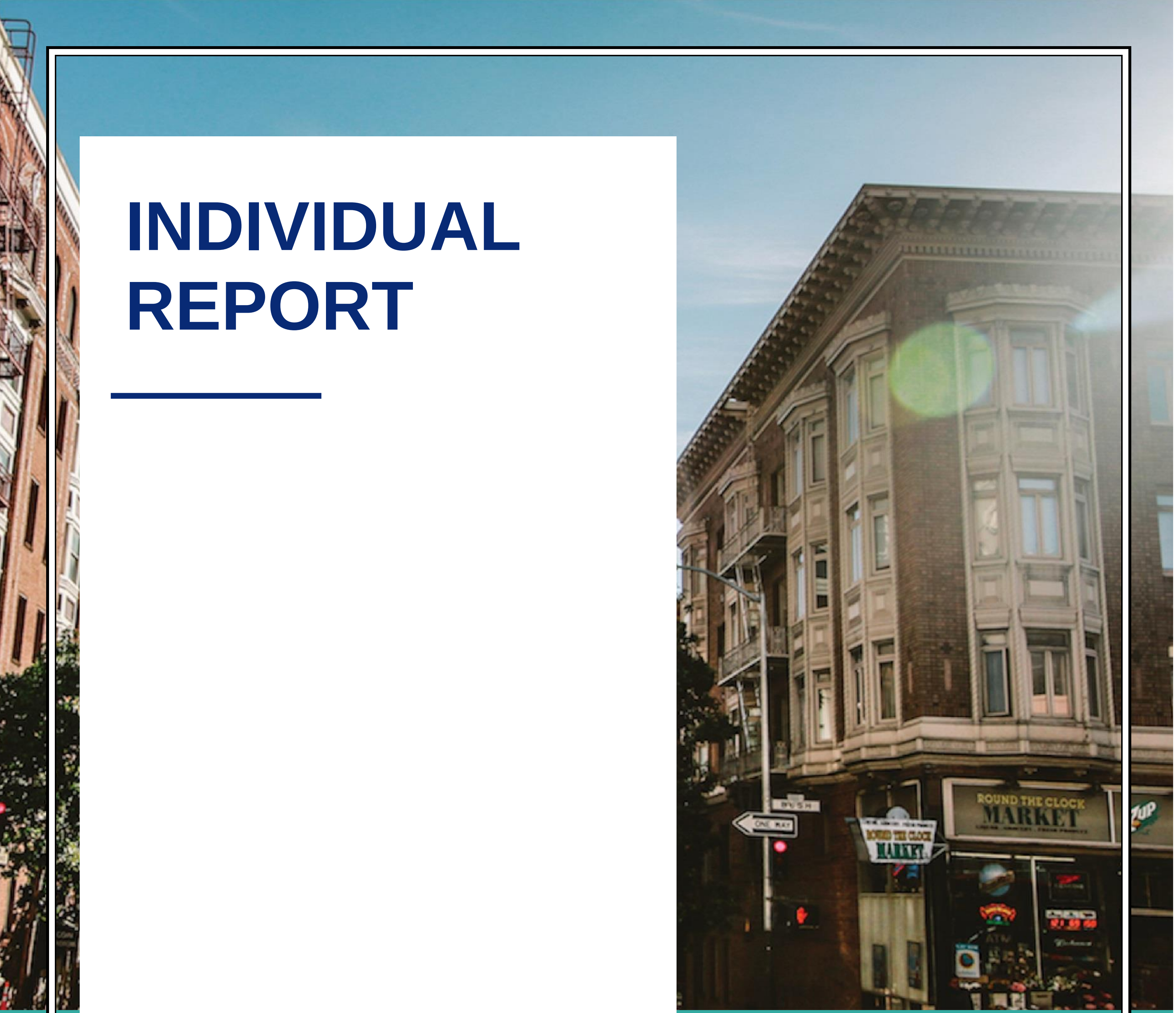


INDIVIDUAL REPORT

19R611-AI and Vision Systems Laboratory

RAGHUL T
21R228

Problem Statement

Perform sharpening filters in the video streams with and without using in-built functions. (Sobel,Prewitt,Canny). Check the performance

CODE:

