ECE558 Project: Blob Detector

Amirhassan Fallah Dizche
afallah@ncsu.edu

December 6, 2018

Abstract

In this project we are asked to implement the Laplacian blob detector. Blob detectors play an important role in feature detection and scale selection and are very useful in many computer vision applications such as object detection and tracking. In this project we are asked to implement the core functions instead of using the available libraries. The details of implementation and functions developed for this project are given in the following. Moreover, the result of blob detection on four given test images and six other images are illustrated. This project is done as part of the requirements for ECE 558 course, fall 2018 semester at NC State university.

1 Introduction

Blob detector consists of several ports. In order to detect blobs, we need to generate a Laplacian of Gaussian (LoG) filter with proper dimensions. We can also use the Difference of Gaussian (DoG) filter that approximate the LoG filter and has lower computational cost. The details of LoG and DoG implementations are discussed later. The next step in blob detection is creating a Laplacian scale-space. Starting from an initial scale for n iterations, we should filter the image with scale-normalized LoG (or DoG).
Then we should save the squared Laplacian response for the current scale, increase the scale by some fixed value (e.g. k), and repeat until end of iteration. When the scale-space is completed it is time to perform non-maximum suppression on it.
Since the detector might detect the same object multiple times, non-maximum suppression is used to make sure that the object is detected only once. After that, we need to find the coordinates of the center of blobs (circles) and their radius in order to be able to display them on the input image. The responses should be compared with a threshold and the ones that are greater than the threshold will be selected as center of circles. The radius of circles is calculated based on the scale of the detection. Multiple functions have been developed to implement the above described processes. The code is written in Python 3 and the functions and steps are discussed in the following sections.

2 Blob Detector

As described in previous section, the bob detector has multiple steps and each step required various functions. These steps and the code that is written to implement the required functions are discussed in the following:

2.1 Laplacian of Gaussian filter

The blob detector requires having a Laplacian of Gaussian filter to be convoluted with the input image. Hence, the first task is to implement this filter. There are two different ways of implementing this filter, both are implemented in the code:

2.1.1 LoG filter:

The first method is to implement the LoG filter exactly based on the following equation.

$$\text{LoG} = -\frac{1}{\pi\sigma^4}(1 - \frac{x^2+y^2}{2\sigma^2})\exp(-\frac{x^2+y^2}{2\sigma^2}) \tag{1}$$

The LoG filter is implemented by the `LoG_filter(sigma)` function in the main code (`afallah_blob.py`). This code will return a LoG filter with proper dimensions scaled with σ^2. This method will generate a more accurate result bus has more computation complexity compared to the approximation that is described next. The LoG filter is illustrated in Figure 1.

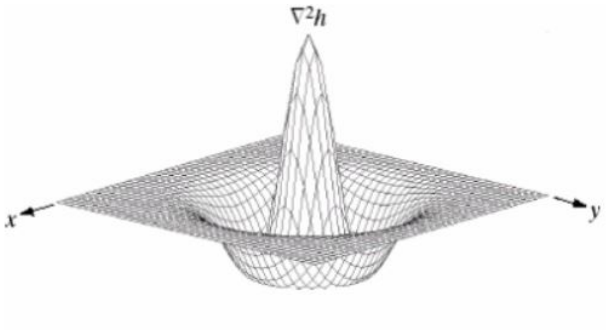


Figure 1: Illustration of LoG filter.

2.1.2 DoG filter:

This method will approximate the LoG filter with parameter σ with difference of two Gaussian functions with variances equal to 1.6σ and σ respectively.

$$G_\sigma = \frac{1}{2\pi\sigma^2}\exp -\frac{x^2+y^2}{2\sigma^2} \tag{2}$$

$$\text{LoG} \approx G_{1.6\sigma} - G_\sigma$$

This filter is implemented via `Dog_filter(sigma)` function in the main code (`afallah_blob.py`). Figure 2 indicates the approximation of LoG filter with DoG filter.

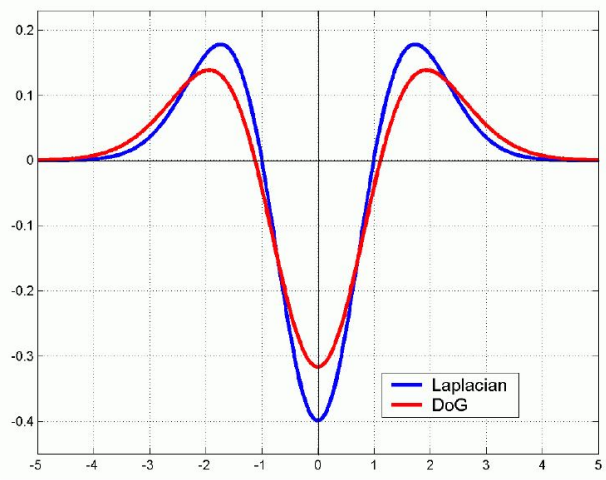


Figure 2: Approximation of LoG filter with DoG.

2.1.3 Two-dimensional Convolution:

In order to make the implementation more efficient, we will keep the size of the filter fixed and reduce the size of input image by down-sampling. In order to avoid aliasing effects in down-sampling the image, a Gaussian blurring filter is applied to the image to remove the high frequency components before down-sampling. The two-dimensional convolutions in this project are implemented using the `Conv_2D.py` function that was written previously for Homework 2 of this course. The `conv2d` function in this python code will do the convolution using `copyedge` padding and `same` output size. This means that the input image will be padded by the edge values around its borders to implement the convolution and the resulting output will have the same dimensions as the input image.

```
conv2d(image, kernel, 'same', 'copyedge')
```

The kernel used in this command is going to be either LoG filter or DoG filter.

