Code gray –

import cv2

import vpi

import numpy as np

# Load the input image

image\_path = 'output/resized\_1.png'  # Replace with your image path

input\_image = cv2.imread(image\_path)

print("Image read")

# Check if the image was loaded correctly

if input\_image is None:

    raise ValueError("Error loading image")

# Convert the image from BGR to RGB format

input\_image\_rgb = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2RGB)

# Step 1: Create a VPI image using the input image dimensions (use shape attributes)

height, width, \_ = input\_image\_rgb.shape  # Extract height, width, and channels

with vpi.Backend.CUDA:

    # Create a VPI image with grayscale format (U8 is 8-bit grayscale)

    output\_vpi\_image = vpi.Image((width, height), vpi.Format.U8)

# Step 2: Convert the VPI image to grayscale manually if needed (since VPI doesn't directly support color conversion)

# Step 3: Threshold to isolate white regions in the grayscale image

threshold\_value = 200 # Threshold to isolate white regions

# Since we don't have access to a grayscale VPI image directly, use OpenCV for thresholding

input\_image\_gray = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2GRAY)  # Convert to grayscale using OpenCV

# Apply the threshold to the grayscale image

\_, mask\_white = cv2.threshold(input\_image\_gray, threshold\_value, 255, cv2.THRESH\_BINARY)

# Step 4: Mask the input image to retain only the white regions

result\_image = cv2.bitwise\_and(input\_image, input\_image, mask=mask\_white)

# Display the input and the result images using OpenCV

cv2.imshow('Input Image', input\_image)

cv2.imshow('Background Eliminated Image (White Regions)', result\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

code gray contour

import cv2

import numpy as np

import vpi

# Load the input image

image\_path = 'output/result\_image.png'  # Replace with your image path

input\_image = cv2.imread(image\_path)

print("Image read")

# Check if the image was loaded correctly

if input\_image is None:

    raise ValueError("Error loading image")

# Convert the image from BGR to RGB format, as VPI expects RGB input

input\_image\_rgb = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2RGB)

# Get the image dimensions

height, width, \_ = input\_image\_rgb.shape

# Step 1: Create a VPI image and convert to grayscale using VPI

with vpi.Backend.CUDA:  # Use CUDA backend for GPU acceleration

    # Create VPI image from the input RGB image

    input\_vpi\_image = vpi.asimage(input\_image\_rgb)

    # Create an empty VPI image for grayscale output

    grayscale\_vpi\_image = vpi.Image((width, height), vpi.Format.U8)

    # Convert RGB to Grayscale using VPI

    with vpi.Stream() as stream:

        input\_vpi\_image.convert(vpi.Format.U8, grayscale\_vpi\_image, stream)

        stream.sync()  # Ensure processing is completed

    # Download the grayscale image from GPU to CPU for further processing

    grayscale\_image = grayscale\_vpi\_image.cpu()

# Step 2: Apply threshold using VPI

threshold\_value = 200

# Use OpenCV for thresholding since contour detection will be done in OpenCV

\_, binary\_thresh = cv2.threshold(grayscale\_image, threshold\_value, 255, cv2.THRESH\_BINARY)

# Step 3: Find contours of the bright regions using OpenCV

contours\_bright, \_ = cv2.findContours(binary\_thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Step 4: Create an output image to draw contours

output\_image = input\_image.copy()

# Draw contours on the output image (Green for bright regions)

cv2.drawContours(output\_image, contours\_bright, -1, (0, 255, 0), 2)

# Step 5: Get bounding box coordinates for each contour

bright\_region\_coords = [cv2.boundingRect(contour) for contour in contours\_bright]

bright\_region\_count = len(bright\_region\_coords)

print("Count of bright regions: ", bright\_region\_count)

# Show the output image using OpenCV

cv2.imshow('Bright regions', output\_image)

# Wait for key press and close the window

cv2.waitKey(0)

cv2.destroyAllWindows()

code red

import cv2

import vpi

import numpy as np

# Load the input image

image\_path = 'output/resized\_1.png' # Replace with your image path

input\_image = cv2.imread(image\_path)

print("Image read")

# Check if the image was loaded correctly

if input\_image is None:

raise ValueError("Error loading image")

# Convert the image from BGR to HSV format

hsv\_image = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2HSV)

# Define thresholds for bright red regions

lower\_bright\_red1 = np.array([0, 150, 150])

upper\_bright\_red1 = np.array([10, 255, 255])

lower\_bright\_red2 = np.array([170, 150, 150])

upper\_bright\_red2 = np.array([180, 255, 255])

# Create masks for the two ranges of red

mask\_bright\_red1 = cv2.inRange(hsv\_image, lower\_bright\_red1, upper\_bright\_red1)

mask\_bright\_red2 = cv2.inRange(hsv\_image, lower\_bright\_red2, upper\_bright\_red2)

