Computer Vision Assignment 1

Sushovit Nanda
21283

September 8, 2025

1 Introduction

This report presents a comprehensive analysis of the assignment questions implemented from scratch using Python and NumPy. Specialized libraries (OpenCV) were used only to load and process the image. The assignment encompasses four main questions with the primary objectives mentioned below:

- Q1: Implement RGB to grayscale conversion and analyze individual color channels.
- Q2: Add Gaussian noise and analyze its effects on image processing.
- Q3: Compare edge detection methods (Sobel, Laplacian) with different filter sizes.
- Q4: Implement the complete Canny edge detector from scratch.

2 Methodology

2.1 Libraries Used

The assignment was carried out using Python with the following key libraries:

- **NumPy**: For numerical computations and array operations.
- **Matplotlib**: For visualization and plotting.
- **OpenCV**: Limited to image loading and color space conversion only.
- **Math**: For mathematical functions and kernel generation.

2.2 Image Chosen

The analysis is performed on a personal RGB image with the following characteristics:

- Dimensions: $1280 \times 914 \times 3$ pixels
- Color channels: 3 (BGR format from OpenCV)
- Data type: uint8 (8-bit unsigned integer, JPEG format)
- Total pixels: 3,509,760

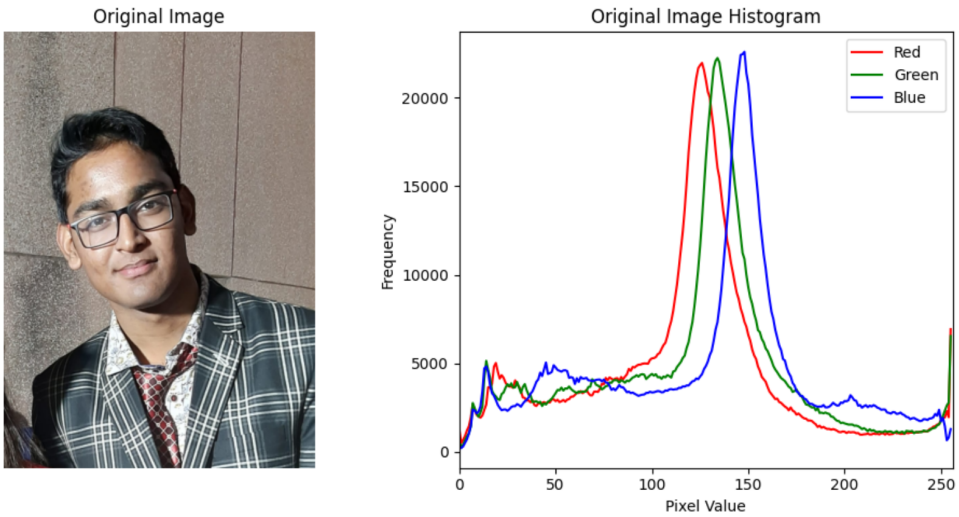


Figure 1: Original RGB Image and its Histogram

The original image shows varying color intensities across the red, green, and blue channels. The histogram analysis reveals the distribution of pixel intensities for each color channel, depicting the color composition and contrast.

3 Question 1: RGB to Grayscale Conversion and Channel Analysis

RGB to grayscale conversion was implemented using the standard luminance formula:

