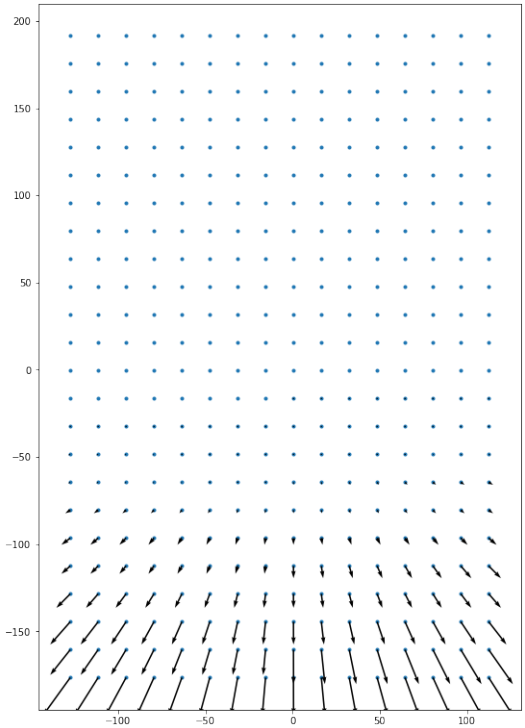
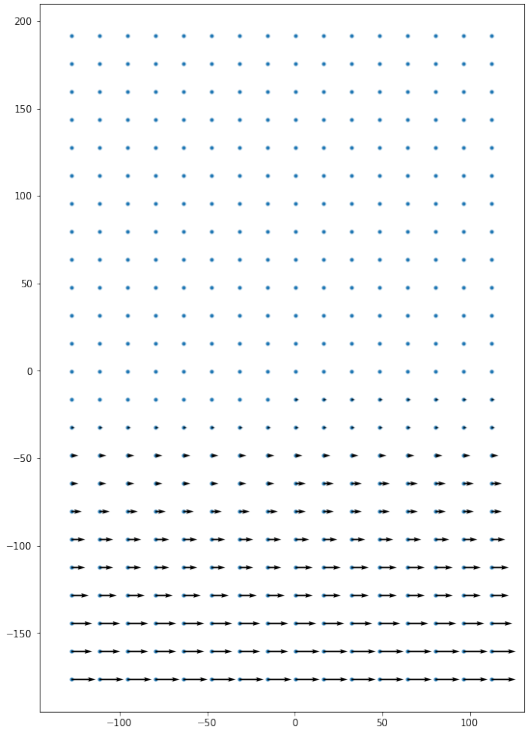
Q1 (b)-

This question requires us to plot the optical flow of the given conditions-

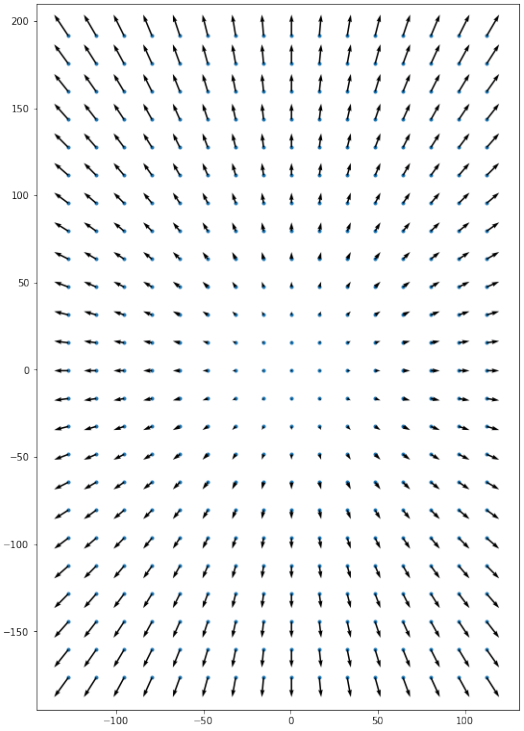
i- Looking forward on a horizontal plane while driving on a flat road.



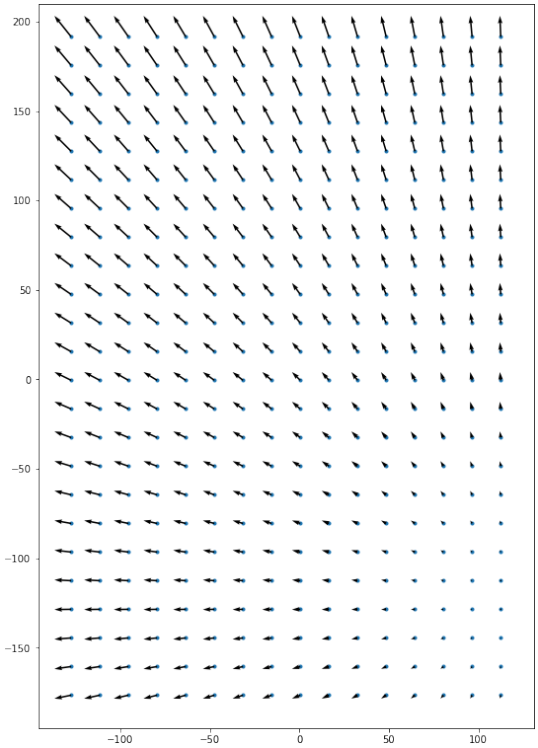
ii- Sitting in a train and looking out over a flat field from a side window.



iii- Flying into a wall head-on.



iv- Flying into a wall but also translating horizontally, and vertically.



v- Counter-clockwise rotating in front of a wall about the Y-axis.

The Output should look like this but **I could not produce the code for ONLY PART V.**

The Code for the others are there in the file on Compass2G.

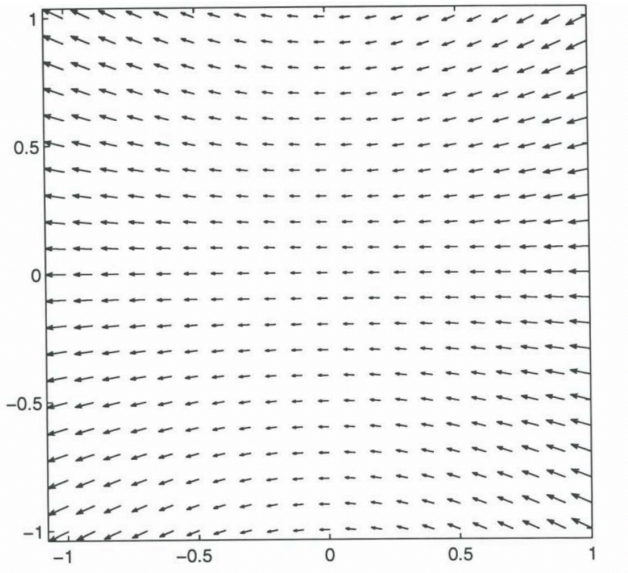


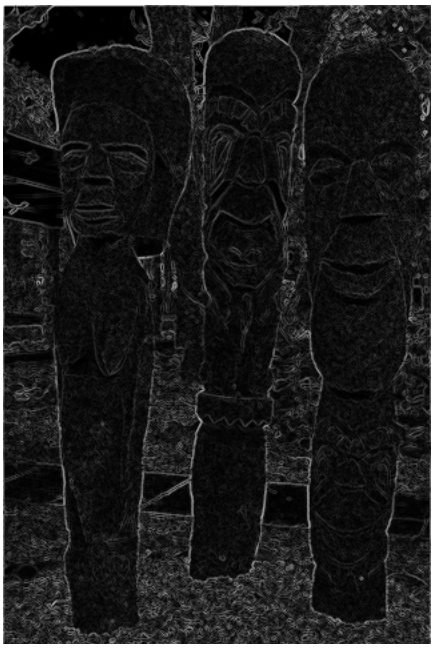
Figure This is how the output should look like for part V.

