SZKOŁA GŁÓWNA HANDLOWA W WARSZAWIE
WARSAW SCHOOL OF ECONOMICS

Student's Name and Surname: **Ngoc Bao Dao**

Student No: **64588**

Specialization: **Big Data – Advanced Analytics**

Python Project

Shape Detection and Analysis using OpenCV

Subject: **Python Programming**

Lecturer: **Dr Paweł Stefan Rubach**

I. Project outline

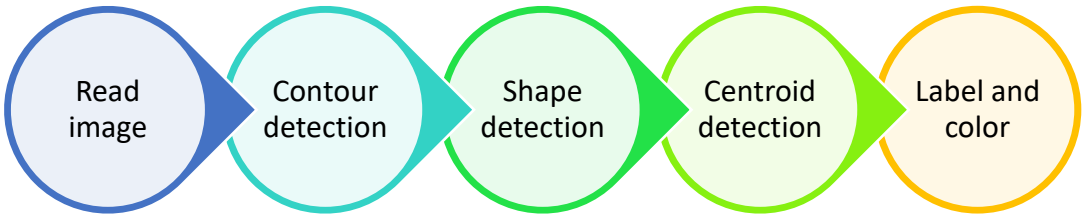
This project is about to create an algorithm to perform a basic Computer Vision task with Python and OpenCV - an Open Computer Vision Library. The project covers some basic functionalities of image processing, including:

1. Compute the center of a contour/shape region.
2. Recognize various shapes, such as circles, squares, rectangles, triangles, and stars using only contour properties.
3. Colour and label the shapes.

The code was being run on Google Colaboratory Notebook, which was built on top of Jupyter Notebook. I chose Google Colab because it is a free cloud service and has been **supported** free **GPU**. Therefore, it is more efficient to practice Python programming language coding skills and develop deep learning algorithms using popular libraries such as Keras, TensorFlow, PyTorch, and **OpenCV**.

II. Methodology

To accomplish this, I leveraged contour approximation, the process of reducing the number of points on a curve to a simpler approximated version.
Then, based on this contour approximation, I examined the number of vertices each shape has. Given the vertex count, it was able to accurately label each of the shapes. Then colored each shape in the image, followed by computing the center of the contour — also called the centroid of the region thanks to image moments.



Flowchart of shape detection

III. Input image

The sample image I used for this project is an image in .jpg and it was cut out from a Word file with a variety of shapes in black colour on the white background as below.

