CV ASSIGNMENT 1

APPROACH for Minimum Enclosing Circle:

- Input: image containing only single white object, and black background
- For the purpose of finding count and locating the objects, the code from HW1 was used.

```
# CODE FROM HW1 TO FIND AND LABEL THE OBJECTS
def code_from_hw1(image):
    """
    Input:
        image: Binary color 2d image matrix.

    Output:
        count: the number of objects detected.
        obj_id: a 2d map indicating which pixel belongs to which object.
            0 indicated, its a black pixel
            i in {1,2,3,...,count} denotes that the pixel belongs to object i.
    """
```

- Firstly, just for the sake of fast convergence, the center is initialized to the center of the **bounding box** found using ((minx+maxx)/2, (miny+maxy)/2). Furthermore, the bounding box was found using the following function.

```
def min_bounding_box(image):
    """
    Input:
        image: Binary color 2d image matrix. Should consist
        of only one object comprising of white pixels.

    Output:
        min_x, max_x, min_y, max_y:
            the 4 required parameters for unique bounding box.
    """
    m,n = image.shape

    min_x,min_y = image.shape
    max_x,max_y = 0,0

    for x in range(m):
        for y in range(n):
            if image[x,y]==1:
                min_x = min(x, min_x)
                min_y = min(y, min_y)
                max_x = max(x, max_x)
                max_y = max(y, max_y)

    return min_x, max_x, min_y, max_y
```

Dushyant Panchal, 2018033

CV ASSIGNMENT 1

- For a given center, the **radius** is calculated by finding the furthest white pixel belonging to the object followed by its distance from the center. The code has been optimized by only considering the points within the bounding box for finding the max distance or the radius.

```python
def calc_radius(image, min_x, max_x, min_y, max_y, x, y):
    """
    Input:
        image: Binary color 2d image matrix.
        min_x, max_x, min_y, max_y: from the bounding box (for computation optimization)
        x, y: current center of the circle for which radius needs to be calculated

    Output:
        r:  the distance of the given center from the furthest
            white pixel within the specified bounding box.
    """
    r = 0
    for i in range(min_x, max_x+1):
        for j in range(min_y, max_y+1):
            if image[i,j]==1:
                dist = math.sqrt( (i-x)**2 + (j-y)**2 )
                r = max(r, dist)
    return r
```

- We use the BFS algorithm to traverse through the neighboring pixels of this center, to search for a smaller possible circle.
- We initialize a queue with ((minx+maxx)/2, (miny+maxy)/2, radius).
- For visited markings, we just use a copy of the image, and mark the visited white pixels as black.
- Consider the pseudocode below for the traversal.

- `While (queue is not empty):`
    - `x,y,r = pop an element from the queue`
    - `If unvisited white neighbor pixel (x1,y1) exist at top/bottom/left/right:`
        - `Set that pixel (x1,y1) as visited`
        - `Calculate the radius r1 considering x1,y1 as the center`
        - `If r1<r:`
            - `Add this pixel for review to the queue (x1,y1,r1)`

- This way, we traverse over possible small circles, and find the one having minimum radius, since a circle having minimum area would be the circle having minimum radius.

- Moreover, the jaccard similarity (Q2) is computed using the following function.

CV ASSIGNMENT 1

```python
def jaccard_score(mask1, mask2):
    """
    Input:
        mask1 and mask2 are the two binary
        image masks consisting of white
        and black pixels only.

    Output:
        The computed jaccard similarity
        between the two masks passed as
        the input.
    """
    m,n = mask1.shape

    intersect = 0
    union = 0

    for i in range(m):
        for j in range(n):
            if mask1[i,j]==1 and mask2[i,j]==1:
                intersect+=1
            if mask1[i,j]==1 or mask2[i,j]==1:
                union+=1

    return intersect/union
```

<u>CODE</u>: **(To be run as python <code_file_path> <image_path>)**

- **Library Imports**

```python
import cv2
import numpy as np
import math
import sys
from collections import deque
```

