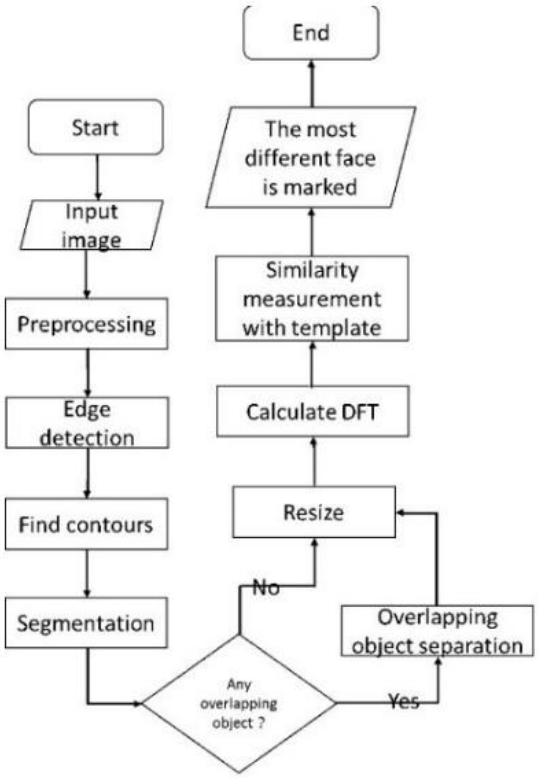
Object Detection and Counting in Image based on Template Object

Abhas Porov (2021EE10781)

Devanshu Ataria (2021EE10162)

Objective of the Project and Steps to be Followed:

In this study, we aim to detect and count similar objects in an image given a template object. Object images are obtained from conditioned environment with fully controlled camera position and lighting. This object image is used for creating template. Gaussian Filter, Pyramid Down and Pyramid Up filters are used for smoothing obtained image. Next the input image will be pre-processed, such as gray-scaling and blurring. After that, edges are detected using Canny algorithm or Binary Threshold and contours of the image are traced. From found contours, sorting and segmentation will be done. Template Matching is then used for identifying objects. Template image is rotated by certain intervals for detecting same object in any orientation. Finally, discrete Fourier transform is calculated and similarity measurement with the template can be done using different calculations such as Euclidian distance, cosine similarity, correlation method or Manhattan distance. Cross correlation is evaluated between template and image for detecting object. Found objects are indicated with a box and then counted.



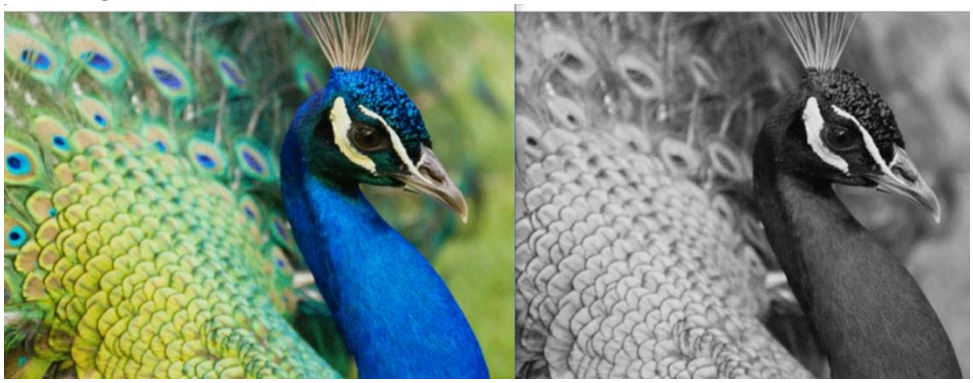
Flowchart of The Project

Uses of the Project:

Object detection is widely used in areas such as surveillance, vehicle tracking, robotics, medical imaging, and manufacturing.

Explanation of the various algorithms used:

1) **Smoothening of image:** The Gaussian smoothing operator is a 2-D convolution operator that is used to 'blur' images and remove detail and noise from it. Each pixel contains a local average corresponding to the neighbourhood pixels present in the region.
2) **Gray-Scaling:** It is the process of converting an image from other colour spaces e.g., RGB, CMYK, HSV, etc. to shades of gray. **T**he value of each pixel represents only the intensity information of the light present in the original image.



