AI VISIONS LAB

7(A) A.**Write a function in Python that accepts a decimal number and returns the equivalent binary number.**

**To make this simple, the decimal number will always be less than 1,024, so the binary number returned**

**will always be less than ten digits long.**

def decimal\_to\_binary(decimal\_number):

# Convert to integer before converting to binary binary\_number = bin(int(decimal\_number))[2:] # Remove the '0b' prefix return binary\_number.zfill(10)

def convert\_and\_display():

decimal\_number = float(input("Enter a decimal number (between 0 and 1023): ")) binary\_number = decimal\_to\_binary(decimal\_number) print(f"The binary representation of {decimal\_number} is: {binary\_number}")

# Example usage convert\_and\_display()

**9(B) Apply a box filter without using in-built function on any image and show the difference on**

**applying a median filter on the same.**

import cv2 import numpy as np

def box\_filter(img, kernel\_size): height, width = img.shape output = np.zeros((height, width), dtype=np.uint8)

for i in range(height - kernel\_size + 1): for j in range(width - kernel\_size + 1): output[i, j] = np.mean(img[i:i+kernel\_size, j:j+kernel\_size]) return output

def median\_filter(img, kernel\_size):

height, width = img.shape output = np.zeros((height, width), dtype=np.uint8)

for i in range(height - kernel\_size + 1): for j in range(width - kernel\_size + 1):

output[i, j] = np.median(img[i:i+kernel\_size, j:j+kernel\_size])

return output

# Read an image image = cv2.imread('IMG\_2325.jpg', cv2.IMREAD\_GRAYSCALE)

# Apply box filter box\_filtered = box\_filter(image, kernel\_size=3)

# Apply median filter median\_filtered = median\_filter(image, kernel\_size=3)

# Display original, box-filtered, and median-filtered images cv2.imshow('Original Image', image) cv2.imshow('Box Filtered', box\_filtered) cv2.imshow('Median Filtered', median\_filtered) cv2.waitKey(0) cv2.destroyAllWindows()

9(a) **a. Remove the glare from the image and display the enhanced image**.

import cv2 import numpy as np

def remove\_glare(image\_path): # Read the input image original\_image = cv2.imread(image\_path)

# Convert the image to grayscale gray = cv2.cvtColor(original\_image, cv2.COLOR\_BGR2GRAY)

# Apply thresholding to create a binary mask of the glare

\_, binary\_mask = cv2.threshold(gray, 200, 255, cv2.THRESH\_BINARY)

# Inpainting: Fill in the glare region using nearby pixel values inpainted\_image = cv2.inpaint(original\_image, binary\_mask, inpaintRadius=3, flags=cv2.INPAINT\_TELEA)

# Display the original and enhanced images cv2.imshow("Original Image", original\_image) cv2.imshow("Enhanced Image", inpainted\_image) cv2.waitKey(0) cv2.destroyAllWindows()

# Example usage: image\_path = "path/to/your/image\_with\_glare.jpg" remove\_glare(image\_path)

**14. Find the number of coins in the given image.**

# Import necessary libraries import cv2

# Read the image from file image = cv2.imread('Screenshot 2024-02-04 at 7.02.01 PM.png')

# Convert the image to grayscale gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian blur to the grayscale image blur = cv2.GaussianBlur(gray, (11, 11), 0)

# Use Canny edge detector to find edges in the blurred image canny = cv2.Canny(blur, 30, 150, 3)

# Dilate the edges to connect nearby edges and close gaps dilated = cv2.dilate(canny, (1, 1), iterations=2)

# Find contours in the dilated image

(cnt, hierarchy) = cv2.findContours(dilated.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

# Convert image to RGB for visualization rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)z

# Draw contours on the RGB image cv2.drawContours(rgb, cnt, -1, (0, 255, 0), 2)

# Print the number of detected coins print('Coins in the image: ', len(cnt))

# Wait for a key press and close all OpenCV windows cv2.waitKey(0) cv2.destroyAllWindows()

1.

# Python program to illustrate # Python program to illustrate # template matching import cv2 import numpy as np # Read the main image

img\_rgb = cv2.imread('TSITP.jpg')

# Convert it to grayscale img\_gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY) # Read the template template = cv2.imread('TSITP1.jpg', 0)

# Store width and height of template in w and h w, h = template.shape[::-1] # Perform match operations.

res = cv2.matchTemplate(img\_gray, template, cv2.TM\_CCOEFF\_NORMED) # Specify a threshold threshold = 0.8

# Store the coordinates of matched area in a numpy array loc = np.where(res >= threshold) # Draw a rectangle around the matched region. for pt in zip(\*loc[::-1]):

cv2.rectangle(img\_rgb, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2) # Show the final image with the matched area. cv2.imshow('Detected', img\_rgb) cv2.waitKey(0)

2. **Demonstrates multiple templates matching to identify a predefined shape in an image.**

import cv2 import numpy as np

def find\_predefined\_shape(image\_path, templates):