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{1}$$

Figure 2: Grayscale Image using Luminance Formula

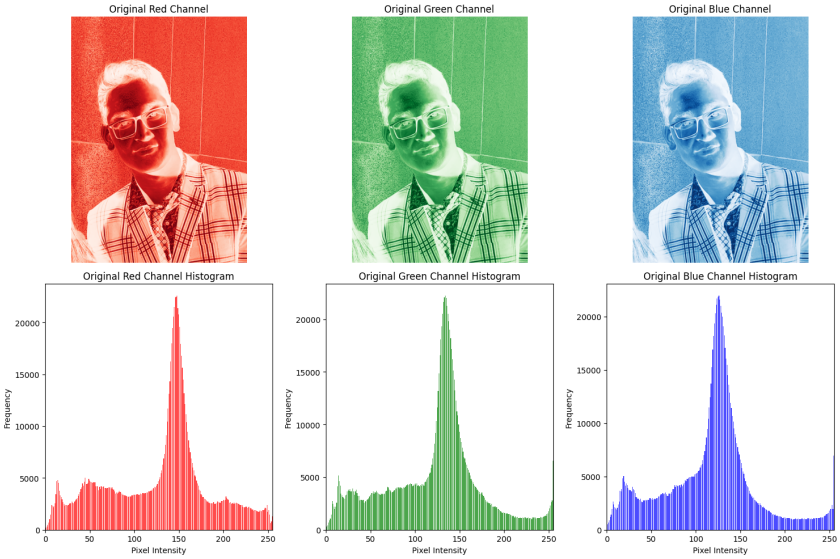


Figure 3: Individual Color Channels and their Histograms

The grayscale conversion in Figure 2 successfully reduced from 3 channels to 1. The channel analysis in Figure 3 shows different intensity patterns across color components. Statistical analysis reveals:

Channel	Mean Intensity	Standard Deviation
Red	128.55	57.12
Green	122.03	53.25
Blue	116.81	50.92

Table 1: Statistical Analysis of Color Channels

The red channel shows highest mean intensity (128.55), indicating warm color tone. All channels exhibit close standard deviations with small difference, suggesting balanced color distribution.

4 Question 2: Gaussian Noise Addition and Analysis

Gaussian noise was added using parameters mean (μ) = 0 and variance (σ^2) = 20 (according to the question), resulting in SNR 20.1 dB.

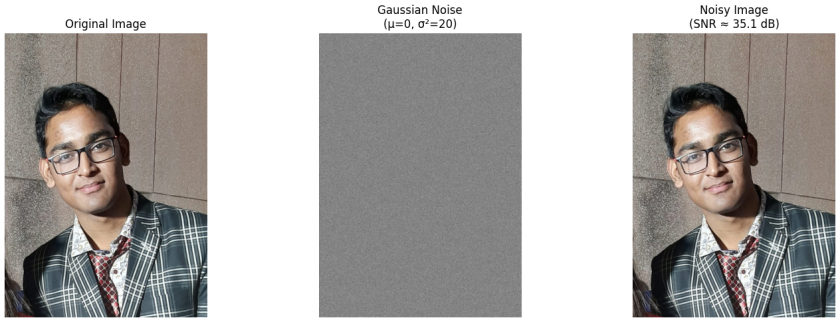
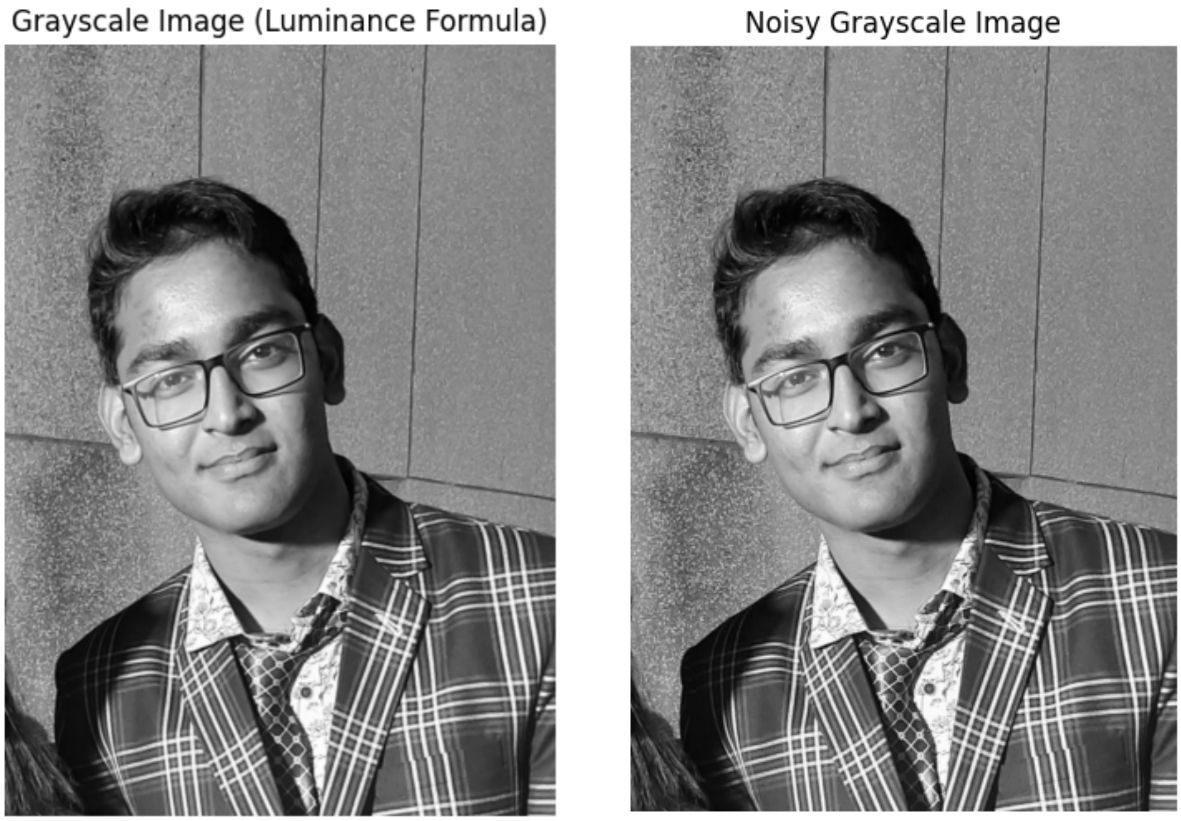