# Combine the masks

mask\_red = cv2.bitwise\_or(mask\_bright\_red1, mask\_bright\_red2)

# Step 1: Load input image into VPI

with vpi.Backend.CUDA:

# Create VPI images for input and masks

vpi\_input\_image = vpi.asimage(input\_image)

vpi\_mask = vpi.Image(mask\_red, vpi.Format.U8)

# Apply the mask to retain red regions

vpi\_result\_image = vpi\_input\_image.multiply(vpi\_mask)

# Download the result back to CPU memory

result\_image = vpi\_result\_image.cpu()

# Display the input and the result images using OpenCV

cv2.imshow('Input Image', input\_image)

cv2.imshow('Red Regions Only', result\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

contour red code

import cv2

import numpy as np

import vpi

# Load the input image

image\_path = 'output/result\_image.png' # Replace with your image path

input\_image = cv2.imread(image\_path)

print("Image read")

# Check if the image was loaded correctly

if input\_image is None:

raise ValueError("Error loading image")

# Convert the image from BGR to HSV format for color detection

hsv\_image = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2HSV)

# Define thresholds for bright red regions

lower\_bright\_red1 = np.array([0, 150, 150])

upper\_bright\_red1 = np.array([10, 255, 255])

lower\_bright\_red2 = np.array([170, 150, 150])

upper\_bright\_red2 = np.array([180, 255, 255])

# Create masks for the two red ranges

mask\_bright\_red1 = cv2.inRange(hsv\_image, lower\_bright\_red1, upper\_bright\_red1)

mask\_bright\_red2 = cv2.inRange(hsv\_image, lower\_bright\_red2, upper\_bright\_red2)

# Combine the masks

mask\_red = cv2.bitwise\_or(mask\_bright\_red1, mask\_bright\_red2)

# Step 1: Apply mask using VPI

height, width, \_ = input\_image.shape

with vpi.Backend.CUDA: # Use CUDA backend for GPU acceleration

# Create VPI image from the input mask

mask\_vpi\_image = vpi.Image(mask\_red)

# Download the mask from GPU to CPU for further processing

mask\_red\_cpu = mask\_vpi\_image.cpu()

# Step 2: Find contours of the red regions using OpenCV

contours\_red, \_ = cv2.findContours(mask\_red\_cpu, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Step 3: Create an output image to draw contours

output\_image = input\_image.copy()

# Draw contours on the output image (Red for red regions)

cv2.drawContours(output\_image, contours\_red, -1, (0, 0, 255), 2)

# Step 4: Get bounding box coordinates for each red contour

red\_region\_coords = [cv2.boundingRect(contour) for contour in contours\_red]

red\_region\_count = len(red\_region\_coords)

print("Count of red regions: ", red\_region\_count)

# Show the output image with red contours

cv2.imshow('Red Regions', output\_image)

# Wait for key press and close the window

cv2.waitKey(0)

cv2.destroyAllWindows()

cropping(template matching) – vpi

import cv2

import numpy as np

import vpi

# Load the input image

image\_path = 'output/result\_image.png' # Replace with your image path

input\_image = cv2.imread(image\_path)

print("Image read")

# Check if the image was loaded correctly

if input\_image is None:

raise ValueError("Error loading image")

# Convert the image from BGR to HSV format for color detection

hsv\_image = cv2.cvtColor(input\_image, cv2.COLOR\_BGR2HSV)

# Define thresholds for bright red regions

lower\_bright\_red1 = np.array([0, 150, 150])

upper\_bright\_red1 = np.array([10, 255, 255])

lower\_bright\_red2 = np.array([170, 150, 150])

upper\_bright\_red2 = np.array([180, 255, 255])

# Create masks for the two red ranges

mask\_bright\_red1 = cv2.inRange(hsv\_image, lower\_bright\_red1, upper\_bright\_red1)

mask\_bright\_red2 = cv2.inRange(hsv\_image, lower\_bright\_red2, upper\_bright\_red2)

# Combine the masks

mask\_red = cv2.bitwise\_or(mask\_bright\_red1, mask\_bright\_red2)

# Step 1: Apply mask using VPI

height, width, \_ = input\_image.shape

with vpi.Backend.CUDA: # Use CUDA backend for GPU acceleration

# Create VPI image from the input mask

mask\_vpi\_image = vpi.Image(mask\_red)

# Download the mask from GPU to CPU for further processing

mask\_red\_cpu = mask\_vpi\_image.cpu()

# Step 2: Find contours of the red regions using OpenCV

contours\_red, \_ = cv2.findContours(mask\_red\_cpu, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Step 3: Create an output image to draw contours

output\_image = input\_image.copy()

