Image Processing Application Using Python and OpenCV

Neelash Kannan Annadurai
Msc Robotics With Industrial Application,
2024-2026
Heriot Watt University
Edinburgh, United Kingdom
na3018@hw.ac.uk

Abstact: A GUI-based image processing system using OpenCV and tkinter implements the user interface as part of this project in the Python framework. The application enables users to perform diverse image operations that involve grayscale conversion while offering both geometric transformation tools and smoothing filters and edge-detection functions alongside intensity adjustment controls. Users can upload an image while picking from the selectable options in the dropdown then adjust threshold values to examine real-time results.

INTRODUCTION

The basis of contemporary computer vision software depends on image processing because machines require this capability to analyze visual content while also enhancing and interpreting it. The core capability of image processing allows technological developments to achieve functions including smartphone facial detection together with autonomous vehicle sign identification. Many individuals struggle to comprehend or put into practice image processing approaches mainly because these techniques involve advanced mathematical theories. Users can perform image processing tasks without difficulty through this project's Graphical User Interface (GUI) system design. Users need not write code because the system enables visual interface interaction through which they can choose images and tasks from dropdown menus for instantaneous display of results. The system enables users to alter parameters in real-time for binarization and edge detection and smoothing filter operations which provides a hands-on learning environment.

SYSTEM DESIGN AND IMPLEMENTATION

The system runs on the Python platform that uses OpenCV for image processing tasks and tkinter for GUI development. A standalone desktop tool runs the application which requires only basic user skills to execute. Users open the application to upload images which should exist in PNG or JPG formats. The application harnesses GUI elements (fig.1.1) to show original images in left-side display with processed versions shown right-side display. After picking a transformation from the dropdown menu additional adjustable parameters appear through sliders according to what task the user selects. Following input of the initial image both the output image alongside its width height and channel are displayed under the original image as addressed in Q.4.1.

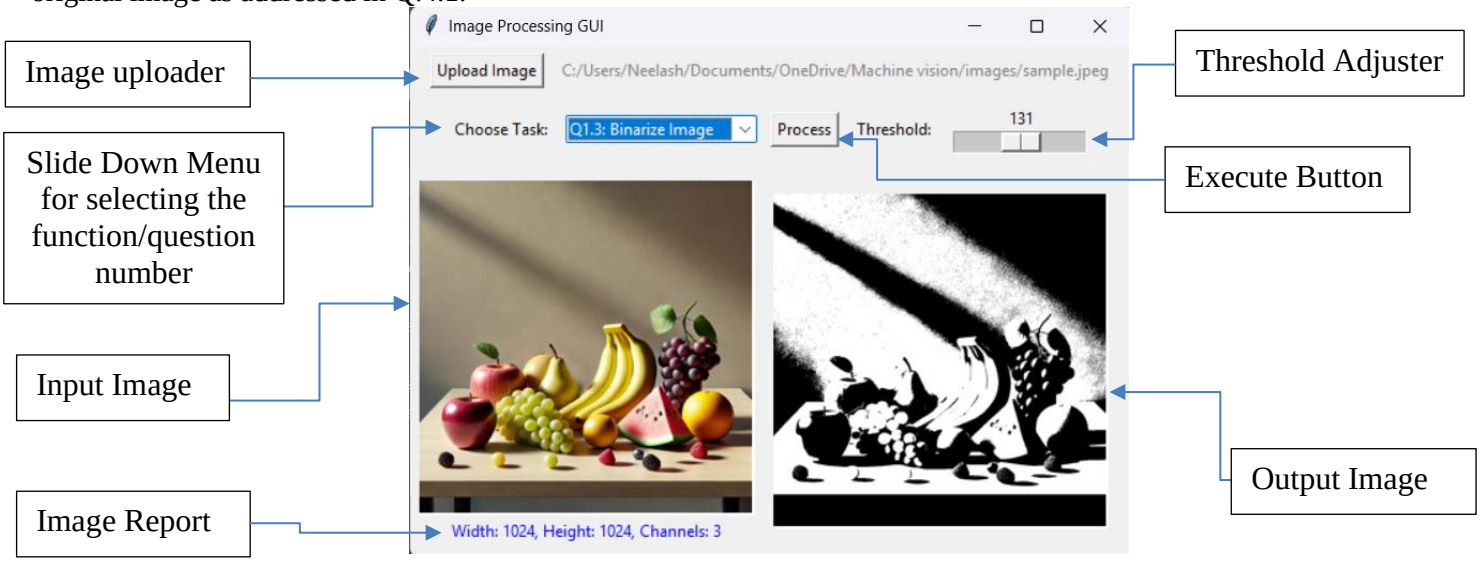


Fig.1.1: Snip of the application created.

Every transformation operates on an independent duplicate of the original image to prevent transformation history from accumulating across different operations. The cloning method produces new results by applying operations on separate image copies to keep their integrity intact.

IMAGE PROCESSING TASKS AND CONCEPTS

Load and Display Images – Q.1.1

The first and most fundamental task in image processing is loading and displaying an image. In this project, images are loaded using the **OpenCV cv2.imread() function**, which reads the image as an array of pixel values. The loaded image is displayed using **Pillow (PIL)** in the GUI. Displaying an image on-screen does not alter its content; it simply renders the visual representation for further analysis. The output and code is mentioned in Below Images.