Q2

Output images which were given as a part of skeleton code-

Sample output 1

Sample Output 2



Sample Output 3

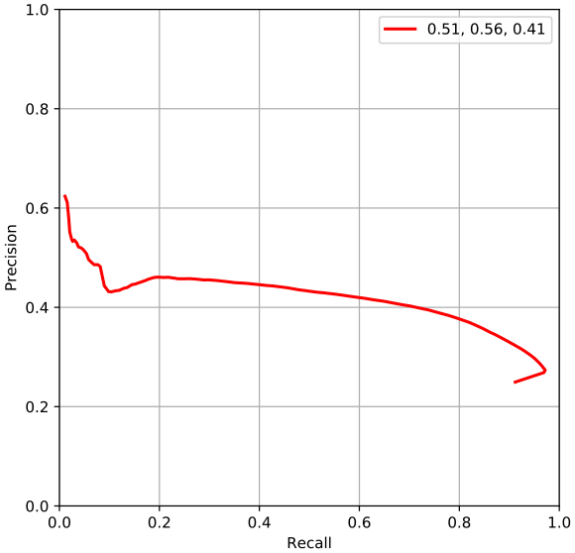
Sample Output 4

Given–

```
          threshold: 0.220000
overall max F1 score: 0.514369
average max F1 score: 0.562687
            area_pr: 0.408983
```

The running time of Given code was: 172.18466305732727

PR plot–



Q2 Task 1-

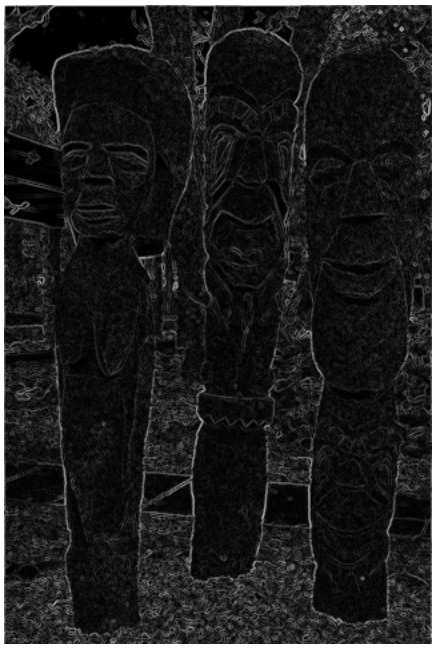
The aim of this task is to produce better edges at the boundaries of the images. The details at the boundaries of the image are missing some detail which was corrected by corrected by using the border reflect function.
The difference can be seen from the following image between results of Given Code and task 1-

Before Task 1

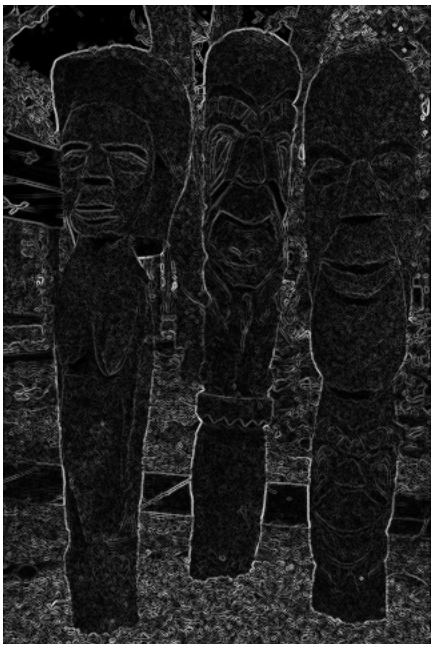
After Task 1

Before Task 1

After Task 1



Before Task 1



After Task 1

Before Task 1

After Task 1

Given–

```
          threshold: 0.220000
overall max F1 score: 0.514369
average max F1 score: 0.562687
            area_pr: 0.408983
```

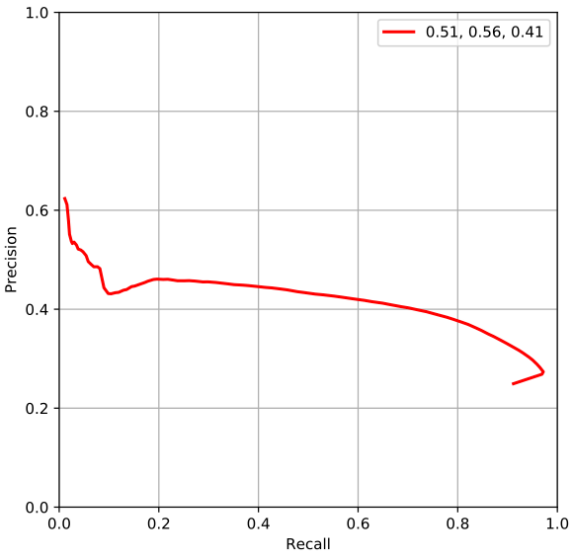
The running time of Given code was: 172.18466305732727

Task 1–

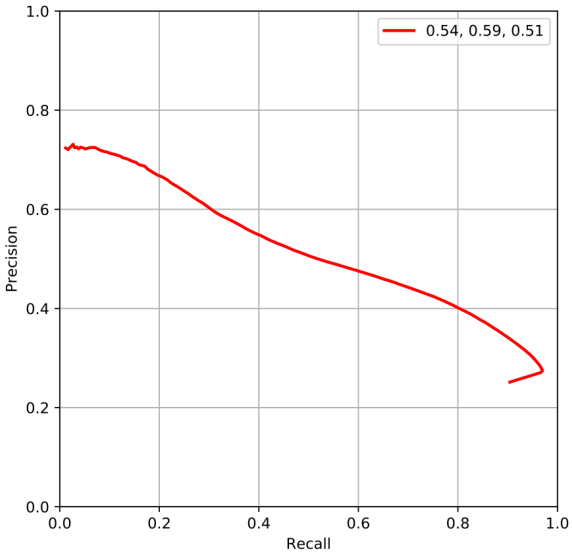
```
          threshold: 0.240000
overall max F1 score: 0.542698
average max F1 score: 0.587614
            area_pr: 0.509440
```

The running time for task 1 was: 177.35533595085144.