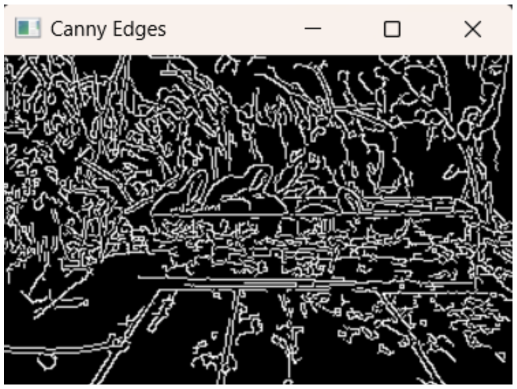
```
import cv2
import numpy as np

def sharpen_without_inbuilt(image):
    # Sharpening using a kernel
    kernel = np.array([[0, -1, 0],[-1, 5, -1],[0, -1, 0]], np.float32)
    sharpened = cv2.filter2D(image, -1, kernel)
    return sharpened

def sharpen_with_sobel(image):
    # Sharpening using Sobel filter
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    sobel_combined = cv2.magnitude(sobel_x, sobel_y)
    return sobel_combined

def sharpen_with_prewitt(image):
    # Sharpening using Prewitt filter
    prewitt_kernel_x = np.array([[1, 1, 1],[0, 0, 0],[-1, -1, -1]], np.float32)
    prewitt_kernel_y = np.array([[-1, 0, 1],[-1, 0, 1],[-1, 0, 1]], np.float32)
    prewitt_x = cv2.filter2D(image, cv2.CV_64F, prewitt_kernel_x)
    prewitt_y = cv2.filter2D(image, cv2.CV_64F, prewitt_kernel_y)
    prewitt_combined = cv2.magnitude(prewitt_x, prewitt_y)
    return prewitt_combined
```

```
# Open a video stream (you can replace '0' with the video file path)
cap = cv2.VideoCapture(RABBIT.mp4')

while True:
    ret, frame = cap.read()

    if not ret:
        print("Error reading video stream")
        break

    # Resize the frame for better visualization
    frame = cv2.resize(frame, (640, 480))

    # Sharpen without in-built function
    sharpened_without_inbuilt = sharpen_without_inbuilt(frame.copy())

    # Sharpen with Sobel
    sharpened_with_sobel = sharpen_with_sobel(frame.copy())

    # Sharpen with Prewitt
    sharpened_with_prewitt = sharpen_with_prewitt(frame.copy())

    #Apply Canny edge detector for comparison
    canny_edges = cv2.Canny(frame, 50, 150)

    # Display the frames
    cv2.imshow('Original', frame)
    cv2.imshow('SharpenedwithoutIn-built', harpened_without_inbuilt.astype(np.uint8))
    cv2.imshow('Sharpened with Sobel', sharpened_with_sobel.astype(np.uint8))
    cv2.imshow('Sharpened with Prewitt', sharpened_with_prewitt.astype(np.uint8))
    cv2.imshow('Canny Edges', canny_edges)
    # Break the loop when 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object and close windows
cap.release()
cv2.destroyAllWindows()
```

INPUT VIDEO

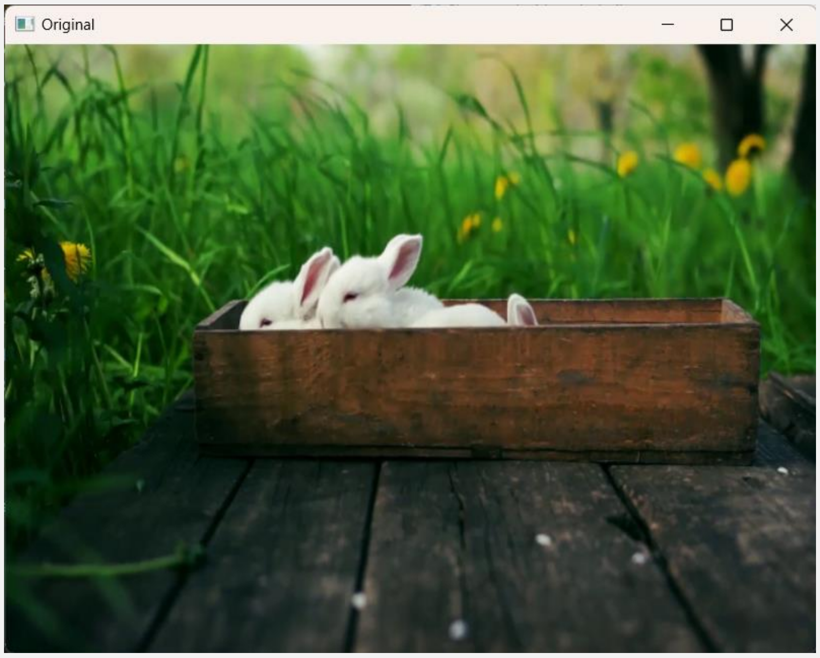
INPUT VIDEO:

OUTPUT VIDEO:

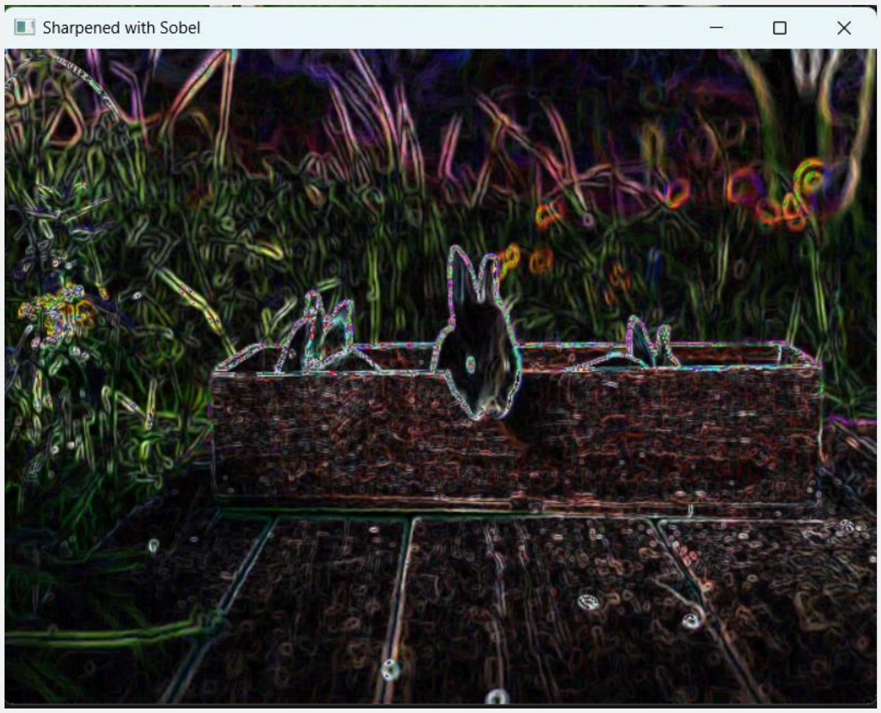


OUTPUT VIDEO

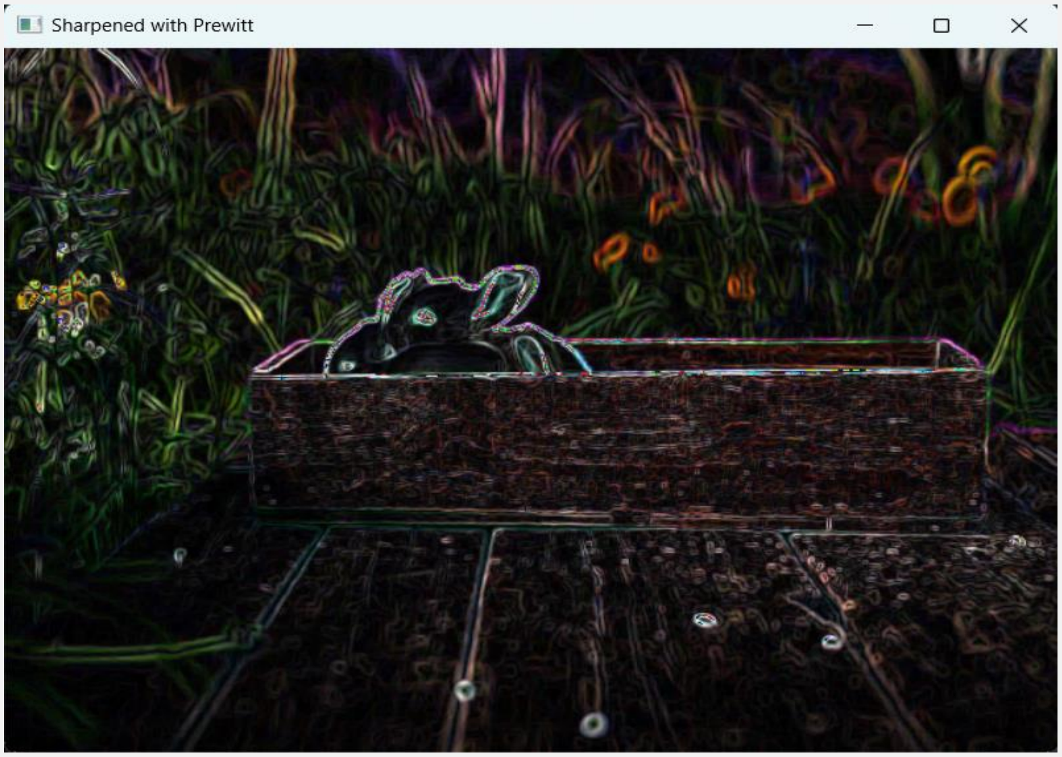
ORIGINAL:



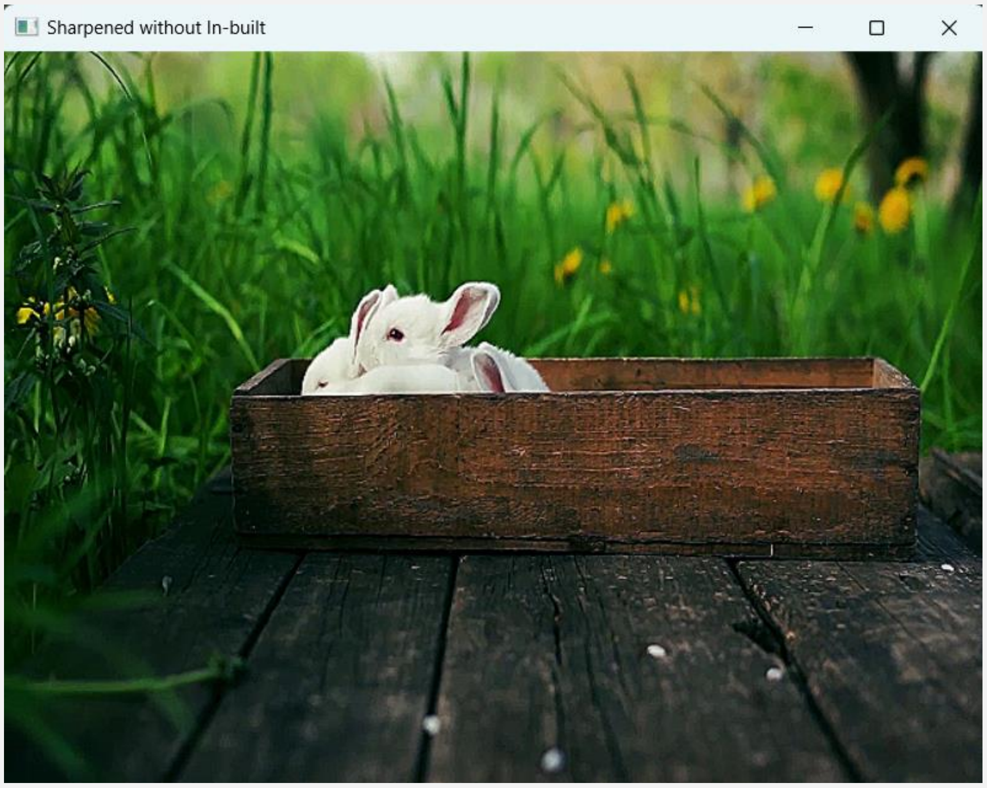
SHARPENED WITH SOBEL:



SHARPENED WITH PREWITT:



SHARPENED WITHOUT IN-BUILT:



OBJECTIVE:

The objective of the provided code is to perform real-time edge detection on a video stream using Sobel, Prewitt, and Canny operators in Python with OpenCV. It compares the results, measures the processing time for each method, and demonstrates custom sharpening functions for Sobel and Prewitt operators.