- **Module Functions**

```python
# CODE FROM HW1 TO FIND AND LABEL THE OBJECTS
def code_from_hw1(image):
    """
    Input:
        image: Binary color 2d image matrix.
```

```
    Output:
        count: the number of objects detected.
        obj_id: a 2d map indicating which pixel belongs to which object.
            0 indicated, its a black pixel
                i in {1,2,3,...,count} denotes that the pixel belongs to
object i.
    """

    # accumulator to count objects
    count = 0

    # dimensions of the image
    m,n = image.shape

    # object id array
    obj_id = np.zeros((m,n))

    # iterate through the pixels
    for a in range(m):
        for b in range(n):

            # if unvisited white pixel found
            if image[a,b]==1:

                # increment the counter to denote new object
                count+=1

                # find all neighboring white pixels and mark
                # them as visited by setting them to black
                queue = deque(((a,b),))
                image[a,b]=0
                obj_id[a,b]=count

                # similar to BFS technique
                while len(queue)>0:
                    i,j = queue.popleft()

                    if i>0 and image[i-1,j]==1:
                        image[i-1,j]=0
```

```
                    obj_id[i-1,j]=count
                    queue.append((i-1,j))

                if j>0 and image[i,j-1]==1:
                    image[i,j-1]=0
                    obj_id[i,j-1]=count
                    queue.append((i,j-1))

                if i<m-1 and image[i+1,j]==1:
                    image[i+1,j]=0
                    obj_id[i+1,j]=count
                    queue.append((i+1,j))

                if j<n-1 and image[i,j+1]==1:
                    image[i,j+1]=0
                    obj_id[i,j+1]=count
                    queue.append((i,j+1))

    # return the final count and object id matrix
    return count, obj_id


def calc_radius(image, min_x, max_x, min_y, max_y, x, y):
    """
    Input:
        image: Binary color 2d image matrix.
        min_x, max_x, min_y, max_y: from the bounding box (for computation
optimization)
          x, y: current center of the circle for which radius needs to be
calculated

    Output:
        r:  the distance of the given center from the furthest
            white pixel within the specified bounding box.
    """
    r = 0
    for i in range(min_x, max_x+1):
        for j in range(min_y, max_y+1):
            if image[i,j]==1:
                dist = math.sqrt( (i-x)**2 + (j-y)**2 )
```

```
                r = max(r, dist)
    return r


def min_bounding_box(image):
    """
    Input:
        image: Binary color 2d image matrix. Should consist
        of only one object comprising of white pixels.

    Output:
        min_x, max_x, min_y, max_y:
            the 4 required parameters for unique bounding box.
    """
    m,n = image.shape

    min_x,min_y = image.shape
    max_x,max_y = 0,0

    for x in range(m):
        for y in range(n):
            if image[x,y]==1:
                min_x = min(x, min_x)
                min_y = min(y, min_y)
                max_x = max(x, max_x)
                max_y = max(y, max_y)

    return min_x, max_x, min_y, max_y


def min_enclosing_circle(image):
    """
    Input:
        image: Binary color 2d image matrix. Should consist
        of only one object comprising of white pixels.

    Output:
        x,y,r: The center coordinates and the radius for the
            calculated minimum enclosing circle.
    """
```