2.2 Scale Space Creation

The next step of implementing blob detector is to create the scale-space of the input image. In order to do that we need two more functions naming `Resize` and `max_replace2`.
The `Resize` function uses bi-linear interpolation to change the size of the given image into desired output dimensions (either smaller or larger). This function is used twice in creating the scale-space. First to down-sample the image, and then two up-sample it after applying the scaled LoG (or DoG) filter.

```
Resize(image, [out_row, out_col])
```

The `image` indicates the input image and [out_row, out_col] indicate the desired output dimensions.
The `max_replace2` function does the non-maximum suppression job. Given size of a neighborhood it will replace the elements of that neighborhood by the maximum value of it.

`max_replace2(array, n)`

where `array` is the input image and `n` is the size of neighborhood for non-maximum suppression.
Using above described functions we can form the the following function that creates the scale-space:

`scale_space_create(image, sigma, n, Filter)`

where `image` is the input image, `sigma` is the initial scale value, `n` is the number of scales to be created, and `Filter` is the type of filter to be applied at each scale (either 'LOG' or 'DOG'). The above function will first create the LoG (or DoG) filter based on its input argument by calling the proper function. Then, it will down-sample the image based on the iteration number (from 1 to n) using the `Resize` function. Then the scaled (by σ^2) LoG (or DoG) filter will be applied on the on the down-sampled image using `conv2d` function. After that, the resulting array will be squared and then up-sampled to its original value using the `Resize` function again. The result will be recorded in the scale-space tensor (three-dimensional matrix). Then, the non-max suppression will be implemented for each scale using the `max_replace2` function, and the results will be recorded in another tensor. Finally the image is convoluted with a Gaussian blurring filter before down-sampling for the next scale, in order to prevent aliasing.

2.3 Blob Detection

Next, we need to find the maximum response scale. In order to do this the function `find_scale` is defined.

`find_scale(org_row, org_col, Scale_Space)`

The above function gets the dimensions of the original input image and the result of the `scale_space_create` function as input. This function will find the scale that has the maximum response to the LoG (or DoG) filter in the scale-space created previously. The result is recorded in a tensor called `max_scale`.
Having these information, we can now start to find the circles. The `find_circles` functions is defined to do this task.

`find_circles(max_scale, threshold, sigma)`

This function gets the result of previous function (`find_scale`) and the values of threshold and σ as the input and returns the coordinates of the center of the circles. It also calculates the radius of each circle based on the values of scale and σ using the following formula:

$$r = \sqrt{2}\sigma \tag{3}$$

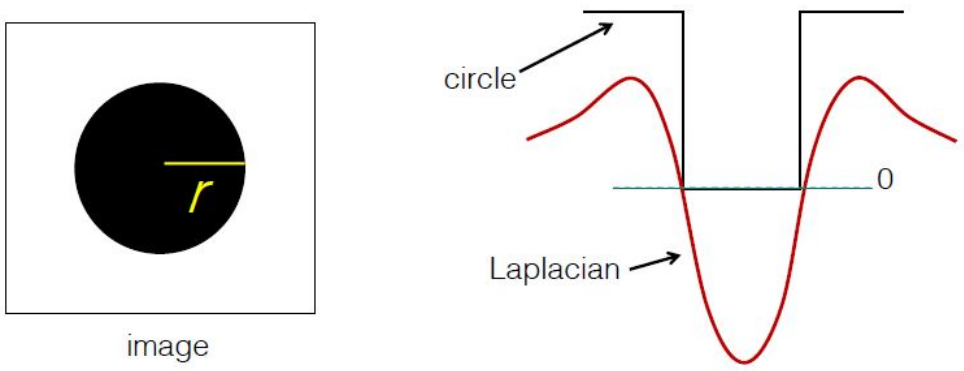


Figure 3: How to find the raduis of detected circles.

Finally, we need a function to depict the results and draw the circles of detected blobs with proper radius and center on the original input image. The `draw_circles` function is developed to handle this task.

```
draw_circles(image, output_name, cx, cy, rad, color='r')
```

This function takes the original image, the desired name to save the output under it, the coordinates of center of circles as well as their radius, and the color of the circles as input. It will iterate over the center of circles and draw them according to the proper radius. The resulting image will be saved under the given name.

2.4 Final Function: blob_detector

The final piece of the puzzle that puts all the above functions to work is the `blob_detector` function. This function calls the above described functions in the required order and gives the output of each of them to the input of the next one so that the desired result will be produced as output of the blob detector.

```
blob_detector(File_name, sigma, n, threshold, Filter, output_name)
```

Input arguments:

- `File_name`: name of the input image (e.g. 'butterfly.jpg')
- `sigma`: σ value (the initial scale value, e.g. 2)
- `n`: number of scales to iterate over and levels in scale pyramid (e.g 10)
- `threshold`: the threshold for blob detection (e.g. 0.01)
- `Filter`: 'LOG' or 'DOG' filter
- `output_name`: desired name to save the resulting output of blob detection (e.g. 'blob_out')

Output:
An image with circles of blobs detected with proper scale drawn on it.