# Read the input image img = cv2.imread(image\_path) gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Initialize a list to store results results = []

# Loop through each template for template\_path in templates: # Read the template image template = cv2.imread(template\_path, 0) w, h = template.shape[::-1]

# Perform template matching res = cv2.matchTemplate(gray\_img, template, cv2.TM\_CCOEFF\_NORMED) threshold = 0.8 loc = np.where(res >= threshold)

# Draw rectangles around the matched areas for pt in zip(\*loc[::-1]): cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2) results.append(pt)

# Display the result cv2.imshow('Result', img) cv2.waitKey(0) cv2.destroyAllWindows()

return results

# Define the image path and templates image\_path = 'image\_to\_search.png' templates = ['template1.png', 'template2.png', 'template3.png']

# Find the predefined shape in the image matched\_points = find\_predefined\_shape(image\_path, templates)

# Print the coordinates of the matched points print("Matched points:", matched\_points)

3. Demonstrate how to implement different template matching algorithms in Python using matrices to find

a predefined sub-matrix within a larger matrix.

import cv2 import numpy as np

def template\_matching(image, template, method=cv2.TM\_CCOEFF\_NORMED):

result = cv2.matchTemplate(image, template, method) min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result) if method in [cv2.TM\_SQDIFF, cv2.TM\_SQDIFF\_NORMED]:

match\_loc = min\_loc else:

match\_loc = max\_loc return match\_loc

# Generate a random larger matrix larger\_matrix = np.random.randint(0, 255, (300, 300)).astype(np.uint8)

# Define a predefined sub-matrix (template) template = np.random.randint(0, 255, (50, 50)).astype(np.uint8)

# Find the predefined sub-matrix within the larger matrix using different template matching methods methods = [cv2.TM\_SQDIFF, cv2.TM\_SQDIFF\_NORMED, cv2.TM\_CCORR, cv2.TM\_CCORR\_NORMED, cv2.TM\_CCOEFF, cv2.TM\_CCOEFF\_NORMED]

for method in methods:

# Make a copy of the larger matrix to draw the rectangle img\_display = larger\_matrix.copy()

# Perform template matching match\_loc = template\_matching(larger\_matrix, template, method)

# Draw a rectangle around the matched region w, h = template.shape[::-1] top\_left = match\_loc bottom\_right = (top\_left[0] + w, top\_left[1] + h)

cv2.rectangle(img\_display, top\_left, bottom\_right, 255, 2)

# Display the result cv2.imshow('Result using Method: {}'.format(method), img\_display) cv2.waitKey(0) cv2.destroyAllWindows()

6. Design real-time edge detection capable of processing video streams with and without using in-built

functions. (Sobel, Prewitt,Canny)

With import cv2 import numpy as np

*# Define the video capture object* cap = cv2.VideoCapture(0)

while True:

*# Capture the video frame by frame*

ret, frame = cap.read()

*# Convert the frame to grayscale*

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

*# Define the Sobel, Prewitt, and Canny filters*

sobel\_x = cv2.Sobel(gray,cv2.CV\_64F, 1, 0, ksize=5) sobel\_y = cv2.Sobel(gray, cv2.CV\_64F, 0,1, ksize=5)

prewitt\_x\_kernel = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])

prewitt\_y\_kernel = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])

prewitt\_x = cv2.filter2D(gray, cv2.CV\_64F, prewitt\_x\_kernel)

prewitt\_y = cv2.filter2D(gray,cv2.CV\_64F, prewitt\_y\_kernel)

canny = cv2.Canny(gray, 30,100)

*# Display the resulting frames*

cv2.imshow('Sobel X', sobel\_x) cv2.imshow('Sobel Y', sobel\_y)

cv2.imshow('Prewitt X', prewitt\_x)

cv2.imshow('Prewitt Y', prewitt\_y)

cv2.imshow('Canny', canny)

*# Exit the loop if 'q' is pressed* if cv2.waitKey(1) & 0xFF == ord('q'): break

*# Release the capture and destroy all windows* cap.release() cv2.destroyAllWindows()

Without

import cv2 import numpy as np

# Define the video capture object cap = cv2.VideoCapture(0)

while True:

# Capture the video frame by frame

ret, frame = cap.read()

# Convert the frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Define the Sobel, Prewitt, and Canny filters

sobel\_x\_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

sobel\_y\_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

prewitt\_x\_kernel = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])

prewitt\_y\_kernel = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])

canny = cv2.Canny(gray,30, threshold2=100)

# Apply the Sobel, Prewitt, and Canny filters

sobel\_x = cv2.filter2D(src=gray, ddepth=cv2.CV\_64F, kernel=sobel\_x\_kernel) sobel\_y = cv2.filter2D(src=gray, ddepth=cv2.CV\_64F, kernel=sobel\_y\_kernel)

prewitt\_x = cv2.filter2D(src=gray, ddepth=cv2.CV\_64F, kernel=prewitt\_x\_kernel)

prewitt\_y = cv2.filter2D(src=gray, ddepth=cv2.CV\_64F, kernel=prewitt\_y\_kernel)