PR plot–



Before Task 1



After Task 1

Q2 Task 2-

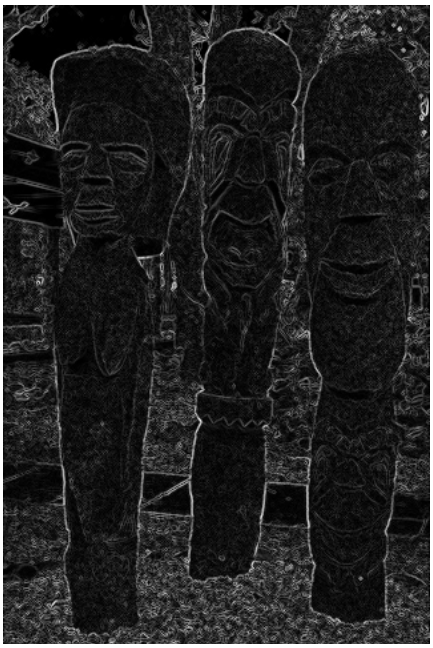
In the task 2 we had to change the filter from the [-1,0,1] to a gaussian filter. The main implementation here was the selection of the parameters. For this case I tried sigma as 2,3,4,5 and 6. The image produced with lower sigma had very thin edges and the image produced by large sigma had very thick edges. Sigma=4 provided with a balance between the thick and the thin edges. So here the Sigma was selected as 4.
The difference can be seen from the following image between results of task 1 and task 2-

Before Task 2

After Task 2

Before Task 2

After Task 2



Before Task 2

After Task 2

Before Task 2

After Task 2

Task 1–

threshold: 0.240000
overall max F1 score: 0.542698
average max F1 score: 0.587614
area_pr: 0.509440

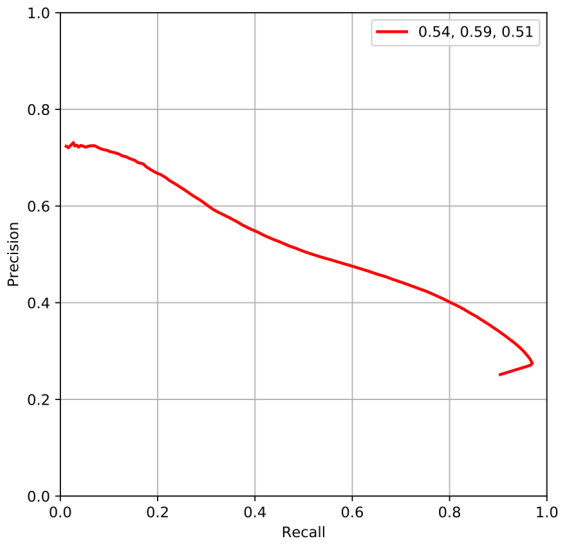
The running time for task 1 was: 177.35533595085144.

Task 2–

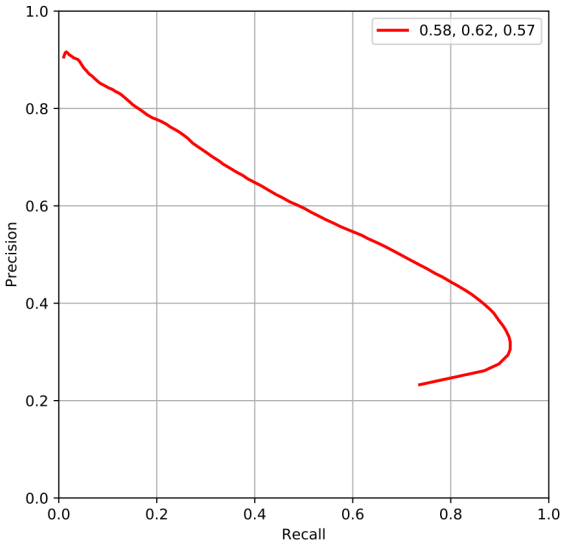
threshold: 0.250000
overall max F1 score: 0.582315
average max F1 score: 0.615587
area_pr: 0.571241

The running time for task 2 was: 179.26073598861694.

Pr Plot–



Before Task 2

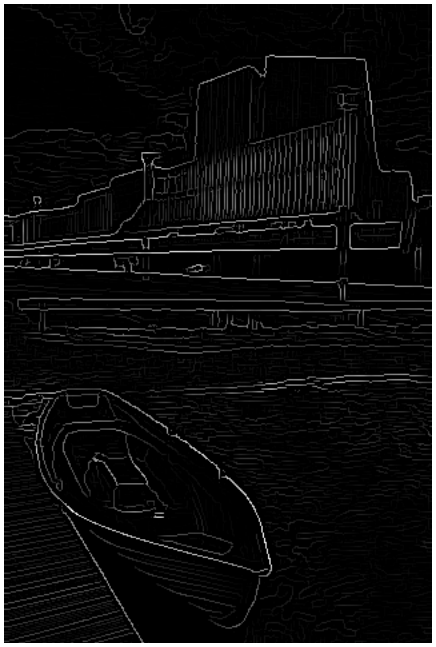


After Task 2

Q2 Task 3-

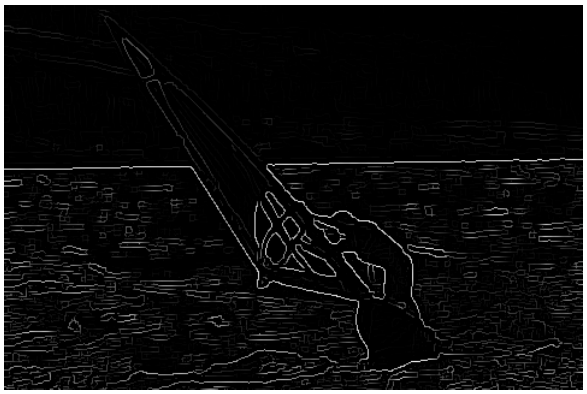
The result of task 2 does not produce thin edges. So, in this part we had to implement the Non-Maximum Suppression. In this method the Gradient of the point was compared with the gradient of its neighbours to produce the edge. To maintain uniformity in the output produced and to depict the results, the sigma used for Task 3 is also kept at 4.
The following are the results for the Comparison of output of task 2 and task 3-

Before Task 3



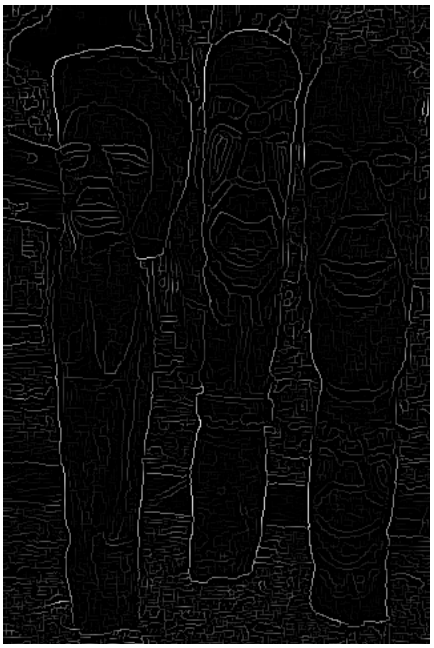
After Task 3

Before Task 3



After Task 3

Before Task 3



After Task 3

Before Task 3

After Task 3

Task 2–

```
          threshold: 0.250000
overall max F1 score: 0.582315
average max F1 score: 0.615587
            area_pr: 0.571241
```