Software Used:

1. **PyCharm IDE**

PyCharm is an Integrated Development Environment (IDE) for the Python programming language.

Developed by JetBrains, it provides a comprehensive set of tools for Python development, including code assistance, graphical debugger, syntax highlighting, project navigation, and more.

2.**Libraries used:**

➢ **NumPy**

NumPy is a powerful numerical computing library for Python.

It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. Essential for scientific and mathematical computing in Python.

➢ **OpenCV-python**

OpenCV (Open-Source Computer Vision Library) is a computer vision and machine learning software library. It contains various tools and functions for image and video processing, including object detection, face recognition, and more. The OpenCV-python package is a Python wrapper for the OpenCV library.

- **Matplotlib**

Matplotlib is a 2D plotting library for Python. It produces static, animated, and interactive visualizations in Python. Commonly used for creating graphs, charts, and other visual representations of data.

WORKING PRINCIPAL:

- The working principle of the provided code involves capturing a video stream, processing each frame using three different edge detection methods (Sobel, Prewitt, and Canny), and displaying the original and sharpened frames in real-time. The code aims to demonstrate the application of these edge detection filters to enhance edges and features in the video.

- Here's a breakdown of the working principle:

- Initialization: The script starts by initializing necessary libraries and functions, including custom functions for sharpening using Sobel and Prewitt operators.

- Video Capture: It opens a video capture object using OpenCV and checks if the capture is successful.

- Processing with Sobel and Prewitt: It processes the video frames in a loop, converting each frame to grayscale and applying the Sobel and Prewitt operators to enhance edges. The sharpened frames are displayed in real-time alongside the original frames.

- Performance Measurement: The script measures the processing time for each method and prints the results. This allows you to compare the computational efficiency of Sobel and Prewitt edge detection.

- Video Processing with Canny: It reopens the video capture object and processes the frames using the Canny edge detection method. The edges obtained using Canny are displayed alongside the original frames.

- Performance Measurement for Canny: The script measures the processing time for the Canny operator and prints the result. This allows you to compare the computational efficiency of the Canny edge detection method.

- Cleanup: Finally, the script releases the video capture object and closes all windows.

CODE OVERVIEW:

This code can be divided into several sections:

1. Importing Libraries: Importing necessary libraries such as `cv2` (OpenCV), `numpy`, and `time`.

2. Sharpening Functions: Two custom functions, `sharpen_image_sobel` and `sharpen_image_prewitt`, are defined to apply sharpening using the Sobel and Prewitt operators, respectively.

3. Video Processing Function: The `process_video` function captures frames from a video stream, converts them to grayscale, applies a specified sharpening filter, and displays the original and sharpened frames in real-time. The loop continues until the user presses 'q'.

4. Main Processing Section: This section opens a video capture object, checks for success, and then processes the video using Sobel, Prewitt, and Canny operators. The processing time for each method is measured and printed.

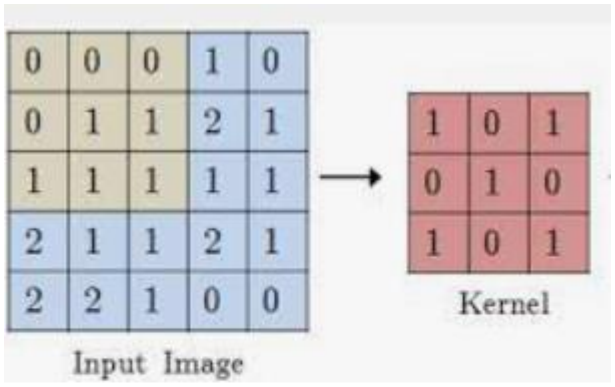
5. Cleanup: After video processing is complete, the video capture object is released, and all OpenCV windows are closed.