CV ASSIGNMENT 1

```
    # to ensure original image is not lost
    image = image.copy()

    m,n = image.shape

    # original copy (will use 'image' as visited array for BFS)
    img = image.copy()

    # finding the bounding box
    min_x,max_x,min_y,max_y = min_bounding_box(image)

    x = (min_x+max_x)//2
    y = (min_y+max_y)//2
    r = math.sqrt( (max_x-min_x)**2 + (max_y-min_y)**2 )//2

    queue = deque([(x,y,r),])
    image[x,y]=0

    # similar to BFS technique
    while len(queue)>0:
        i,j,r_cur = queue.popleft()

        if r_cur<r:
            r = r_cur
            x = i
            y = j

        if i>0 and image[i-1,j]==1:
            image[i-1,j]=0
            r_nxt = calc_radius(img, min_x, max_x, min_y, max_y, i-1, j)
            if r_nxt<r_cur:
                queue.append((i-1,j,r_nxt))

        if j>0 and image[i,j-1]==1:
            image[i,j-1]=0
            r_nxt = calc_radius(img, min_x, max_x, min_y, max_y, i, j-1)
            if r_nxt<r_cur:
                queue.append((i,j-1,r_nxt))

        if i<m-1 and image[i+1,j]==1:
```

```
            image[i+1,j]=0
            r_nxt = calc_radius(img, min_x, max_x, min_y, max_y, i+1, j)
            if r_nxt<r_cur:
                queue.append((i+1,j,r_nxt))

        if j<n-1 and image[i,j+1]==1:
            image[i,j+1]=0
            r_nxt = calc_radius(img, min_x, max_x, min_y, max_y, i, j+1)
            if r_nxt<r_cur:
                queue.append((i,j+1,r_nxt))

    return x,y,r


def jaccard_score(mask1, mask2):
    """
    Input:
        mask1 and mask2 are the two binary
        image masks consisting of white
        and black pixels only.

    Output:
        The computed jaccard similarity
        between the two masks passed as
        the input.
    """
    m,n = mask1.shape

    intersect = 0
    union = 0

    for i in range(m):
        for j in range(n):
            if mask1[i,j]==1 and mask2[i,j]==1:
                intersect+=1
            if mask1[i,j]==1 or mask2[i,j]==1:
                union+=1

    return intersect/union
```