3 Blob Detection Results

The results of applying the blob detector on two sets of images is presented in this section. The first set is the testing images given by the project description. The second set of images are chosen by the student to demonstrate the effectiveness and performance of the code. More details on computation time and parameters used are provided in table 1.

Four test images: These images include 'butterfly', 'Einstein', 'fishes', and 'sunflowers' images. The results are depicted in figures 4-7.

Student Images: Six images are tested in this section. Images are selected to cover both images of natural objects and scenes and the synthetic images created by computer graphics programs. These images include: 'rocks', 'Hunt' (library), 'ECE' (logo), 'circles' (synthetic), 'Hunt-2' (library), and 'solar system'. The results are depicted in figures 8-13.

3.1 Experiment

The blob detector is applied on each image with six different configurations. The parameters for each blob detector is as follows:

1. The initial scale (σ) = 2, Scale size = 10, threshold = 0.01, Filter type = DoG.
2. The initial scale (σ) = 2, Scale size = 15, threshold = 0.01, Filter type = DoG.
3. The initial scale (σ) = 2, Scale size = 10, threshold = 0.01, Filter type = LoG.
4. The initial scale (σ) = 2, Scale size = 15, threshold = 0.01, Filter type = LoG.
5. The initial scale (σ) = 5, Scale size = 10, threshold = 0.01, Filter type = DoG.
6. The initial scale (σ) = 5, Scale size = 10, threshold = 0.01, Filter type = LoG.

Note these values for initial scale and threshold are selected using try and error.

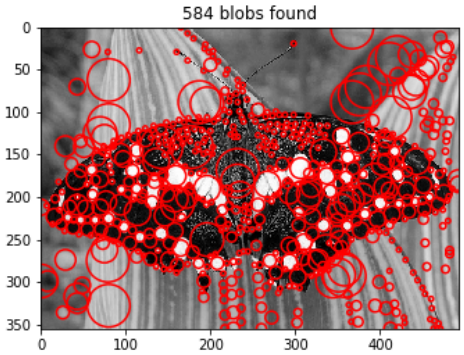
3.2 Example Command

In order to run the blob detector one need to have the 'Numpy' and 'Pillow' libraries installed. If they are not already installed they can be easily installed using the following command:

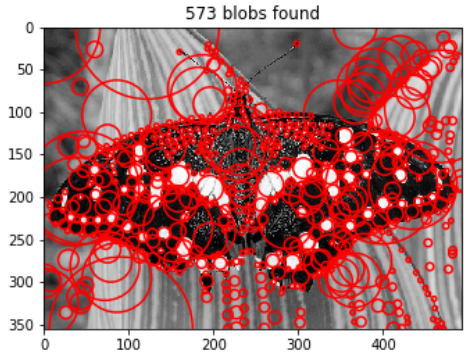
```
pip install pillow,   pip install Numpy
```

After installing the above libraries, one should run the 'afallah_blob.py' file. Then in order to apply it on an image the following command should be used:

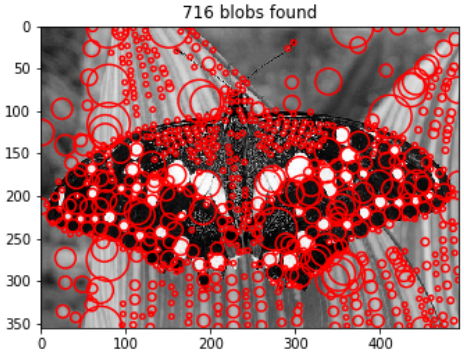
```
blob_detector('butterfly.jpg', 2, 10, 0.01, 'DOG', 'buttefly_blob')
```



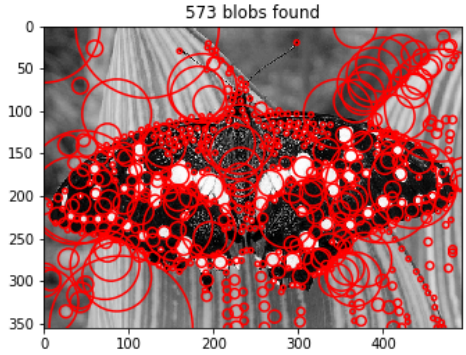
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



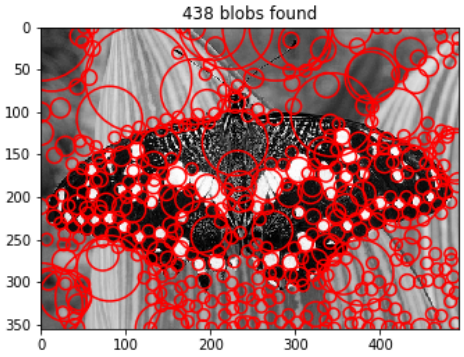
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



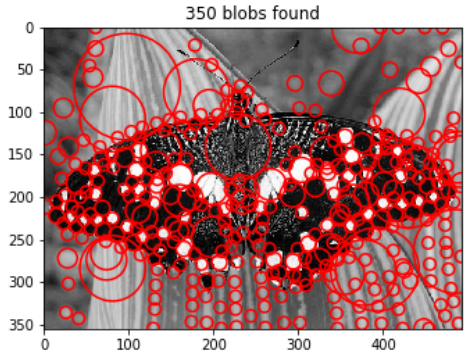
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



