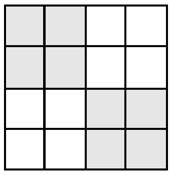
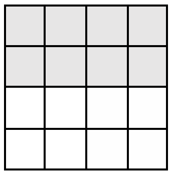
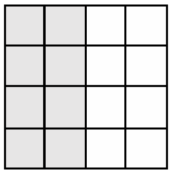
HW 2 – EE655 (31/01/2025)
AYUSH KIRAN BADGUJAR (230259)

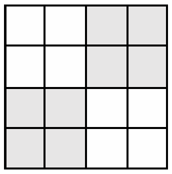
AIM:
This application calculates the response of six Haar-like filters on a given image using the Integral Image concept. The filters draw attention to various aspects of the image, including contrasts and edges. The application speeds up and improves feature extraction by optimizing computation through the use of integral images.

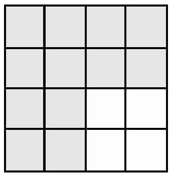
FILTERS USED:











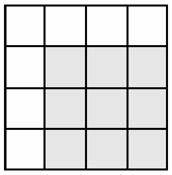
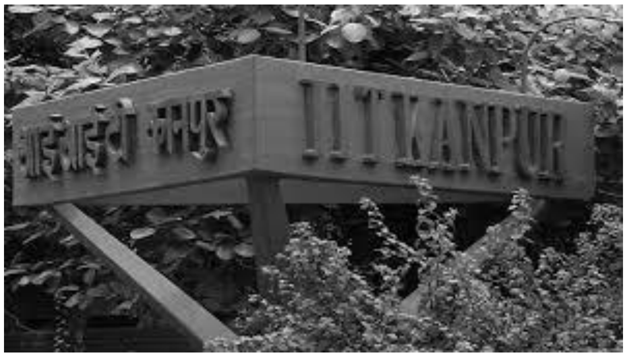
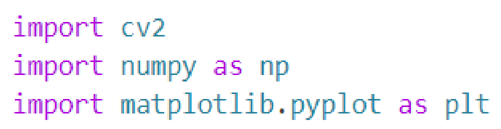


IMAGE ON WHICH FILTER IS USED:



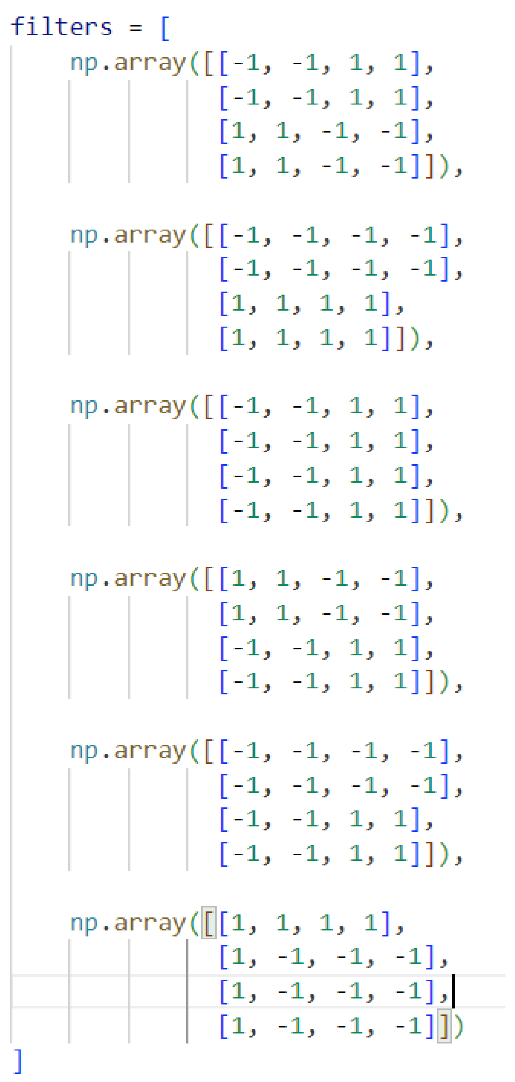
CODE PARTS:

1)IMPORT OF REQUIRED LIBRARIES



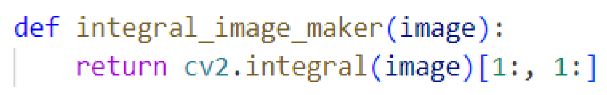
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

2)DEFINING HAAR LIKE FILTERS:



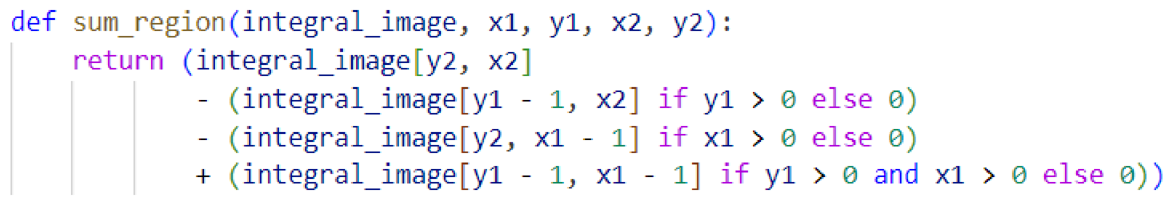
```python
filters = [
    np.array([[-1, -1, 1, 1],
              [-1, -1, 1, 1],
              [1, 1, -1, -1],
              [1, 1, -1, -1]]),

    np.array([[-1, -1, -1, -1],
              [-1, -1, -1, -1],
              [1, 1, 1, 1],
              [1, 1, 1, 1]]),

    np.array([[-1, -1, 1, 1],
              [-1, -1, 1, 1],
              [-1, -1, 1, 1],
              [-1, -1, 1, 1]]),

    np.array([[1, 1, -1, -1],
              [1, 1, -1, -1],
              [-1, -1, 1, 1],
              [-1, -1, 1, 1]]),

    np.array([[-1, -1, -1, -1],
              [-1, -1, -1, -1],
              [-1, -1, 1, 1],
              [-1, -1, 1, 1]]),

    np.array([[1, 1, 1, 1],
              [1, -1, -1, -1],
              [1, -1, -1, -1],
              [1, -1, -1, -1]])
]
```

3) MAKING THE INTEGRAL IMAGE: **cv2.integral(image)** computes the integral image where each pixel represents the sum of all pixels above and to the left.
[1:, 1:] removes the first row and column to match the original image size