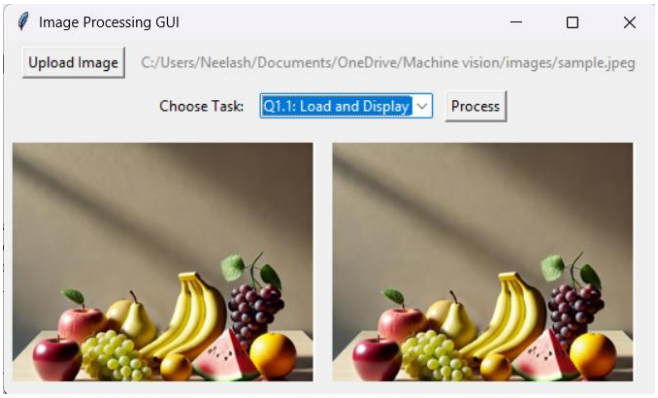


Fig.2.1 Image is Loaded in the GUI and displayed in Right Side

```
def upload_image(self):
    self.image_path = filedialog.askopenfilename(filetypes=[("Image Files", "*.png;*.jpg;*.jpeg")])
    if self.image_path:
        self.file_label.config(text=self.image_path)
        self.original_image = cv2.imread(self.image_path)
        if self.original_image is None:
            messagebox.showerror("Error", "Failed to load image!")
            return
        self.display_image(self.original_image, self.canvas_original)
        height, width, channels = self.original_image.shape
        image_details = f"Width: {width}, Height: {height}, Channels: {channels}"
        # Display image details below the left canvas.
        self.image_info_label.config(text=image_details)
```

Fig.1.1 Code for Displaying image in the GUI Window.

Grayscale and HSV Conversion – Q.1.2

A color image always have three fundamental channels which are Red [R], Green [G] and Blue [B]. Most of the image processing operations work best on grayscale versions rather than color images because color information proves unnecessary for these operations and processing time reduces. The grayscale conversion process produces images that contain only one channel that shows pixel intensities from absolute black at 0 to total white at 255. The intensity values provide enough information for both edge detection and binarization operations after the system simplifies them. Another vital image transformation applies the HSV (Hue, Saturation, Value) color model. HSV divides image elements into color components while maintaining separate representation of brightness values since it offers distinct color-based and intensity-based operations unlike RGB systems. Each pixel in RGB displays hue as color type and satiation as color intensity and value represents brightness.

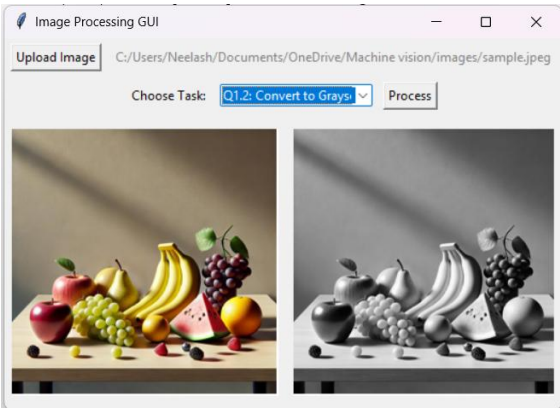


Fig.2.2 Greyscale

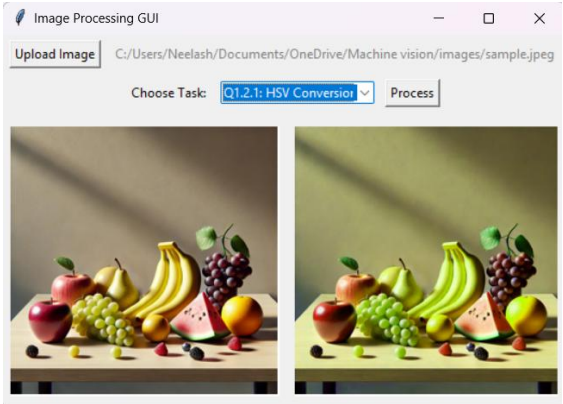


Fig2.3 HSV

Binarization – Q.1.3

Binarization is a process that converts an image into two distinct pixel values: **black and white**. This is achieved by applying a **threshold**, where pixels **above** a certain value are set to white, and those **below** are set to black.

The threshold of **161** is chosen (on a scale of 0 to 255), any pixel with an intensity greater than **161** will be turned white, while others become black. This technique is used in **document scanning (OCR)**, where text needs to be isolated from the background. The Binarized images with varying threshold are added in next page.

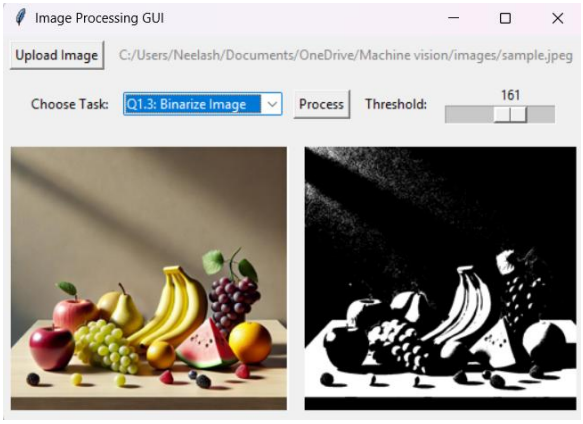


Fig.2.3 Binarized Image

Geometric Transformations: Translation and Rotation – Q.2

Geometric transformations allow modification of an image's **position and orientation**.