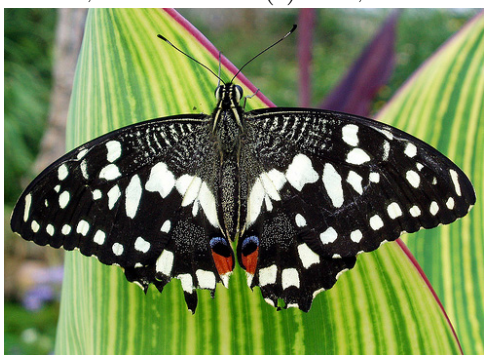
(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG

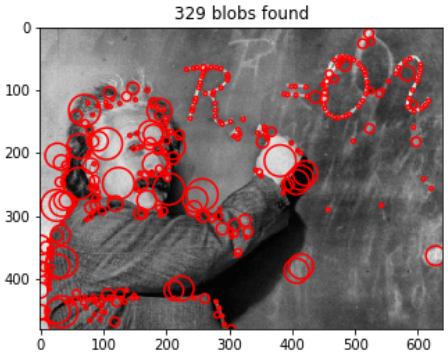


(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

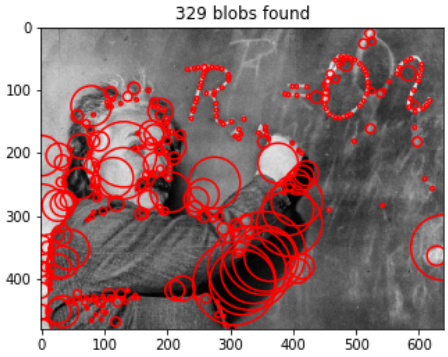


(g) Original Image

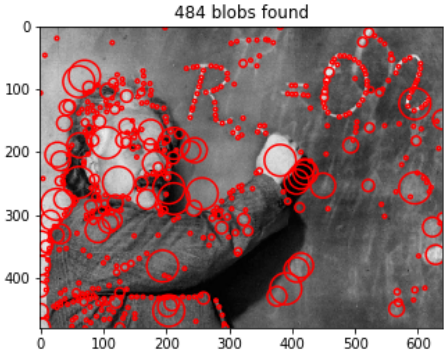
Figure 4: Test Image: 'butterfly'.



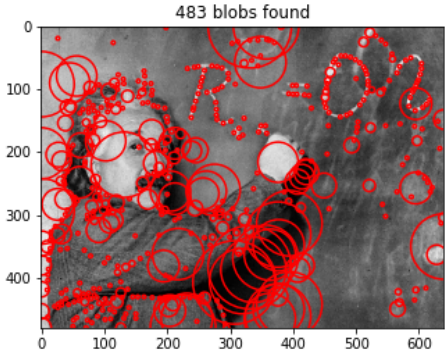
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



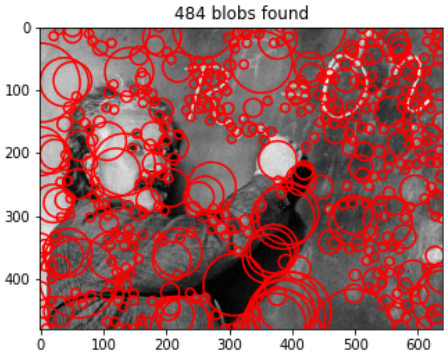
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



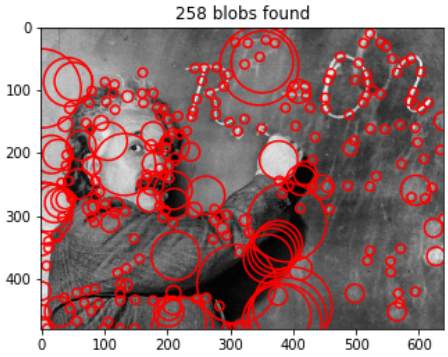
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



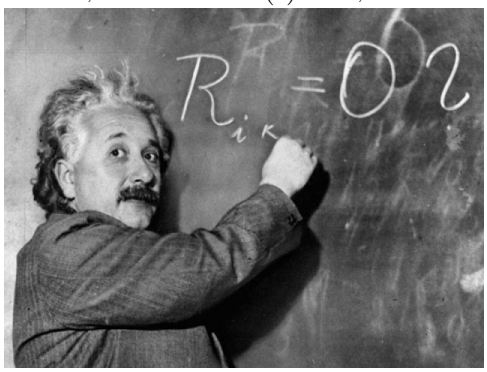
(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG

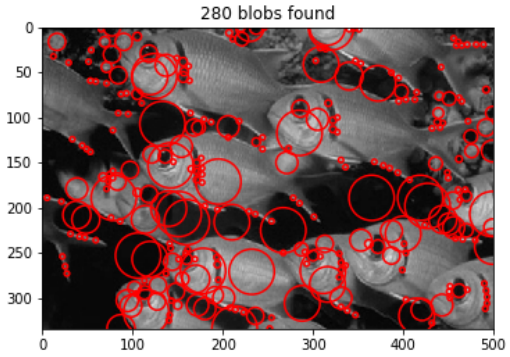


(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

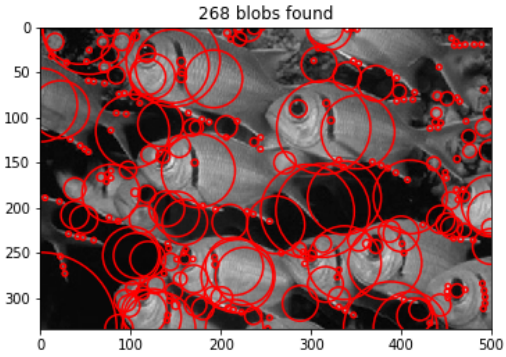


(g) Original Image

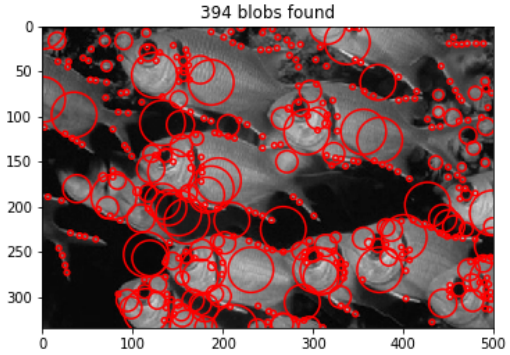
Figure 5: Test Image: 'Einstein'.



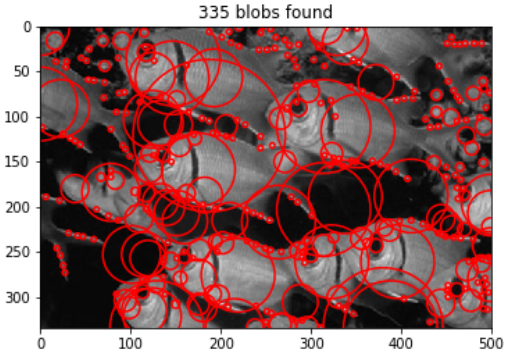
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



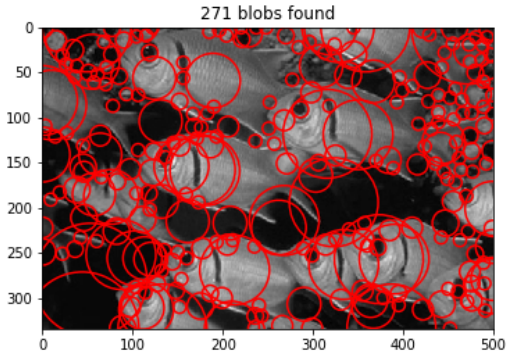
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



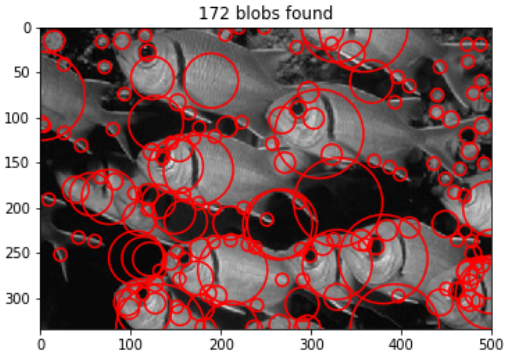
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



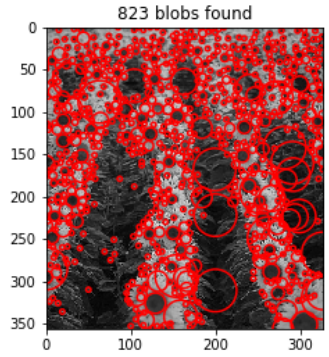
(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG



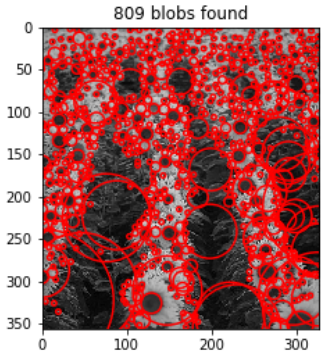
(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

(g) Original Image

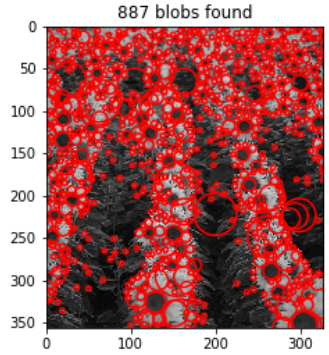
Figure 6: Test Image: 'fishes'.



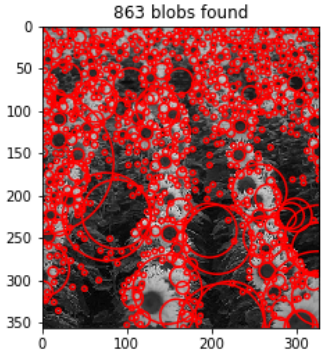
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



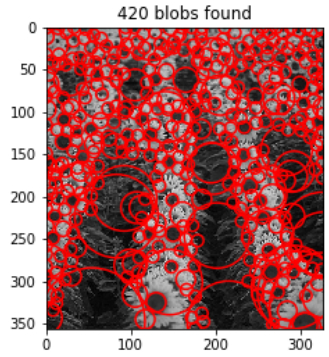
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



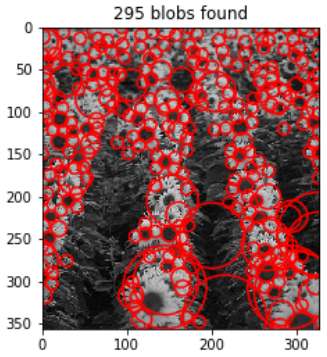
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