Conclusion:

In conclusion, the code successfully demonstrates real-time edge detection on a video stream through the application of Sobel, Prewitt, and Canny operators using OpenCV in Python. The comparison of processing times for each method provides insights into their computational efficiency. The inclusion of custom sharpening functions enhances the code's flexibility and adaptability. Overall, the script offers a practical foundation for individuals interested in exploring and implementing edge detection techniques in video processing applications. Its interactive display of original and processed frames facilitates a visual understanding of the impact of different edge detection methods.

PROBLEM STATEMENT:

Write a python program for convolution operation for the given pixel and kernel values



CODE:

```
def convolution(input_matrix, kernel):
    input_height, input_width = len(input_matrix), len(input_matrix[0])
    kernel_height, kernel_width = len(kernel), len(kernel[0])

    # Calculate the output matrix dimensions
    output_height = input_height - kernel_height + 1
    output_width = input_width - kernel_width + 1

    # Initialize the output matrix with zeros
    output_matrix = [[0] * output_width for _ in range(output_height)]

    # Perform convolution
    for i in range(output_height):
        for j in range(output_width):
            # Extract the region of interest (ROI) from the input matrix
            roi = [row[j:j + kernel_width] for row in input_matrix[i:i + kernel_height]]
```

```
# Perform element-wise multiplication and sum to get the convolution result
            output_matrix[i][j] = sum([sum([roi_row[col] * kernel_row[col] for col
in range(kernel_width)]) for roi_row, kernel_row in zip(roi, kernel)])

    return output_matrix

if __name__ == "__main__":
    # Input pixel values
    input_matrix = [
        [0,0,0,1,0],
        [0,1,1,2,1],
        [1,1,1,1,1],
        [2,1,1,2,1],
        [2,2,1,0,0]
    ]

    # Kernel values
    kernel = [
        [1, 0, 1],
        [0, 1, 0],
        [1, 0, 1]
    ]

    # Perform convolution
    result = convolution(input_matrix, kernel)

    # Display the result
    for row in result:
        print(row)
```

OUTPUT:

INPUT MATRIX:

```
 [
        [0,0,0,1,0],
        [0,1,1,2,1],
        [1,1,1,1,1],
        [2,1,1,2,1],
        [2,2,1,0,0]
 ]
```

KERNEL MATRIX:

```
[
        [1, 0, 1],
        [0, 1, 0],
        [1, 0, 1]
]
```

OUTPUT MATRIX:

```
[3, 4, 4]
[5, 7, 5]
[6, 5, 5]
```

PROBLEM STATEMENT:

Analyse histograms and experiment on finding a Gray Level threshold that generates accurate regions for the coin image. Do these experiments both for the filtered and non-filtered images? Repeat the same with automatic thresholding. Analyse which thresholding yields an accurate result.

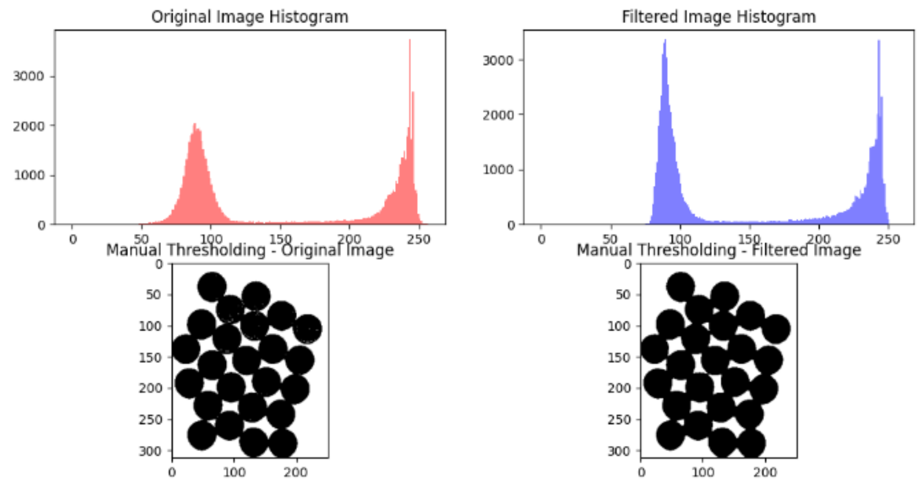
CODE:

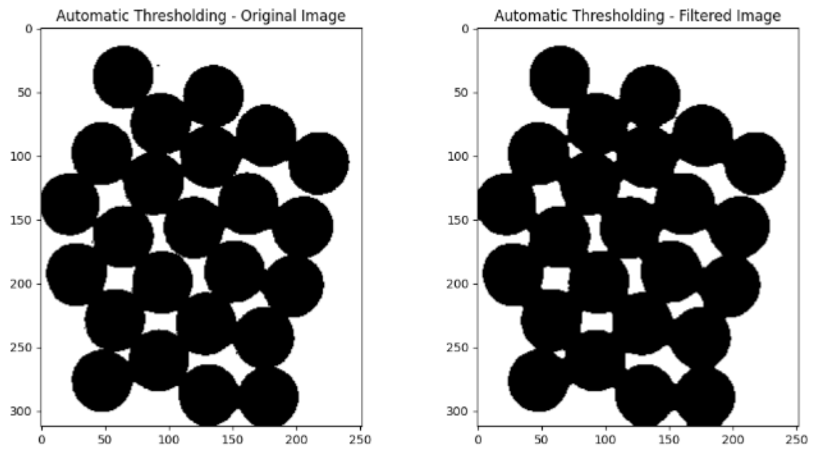
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def custom_noise_reduction(image):
    # Example: Use a bilateral filter for noise reduction
    denoised_image = cv2.bilateralFilter(image, 9, 75, 75)
    return denoised_image

# Read the coin image
image_path = r"C:\Users\RAGHUL\Downloads\12345.jpg"
original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Custom noise reduction for filtered image
filtered_image = custom_noise_reduction(original_image)

# Plot histograms for both filtered and non-filtered images
plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.hist(original_image.flatten(), bins=256, range=[0,256], color='r', alpha=0.5)
plt.title('Original Image Histogram')

plt.subplot(2, 2, 2)
plt.hist(filtered_image.flatten(), bins=256, range=[0,256], color='b', alpha=0.5)
plt.title('Filtered Image Histogram')
```

```
# Manual thresholding experiments
manual_threshold = 120  # You can experiment with different threshold values

_, manual_result_original = cv2.threshold(original_image, manual_threshold, 255,
cv2.THRESH_BINARY)
_, manual_result_filtered = cv2.threshold(filtered_image, manual_threshold, 255,
cv2.THRESH_BINARY)

plt.subplot(2, 2, 3)
plt.imshow(manual_result_original, cmap='gray')
plt.title('Manual Thresholding - Original Image')

plt.subplot(2, 2, 4)
plt.imshow(manual_result_filtered, cmap='gray')
plt.title('Manual Thresholding - Filtered Image')

plt.show()

# Automatic thresholding experiments
_, auto_result_original = cv2.threshold(original_image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
_, auto_result_filtered = cv2.threshold(filtered_image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(auto_result_original, cmap='gray')
plt.title('Automatic Thresholding - Original Image')

plt.subplot(1, 2, 2)
plt.imshow(auto_result_filtered, cmap='gray')
plt.title('Automatic Thresholding - Filtered Image')

plt.show()
```

INPUT IMAGE:

OUTPUT:





NAME: RAHIL ALTHAF WANI
ROLL NO: 21R229

Example 3: Different Template Matching Algorithms

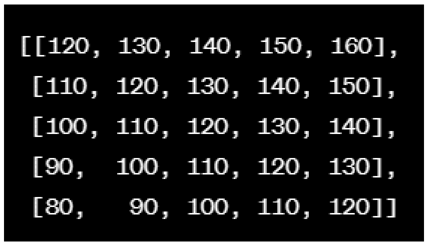
Aim:
Implement different template matching algorithms in Python using matrices to find a predefined sub-matrix within a larger matrix.

Software Packages Used:
- Python
- NumPy

OpenCV Inbuilt Functions Used:
- `cv2.matchTemplate`: Applies template matching using various algorithms.
- Drawing Functions (`cv2.rectangle`): Used to draw rectangles on the image.

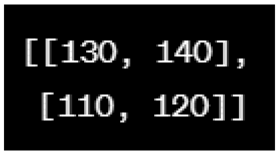
Explanation:
This example showcases different template matching algorithms provided by OpenCV, such as `cv2.TM_CCOEFF`, `cv2.TM_SQDIFF`, etc. The code applies these algorithms to find the best match for a template within a larger image. The results are then displayed to compare the effectiveness of each algorithm.

Input Matrix



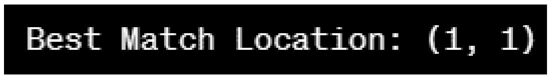
```
[[120, 130, 140, 150, 160],
 [110, 120, 130, 140, 150],
 [100, 110, 120, 130, 140],
 [90,  100, 110, 120, 130],
 [80,   90, 100, 110, 120]]
```

Template