- **Translation** moves the image **horizontally and vertically** without changing its content. This is commonly used in applications such as **object tracking** and image alignment. The Output for **Question 2.1** is in the Fig.2.4

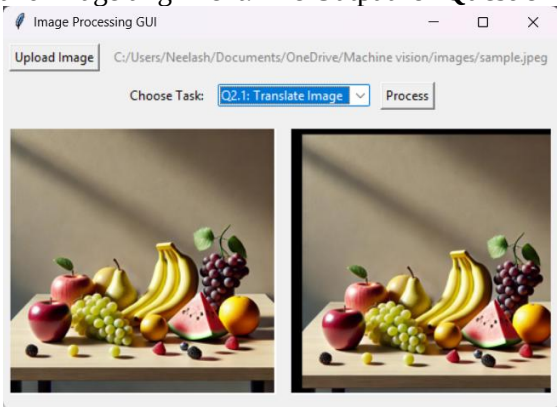


Fig.2.4 Snip of the Image which translates the left side input image and shows output in Right.

- **Rotation** reorients an image by a specified **angle**, typically around its center. This transformation is useful in applications like **medical imaging**, where scans might need alignment before analysis. The Output for **Question 2.2** is in the Fig.2.5

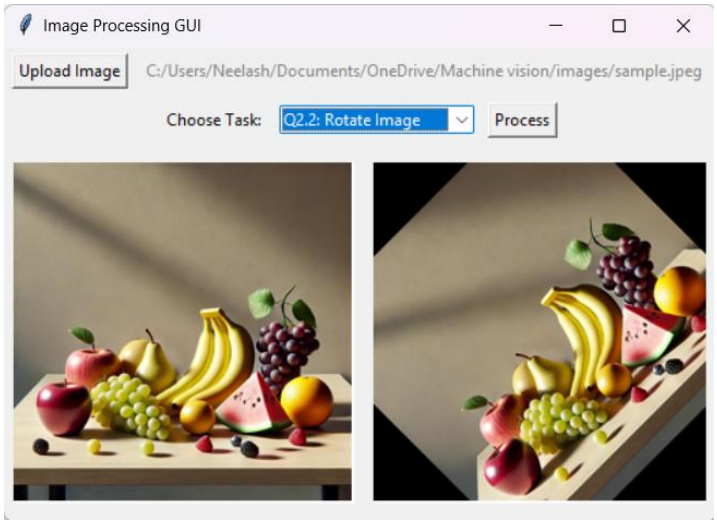


Fig.2.5. Input Image being rotated

Filtering: Mean and Gaussian Smoothing – Q.3

Noise in images can arise due to **lighting conditions, sensor errors, or compression artifacts**. To reduce noise while preserving important features, **smoothing filters** are used.

- **Mean Filter:** Averages pixel values in a small neighbourhood, making the image **less grainy** as mentioned in Fig.2.6.1. However, it tends to **blur edges**. The images for varying the kernel size are attached below. (Question 3.1)
- **Gaussian Filter:** Similar to the mean filter but applies a **weighted average** where pixels closer to the centre contribute more to the smoothing process. This produces **better edge preservation** while reducing noise as mentioned in Fig.2.6.2. (Question 3.2)

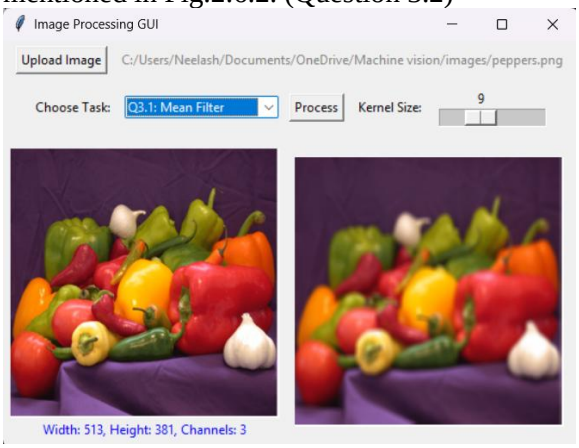


Fig.2.6.1 Mean Filter

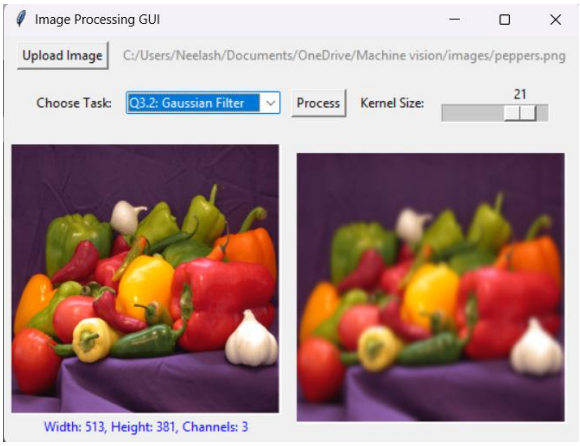


Fig.2.6.2 Gaussian Filter

The Output images for varying Kernel size for the Mean Filter is attached in below Fig.2.6.3 for **Question 3.1.**