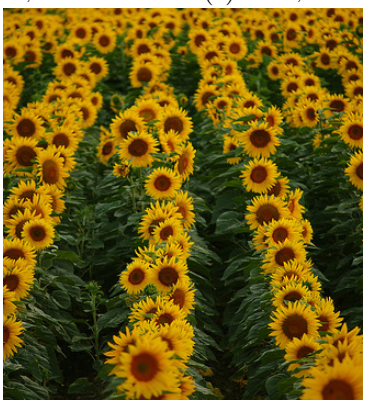
(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG

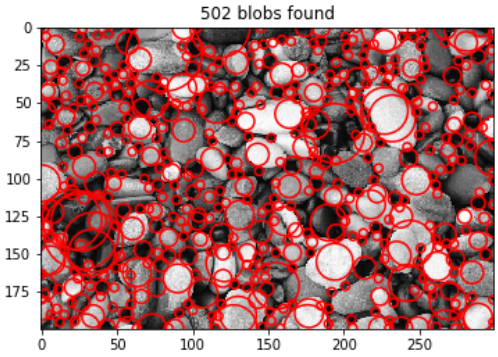


(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

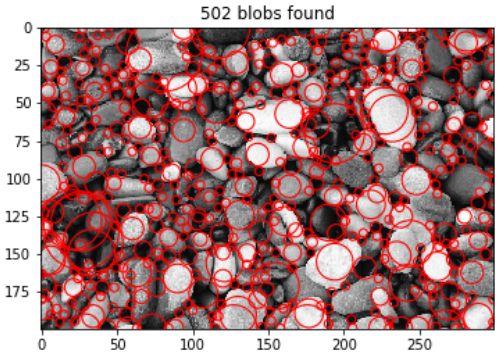


(g) Original Image

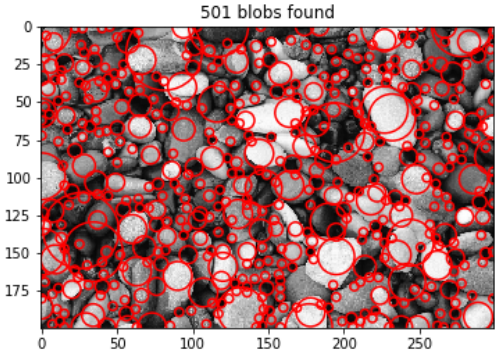
Figure 7: Test Image: 'sunflowers'.



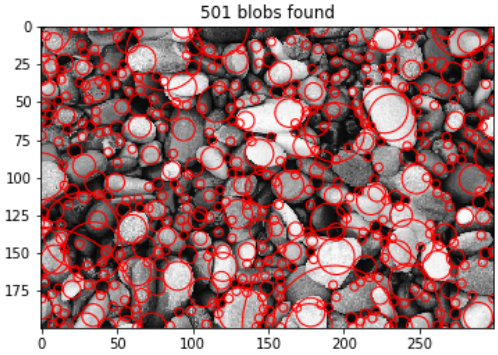
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



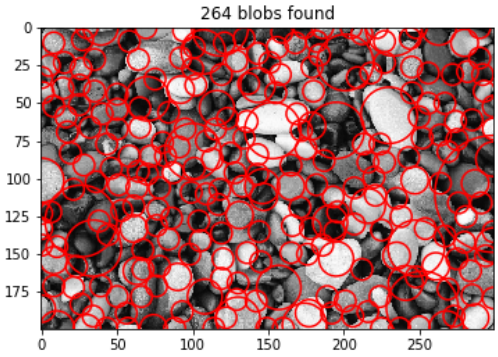
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



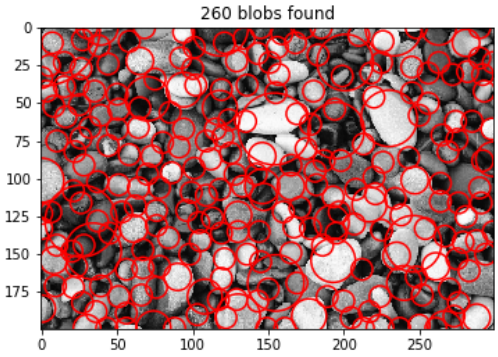
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



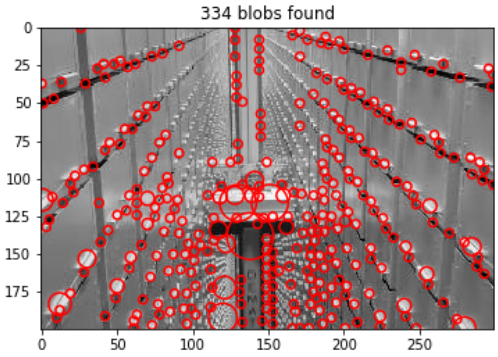
(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG



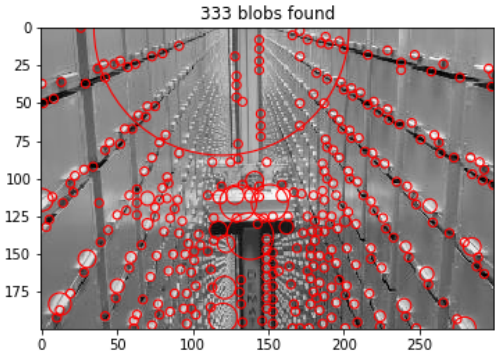
(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

(g) Original Image

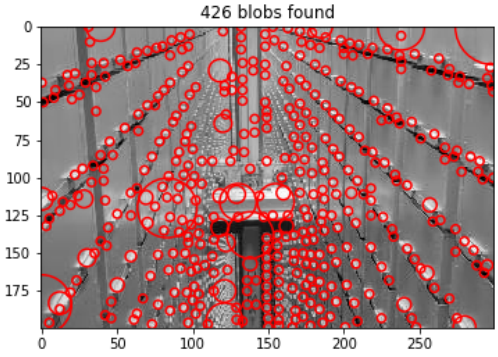
Figure 8: Student Image: 'rocks'.