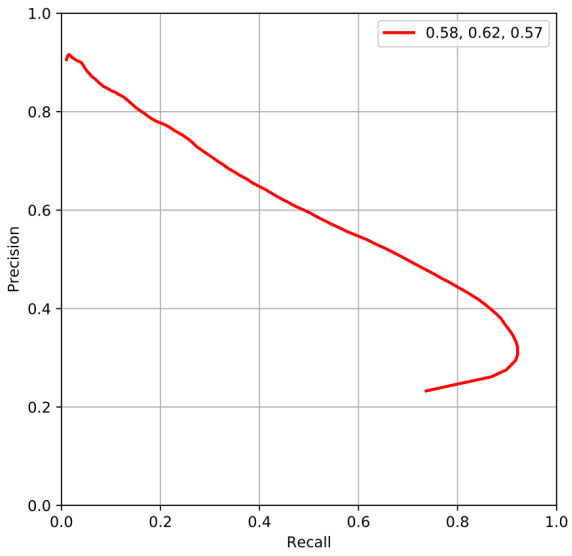
The running time for task 2 was: 179.26073598861694.

Task 3–

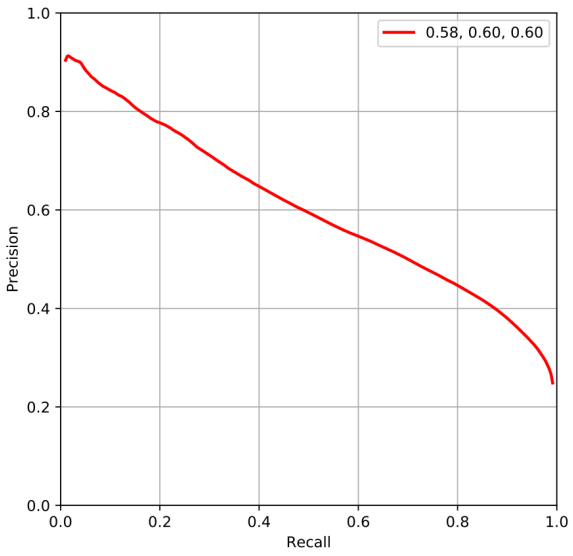
```
          threshold: 0.250000
overall max F1 score: 0.583430
average max F1 score: 0.601078
            area_pr: 0.598545
```

The running time for task 3 was: 166.61612892150879.

PR Plot–



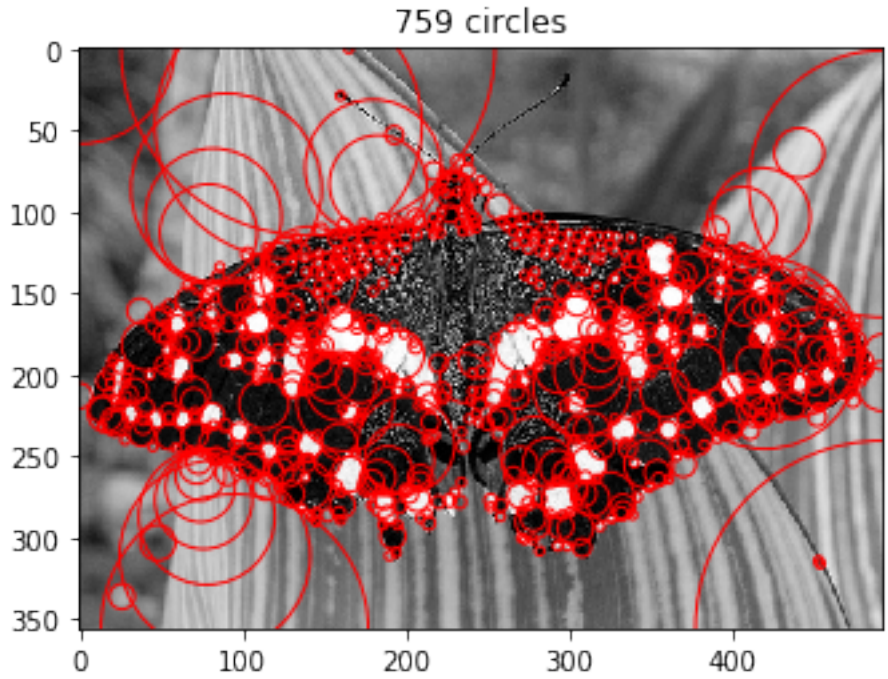
Before Task 3



After Task 3

Question 3-
This question requres us to implement a blob detector.
Task 1- Basic Implementation-
In this task a Laplacian blob detector was generated by increasing the scale by a factor of k.
The result of the Images are as follows-