Figure 4: Original Image, Gaussian Noise Pattern, and Noisy Image



(a) Clean Grayscale (b) Noisy Grayscale

Figure 5: Clean vs Noisy Grayscale Images Comparison

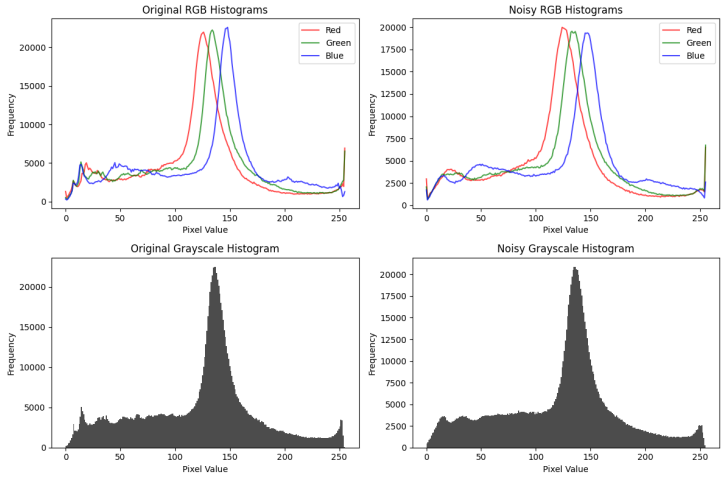


Figure 6: Clean vs Noisy Histograms Comparison

The noise addition in Figure 4 shows random intensity variations across the image. Comparing clean vs noisy grayscale images in Figure 5, noise slightly increased standard deviation (53.75 vs. 53.67 original). The histogram comparison in Figure 6 shows increased spread in noisy distributions.

Mean ($\mu = 0$) preserves brightness. Non-zero values would cause systematic shifts. Variance ($\sigma^2 = 20$) controls noise intensity. Due to the relatively low noise variance, the

noisy image is not significantly different from the original, making it difficult to distinguish significant differences in subsequent processing results through visual inspection. Lower values provide less visible noise but are less realistic, while higher values create more visible noise but are more challenging for testing. The chosen value provides noticeable noise while maintaining image recognizability, suitable for the assignment.

5 Question 3: Edge Detection with Various Filters and Noise Conditions

Edge detection was performed using Sobel and Laplacian operators with filter sizes 3×3, 5×5, and 7×7 on both clean and noisy images.

(a) Clean Image Results (b) Noisy Image Results

Figure 7: Sobel Edge Detector Results (Combined X+Y) for Different Filter Sizes

(a) Clean Image Results (b) Noisy Image Results

Figure 8: Sobel Operator in Horizontal and Vertical Directions

(a) Clean Image Results (b) Noisy Image Results

Figure 9: Laplacian Edge Detector Results for Different Filter Sizes

The Sobel results in Figure 7 show that for clean images, the 3×3 filter produces

clear, well-defined edges with good detail preservation, while 5×5 and 7×7 filters result in disturbed and less distinct edge detection. For noisy images, due to the relatively low noise variance ($^2 = 20$), the noisy image is not significantly different from the original, making it difficult to distinguish significant differences in edge detection results through visual inspection. The 3×3 filter shows noticeable edges despite noise, while larger filters (5×5, 7×7) produce very disturbed edge detection with poor edge quality.