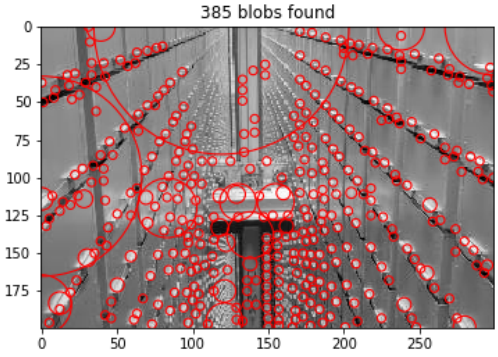
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



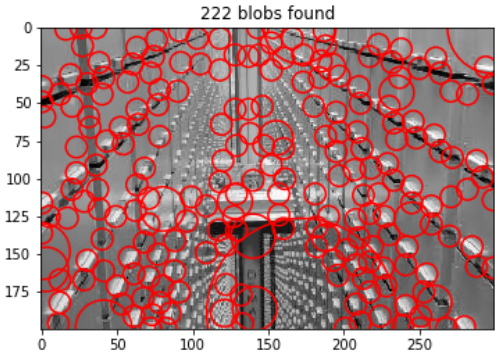
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



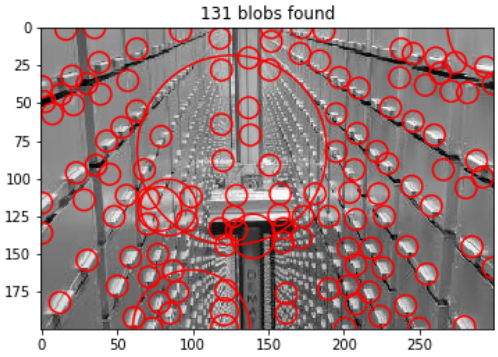
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



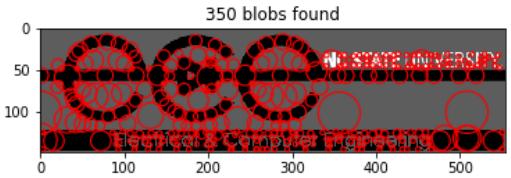
(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG



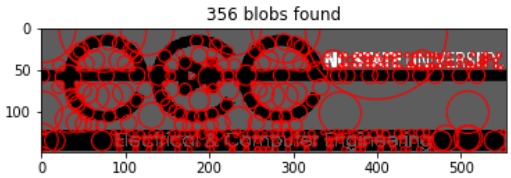
(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

(g) Original Image

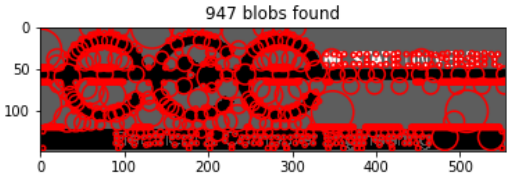
Figure 9: Student Image: 'Hunt'.



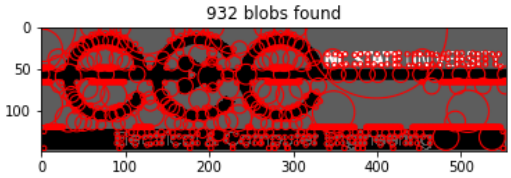
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



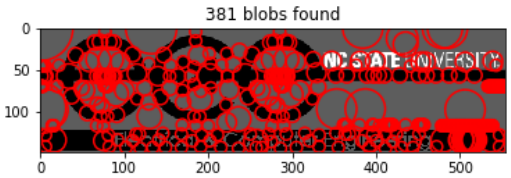
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



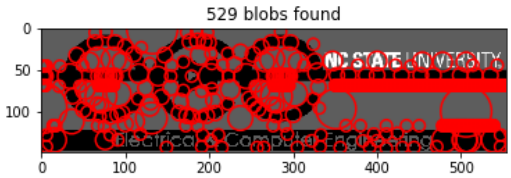
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



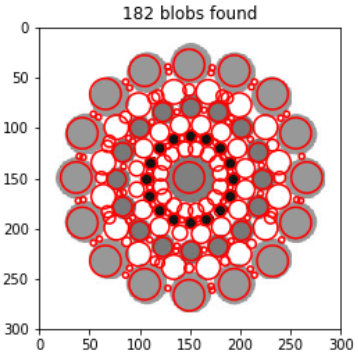
(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG



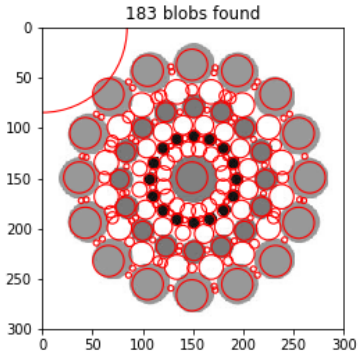
(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

(g) Original Image

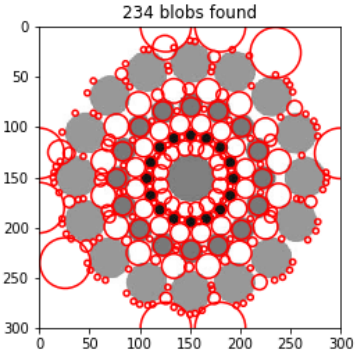
Figure 10: Student Image: 'ECE'.