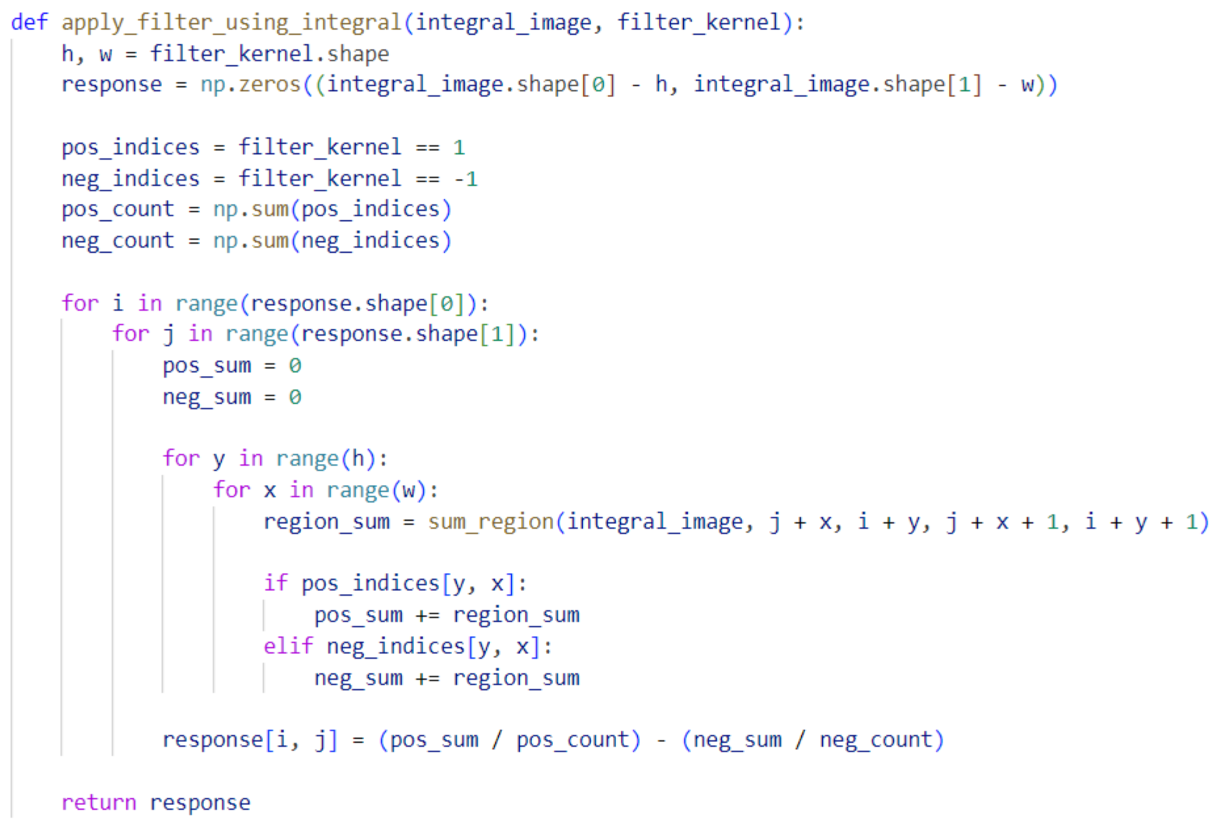
```python
def integral_image_maker(image):
    return cv2.integral(image)[1:, 1:]
```

4) SUMMING PIXEL VALUES IN A REGION: This function efficiently computes the sum of pixel values in a rectangular region using the integral image. Reduces computation time from O(n^2) to O(1).



```
def sum_region(integral_image, x1, y1, x2, y2):
    return (integral_image[y2, x2]
            - (integral_image[y1 - 1, x2] if y1 > 0 else 0)
            - (integral_image[y2, x1 - 1] if x1 > 0 else 0)
            + (integral_image[y1 - 1, x1 - 1] if y1 > 0 and x1 > 0 else 0))
```

5) APPLYING FILTER USING INTEGRAL IMAGE: Extracts regions from the integral image and applies the Haar-like filter. “pos_indices” and “neg_indices” separate positive and negative regions. Computes weighted differences between positive and negative pixel sums. Stores filtered results in a response matrix.



```
def apply_filter_using_integral(integral_image, filter_kernel):
    h, w = filter_kernel.shape
    response = np.zeros((integral_image.shape[0] - h, integral_image.shape[1] - w))

    pos_indices = filter_kernel == 1
    neg_indices = filter_kernel == -1
    pos_count = np.sum(pos_indices)
    neg_count = np.sum(neg_indices)

    for i in range(response.shape[0]):
        for j in range(response.shape[1]):
            pos_sum = 0
            neg_sum = 0

            for y in range(h):
                for x in range(w):
                    region_sum = sum_region(integral_image, j + x, i + y, j + x + 1, i + y + 1)

                    if pos_indices[y, x]:
                        pos_sum += region_sum
                    elif neg_indices[y, x]:
                        neg_sum += region_sum

            response[i, j] = (pos_sum / pos_count) - (neg_sum / neg_count)

    return response
```

