

# CS477 – Computer Vision

## Assignment 1



By  
**Muhammad Umer** (CMS – 345834)  
**Muhammad Ahmed Mohsin** (CMS – 333060)  
**Muhammad Saad** (CMS – 333414)

Instructor  
**Dr. Mohsin Kamal**

School of Electrical Engineering and Computer Science (SEECS)  
National University of Sciences and Technology (NUST)  
Islamabad, Pakistan

October 19, 2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Task 1 – Filtering Techniques</b>	<b>2</b>
<b>3</b>	<b>Task 2 – Edge Detection</b>	<b>4</b>
<b>4</b>	<b>Task 3 – Histogram Representation</b>	<b>6</b>
<b>5</b>	<b>Task 4 – Gray and Binary Representation</b>	<b>8</b>
<b>6</b>	<b>Task 5 – Corner Detection</b>	<b>10</b>

# 1 Introduction

In this assignment, we delve into several fundamental image processing tasks, each aimed at demonstrating different aspects of image manipulation and analysis. The primary objective of this assignment is to develop a comprehensive understanding of various image processing techniques and their applications.

## Deliverables

### 1. Filtering Techniques:

*Take any image and perform Gaussian filtering, box filtering and median filtering. Also provide comments on the results.*

### 2. Edge Detection:

*Take any image and detect edges by using Sobel and Prewitt Kernel. Comment on both results.*

### 3. Histogram Representation:

*Take any image and plot its histogram representation.*

### 4. Gray and Binary Representation:

*Take a colored image and provide its gray and binary representation.*

### 5. Corner Detection:

*Take any image and detect the corners by employing Harris corner detection.*

## Author Contributions

Mention each author's contribution at the end of each question (Mandatory).

- **Author 1:** –
- **Author 2:** –
- **Author 3:** –

## Grading Scheme

- **Q1: 10 Marks**
- **Q2: 10 Marks**
- **Q3: 10 Marks**
- **Q4: 10 Marks**
- **Q5: 10 Marks**
- **Total: 50 marks**

## Copying

Copying is highly discouraged and it will lead to a significant loss (90-95 %) of marks.

*\*Copying includes using sentences, variables, code, formats from others. Discussion is appreciated, but attempt the tasks on your own (which would make it look original).*

## 2 Task 1 – Filtering Techniques

```
1 import cv2
2 import numpy as np
3
4 def get_gaussian_kernel(size, sigma=-1):
5     """
6         Generates a Gaussian kernel of the specified size and sigma.
7
8         If sigma = -1, then sigma is automatically calculated
9         as (size - 1) / 6.0, as per the OpenCV documentation.
10    """
11    if sigma < 0:
12        sigma = (size - 1) / 6.0
13
14    # Generate a grid using the specified size
15    axis = np.linspace(-(size - 1) / 2.0, (size - 1) / 2.0, size)
16    x, y = np.meshgrid(axis, axis)
17
18    # Create the kernel
19    gaussian_kernel = np.exp(-0.5 * (x**2 + y**2) / sigma**2)
20
21    return gaussian_kernel / np.sum(gaussian_kernel)
22
23 # Read image
24 img = cv2.imread("./lenna.png")
25
26 # Define kernel size
27 kernel_size = 7
28
29 # Define kernels
30 box_kernel = (1 / kernel_size**2) * np.ones((kernel_size,
31                                                 kernel_size))
32 gaussian_kernel = get_gaussian_kernel(kernel_size, 5)
33
34 # Apply filters
35 box_filtered = cv2.filter2D(img, -1, box_kernel)
36 gaussian_filtered = cv2.filter2D(img, -1, gaussian_kernel)
37 median_filtered = cv2.medianBlur(img, kernel_size)
38
39 # Save images
40 cv2.imwrite("report/figs/task1/box_filtered.png", box_filtered)
41 cv2.imwrite("report/figs/task1/gaussian_filtered.png",
42             gaussian_filtered)
43 cv2.imwrite("report/figs/task1/median_filtered.png", median_filtered)
44
45 # Show images
46 cv2.imshow("Original", img)
47 cv2.imshow("Box Filtered", box_filtered)
48 cv2.imshow("Gaussian Filtered", gaussian_filtered)
49 cv2.imshow("Median Filtered", median_filtered)
50
51 cv2.waitKey(0)
52 cv2.destroyAllWindows()
```

Python Code for Filtering Techniques

## Author Contributions & Comments

- **Author 1:** – NIL
- **Author 2:** – Implemented the Gaussian kernel function and the Gaussian filtering. Also, put forth a layout for showing and saving images.
- **Author 3:** – Implemented the box kernel function and box filtering. Also, implemented the median filtering.
- **Comments:** Given that the original image had realistic noise, the Gaussian filtered image was the least blurred while effectively suppressing noise. On the other hand, the median filter suppressed noise, but at the cost of losing edge clarity, as expected since the median filter is better suited for salt-and-pepper noise. Similarly, the box filter yielded a uniformly blurred image, losing information about edges.

