Image template matching

Project Report

EC 452: Image Processing

Submitted to

Prof. Robinson Paul

By

Kaiwalya Patil

Enrollment No: 160080111049

BIRLA VISHVAKARMA MAHAVIDYALAYA

Contents

ABSTRACT

Image processing methods and techniques for Template Matching are developed. Matching two photos in order to compare a certain portion of one image into other image and checking if the part is present of not can be used in various systems, making computer vision systems more enhanced and well equipped. Since the template we want to match, may not always be of the same dimensions when compared to the image which we are matching on, we loop this process by scaling down the dimensions of the image on which the template is matched upon. With increasing number of loops and scaling down of the dimensions, we get different correlations values between template and image. The one that is the most is chosen, and a rectangle is drawn using the coordinates corresponding to the highest correlation value. This way, we can find even the smallest part in any image. The same matching of templates and image can be used in various systems to match and check if the image is as per the desired output.

INTRODUCTION

This project is based on application of Image processing, Template Matching. This is used in finding a specific template which is passed as an input, this template is search in a different image. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. The methods for this include using a function from Open-CV Python, called matchTemplate(). There are various methods as argument in matchTemplate(), after trying all of the methods, we come to a conclusion “cv2.TM_CCOEFF” works better than any of other 8 methods. We use Canny edge detection methods for edge detection of image. The results of this project can be used in various applications, some of them could be: capturing location of people committing crime, in a image matching database, to track a specific object/person in real time in sports.

OBJECTIVE

With progressing technology, the algorithms are becoming more and more computation hungry. Template matching plays a very important role, when it comes to applications like image matching in a large database or constantly monitoring surveillance cameras in search of a specific person. So, this project aims to solve this by a basic pixel value matching and correlation values. Some other computation hungry algorithms include extraction of facial features, or training a generalized model for further classification. Some of these may even be needed, at times when the detection is complex. But most of the times, a simple matching solves the task. It becomes more powerful when it is combined with loops and scaling down of dimensions of image. Hence, our objective is to match the template with accurate part of image in best possible way, using very less computational power.

STRATEGY AND TECHNOLOGY

The project used various libraries which are available in Python 3.x.x. The libraries that are used are as follows:

1. NumPy:

Also known as Numerical Python, this is mainly used to arrays and numerical computation. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. It is very well optimized. We use this library in this project since, an image, on a deeper level is a matrix. So, to play with the values on deeper level, we need this.

2. Open-CV Python:

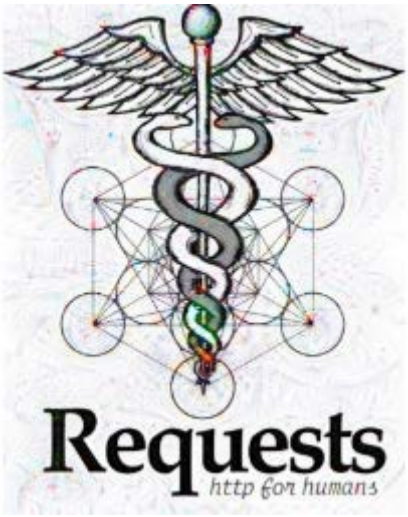
A very popular library across all the coding languages available. It's base is written in C++ and is very well optimized. This library is mostly used while working with images, videos that aim towards making new changes in computer vision. We use this library in various parts of the project, each part will be demonstrated with code in further sections. The main uses are in reading an image, using edge detection, using template matching, scaling down the size of image.

3. Imutils:

(imutils doesn't have any logo as of now, this is just an application of the library)

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and both Python 2.7 and Python 3. We use this to resize the image according to the scale, and keep track of the ratio of the resizing.

4. Requests:



Requests allows you to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, thanks to urllib3. We use this to make a call to API which helps us to remove background of the image, this helps in better template matching. Observations are this doesn't always give good results.

PROGRAM CODE AND ANALYSIS

In this section we would analyze main parts of code and understand its significance. Let us look into the working of entire code and then get a deeper insight of each part. The sections in which code can be divided are as follows:

1. Importing of libraries: All the above-mentioned libraries are imported and are frequently used in various parts of code, so that achieving specific tasks becomes easier, more executable and remains optimized.

```
import numpy as np
import imutils
import cv2
import requests
```

2. Getting started with importing and reading image:
   We read the image named “no-bg.png” using the Open-CV python library. We use the function “imread” for the same. Next, we used “cvtColor” function and pass image and BGR2GRAY as arguments to it, this helps in changing the RGB colored image into a gray color image. The we use “Canny” function for edge detection. First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Third argument is aperture_size. The next path is storing the height and width of image in two variables. The final line gives the path to the main image on which the template is to be matched.

```
template = cv2.imread('no-bg.png')
template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
template = cv2.Canny(template, 50, 200)
(tH, tW) = template.shape[:2]
cv2.imshow("Template", template)
pt = 'image2.png'
```

3. Starting to loop and try to match template:
   We use the variable “imagePath” as out iterable variable inside for loop. We loop this into the image path that we have provided. The same process of reading the image and converting it to gray scale, as mentioned above, is repeated for this image. This image is the main image on which the template needs to be matched.

```
for imagePath in glob.glob(pt):
    image = cv2.imread(imagePath)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    found = None
```