6) IMAGE LOADING AND PROCESSING: Loads the image in grayscale using cv2.imread(). Checks for loading errors and exits if the image is not found. Generates an integral image using “integral_image_maker()”. Applies all six filters using “apply_filter_using_integral()”. Displays results in a 2×3 grid using matplotlib.

```python
def main():
    image_path = r"C:\Users\Ayush\OneDrive\Desktop\iitk_ee655_hw2.png"
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    if image is None:
        print(f"Error: Unable to load image from {image_path}")
        return

    integral_image = integral_image_maker(image)

    responses = [apply_filter_using_integral(integral_image, kernel) for kernel in filters]

    fig, axes = plt.subplots(2, 3, figsize=(12, 8))

    for idx, ax in enumerate(axes.flat):
        ax.imshow(responses[idx], cmap='gray')
        ax.set_title(f"Filter {idx+1}")
        ax.axis("off")

    plt.show()

if __name__ == "__main__":
    main()
```

OUTPUT:

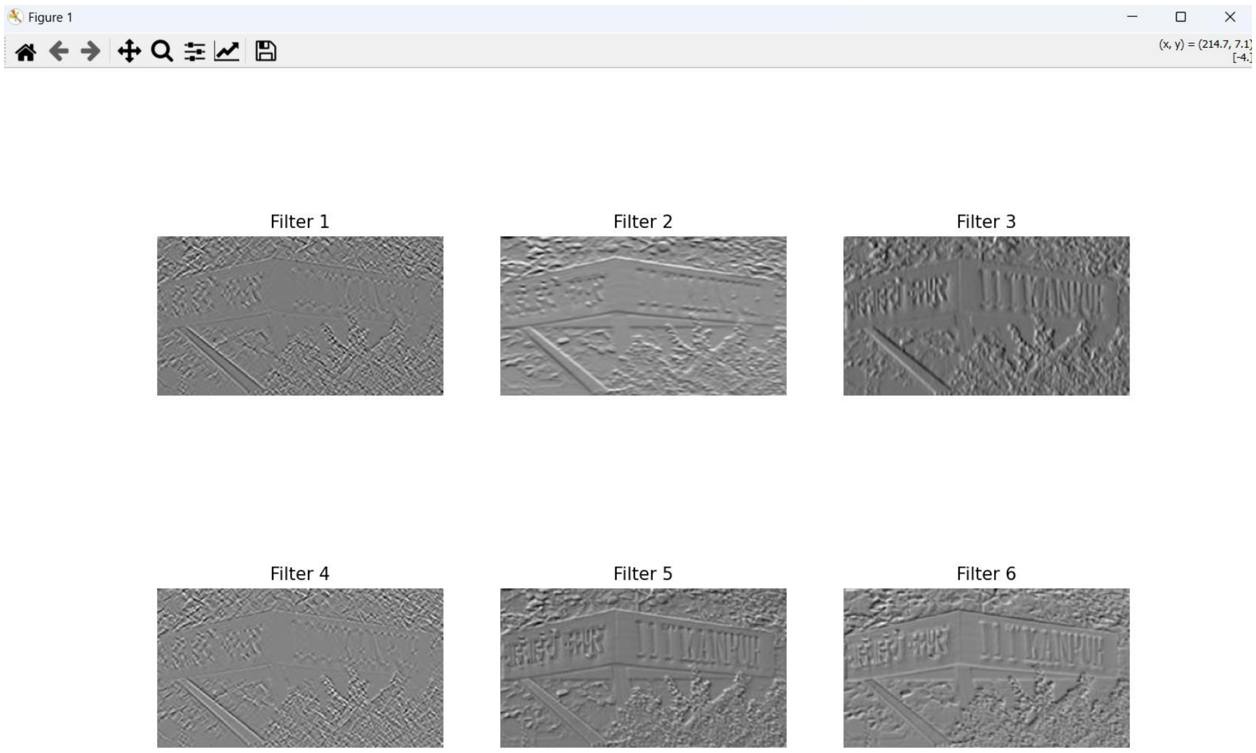


Figure 1
(x, y) = (214.7, 7.1)
[-4.]
Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Filter 6