## Results



Figure 2.1: Original (Noisy)



Figure 2.2: Box Filtered



Figure 2.3: Median Filtered



Figure 2.4: Gaussian Filtered

### 3 Task 2 – Edge Detection

```
1 import cv2
2 import numpy as np
3
4 # Read image
5 img = cv2.imread("./lenna.png")
6
7 # Convert to gray
8 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9
10 # Define kernels
11 sobel_x = np.array([[ -1,  0,  1], [ -2,  0,  2], [ -1,  0,  1]])
12 sobel_y = np.array([[ -1, -2, -1], [ 0,  0,  0], [ 1,  2,  1]])
13
14 prewitt_x = np.array([[ -1,  0,  1], [ -1,  0,  1], [ -1,  0,  1]])
15 prewitt_y = np.array([[ -1, -1, -1], [ 0,  0,  0], [ 1,  1,  1]])
16
17 # Apply filters
18 sobel_x_filtered = cv2.filter2D(gray, -1, sobel_x)
19 sobel_y_filtered = cv2.filter2D(gray, -1, sobel_y)
20
21 prewitt_x_filtered = cv2.filter2D(gray, -1, prewitt_x)
22 prewitt_y_filtered = cv2.filter2D(gray, -1, prewitt_y)
23
24 # Save images
25 cv2.imwrite("report/figs/task2/sobel_x_filtered.png",
26             sobel_x_filtered)
27 cv2.imwrite("report/figs/task2/sobel_y_filtered.png",
28             sobel_y_filtered)
29 cv2.imwrite("report/figs/task2/prewitt_x_filtered.png",
30             prewitt_x_filtered)
31 cv2.imwrite("report/figs/task2/prewitt_y_filtered.png",
32             prewitt_y_filtered)
33
34 # Show images
35 cv2.imshow("Original", img)
36 cv2.imshow("Sobel X Filtered", sobel_x_filtered)
37 cv2.imshow("Sobel Y Filtered", sobel_y_filtered)
38 cv2.imshow("Prewitt X Filtered", prewitt_x_filtered)
39 cv2.imshow("Prewitt Y Filtered", prewitt_y_filtered)
40 cv2.waitKey(0)
41 cv2.destroyAllWindows()
```

Python Code for Edge Detection

### Author Contributions & Comments

- **Author 1:** – NIL
- **Author 2:** – Implemented the Prewwit kernel functions and the Prewitt filtering.
- **Author 3:** – Implemented the Sobel kernel functions and the Sobel filtering.
- **Comments:** From the figures below, we observe that Sobel filter results in more accurate edges than the Prewitt filter. This is attributed to the fact that the Sobel filter gives more weight to the pixels closer to the center of the mask, making it more sensitive to edges,

however, this also makes it more sensitive to noise. On the other hand, the Prewitt filter gives equal weight to all pixels in the mask, making it less sensitive to noise, but also less sensitive to edges.

## Results



Figure 3.1: Prewitt X Filtered

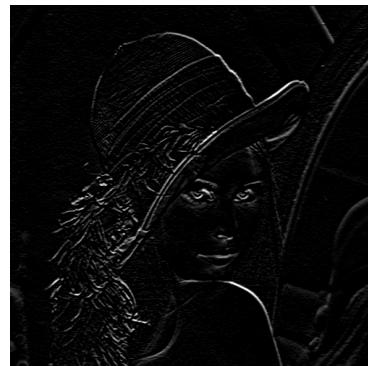


Figure 3.2: Prewitt Y Filtered



Figure 3.3: Sobel X Filtered



Figure 3.4: Sobel Y Filtered

## 4 Task 3 – Histogram Representation

```
1  import cv2
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  def compute_histogram(img):
6      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7      rows, cols = gray.shape
8
9      hist = np.zeros(256)
10
11     # Iterate over image and increment histogram
12     for i in range(rows):
13         for j in range(cols):
14             hist[gray[i, j]] += 1
15
16     return hist
17
18 plt.rcParams["mathtext.fontset"] = "stix"
19 plt.rcParams["font.family"] = "STIXGeneral"
20
21 # Read image
22 img = cv2.imread("./lenna.png")
23
24 # Compute histogram
25 hist = compute_histogram(img)
26
27 # Plot and histogram
28 plt.figure(figsize=(16, 6))
29 bars = plt.bar(np.arange(256), hist, color="red", edgecolor="black")
30 plt.xlabel("Pixel Intensity", fontsize=20)
31 plt.ylabel("Frequency", fontsize=20)
32 plt.xticks(fontsize=20)
33 plt.yticks(fontsize=20)
34 plt.xlim(0, 256)
35 plt.grid(alpha=0.5)
36 plt.legend([bars], ["Pixel Intensity"], fontsize=20)
37 plt.tight_layout()
38 plt.savefig("report/figs/task3/histogram.png", dpi=300)
39 plt.show()
```