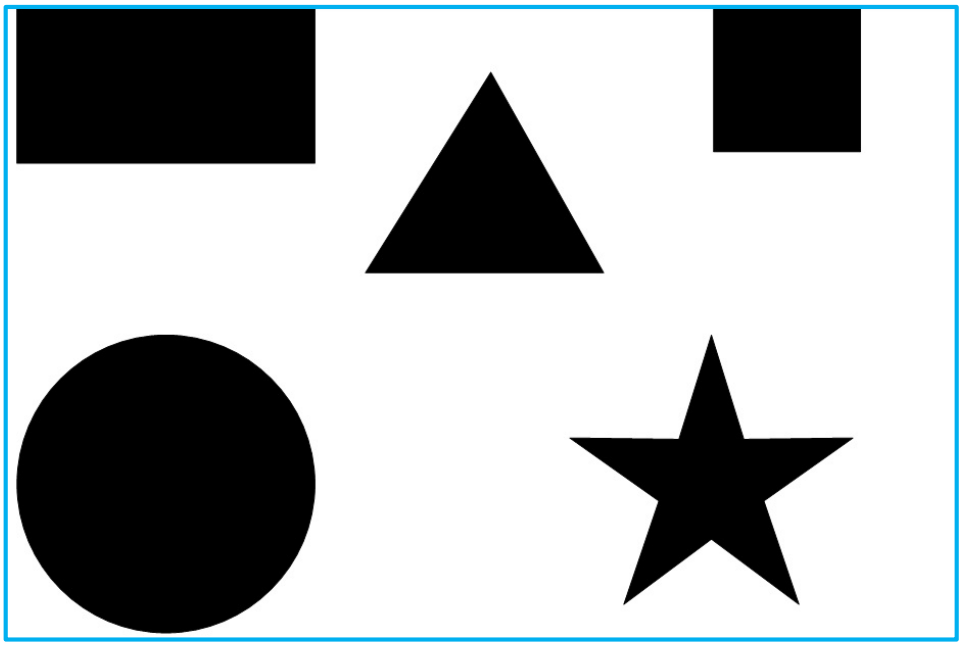


Image used for this project

IV. Steps involved

1) Install packages

To implement this project, the following packages of Python 3 have to be downloaded and installed: *NumPy* and *cv2*. Exceptionally, as this project was run on Google Colab, cv2_imshow - a function to show image in OpenCV has to be installed as well.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
```

2) Read an image

First, a sample image in which processing is to be applied is to be read. It is done using a pre-defined Python function: CV2.imread(). Then I will need to perform a bit of image pre-processing, including:

- Conversion to grayscale.
- Binarization of the image. Typically edge detection and thresholding are used for this process. In this project, we'll be applying thresholding.

```
# Load and then gray scale image

image = cv2.imread('someshapes.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2_imshow(image)
cv2.waitKey(0)

# Threshold
ret, thresh = cv2.threshold(gray, 127, 255, 1)
```

Now I am ready to leverage contour properties to actually label and identify shapes in the image.

3) Contour extraction

The next step is to find the location of these shapes in the threshold image using contour detection.

A call to cv2.findContours returns the set of outlines (i.e., contours) that correspond to each of the black blobs on the image. There are three arguments in cv.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy. Contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

Retrieval types:

Cv2.RETR_LIST – Retrieves all contours.
Cv2.RETR_EXTERNAL – Retrieves external or outer contours only.
Cv2.RETR_COMP - Retrieves all in a 2-level hierarchy.
Cv2.RETR_TREE – Retrieves all in full hierarchy.

```
# Extract Contours
_, contours, hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
```

Since I chose retrieval mode as ***cv2.RETR_LIST***, it will retrieve all contours in the image.
I am now ready to process each of the contours.

4) Contour approximation

I started looping over each of the individual contours, followed by getting approximate polygons.

```
# Loop over the contours
for cnt in contours:

    # Get approximate polygons
    approx = cv2.approxPolyDP(cnt, 0.01*cv2.arcLength(cnt,True),True)
```

In order to perform shape detection, I will be using contour approximation.

As the name suggests, contour approximation is an algorithm for reducing the number of points in a curve with a reduced set of points — thus the term approximation. This algorithm is commonly known as the Ramer-Douglas-Peucker algorithm, or simply the split-and-merge algorithm.

Contour approximation is predicated on the assumption that a curve can be approximated by a series of short line segments. This leads to a resulting approximated curve that consists of a subset of points that were defined by the original curve. Contour approximation is actually already implemented in OpenCV via the cv2.approxPolyDP method.

Cv2.approxPolyDP(Contour, Approximation Accuracy, Closed)

- Contour – is the individual contour we wish to approximate.
- Approximation Accuracy - Important parameter is determining the accuracy of the approximation. Small values give precise- approximations, large values give more generic approximation. A good rule of thumb is less than 5% of the contour perimeter. In our case, I chose 1% to get the more precise approximation.
- Closed: a Boolean value that states whether the approximate contour should be open or closed.

5) Shapes detection:

Given our approximated contour, I can move on to performing shape detection.

A contour consists of a list of vertices so I can check the number of entries in this list to determine the shape of an object.

For example, if the approximated contour has three vertices, then it must be a triangle.

```
    if len(approx) == 3:
        shape_name = "Triangle"
        cv2.drawContours(image,[cnt],0,(0,255,0),-1)

        # Find contour center to place text at the center
        M = cv2.moments(cnt)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])
        cv2.putText(image, shape_name, (cx-50, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)
```

If a contour has four vertices, then it must be either a square or a rectangle.

To determine which, I compute the aspect difference of the shape, which is simply the width of the contour bounding box minus the height. If the aspect difference is less than or equal 3, then I am examining a square (since all sides have approximately equal length). Otherwise, the shape is a rectangle.