Image 1-
Values- K=1.4, Level= 14



Image– butterfly.jpg
Run time– 9.5945866529876566

Image 2-
Values- K=1.2, Level= 12

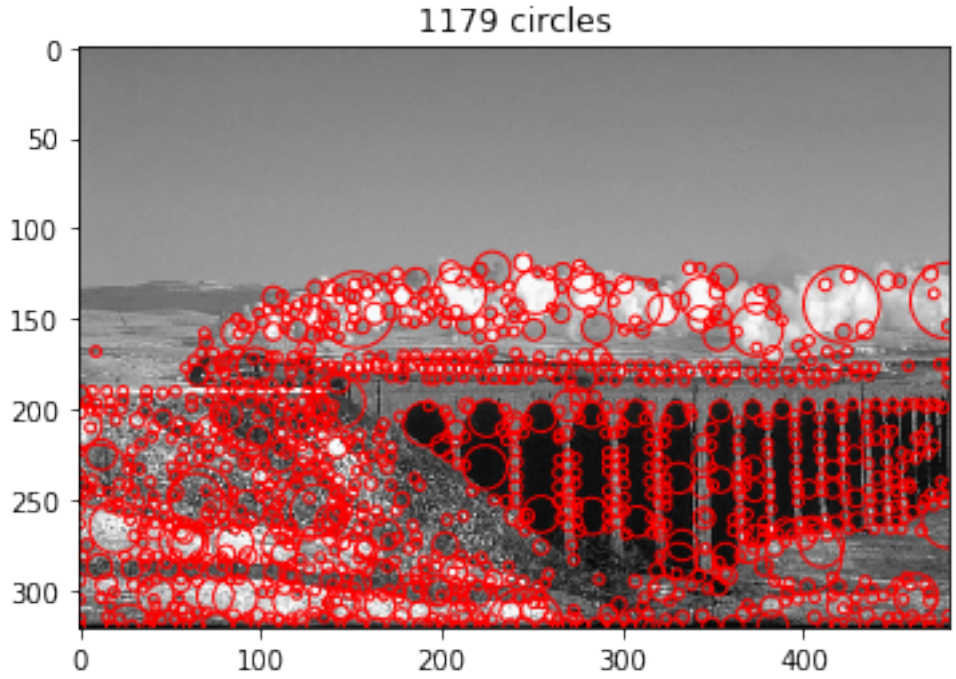


Image- 182053.jpg
Run time- 6.2275385637849568

Image 3-
Values- K=1.2, Level= 12

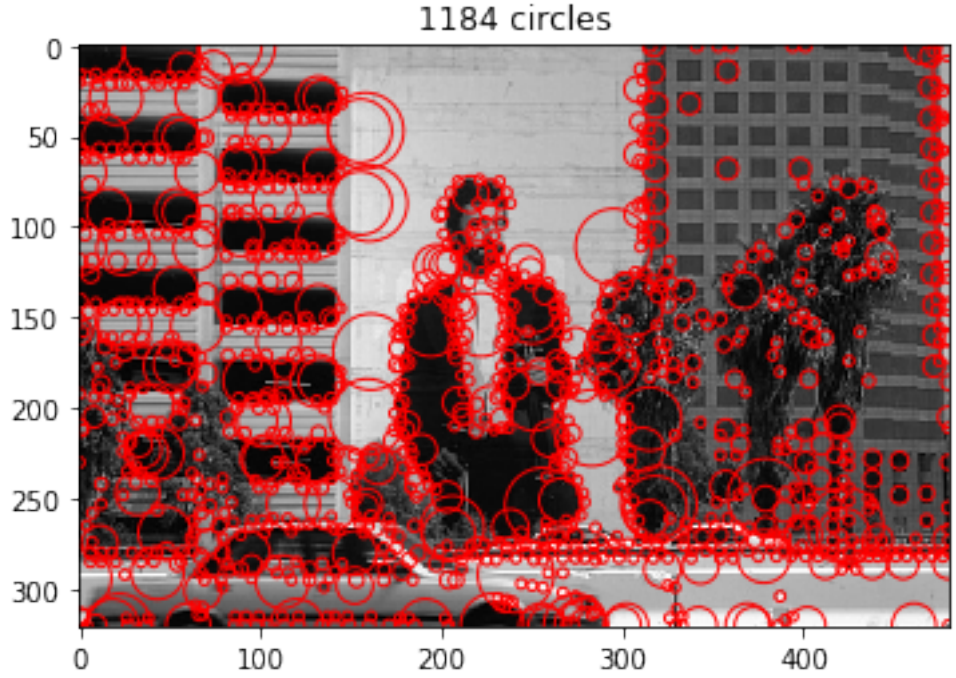


Image- 119082.jpg
Run time- 6.0643819465374573

Image 4-
Values- K=1.3, Level= 12

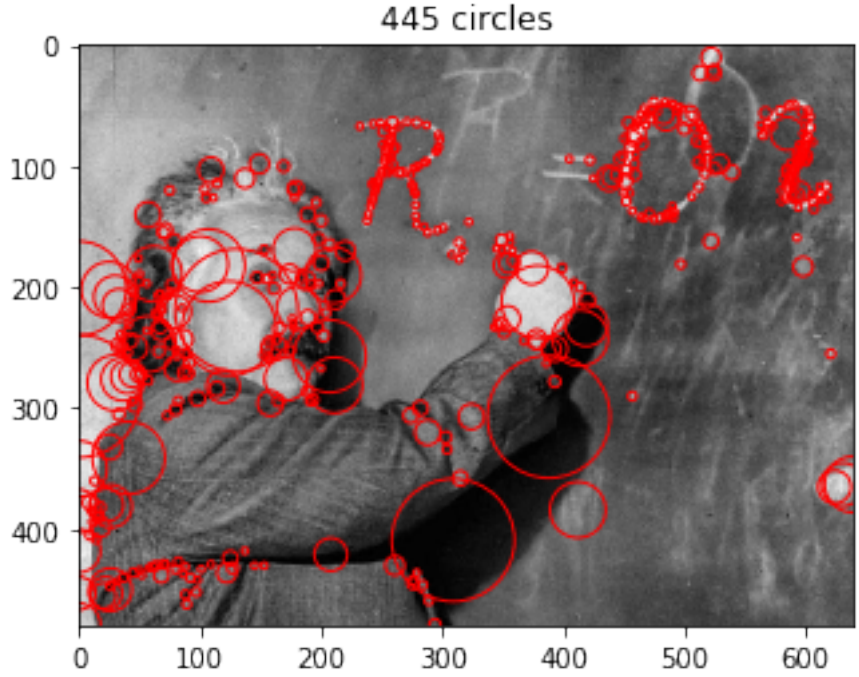


Image- einstein.jpg
Run time- 6.0643819465374573

Image 5-
Values- K=1.3, Level= 12

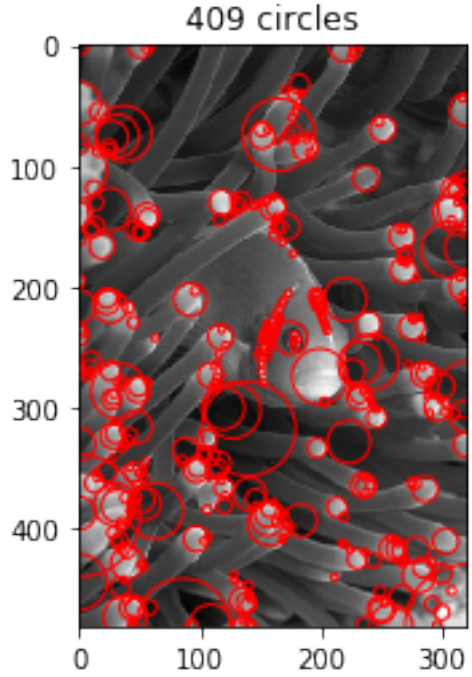


Image- 210088.jpg
Run time- 5.5916453776449756

Image 6-
Values- K=1.3, Level= 12

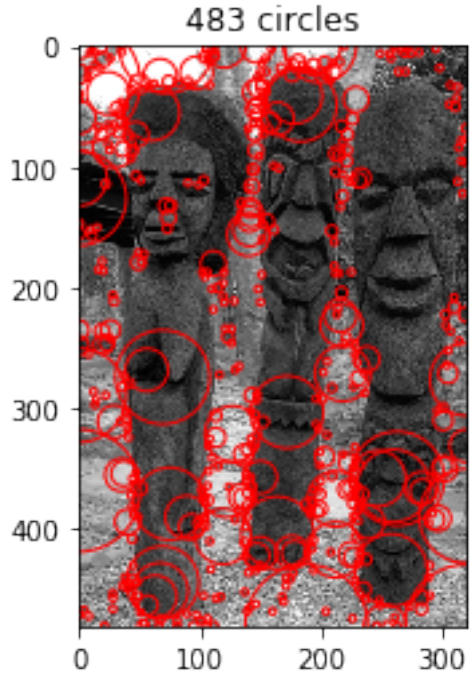


Image- 101085.jpg
Run time- 5.3674537792844664

Image 7-
Values- K=1.3, Level= 13

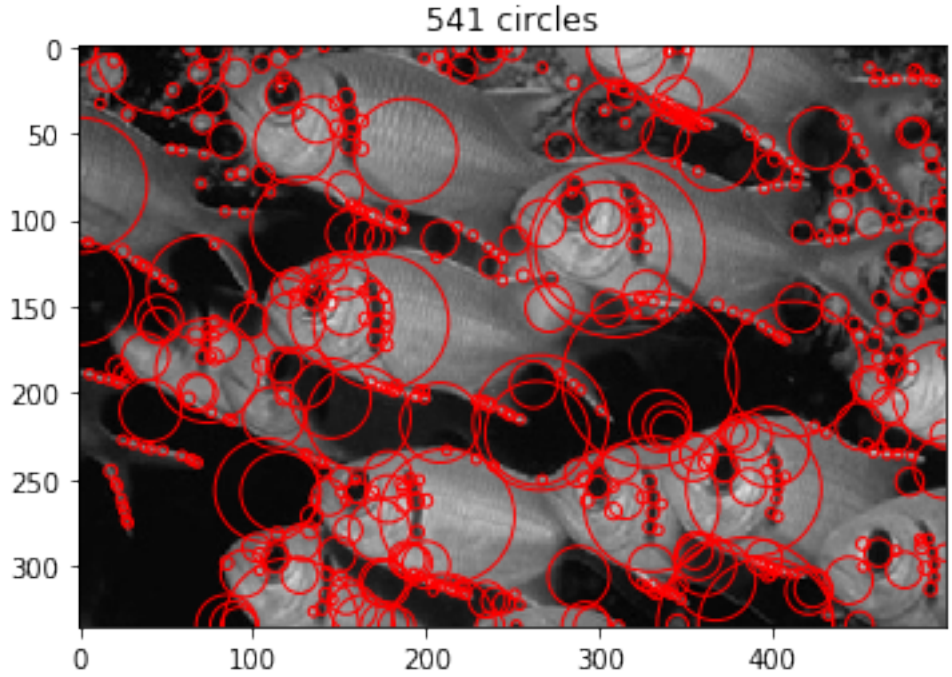


Image- fishes.jpg
Run time- 6.6647421126458309

Image 8-
Values- K=1.3, Level= 13

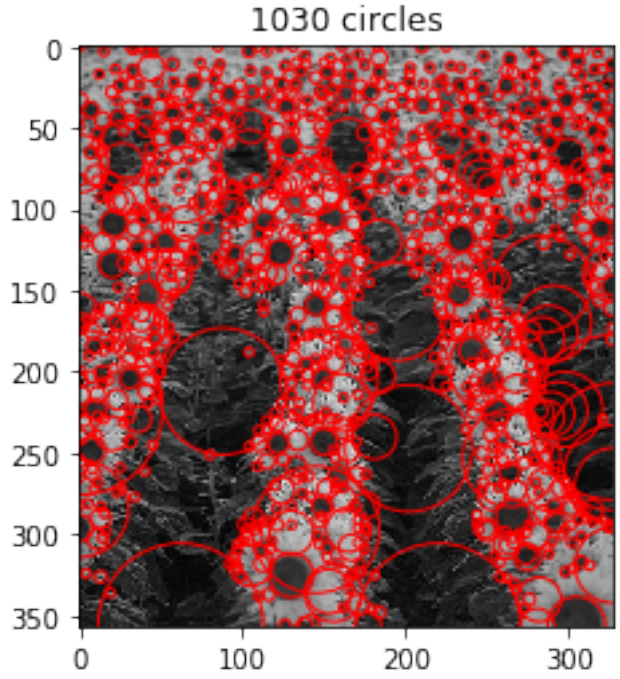


Image- sunflowers.jpg
Run time- 5.4078085664527845

Interesting Implementations-

- The parameters like k and level vary from one image to another. Multiple combinations were produced to select the values that produced the best possible output.
- The recommended scipy functions were used to produce the required results.
- The Output of the parameters and the time is shown as follows-

Image	K	Sigma	Level	Time	Number of Blobs
butterfly.jpg	1.4	2	14	9.5945866529876566	759
182053.jpg	1.2	2	12	6.2275385637849568	1179
119082.jpg	1.2	2	12	6.0643819465374573	1184
einstein.jpg	1.3	2	12	6.0643819465374573	445
210088.jpg	1.3	2	12	5.5916453776449756	409
101085.jpg	1.3	2	12	5.3674537792844664	483
fishes.jpg	1.3	2	13	6.6647421126458309	541
sunflowers.jpg	1.3	2	13	5.4078085664527845	1030

Task 2- Efficient Implementation-
In this task a Laplacian blob detector was generated by downsampling the image by a factor of k.
The result of the Images are as follows-