4. Continuation of loop for resize and matching:
   We resize the image according to the scale, and keep track of the ratio of the resizing. This is done by imutils “resize” function. If the resized image is smaller than the template, then break from the loop and detect edges in the resized image by Canny method, and apply template matching to find the template in the image. Check to see if the iteration should be visualized, if we have found a new maximum correlation value, then update the bookkeeping variable (maxVal and maxLoc are stored in variable named “found”). If not, we conclude that the value, we got has maximum correlation.

```
for scale in np.linspace(0.2, 1.0, 20)[::-1]:
        resized = imutils.resize(gray, width = int(gray.shape[1] * scale))
        r = gray.shape[1] / float(resized.shape[1])
        if resized.shape[0] < tH or resized.shape[1] < tW:
            break
        edged = cv2.Canny(resized, 50, 200)
        result = cv2.matchTemplate(edged, template, cv2.TM_CCOEFF)
        (_, maxVal, _, maxLoc) = cv2.minMaxLoc(result)
        if found is None or maxVal > found[0]:
            found = (maxVal, maxLoc, r)
```

5. Getting out loop, storing the results:
   Now, since we have the stating and ending values of the pixels, that has matched into our image, we draw a red colored rectangle on the detected area. This drawing of rectangle is again done by cv2 function. This takes the starting, ending pixel values and the boldness of the bounding box as the arguments. Finally, after applying the bounding box, we use imwrite function of cv2, which is used to store the results into our working directory.

```
    (_, maxLoc, r) = found
    (startX, startY) = (int(maxLoc[0] * r), int(maxLoc[1] * r))
    (endX, endY) = (int((maxLoc[0] + tW) * r), int((maxLoc[1] + tH) * r))
    cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 10)
    cv2.imwrite("Imagexx.png", image)
```

6. Removing background using API call:
   This is optional background removal part of code. Some images may not be directly matched to the template. So, those images need to go through background removal process. We pass the image into the API and get an output which has no background. This is completely optional and this is done just after importing libraries.

```
response = requests.post(
    'https://api.remove.bg/v1.0/removebg',
    files={'image_file': open('template4.png', 'rb')},
    data={'size': 'auto'},
    headers={'X-Api-Key': '5JFrGRc7B9ktsMFwEnBp6Pmg'},
)
if response.status_code == requests.codes.ok:
    with open('no-bg.png', 'wb') as out:
        out.write(response.content)
else:
    print("Error:", response.status_code, response.text)
```

SIMULATION RESULTS

Image 1:

Original Image

Template to be matched

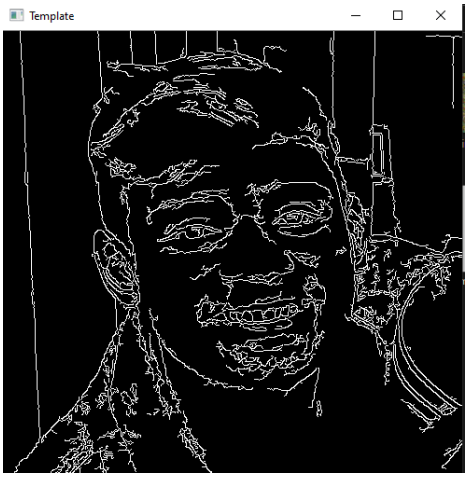
Canny Detection

Resulting image with template matched

Image 2:

Original image

Template to be matched



Canny edge detection

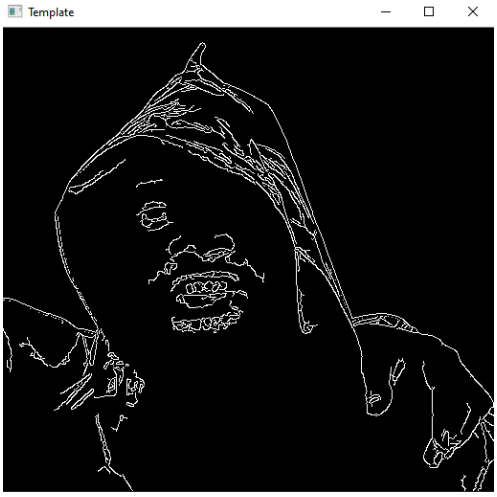
Resulting image with template matched

Image 3(With background removal):

Original image

Main template

Template to be matched (Background removed)



Canny edge detection

Final results

Future Work

The future work that could be added to this for better results could be:

1. The model could be made smart by naming the template. So, if the various templates are named, say A, B, C…, when the template named A is matched, we get a count of it. This helps in tracking an object or person in a scenario.
2. The API dependency could be removed, but coding in such a way that the background removal process is done via open-cv python.
3. All of this can be shifted to a webcam, so constantly monitoring is possible.
4. A webcam feed can be converted into multiple number of images, and then those individual images can be deblurred by using Weiner filters. And then this kind of template matching can be applied.
5. Before matching template, we can add/remove facial features like mustache, beard etc. So this becomes helpful in identification of a criminal who is roaming openly in cities.

CONCLUSION

Hence, by using Template Matching, an application of image processing, we can match the template to any image and get our desired output. This can be used in various domains from making and image database to counting of an object. The combination of Canny edge detection method with template matching proves to be really helpful in this. And on top of this, the resizing of the main image to match template solves any issues of dimension mismatch.

REFERENCES

- Canny edge detection: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html
- Canny edge detection: https://www.geeksforgeeks.org/real-time-edge-detection-using-opencv-python/
- Imutils: https://github.com/jrosebr1/imutils
- Open-CV python: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
- Weiner filter MATLAB: https://in.mathworks.com/help/images/deblurring-images-using-a-wiener-filter.html
- Background image removal: https://flothesof.github.io/removing-background-scikit-image.html