```
elif len(approx) == 4:
    x,y,w,h = cv2.boundingRect(cnt)
    M = cv2.moments(cnt)
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])

    # Check to see if 4-side polygon is square or rectangle
    # cv2.boundingRect returns the top left and then width and
    if abs(w-h) <= 3:
        shape_name = "Square"

        # Find contour center to place text at the center
        cv2.drawContours(image, [cnt], 0, (0, 125 ,255), -1)
        cv2.putText(image, shape_name, (cx-50, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)
    else:
        shape_name = "Rectangle"

        # Find contour center to place text at the center
        cv2.drawContours(image, [cnt], 0, (0, 0, 255), -1)
        M = cv2.moments(cnt)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])
        cv2.putText(image, shape_name, (cx-50, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)
```

If a contour has ten vertices, I can label it as a star and if the contour has more than fifteen vertices, I can make the assumption that the shape I am examining is a circle.

```
    elif len(approx) == 10:
        shape_name = "Star"
        cv2.drawContours(image, [cnt], 0, (255, 255, 0), -1)
        M = cv2.moments(cnt)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])
        cv2.putText(image, shape_name, (cx-50, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)



    elif len(approx) >= 15:
        shape_name = "Circle"
        cv2.drawContours(image, [cnt], 0, (0, 255, 255), -1)
        M = cv2.moments(cnt)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])
        cv2.putText(image, shape_name, (cx-50, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)

    cv2_imshow(image)
    cv2.waitKey(0)

cv2.destroyAllWindows()
```

6) Centroid detection, label and colour the shape

The last step is to draw contours with colours over the shapes, compute the center of the contour and then put the label in the center of each shape.

In computer vision and image processing, image moments are often used to characterize the shape of an object in an image. These moments capture basic statistical properties of the shape, including the area of the object, the centroid (i.e., the center (x, y) - coordinates of the object), orientation, along with other desirable properties.

```
  cv2.drawContours(image,[cnt],0,(0,255,0),-1)