3) **Blurring:** Blurring an image makes the edges less sharp so as to remove unnecessary information and average out rapid changes in pixel intensity.
4) **Contour Detection:** Binary Threshold technique creates a binary image from a gray image based on the threshold values and then the edges are detected using Canny Algorithm.
5) **Sorting and Segmentation:** Image segmentation is a commonly used technique in digital image processing and analysis to partition an image into multiple parts or regions, often based on the characteristics of the pixels in the image. It helps in reducing the complexity of the image to make further processing or analysis of the image simpler.
6) **Template Matching:** It is a technique for finding a template image in the source image. The algorithm works by comparing the template for each part of the source image by sliding it one pixel at a time. This process is known as cross-correlation**.**
7) **Counting:** Based on the similarity values at each pixel we can count the number of times the similarity exceeded a previously set value. This will tell us the number of times the template object appears in the image.

Code (in python):

Used open-cv python library to achieve the task:

1) Using Cross-Correlation:

```
import cv2
import numpy
from matplotlib import pyplot

img_rgb = cv2.imread('<Source Image>') # Reading Source image
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY) # Changing image to
grayscale
template = cv2.imread('<Template Image>',0) # Reading Template Image in
grayScale
height, width = template.shape[::] # Finding height and width of template
image

cross_corr_check = cv2.matchTemplate(img_gray, template, cv2.TM_CCOEFF_NORMED)
# Using cross-correlation to find difference between the two images

threshold = 0.95 # Pick only values above 0.95. For Cross-Correlation-
Normalized Larger values = good fit ( 1 being the best )

location = numpy.where( cross_corr_check >= threshold)
# Outputs 2 arrays. Combine these arrays to get x,y coordinates - take x from
one array and y from the other.

# ZIP function is an iterator of tuples where first item in each iterator is
paired together, then the second item and then third, etc.

count=0 # Counter to count number of times the fit is approximately matched
for point in zip(*location[::-1]): #-1 to swap the values as we assign x and y
coordinate to draw the rectangle.
    # Draw rectangle around each object. We know the top left (point), draw
rectangle to match the size of the template image.
    cv2.rectangle(img_rgb, point, (point[0] + width, point[1] + height), (0,
0, 255), 2)  # Red rectangles with thickness 4.
    count+=1

cv2.putText(img_rgb, "Count="+str(count), org=(10, 30),
fontFace=cv2.FONT_HERSHEY_TRIPLEX, fontScale=1, color=(255, 0, 0),
thickness=2)
# Putting no. of times the template appears on image on the image

cv2.imshow("Matched image", img_rgb)
cv2.waitKey()
cv2.destroyAllWindows()
```

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient

x_i = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

y_i = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable

2) Using Square-Difference Method:

```
import cv2
import numpy
from matplotlib import pyplot

img_rgb = cv2.imread('<Source Image>') # Reading Source image
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY) # Changing image to
grayscale
template = cv2.imread('<Template Image>',0) # Reading Template Image in
grayScale
height, width = template.shape[::] # Finding height and width of template
image

sq_diff_check = cv2.matchTemplate(img_gray, template, cv2.TM_SQDIFF)# Using
square-difference to find difference between the two images

upper_bnd= 10**5 # Pick only values below 10^5. For Square-Difference-
Normalized Smaller values = good fit ( 0 being the best )

location = numpy.where(sq_diff_check <= upper_bnd)
# Outputs 2 arrays. Combine these arrays to get x,y coordinates - take x from
one array and y from the other.

# ZIP function is an iterator of tuples where first item in each iterator is
paired together, then the second item and then third, etc.

count=0 # Counter to count number of times the fit is approximately matched
for point in zip(*location[::-1]): #-1 to swap the values as we assign x and y
coordinate to draw the rectangle.
    # Draw rectangle around each object. We know the top left (point), draw
rectangle to match the size of the template image.
    cv2.rectangle(img_rgb, point, (point[0] + width, point[1] + height), (0,
0, 255), 2)  # Red rectangles with thickness 4.
    count+=1

cv2.putText(img_rgb, "Count="+str(count), org=(10, 30),
fontFace=cv2.FONT_HERSHEY_TRIPLEX, fontScale=1, color=(255, 0, 0),
thickness=2)
# Putting no. of times the template appears on image on the image

cv2.imshow("Matched image", img_rgb)
cv2.waitKey()
cv2.destroyAllWindows()
```