Python Code for Histogram Representation

### Author Contributions & Comments

- **Author 1:** – Implemented the iterative histogram computation function.
- **Author 2:** – Made the layout for the histogram plot.
- **Author 3:** – NIL
- **Comments:** To compute the histogram, we initialize an array of size equal to the number of possible pixel intensities (bit-depth of the image). Then, after converting the image to grayscale, we iterate over each pixel of the image and increment the corresponding index of the histogram array. Finally, we visualize the histogram using a bar graph.

## Results

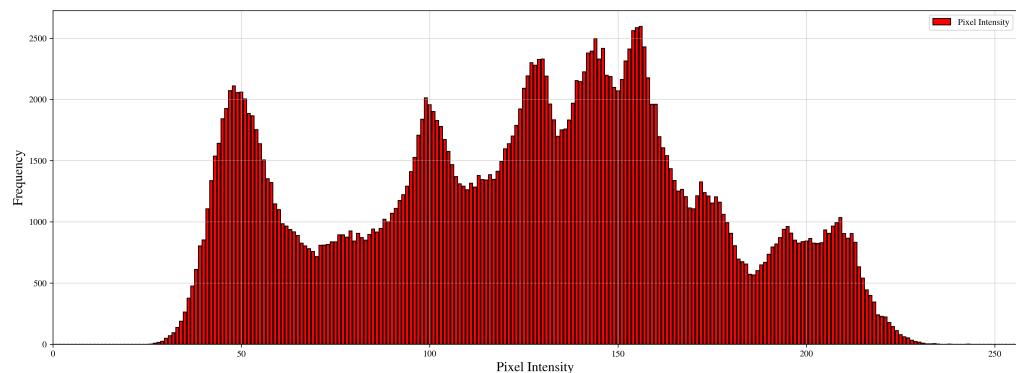


Figure 4.1: Histogram Representation

## 5 Task 4 – Gray and Binary Representation

```
1      import cv2
2
3      def rgb2gray(rgb):
4          """Convert RGB image to grayscale
5
6              Standard formula for converting RGB to grayscale
7              Reference:
8                  https://w.wiki/7pNM
9              """
10
11         # Get the red, green, and blue channels of the image
12         red_channel = rgb[:, :, 0]
13         green_channel = rgb[:, :, 1]
14         blue_channel = rgb[:, :, 2]
15
16         # Calculate the grayscale value for each pixel using the
17         # standard formula
18         gray = (0.2989 * red_channel + 0.5870 * green_channel +
19                 0.1140 * blue_channel)
20
21         return gray
22
23     # Read image
24     img = cv2.imread("./lenna.png")
25
26     # Convert to gray
27     gray = rgb2gray(img) # divide by 255 to normalize
28     norm_gray = gray / 255
29
30     # Convert to binary
31     threshold = 127 # empirically found
32     binary = gray.copy()
33     binary[binary < threshold] = 0
34     binary[binary >= threshold] = 255
35
36     # Save images
37     cv2.imwrite("report/figs/task4/gray.png", gray)
38     cv2.imwrite("report/figs/task4/binary.png", binary)
39
40     # Show images
41     cv2.imshow("Original", img)
42     cv2.imshow("Gray", norm_gray)
43     cv2.imshow("Binary", binary)
44     cv2.waitKey(0)
45     cv2.destroyAllWindows()
```

Python Code for Gray and Binary Representation

### Author Contributions & Comments

- **Author 1:** – Implemented the RGB to grayscale conversion function.
- **Author 2:** – Implemented the grayscale to binary conversion function.
- **Author 3:** – NIL

- **Comments:** To convert an RGB image to grayscale, we utilize the standard formula ([Converting color to grayscale](#)). To convert a grayscale image to binary, we first empirically find a threshold value, or use a standard value of 127, and then set all pixels below the threshold to 0 and all pixels above the threshold to 255.