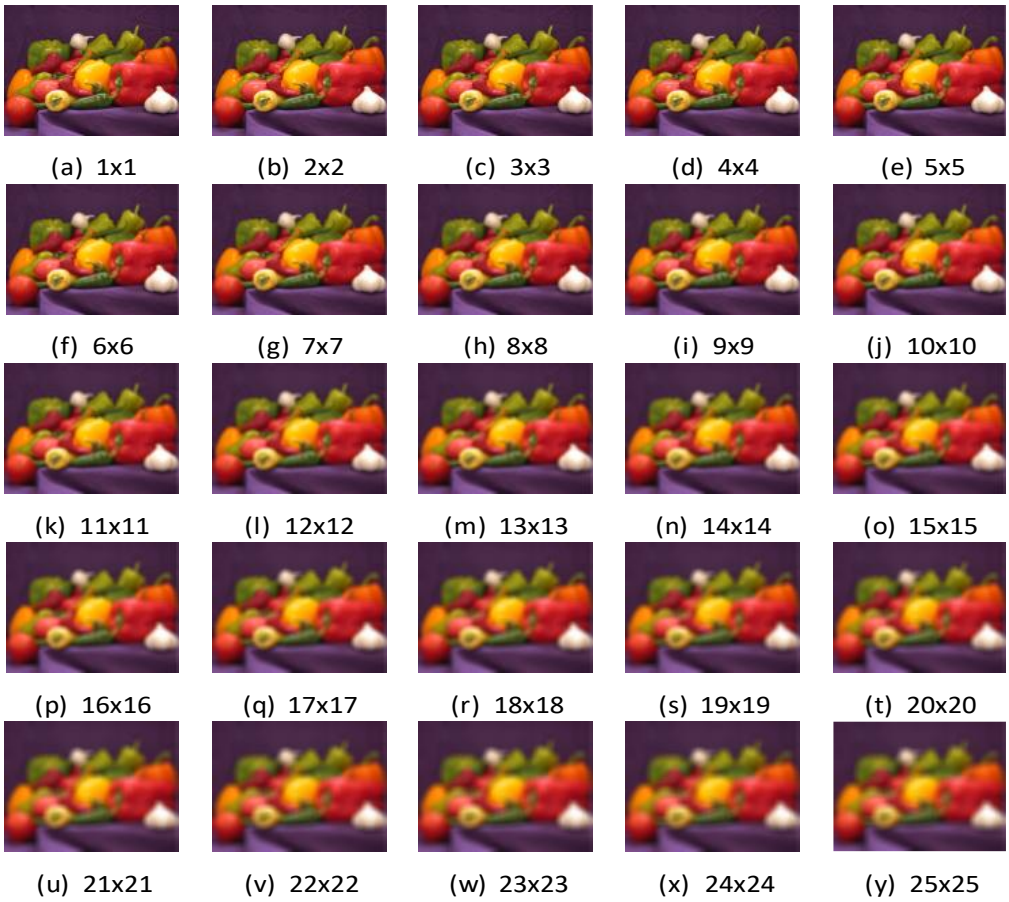


Fig.2.6.3. Mean Filter

The Output images for varying Kernel size for the Gaussian Filter is attached in below Fig.2.6.4 for **Question 3.2**.

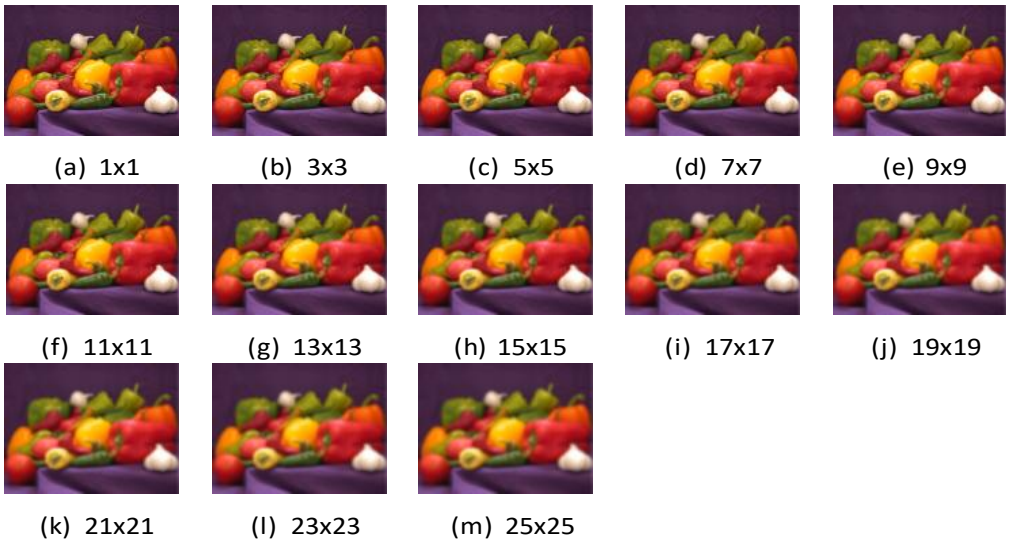


Fig.2.6.4. Gaussian Filter

Canny Edge Detection – Q.3.3

The boundaries of image objects get defined through edges. The Canny Edge Detection algorithm identifies edges through an evaluation of sudden intensity variations. The process involves:

1. The implementation of a Gaussian filter smooths images for reducing noise.
2. Identification of locations where intensity changes rapidly occurs (gradients).
3. The thresholding process creates two classifications of edges that are strong and weak.

Detecting edges through this method finds extensive use in object recognition systems operated by autonomous vehicles which need to detect lane boundaries.