  # Find contour center to place text at the center
  M = cv2.moments(cnt)
  cx = int(M['m10'] / M['m00'])
  cy = int(M['m01'] / M['m00'])
  cv2.putText(image, shape_name, (cx-50, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)
```

Here I am only interested in the center of the contour, which I computed using cv2.moments with x position as cx -50 and y position as cy. The reason I chose x position as cx -50 because it starts from the bottom right corner of the shape, so it is nicer to pull back the text a little more so the whole text will be inside.

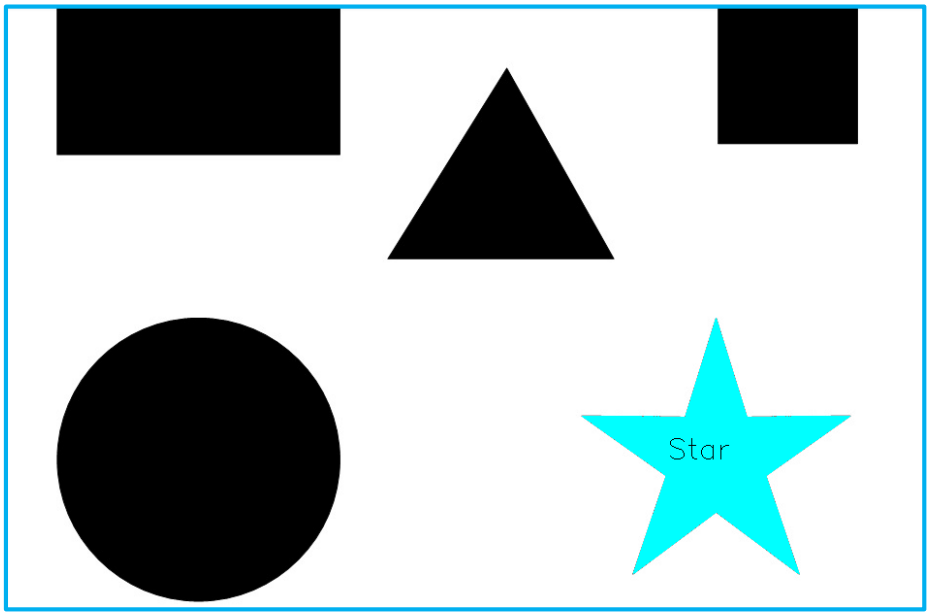
From there, I handle:

- Drawing the contour over the current shape with specific colour by making a call to cv2.drawContours.
- Placing the label of shape at the center (cX -50, cY) - coordinates of the shape.

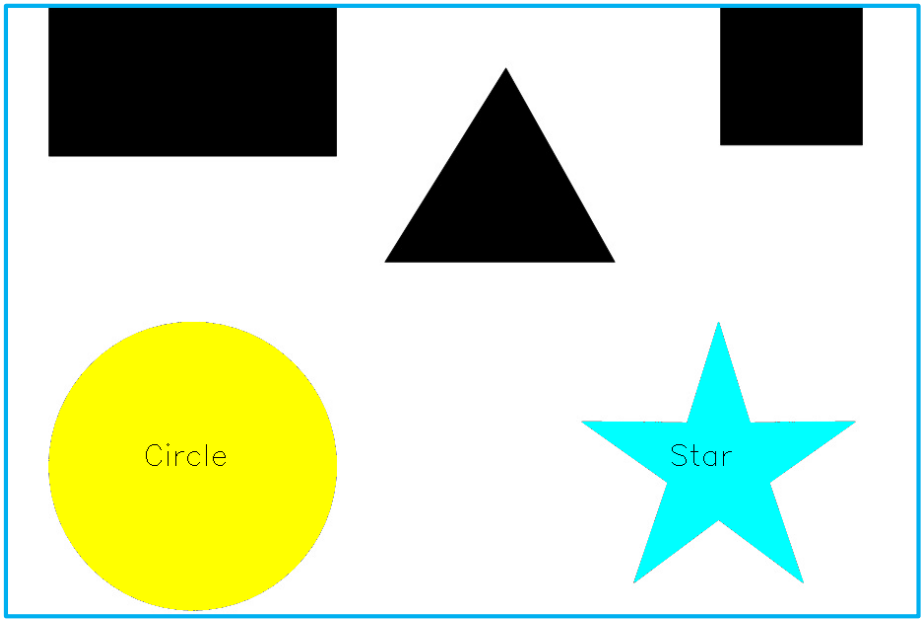
7) Result display:

Finally, I display the results on the screen.

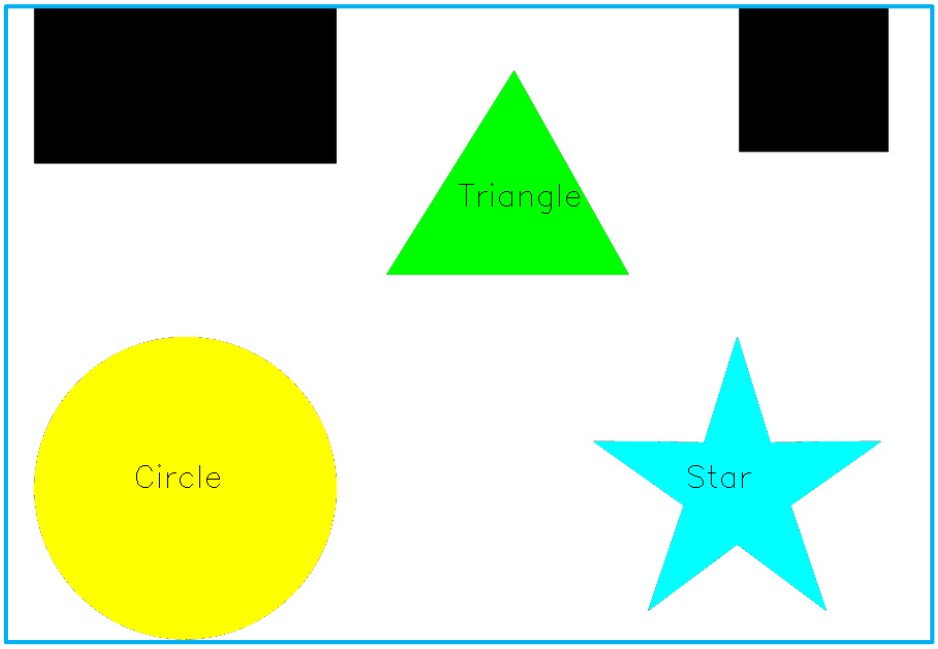
The Star was detected first and coloured with the light blue.