## Results



Figure 5.1: Original



Figure 5.2: Gray



Figure 5.3: Binary

## 6 Task 5 – Corner Detection

```
1  import cv2
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from numba import jit
5
6  def get_gaussian_kernel(size, sigma=-1):
7      """
8          Generates a Gaussian kernel of the specified size and sigma.
9
10         If sigma = -1, then sigma is automatically calculated
11         as (size - 1) / 6.0, as per the OpenCV documentation.
12     """
13     if sigma < 0:
14         sigma = (size - 1) / 6.0
15
16     # Generate a grid using the specified size
17     axis = np.linspace(-(size - 1) / 2.0, (size - 1) / 2.0, size)
18     x, y = np.meshgrid(axis, axis)
19
20     # Create the kernel
21     gaussian_kernel = np.exp(-0.5 * (x**2 + y**2) / sigma**2)
22
23     return gaussian_kernel / np.sum(gaussian_kernel)
24
25     # Details of JIT in comments
26     @jit(nopython=True)
27     def non_max_suppression(H_response, win_height, win_width):
28         """
29             Performs non-maximum suppression on the Harris response.
30         """
31         for i in range(win_height, H_response.shape[0] - win_height):
32             for j in range(win_width, H_response.shape[1] - win_width):
33                 if H_response[i, j] == 0:
34                     continue
35                 H_slice = H_response[i - win_height : i + win_height + 1,
36                                       j - win_width : j + win_width + 1]
37                 if H_response[i, j] != H_slice.max():
38                     H_response[i, j] = 0
39
40     return H_response
41
42     @jit(nopython=True)
43     def get_harris_matrix(I_x, I_y, w, i, j, win_height, win_width):
44         """
45             Returns the Harris matrix for the pixel at (i, j).
46         """
47         I_x_slice = I_x[i - win_height : i + win_height + 1,
48                         j - win_width : j + win_width + 1]
49         I_y_slice = I_y[i - win_height : i + win_height + 1,
50                         j - win_width : j + win_width + 1]
51         M_xx = np.sum(I_x_slice**2 * w)
52         M_yy = np.sum(I_y_slice**2 * w)
53         M_xy = np.sum(I_x_slice * I_y_slice * w)
54         return np.array([[M_xx, M_xy], [M_xy, M_yy]])
55
56     @jit(nopython=True)
57     def get_corners(M, k):
```

```

57      """
58      Returns the cornerness of the Harris matrix.
59      """
60      return np.linalg.det(M) - k * np.trace(M) ** 2
61
62      # Read image
63      img = cv2.imread("./task5.png")
64
65      # Convert to gray
66      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
67      gray = gray.astype(np.float32) / gray.max()
68
69      # Compute Ix and Iy
70      sobel_x = np.array([[1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
71      sobel_y = np.array([[1, -2, -1], [0, 0, 0], [1, 2, 1]])
72
73      I_x = cv2.filter2D(gray, -1, sobel_x)
74      I_y = cv2.filter2D(gray, -1, sobel_y)
75
76      # Harris corner detection
77      kernel_size = 9
78      window_shape = (kernel_size, kernel_size)
79      win_height, win_width = ((window_shape[0] - 1) // 2,
80                               (window_shape[1] - 1) // 2)
81      w = get_gaussian_kernel(kernel_size)
82
83      H_response = np.zeros(gray.shape, dtype=np.float32)
84      k = 0.04
85
86      for i in range(win_height, gray.shape[0] - win_height):
87          for j in range(win_width, gray.shape[1] - win_width):
88              M = get_harris_matrix(I_x, I_y, w, i, j,
89                                    win_height, win_width)
90              H_response[i, j] = get_cornerness(M, k)
91
92      # Thresholding
93      # Empirically found values
94      threshold = 0.05 * H_response.max()
95      H_response[H_response < threshold] = 0
96
97      # Non-maximum suppression
98      H_response = non_max_suppression(H_response, win_height, win_width)
99
100     # Get corners
101     y_c, x_c = np.nonzero(H_response)
102
103     # Draw circles on the image
104     for i in range(len(x_c)):
105         cv2.circle(img, (x_c[i], y_c[i]), 5, (0, 0, 255), -1)
106
107     # Save the image
108     cv2.imwrite("report/figs/task5/corners.png", img)
109
110     # Display the image with circles
111     cv2.imshow("Corners", img)
112     cv2.waitKey(0)
113     cv2.destroyAllWindows()

```

Python Code for Corner Detection

## Author Contributions & Comments

- **Author 1:** – Implemented functions for computing the Harris matrix and the cornerness of the matrix. Also proposed the usage of JIT compilation to optimize the code.
- **Author 2:** – Implemented the Gaussian kernel function and the non-maximum suppression function. Also, implemented the placement of circles on the image.
- **Author 3:** – Patched the code together and implemented the Harris corner detection algorithm with thresholding.
- **Comments:** To perform Harris corner detection, we first compute the image gradients  $I_x$  and  $I_y$  using the Sobel operator. Then, we compute the Harris matrix for each pixel in the image, and use it to compute the cornerness of the pixel. Finally, we threshold the cornerness values and perform non-maximum suppression to obtain the corners.  
Additionally, to optimize the code, we use the `numba` library to Just-In-Time (JIT) compile the functions to boost the execution time incurred by the `for` loops. The JIT compilation is done by adding the `@jit(nopython=True)` decorator above the function definition.

## Results



Figure 6.1: Detected Corners