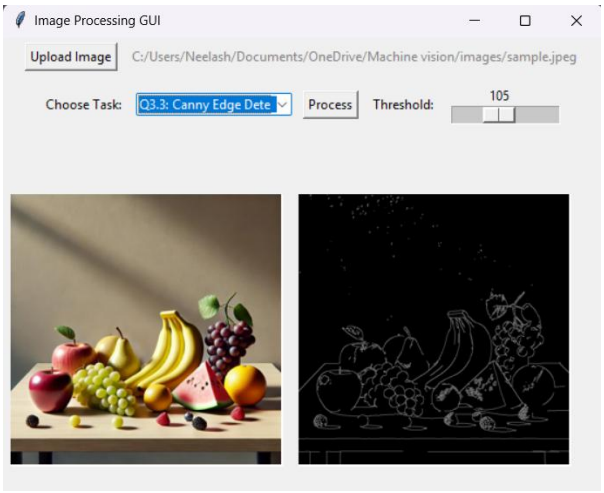
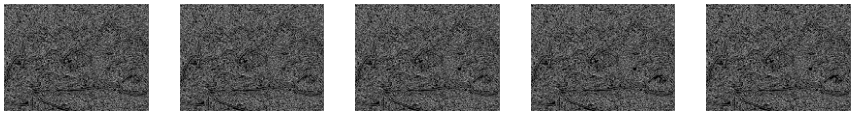
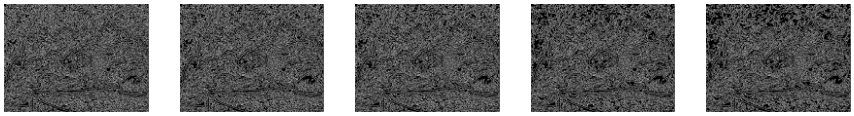


Fig.2.7. Canny Edge Detection

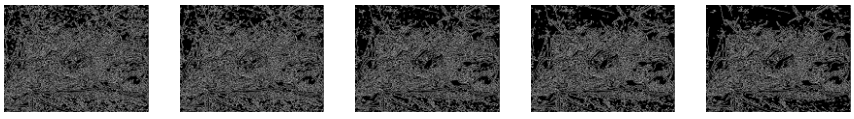
The Output images of Canny Edge detection using the pepper image taken from MATLAB for **Q.3.3** are attached below by varying threshold from 0 - 255.



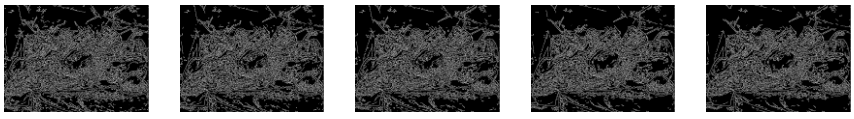
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4)



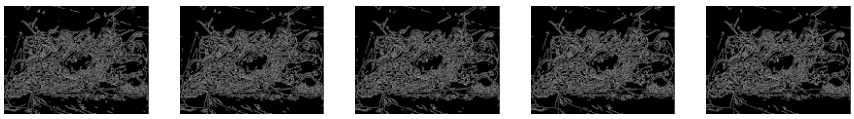
(0, 5) (0, 6) (0, 7) (0, 8) (0, 9)



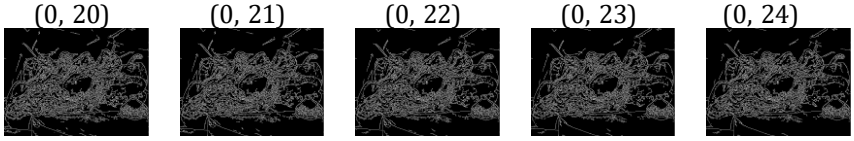
(0, 10) (0, 11) (0, 12) (0, 13) (0, 14)



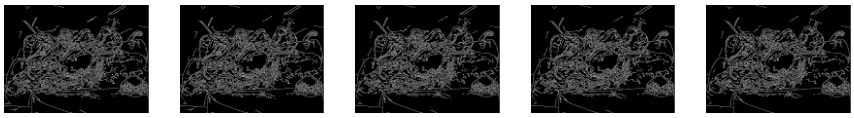
(0, 15) (0, 16) (0, 17) (0, 18) (0, 19)



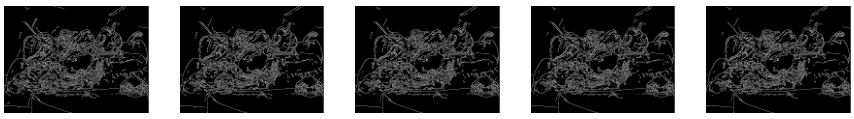
(0, 20) (0, 21) (0, 22) (0, 23) (0, 24)



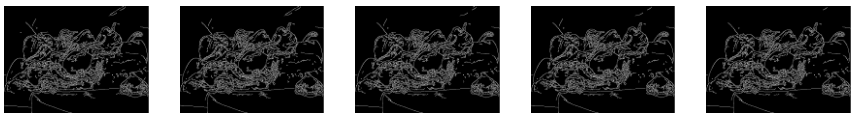
(0, 25) (0, 26) (0, 27) (0, 28) (0, 29)



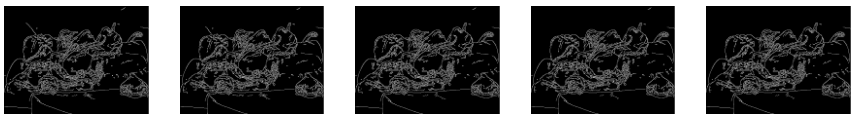
(0, 30) (0, 31) (0, 32) (0, 33) (0, 34)



(0, 35) (0, 36) (0, 37) (0, 38) (0, 39)



(0, 40) (0, 41) (0, 42) (0, 43) (0, 44)



(0, 45) (0, 46) (0, 47) (0, 48) (0, 49)

(0, 50) (0, 51) (0, 52) (0, 53) (0, 54)

(0, 55) (0, 56) (0, 57) (0, 58) (0, 59)

(0, 60) (0, 61) (0, 62) (0, 63) (0, 64)


(0, 65)

(0, 66)

(0, 67)

(0, 68)

(0, 69)


(0, 70)