```
[[130, 140],
 [110, 120]]
```

O/P Matrix



```
Best Match Location: (1, 1)
```

Code:

```
import numpy as np
import cv2
def template_matching(image, template):
    image_height, image_width = image.shape
    template_height, template_width = template.shape
    best_match_value = float('inf')
    best_match_location = (0, 0)
    for y in range(image_height - template_height + 1):
        for x in range(image_width - template_width + 1):
            region = image[y:y+template_height, x:x+template_width]
            ssd = np.sum((region - template)**2)
            if ssd < best_match_value:
                best_match_value = ssd
                best_match_location = (x, y)
    return best_match_location

image = np.random.randint(0, 255, size=(100, 100), dtype=np.uint8)
template = image[20:40, 30:50].copy()

best_match = template_matching(image, template)

result_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
cv2.rectangle(result_image, best_match, (best_match[0] + template.shape[1], best_match[1] + template.shape[0]), (0, 0, 255), 2)

cv2.imshow('Input Image', image)
cv2.imshow('Template', template)
cv2.imshow('Result', result_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

NAME: Reegan Abishek
ROLL NO: 21R230

Problem statement:

To perform convolution operation for given pixel values and kernel values.

CODE:

```
import numpy as np
# Define the input matrix
mat = np.matrix('0 0 0 1 0; 0 1 1 2 1; 1 1 1 1 1; 2 1 1 2 1 ; 2 2 1 0 0')
print("Input Matrix:")
print(mat)
# Define the kernel for convolution
kernel = np.matrix('1 0 1; 0 1 0; 1 0 1')
print("\nKernel:")
print(kernel)
# Define a function to calculate the average multiplication for convolution
def mulnavg(mat1, mat2):
    avg = 0
    for i in range(0, 3):
        for j in range(0, 3):
            avg = avg + mat1[i, j] * mat2[i, j]
    return avg / 9
# Create a padded matrix for convolution
padd = np.zeros([7, 7], dtype=int)
padd[1:6, 1:6] = mat
print("\nPadded Matrix:")
print(padd)
# Perform convolution and print the result
print("\nConvolution Result:")
for i in range(0, 5):
    for j in range(0, 5):
        # Extract a 3x3 matrix for convolution
        mat1 = padd[i:i + 3, j:j + 3]
        # Calculate the convolution result and print it
        result = int(mulnavg(mat1, kernel))
        print(result, end=' ')
    print()
```

OUTPUT:

```
C:\Users\areeg\PycharmProjects\pythonProject\.venv\Scripts\python.exe
C:\Users\areeg\PycharmProjects\pythonProject\convolution.py
[[0 0 0 1 0]
 [0 1 1 2 1]
 [1 1 1 1 1]
 [2 1 1 2 1]
 [2 2 1 0 0]]
[[1 0 1]
 [0 1 0]
 [1 0 1]]
[[0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0]
 [0 0 1 1 2 1 0]
 [0 1 1 1 1 1 0]
 [0 2 1 1 2 1 0]
 [0 2 2 1 0 0 0]
 [0 0 0 0 0 0 0]]
0

Process finished with exit code 0
```

Explaination:

The convolutions are used to extract features from images. We use a function “mulnavg” to get the multiple the given 3*3 matrix and to get the avg. we convert the 5*5 matrix to 7*7 matrix by padding to fill the edge with ‘0’ elements for convolution. We would have given the mat1 as padd[i:i+3;j:j+3], so when convolution the 3*3 matrix can be selected and move as 3 through 7*7. Then we just print the result after the mat1 convolution finishes.