters. Gaussian Smoothing, Laplacian Edges, and Smoothed + Edges, across three kernel sizes (3x3, 5x5, 7x7) on a noisy image of a person with arms crossed.

The Gaussian Smoothing column shows progressively smoother versions of the original image as the kernel size increases (3x3, 5x5, 7x7), reducing noise but retaining the subject's outline more clearly with larger kernels.

The Laplacian Edges column reveals highly noisy, speckled outputs for all kernel sizes, indicating the filter's sensitivity to noise, with minimal discernible structure due to amplified artifacts. The Smoothed + Edges column, which combines Gaussian blur followed by Laplacian, attempts to balance noise reduction and edge detection, showing faint outlines of the subject that become slightly more defined with larger kernels (7x7), though still heavily obscured by residual noise.

Overall, the visualization highlights that while Gaussian smoothing effectively reduces noise, the Laplacian-based methods struggle with the image's noise level, and the combined approach offers a compromise but fails to fully recover clear edges.

This experiment demonstrates that Gaussian smoothing effectively reduces noise in a noisy image, with larger kernel sizes (e.g., 7x7) providing clearer outlines, though at the cost of some detail. The Laplacian filter, while useful for edge detection, amplifies noise significantly, rendering it less effective without preprocessing. The combined Gaussian + Laplacian approach offers a balanced solution, improving edge visibility in noisy conditions, but its performance is limited by the extent of initial noise, suggesting the need for further optimization or preprocessing techniques.

**References**

GeeksforGeeks. (n.d.). *Normal distribution in statistics*. In *GeeksforGeeks*. Retrieved September 30, 2025, from <https://www.geeksforgeeks.org/maths/normal-distribution/#google_vignette>

Siegrist, K. (n.d.). *The Laplace distribution*. In *Probability, mathematical statistics, and stochastic processes*. LibreTexts. Retrieved September 30, 2025, from <https://stats.libretexts.org/Bookshelves/Probability_Theory/Probability_Mathematical_Statistics_and_Stochastic_Processes_(Siegrist)/05%3A_Special_Distributions/5.28%3A_The_Laplace_Distribution>

TheAILearner. (2019, May 25). *Laplacian of Gaussian (LoG)*. TheAILearner. <https://theailearner.com/2019/05/25/laplacian-of-gaussian-log/>

Wikipedia. (n.d.-a). *Laplace distribution*. In *Wikipedia*. Retrieved September 30, 2025, from <https://en.wikipedia.org/wiki/Laplace_distribution>

Wikipedia. (n.d.-b). *Normal distribution*. In *Wikipedia*. Retrieved September 30, 2025, from <https://en.wikipedia.org/wiki/Normal_distribution>