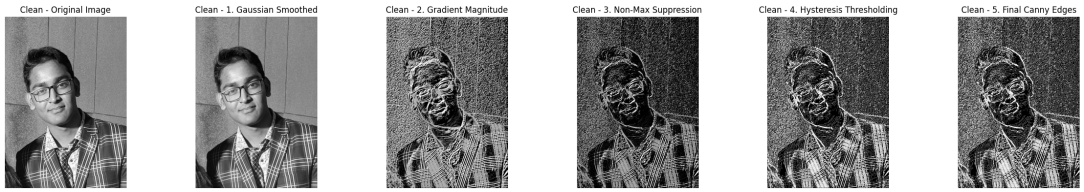
The directional Sobel results in Figure 8 demonstrate that Sobel_X effectively detects vertical edges while Sobel_Y detects horizontal edges, with the combined version capturing edges in all directions, as we can see clearly from the highlighted vertical edges shown on my coat patterns. However, the edge quality degrades significantly with larger filter sizes.

The Laplacian results in Figure 9 show that 3×3 and 5×5 filters produce proper, complete edges with good detail, while the 7×7 filter results in incomplete, light edges with poor edge detection quality.

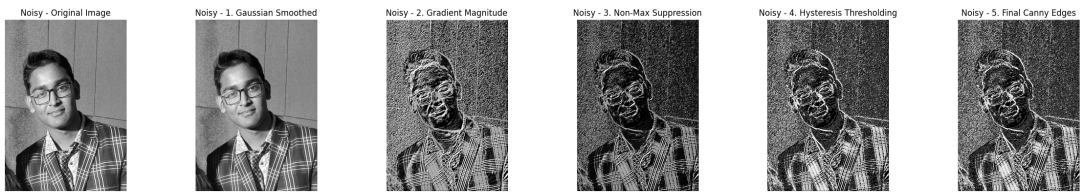
The results show that smaller filters (3×3) are optimal for edge detection quality, while larger filters (5×5, 7×7) significantly degrade edge detection performance despite providing noise reduction. For clean images, 3×3 filters provide the best edge detection results. For noisy images, due to the low noise variance, the differences between clean and noisy results are minimal and difficult to distinguish visually.

6 Question 4: Canny Edge Detector Implementation from Scratch

The Canny edge detector was implemented from scratch with six complete steps: (1) Original image, (2) Gaussian smoothing ($\sigma = 1.4$ provides optimal noise reduction without excessive blurring, 3×3 kernel provides good balance between noise reduction and edge preservation), (3) Gradient magnitude computation using Sobel filters, (4) Non-maximum suppression for edge thinning, (5) Hysteresis thresholding (high threshold = 15% balances strong edge detection with false positive reduction, low threshold = 5% allows weak edge connection), and (6) Final Canny edges.



(a) Clean Image - Complete Canny Algorithm Steps



(b) Noisy Image - Complete Canny Algorithm Steps

Figure 10: Complete Step-by-Step Canny Algorithm Visualization (6 Steps)

The step-by-step visualization in Figure 10 demonstrates the complete Canny algorithm implementation. Each step progressively refines edge detection: (1) Original image provides the input, (2) Gaussian smoothing reduces noise while preserving edge structure, (3) Gradient magnitude computation identifies potential edges with clear boundaries, (4) Non-maximum suppression creates thin, single-pixel-wide edge candidates, (5) Hysteresis thresholding connects weak edges to strong ones, and (6) Final Canny edges show continuous, well-defined contours.

For noisy images, the same six-step process is applied, but noise affects each step. However, due to the relatively low noise variance ($\sigma^2 = 20$), the noisy image is not significantly different from the original, making it difficult to distinguish significant differences in edge detection results through visual inspection. The major edges remain visible through the process, but some noise-induced artifacts may appear in the final output.