CV ASSIGNMENT 1

- **The Main Driving Code**

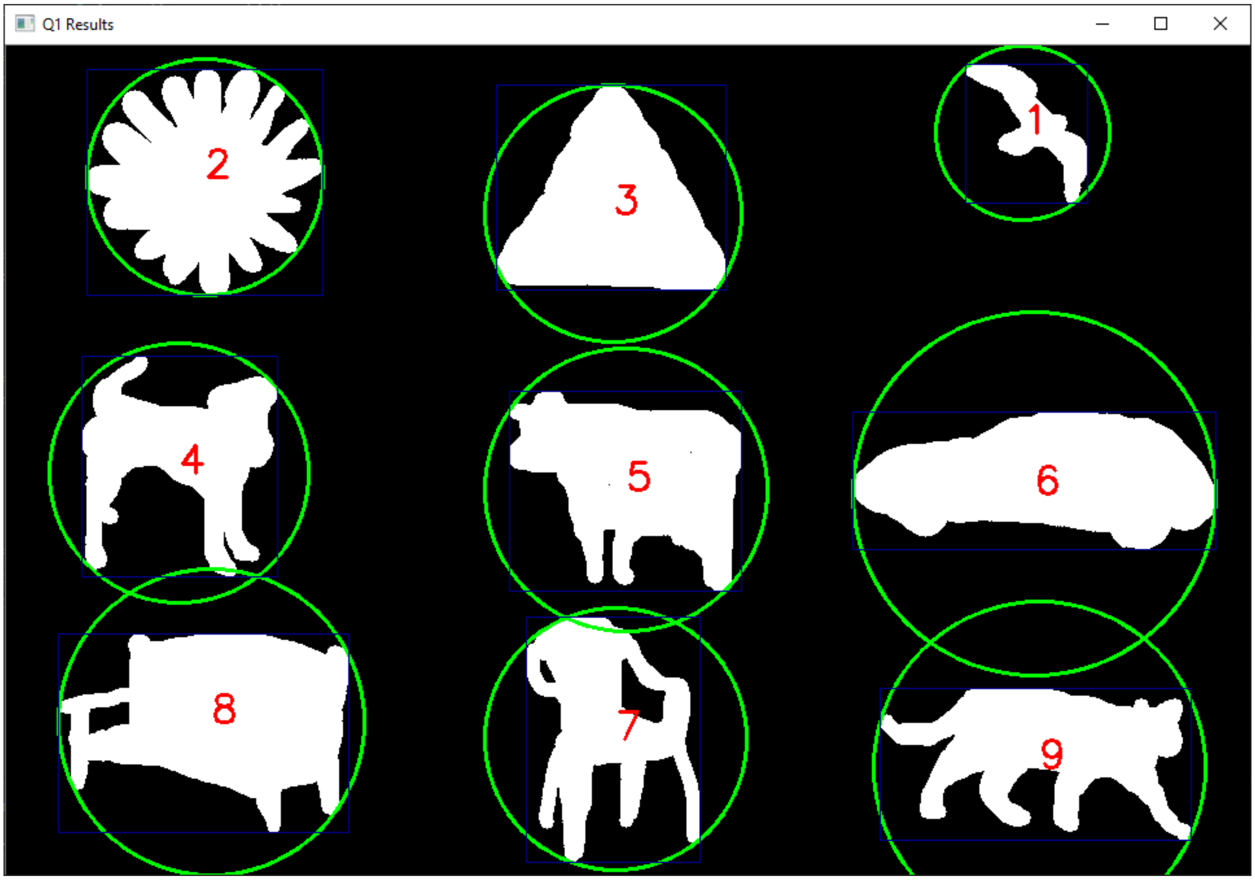
```
if __name__=="__main__":
    # input the image to np array
    image = cv2.imread(sys.argv[1], cv2.IMREAD_GRAYSCALE)

    # threshold to convert to binary image
    image[image<128] = 0
    image[image>=128] = 1

    # get the objects
    count, object_id = code_from_hw1(image.copy())


    boxes = []
    circles = []
    for i in range(1,count+1):
        img = np.zeros(image.shape)

        # image containing only object 'i'
        img[object_id==i] = 1

        # get minimum bounding boxes
        box = min_bounding_box(img)
        boxes.append(box)

        # get minimum enclosing circles
        circle = min_enclosing_circle(img)
        circles.append(circle)

        # get circle mask for jackard similarity
        circle_mask = np.zeros(image.shape)
        min_x, max_x, min_y, max_y = box
        x_c, y_c, r = circle
        for x in range(min_x, max_x+1):
            for y in range(min_y, max_y+1):
                within_circle = ( ((x-x_c)**2 + (y-y_c)**2) <= r*r )
                if within_circle:
                    circle_mask[x,y] = 1

        # calculate jaccard similarity
```

```
        jaccard = jaccard_score(img, circle_mask)

        # print the results
                      print(f"Center:({x_c},{y_c}),   Radius:{r},   Jaccard
Score:{jaccard}")


    # read a color instance of the image
    img = cv2.imread(sys.argv[1])

    # add the enclosing circles to the image
    for i in range(count):
        x,y,r = circles[i]
        x=int(x)
        y=int(y)
        r=int(r)
        img = cv2.circle(img, (y,x), r, (0,255,0), 2)

    # add the enclosing boxes to the image
    for i in range(count):
        x1,x2,y1,y2 = boxes[i]
        img = cv2.rectangle(img, (y1, x1), (y2, x2), (150,0,0), 1)

    # add text to identify object by id
    for i in range(count):
        x,y,r = circles[i]
        x=int(x)
        y=int(y)
        img = cv2.putText(img, str(1+i), (y, x), cv2.FONT_HERSHEY_SIMPLEX,
1, (0,0,255), 2)

    # display the results
    cv2.imshow('Q1 Results', img)
    cv2.waitKey(0)
```

RESULTS:

CV ASSIGNMENT 1

Q1: Enclosing Circles Visualized



Centers and Radii of Enclosing Circles and Q2:Jaccard Similarities

1. Center:(67,784), Radius:67.00746227100382, Jaccard Score:0.3383947939262473
2. Center:(101,153), Radius:91.7877987534291, Jaccard Score:0.7409113593612039
3. Center:(129,468), Radius:99.84988733093293, Jaccard Score:0.6591141129348936
4. Center:(329,133), Radius:100.4987562112089, Jaccard Score:0.5165750258984779
5. Center:(342,478), Radius:109.04127658827184, Jaccard Score:0.6300254452926208

Dushyant Panchal, 2018033

6. Center:(345,793), Radius:140.1748907615055, Jaccard Score:0.6999006066422182

7. Center:(534,470), Radius:101.04454463255303, Jaccard Score:0.5296036535638559

8. Center:(521,158), Radius:118.6086000254619, Jaccard Score:0.647756371174247

9. Center:(556,797), Radius:128.16005617976296, Jaccard Score:0.5391245743473326