(0, 71)

(0, 72)

(0, 73)

(0, 74)


(0, 75)

(0, 76)

(0, 77)

(0, 78)

(0, 79)


(0, 80)

(0, 81)

(0, 82)

(0, 83)

(0, 84)


(0, 85)

(0, 86)

(0, 87)

(0, 88)

(0, 89)


(0, 90)

(0, 91)

(0, 92)

(0, 93)

(0, 94)


(0, 95)

(0, 96)

(0, 97)

(0, 98)

(0, 99)


(0, 100)

(0, 101)

(0, 102)

(0, 103)

(0, 104)

(0, 105)

(0, 106)

(0, 107)

(0, 108)

(0, 109)


(0, 110)

(0, 111)

(0, 112)

(0, 113)

(0, 114)


(0, 115)

(0, 116)

(0, 117)

(0, 118)

(0, 119)


(0, 120)

(0, 121)

(0, 122)

(0, 123)

(0, 124)


(0, 125)

(0, 126)

(0, 127)

(0, 128)

(0, 129)

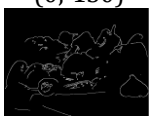

(0, 130)

(0, 131)

(0, 132)

(0, 133)

(0, 134)


(0, 135)

(0, 136)

(0, 137)

(0, 138)

(0, 139)


(0, 140)

(0, 141)

(0, 142)

(0, 143)

(0, 144)


(0, 145)

(0, 146)

(0, 147)

(0, 148)

(0, 149)

(0, 150)

(0, 151)

(0, 152)

(0, 153)

(0, 154)


(0, 155)

(0, 156)

(0, 157)

(0, 158)

(0, 159)


(0, 160)

(0, 161)

(0, 162)

(0, 163)

(0, 164)


(0, 165)

(0, 166)

(0, 167)

(0, 168)

(0, 169)


(0, 170)

(0, 171)

(0, 172)

(0, 173)

(0, 174)


(0, 175)

(0, 176)

(0, 177)

(0, 178)

(0, 179)


(0, 180)

(0, 181)

(0, 182)

(0, 183)

(0, 184)


(0, 185)

(0, 186)

(0, 187)

(0, 188)

(0, 189)


(0, 190)

(0, 191)

(0, 192)

(0, 193)

(0, 194)

(0, 195)

(0, 196)

(0, 197)

(0, 198)

(0, 199)


(0, 200)

(0, 201)

(0, 202)

(0, 203)

(0, 204)


(0, 205)

(0, 206)

(0, 207)

(0, 208)

(0, 209)


(0, 210)

(0, 211)

(0, 212)

(0, 213)

(0, 214)


(0, 215)

(0, 216)

(0, 217)

(0, 218)

(0, 219)

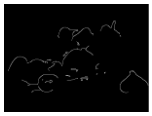
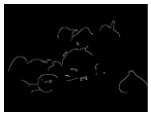
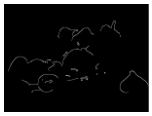
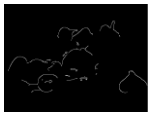
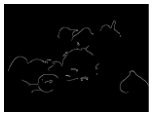

(0, 220)

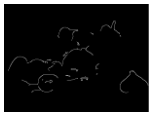
(0, 221)

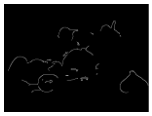
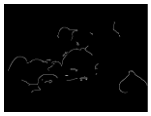
(0, 222)

(0, 223)

(0, 224)


(0, 225)

(0, 226)

(0, 227)

(0, 228)

(0, 229)


(0, 230)

(0, 231)

(0, 232)

(0, 233)

(0, 234)


(0, 235)

(0, 236)

(0, 237)

(0, 238)

(0, 239)

(0, 240)

(0, 241)

(0, 242)

(0, 243)

(0, 244)

(0, 245) (0, 246) (0, 247) (0, 248) (0, 249)

(0, 250) (0, 251) (0, 252) (0, 253) (0, 254)

(0, 255)

Intensity Range Reduction

In some applications, it is necessary to **reduce the intensity range** of an image while keeping its **color information** intact. This process limits the brightness values, making an image appear as if captured with a **lower dynamic range camera**.Unlike grayscale conversion, which removes color entirely, intensity reduction **maintains the hues but compresses brightness variations**.

Fig.2.8. Intensity Range Reduction

Attached to the next pages are grayscale images with intensity ranges reduced from [0, 255] to [0, N] for different N values from 255 down to 8 as described in Q.4.2. The clear distortions occur between the coordinates (0, 28) and (0, 32) across Question 4.3.

(0, 8) (0, 9)

(0, 10) (0, 11) (0, 12) (0, 13) (0, 14)

(0, 15) (0, 16) (0, 17) (0, 18) (0, 19)

(0, 20) (0, 21) (0, 22) (0, 23) (0, 24)

(0, 25) (0, 26) (0, 27) (0, 28) (0, 29)

(0, 30) (0, 31) (0, 32) (0, 33) (0, 34)

(0, 35) (0, 36) (0, 37) (0, 38) (0, 39)

(0, 40) (0, 41) (0, 42) (0, 43) (0, 44)

(0, 45) (0, 46) (0, 47) (0, 48) (0, 49)

(0, 50) (0, 51) (0, 52) (0, 53) (0, 54)

(0, 55) (0, 56) (0, 57) (0, 58) (0, 59)