Image 1-
Values- K=1.4, Level= 14

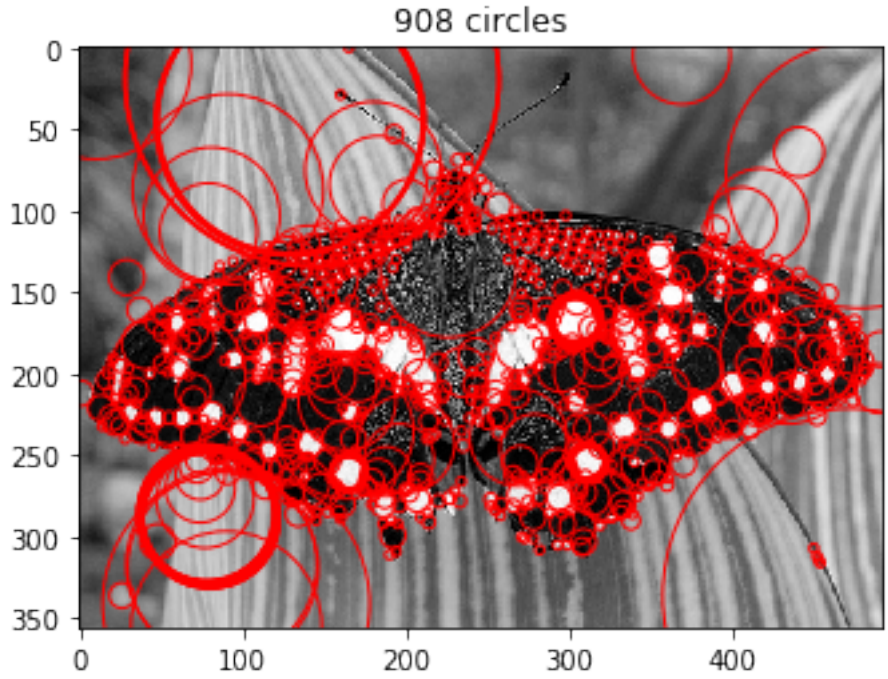


Image- butterfly.jpg
Run time- 8.7865764633411796

Image 2-
Values- K=1.2, Level= 12

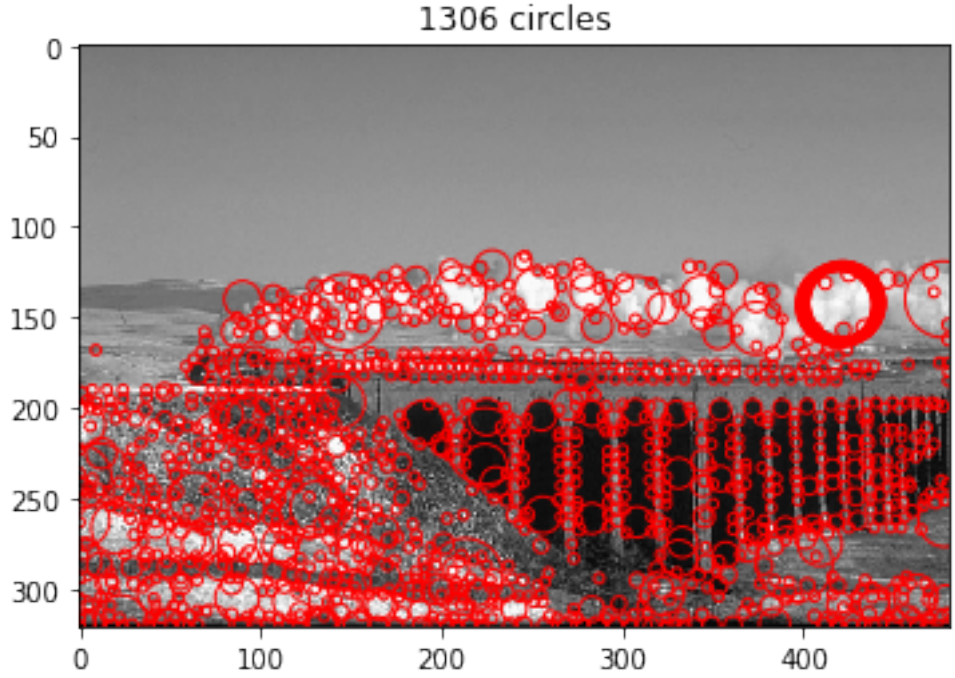


Image- 182053.jpg
Run time- 6.0356438746556291

Image 3-
Values- K=1.2, Level= 12

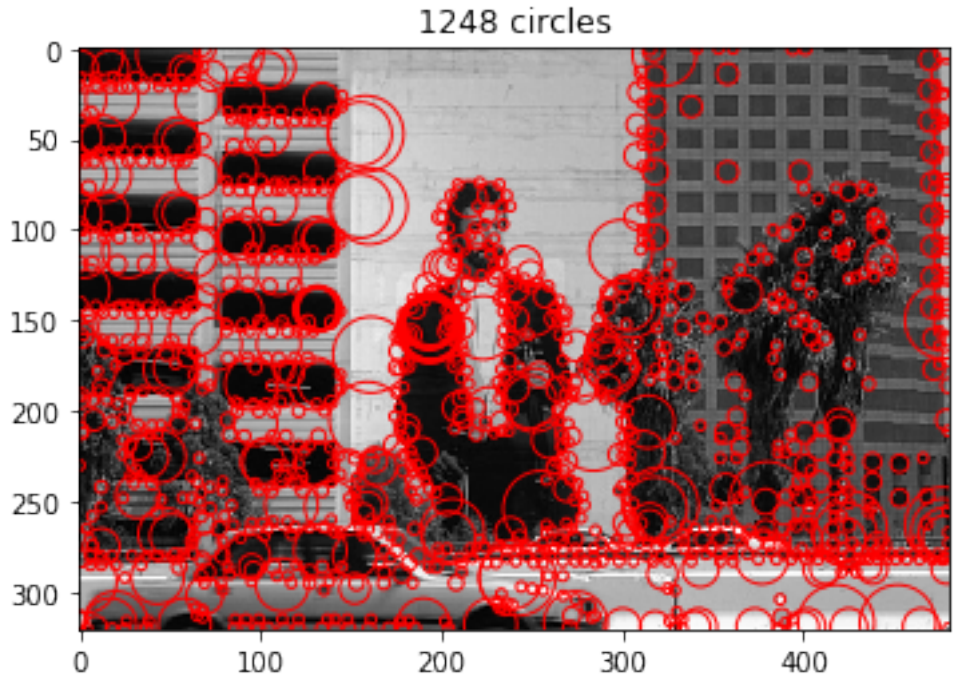


Image- 119082.jpg
Run time- 5.7865745639056087

Image 4-
Values- K=1.3, Level= 12

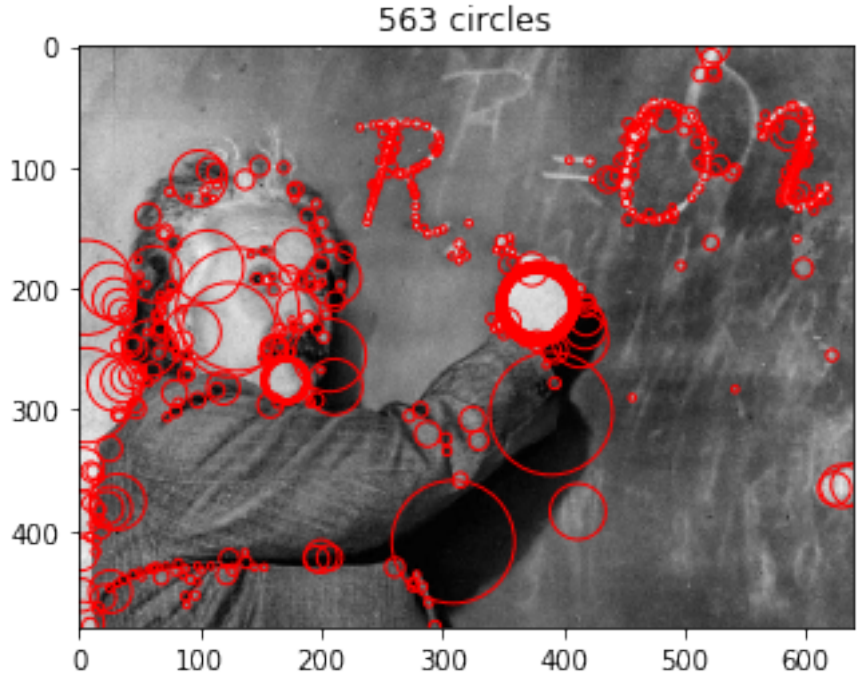


Image- einstein.jpg
Run time- 5.8954725494308674

Image 5-
Values- K=1.3, Level= 12

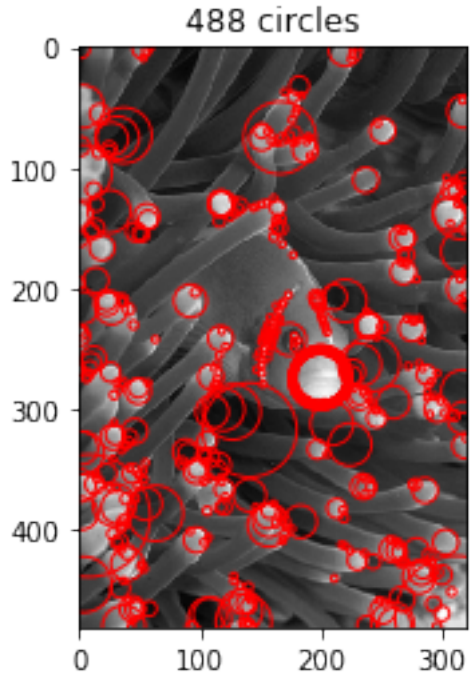


Image- 210088.jpg
Run time- 5.2765038592658522

Image 6-
Values- K=1.3, Level= 12

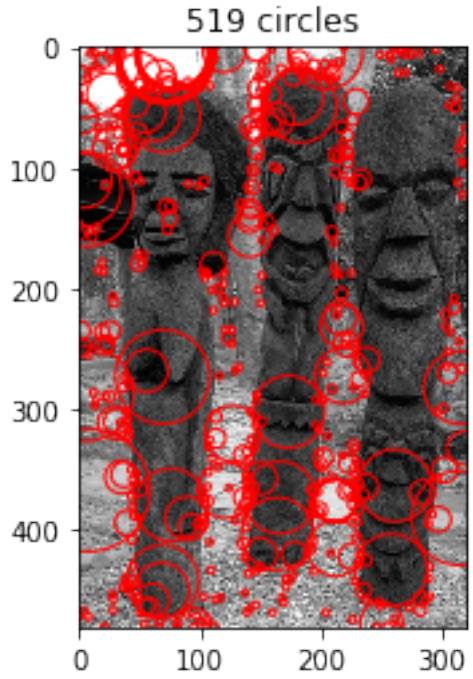


Image- 101085.jpg
Run time- 4.9876573916573385

Image 7-
Values- K=1.3, Level= 13

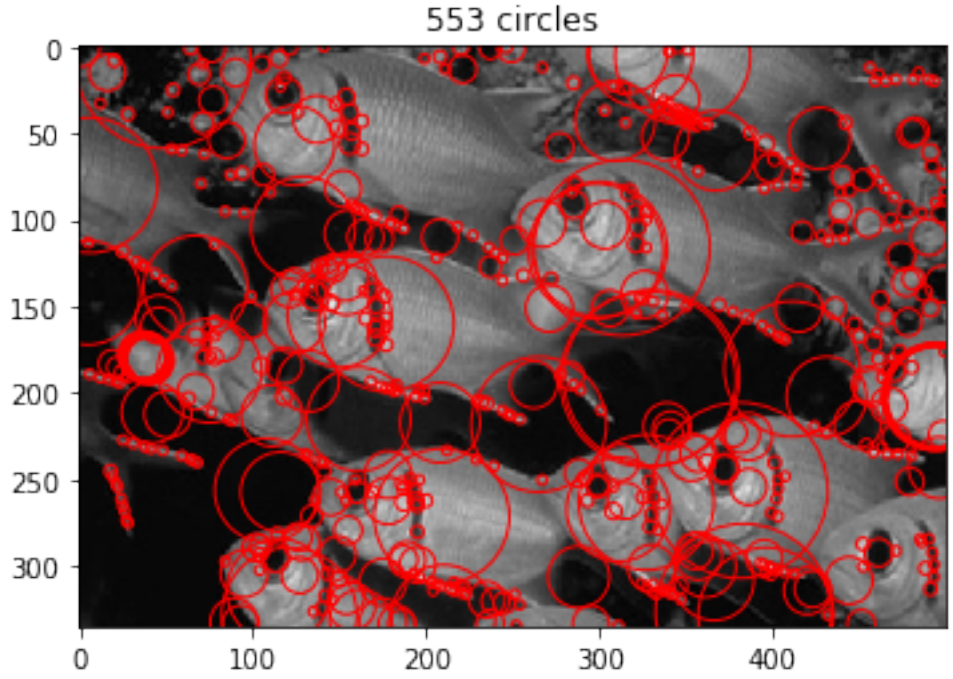


Image- fishes.jpg
Run time- 6.2277638563827564

Image 8-
Values- K=1.3, Level= 13

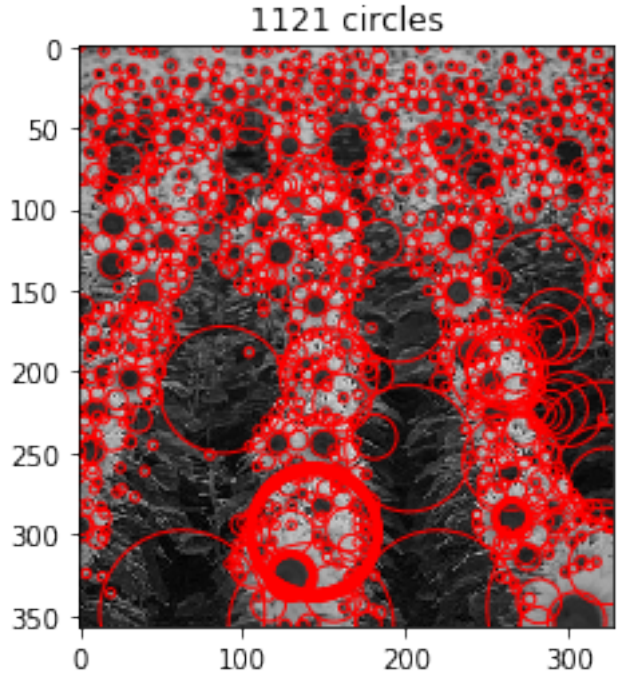


Image- sunflowers.jpg
Run time- 5.1764829104658376

Interesting Implementations and comparison-

- The parameters like k and level vary from one image to another. Multiple combinations were produced to select the values that produced the best possible output.
- The recommended scipy functions were used to produce the required results.
- The down sampling method was used to speed up the blob detection.
- There were more blobs detected by the efficient method than by the basic method.
- After trying out with a lot of values, it was observed that increasing Kernel size will increase time.
- The Output of the parameters and the time is shown as follows-

Image	K	Sigma	Level	Time (Basic)	Time (efficient)	Number of Blobs
butterfly.jpg	1.4	2	14	9.5945866529876566	8.7865764633411796	908
182053.jpg	1.2	2	12	6.2275385637849568	6.0356438746556291	1306
119082.jpg	1.2	2	12	6.0643819465374573	5.7865745639056087	1248
einstein.jpg	1.3	2	12	6.0643819465374573	5.8954725494308674	563
210088.jpg	1.3	2	12	5.5916453776449756	5.2765038592658522	488
101085.jpg	1.3	2	12	5.3674537792844664	4.9876573916573385	519
fishes.jpg	1.3	2	13	6.6647421126458309	6.2277638563827564	553
sunflowers.jpg	1.3	2	13	5.4078085664527845	5.1764829104658376	1121