# Display the resulting frames

cv2.imshow('Sobel X', sobel\_x)

cv2.imshow('Sobel Y', sobel\_y)

cv2.imshow('Prewitt X', prewitt\_x)

cv2.imshow('Prewitt Y', prewitt\_y)

cv2.imshow('Canny', canny)

# Exit the loop if 'q' is pressed if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the capture and destroy all windows

*cap.release()* cv2.destroyAllWindows()

17,3 Pixel values:

180 160 160 140 120

110 110 120 140 120

110 140 120 120 140

120 160 160 170 170

170 120 110 140 110

For the above pixel values perform Image intensity transformations (3 members)

import numpy as np

# Given pixel values pixel\_values = np.array([ [180, 160, 160, 140, 120],

[110, 110, 120, 140, 120],

[110, 140, 120, 120, 140], [120, 160, 160, 170, 170],

[170, 120, 110, 140, 110] ])

# Linear Scaling min\_val = np.min(pixel\_values) max\_val = np.max(pixel\_values) linear\_scaled\_values = ((pixel\_values - min\_val) / (max\_val - min\_val)) \* 255

# Logarithmic Transformation log\_transformed\_values = np.log1p(pixel\_values)

# Power-law (Gamma) Transformation gamma = 0.5 gamma\_transformed\_values = np.power(pixel\_values, gamma)

# Display the original and transformed pixel values print("Original Pixel Values:") print(pixel\_values) print("\nLinear Scaled Values:") print(linear\_scaled\_values.astype(int)) print("\nLogarithmic Transformed Values:") print(log\_transformed\_values.astype(int)) print("\nGamma Transformed Values:") print(gamma\_transformed\_values.astype(int))

19) View the below pixel values in histogram and perform histogram equalization. Display the new pixelvales

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

def display\_histogram\_and\_equalize(matrix): # Display the pixel values in a histogram plt.hist(matrix.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7) plt.title('Original Histogram') plt.xlabel('Pixel Value') plt.ylabel('Frequency') plt.show()

# Apply histogram equalization equalized\_matrix = cv2.equalizeHist(matrix)

# Display the pixel values in a histogram after equalization plt.hist(equalized\_matrix.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7) plt.title('Equalized Histogram') plt.xlabel('Pixel Value') plt.ylabel('Frequency') plt.show()

# Display the original and equalized matrices print("Original Matrix:") print(matrix) print("\nEqualized Matrix:") print(equalized\_matrix)

# Example usage with an 8x8 matrix input\_matrix = np.array([[10, 20, 30, 40, 50, 60, 70, 80],

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

[170, 180, 190, 200, 210, 220, 230, 240],

[250, 20, 30, 40, 50, 60, 70, 80],

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

[170, 180, 190, 200, 210, 220, 230, 240],

[250, 20, 30, 40, 50, 60, 70, 80],

[90, 100, 110, 120, 130, 140, 150, 160]], dtype=np.uint8) display\_histogram\_and\_equalize(input\_matrix) 5.

22. Construct a 5\*5 matrix with elements starting with ASCII values of your name followed by ASCII

values of your register number. The first letter of each word and initials should be in capital. Space

should be treated as an element. The balance elements should be filled with the pattern 2x , where x is

the last digit of your register number. Perform thresholding for the above matrix.

import cv2 import numpy as np

def generate\_matrix(name, register\_number):

# Convert name and register number to ASCII representation name\_ascii = [ord(char) for char in name] register\_number\_ascii = [int(digit) for digit in str(register\_number)]

# Ensure proper capitalization for the first letter of each word and initials name = name.title()

# Create a 5x5 matrix to hold the generated data matrix = np.zeros((5, 5), dtype=np.uint8)

# Fill the matrix with ASCII values of name and register number idx = 0 for i in range(5): for j in range(5): if idx < len(name\_ascii):

matrix[i, j] = name\_ascii[idx] idx += 1 elif idx < len(name\_ascii) + len(register\_number\_ascii): matrix[i, j] = register\_number\_ascii[idx - len(name\_ascii)] idx += 1 else:

last\_digit = register\_number\_ascii[-1] matrix[i, j] = 2 \* last\_digit return matrix

# Generate the matrix your\_name = "shudhi" # Change this to your name with proper capitalization your\_register\_number = 241 # Change this to your register number generated\_matrix = generate\_matrix(your\_name, your\_register\_number)

# Perform thresholding

\_, thresholded\_matrix = cv2.threshold(generated\_matrix, 127, 255, cv2.THRESH\_BINARY)

# Display the matrices print("Generated Matrix:") print(generated\_matrix) print("\nThresholded Matrix:") print(thresholded\_matrix)

8.