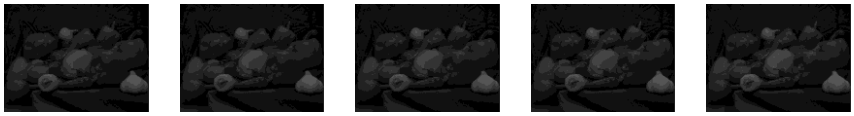
(0, 60) (0, 61) (0, 62) (0, 63) (0, 64)

(0, 65) (0, 66) (0, 67) (0, 68) (0, 69)

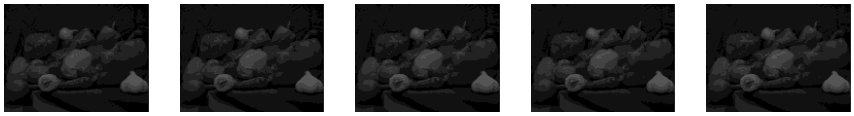
(0, 70) (0, 71) (0, 72) (0, 73) (0, 74)

(0, 75) (0, 76) (0, 77) (0, 78) (0, 79)

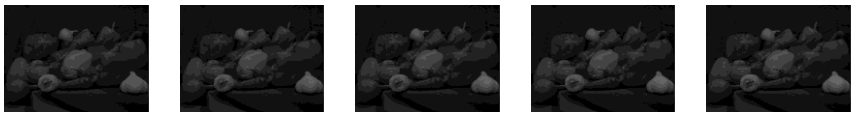
(0, 80) (0, 81) (0, 82) (0, 83) (0, 84)



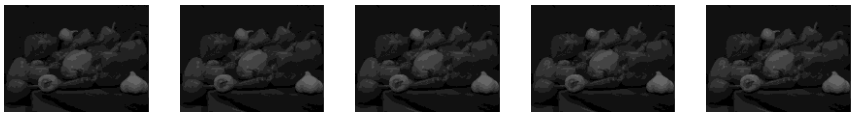
(0, 85) (0, 86) (0, 87) (0, 88) (0, 89)



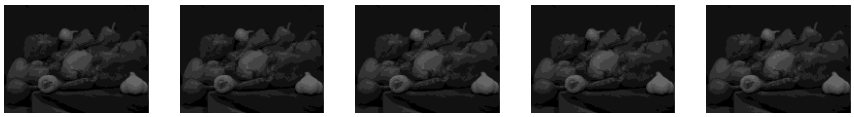
(0, 90) (0, 91) (0, 92) (0, 93) (0, 94)



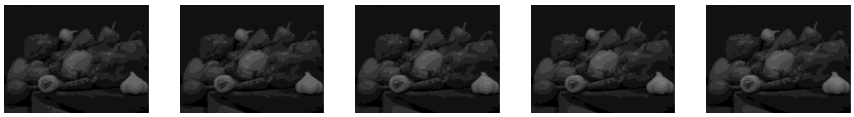
(0, 95) (0, 96) (0, 97) (0, 98) (0, 99)



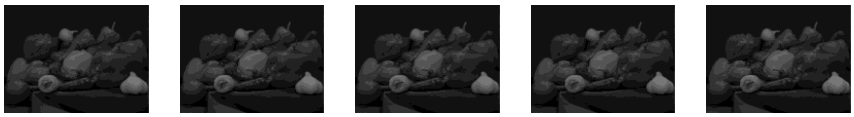
(0, 100) (0, 101) (0, 102) (0, 103) (0, 104)



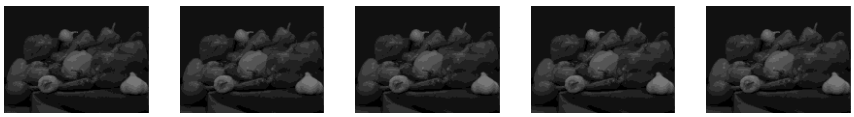
(0, 105) (0, 106) (0, 107) (0, 108) (0, 109)



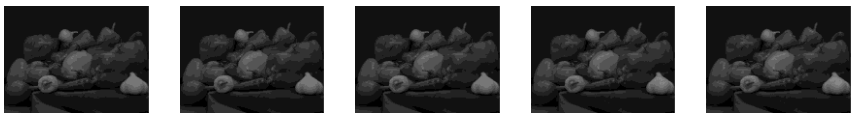
(0, 110) (0, 111) (0, 112) (0, 113) (0, 114)



(0, 115) (0, 116) (0, 117) (0, 118) (0, 119)



(0, 120) (0, 121) (0, 122) (0, 123) (0, 124)

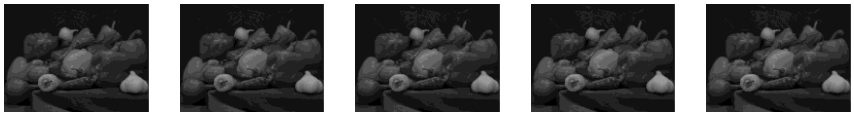


(0, 125) (0, 126) (0, 127) (0, 128) (0, 129)

(0, 130) (0, 131) (0, 132) (0, 133) (0, 134)

(0, 135) (0, 136) (0, 137) (0, 138) (0, 139)

(0, 140) (0, 141) (0, 142) (0, 143) (0, 144)



(0, 145) (0, 146) (0, 147) (0, 148) (0, 149)


(0, 150)

(0, 151)

(0, 152)

(0, 153)

(0, 154)


(0, 155)

(0, 156)

(0, 157)

(0, 158)

(0, 159)


(0, 160)

(0, 161)

(0, 162)

(0, 163)

(0, 164)


(0, 165)

(0, 166)

(0, 167)

(0, 168)

(0, 169)


(0, 170)

(0, 171)