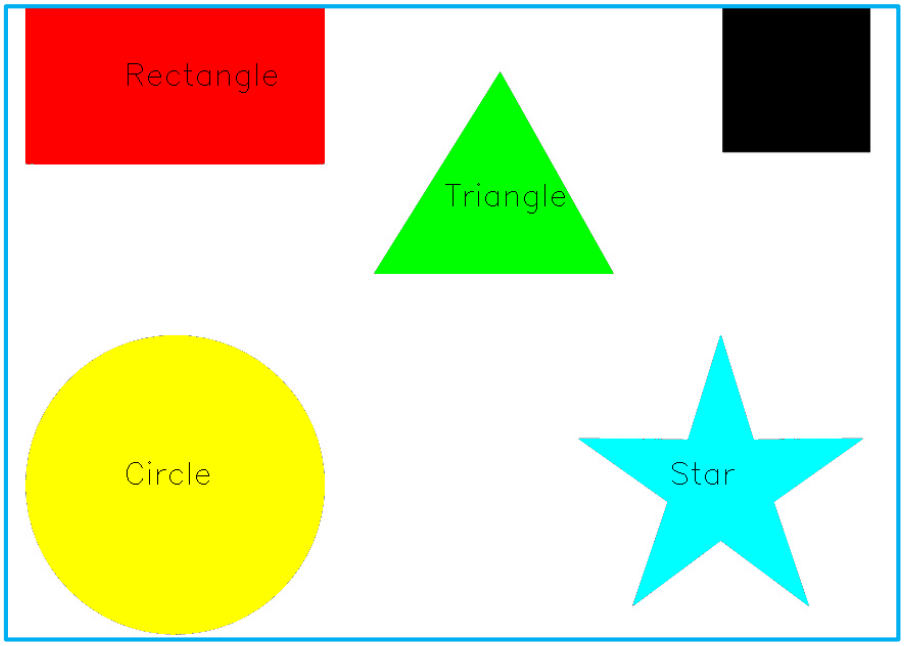
Next was the circle with yellow.



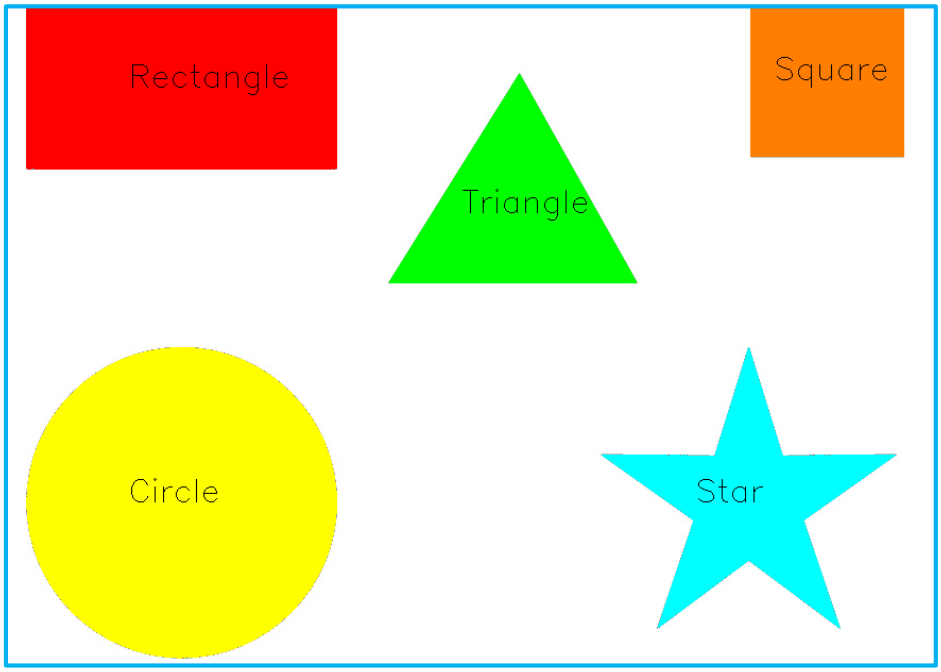
Triangle was processed thirdly with green colour.



Followed by Triangle with Red colour.



And finally, the square was also recognized with orange colour.



As seen from the above, script loops over each of the shapes individually, performs shape detection on each one, and then draws the name of the shape on the object.

V. Summary

Computer vision can be used to solve the most intriguing problems with utmost sophistication. All the basics regarding the detection technique along with the way to achieve it have been profoundly discussed.

In this project, I have successfully performed shape detection in the given sample image with OpenCV and Python. The project was done thanks to the contour approximation and image moments method.

During the course of programming, I used Python on Google Colaboratory for Computer Vision, I do prefer Python on Google Colab because it takes less simulation time and I find it easy to implement.