# Draw contours on the output image (Red for red regions)

cv2.drawContours(output\_image, contours\_red, -1, (0, 0, 255), 2)

# Step 4: Get bounding box coordinates for each red contour

red\_region\_coords = [cv2.boundingRect(contour) for contour in contours\_red]

red\_region\_count = len(red\_region\_coords)

print("Count of red regions: ", red\_region\_count)

# Show the output image with red contours

cv2.imshow('Red Regions', output\_image)

# Wait for key press and close the window

cv2.waitKey(0)

cv2.destroyAllWindows()

cropping(template matching) – cv2

import cv2

import numpy as np

import matplotlib.pyplot as plt

import vpi

# Load the main image

main\_image = cv2.imread('/content/drive/MyDrive/cropping\_model/4.png')

# Load the template image

template = cv2.imread('/content/drive/MyDrive/Amit\_sir2/1.png')

# Check if the images are loaded successfully

if main\_image is None or template is None:

print("Error: Could not load the images.")

exit()

# Step 1: Convert images to grayscale using VPI

with vpi.Backend.CUDA: # Use CUDA backend for GPU acceleration

# Convert main image to grayscale

vpi\_main\_image = vpi.asimage(main\_image)

main\_gray\_vpi = vpi.Image(main\_image.shape[:2], vpi.Format.U8)

with vpi.Stream() as stream:

vpi\_main\_image.convert(vpi.Format.U8, main\_gray\_vpi, stream)

stream.sync()

main\_gray = main\_gray\_vpi.cpu() # Download grayscale image to CPU memory

# Convert template image to grayscale

vpi\_template\_image = vpi.asimage(template)

template\_gray\_vpi = vpi.Image(template.shape[:2], vpi.Format.U8)

with vpi.Stream() as stream:

vpi\_template\_image.convert(vpi.Format.U8, template\_gray\_vpi, stream)

stream.sync()

template\_gray = template\_gray\_vpi.cpu() # Download grayscale image to CPU memory

# Step 2: Perform template matching using OpenCV (since VPI doesn't have a direct function for it)

result = cv2.matchTemplate(main\_gray, template\_gray, cv2.TM\_CCOEFF\_NORMED)

# Step 3: Define a threshold to consider a match

threshold = 0.87

# Find locations where the result is above the threshold

locations = np.where(result >= threshold)

# If there are no matches, print "False" and exit

if len(locations[0]) == 0:

print("False")

exit()

# Step 4: Mark the matches on the original image

for loc in zip(\*locations[::-1]):

cv2.rectangle(main\_image, loc, (loc[0] + template.shape[1], loc[1] + template.shape[0]), (0, 255, 0), 2)

# Show the result using matplotlib

plt.imshow(cv2.cvtColor(main\_image, cv2.COLOR\_BGR2RGB))

plt.title('Template Matching Result')

plt.axis('off')

plt.show()

# Step 5: Extract the first matched region

top\_left = (locations[1][0], locations[0][0])

bottom\_right = (top\_left[0] + template.shape[1], top\_left[1] + template.shape[0])

matched\_region = main\_image[top\_left[1]:bottom\_right[1], top\_left[0]:bottom\_right[0]]

# Step 6: Resize the matched region using VPI

output\_width, output\_height = 1920, 1080

with vpi.Backend.CUDA:

vpi\_matched\_region = vpi.asimage(matched\_region)

resized\_matched\_region\_vpi = vpi.Image((output\_width, output\_height), vpi.Format.RGB8)

with vpi.Stream() as stream:

vpi\_matched\_region.rescale(resized\_matched\_region\_vpi, stream=stream, interpolation=vpi.Interpolation.LINEAR)

stream.sync()

resized\_matched\_region = resized\_matched\_region\_vpi.cpu() # Download resized image to CPU memory

# Save the resized matched region to a file

cv2.imwrite('/content/drive/MyDrive/cropped\_output/matched\_region.png', resized\_matched\_region)

# Print "True"

print("True")

# Display the resized cropped image

plt.imshow(cv2.cvtColor(resized\_matched\_region, cv2.COLOR\_BGR2RGB))

plt.title('Matched Region')

plt.axis('off')

plt.show()

Trial code –

import vpi  
import numpy as np  
from PIL import Image  
  
kernel = [[1,0,-1],  
                [0,0,0],  
                [-1,0,1]]  
  
input = vpi.asimage(np.asarray(Image.open('Input .png)))  
  
with vpi.Backend.CUDA:  
              output = input.convert(vpi.Format.U8)\.convolution(kernel)  
Image.fromarray(output.cpu()).save('output.png')  
what this code performs