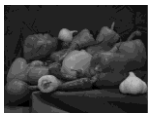
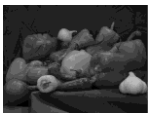
(0, 172)

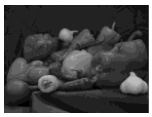
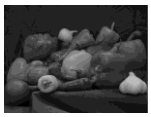
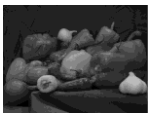
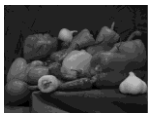
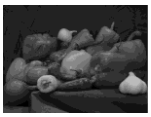
(0, 173)

(0, 174)


(0, 175)

(0, 176)

(0, 177)

(0, 178)

(0, 179)


(0, 180)

(0, 181)

(0, 182)

(0, 183)

(0, 184)

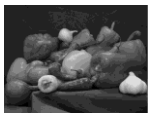
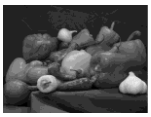

(0, 185)

(0, 186)

(0, 187)

(0, 188)

(0, 189)

(0, 190)	(0, 191)	(0, 192)	(0, 193)	(0, 194)
				
(0, 195)	(0, 196)	(0, 197)	(0, 198)	(0, 199)
				
(0, 200)	(0, 201)	(0, 202)	(0, 203)	(0, 204)
				
(0, 205)	(0, 206)	(0, 207)	(0, 208)	(0, 209)
				
(0, 210)	(0, 211)	(0, 212)	(0, 213)	(0, 214)
				
(0, 215)	(0, 216)	(0, 217)	(0, 218)	(0, 219)
				
(0, 220)	(0, 221)	(0, 222)	(0, 223)	(0, 224)
				
(0, 225)	(0, 226)	(0, 227)	(0, 228)	(0, 229)
				
(0, 230)	(0, 231)	(0, 232)	(0, 233)	(0, 234)

(0, 235) (0, 236) (0, 237) (0, 238) (0, 239)

(0, 240) (0, 241) (0, 242) (0, 243) (0, 244)

(0, 245) (0, 246) (0, 247) (0, 248) (0, 249)

(0, 250) (0, 251) (0, 252) (0, 253) (0, 254)

(0, 255)

Histogram Equalization

The image contrast enhancement mechanism of Histogram Equalization performs intensity value redistribution that spans the entire range from 0 to 255 as shown in Fig.2.9. Contrast enhancement through image distribution methods provides medical professionals with better visibility of crucial details in radiological and magnetic resonance imaging examinations.

Fig.2.9 Histogram Equalization

CHALLENGES FACED

1. **Real-Time Updates**: Making the GUI respond quickly to slider changes (e.g., kernel size, thresholds) without freezing.

2. **Color Adjustments**: Keeping HSV/HSL values valid (e.g., hue between 0–180) to avoid color errors.
3. **Filter Kernels**: Ensuring kernel sizes were odd numbers (3, 5, 7, etc.) for proper filtering.
4. **Quality Loss**: Reducing intensity levels (e.g., to 32 or 16) caused visible "banding" in images.
5. **Rotation Gaps**: Rotated images had empty corners, needing manual fixes.
6. **Edge Detection**: Balancing noise and edge detail with threshold sliders.
7. **Slow Processing**: Large images or complex filters slowed down the app.

These issues were solved by testing, simplifying code, and using OpenCV's built-in tools.

CONCLUSION

This report successfully implements core image processing techniques, including image loading, grayscale/HSV/HSL conversion, geometric transformations (translation by 50/30 pixels and 45° rotation), mean/Gaussian filtering, Canny edge detection, intensity range reduction, and histogram equalization, using a Python-based GUI. Key findings demonstrate noticeable intensity distortion at **N = 32** during range reduction, the critical role of kernel size in smoothing filters, and the effectiveness of histogram equalization in enhancing contrast. The interactive interface enabled real-time parameter exploration, revealing practical trade-offs (e.g., blur vs. noise reduction, edge detail vs. threshold sensitivity). Limitations include fixed sigma in Gaussian filtering and linear intensity scaling artifacts, which could be improved with adaptive parameter controls. The project fulfills the assignment's objectives, providing a foundational understanding of image processing workflows and their visual outcomes, with insights for future extensions such as advanced filters or edge-preserving techniques.

REFERENCES

1. freeCodeCamp, "OpenCV Python Tutorial for Beginners," YouTube ; https://youtu.be/8O-FW4Wm10s (Accessed: 10 February 2023).
2. Murtaza's Workshop – Robotics and AI, "Color Spaces in OpenCV | Python," YouTube ; https://youtu.be/6jP3r7WYqa4 (Accessed: 10 February 2023).
3. ProgrammingKnowledge, "Image Translation and Rotation using OpenCV," YouTube ; https://youtu.be/4Kp4XAAmkeY (Accessed: 10 February 2023).
4. CodeWithHarry, "Image Smoothing Techniques in OpenCV," YouTube; https://youtu.be/C_zFhWdM4ic (Accessed: 10 February 2023).
5. Sentdex, "Canny Edge Detection – OpenCV Python Tutorial," YouTube ; https://youtu.be/4n3ul3tEy_c (Accessed: 10 February 2023).
6. Pysource, "Histogram Equalization in OpenCV," YouTube ; https://youtu.be/7LW_75E3A1Q (Accessed: 10 February 2023).
7. Parwiz Forogh, "OpenCV with Tkinter – Display Images in GUI," YouTube ; https://youtu.be/3rr0JN8QkZk (Accessed: 10 February 2023).