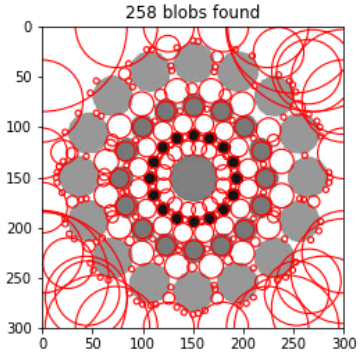
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



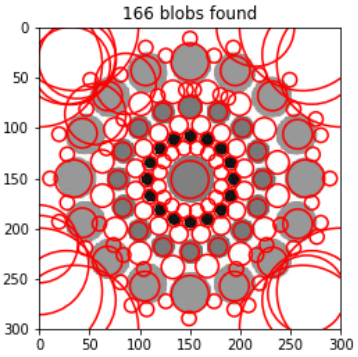
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



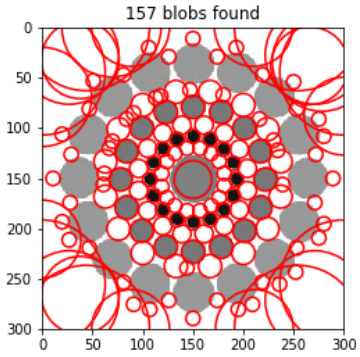
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



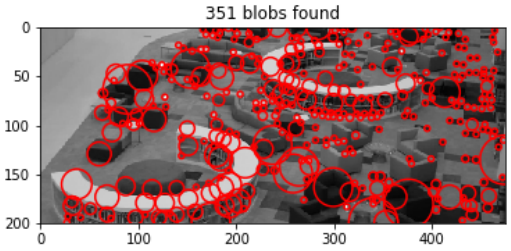
(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG



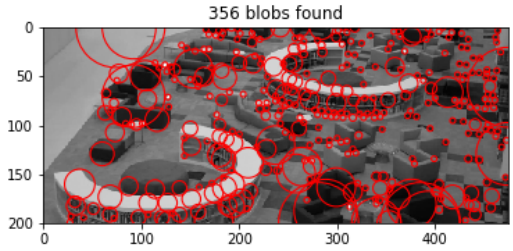
(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

(g) Original Image

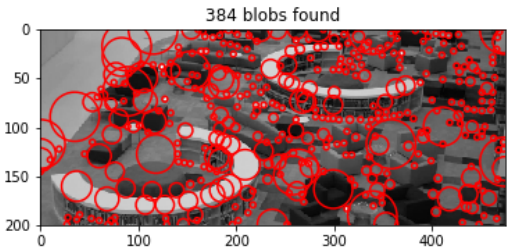
Figure 11: Student Image: 'circles'.



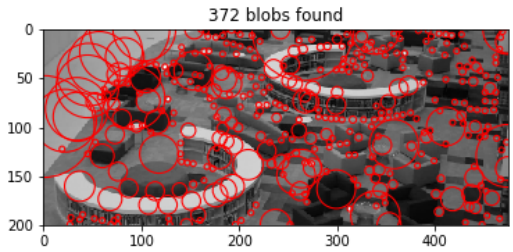
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



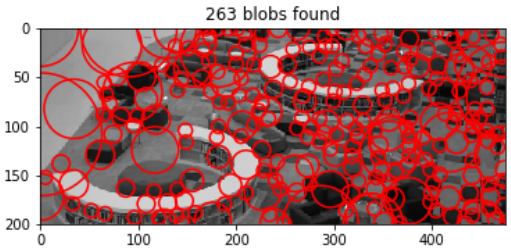
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



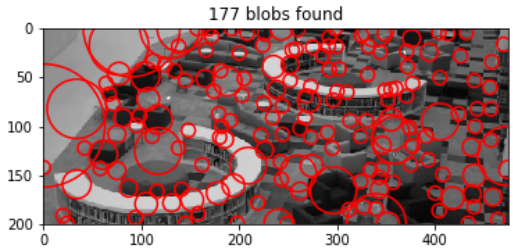
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



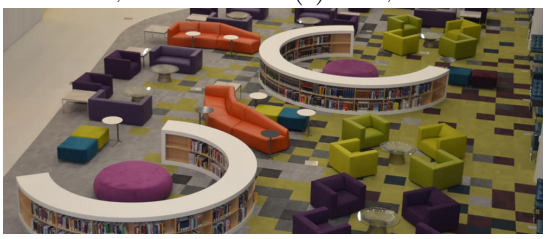
(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG

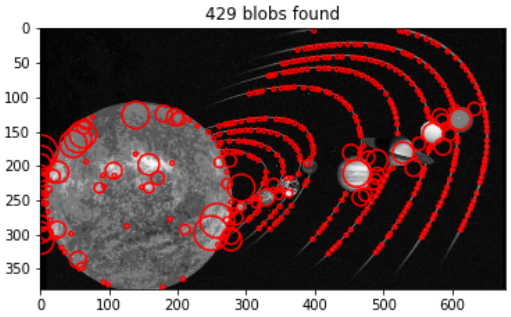


(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG

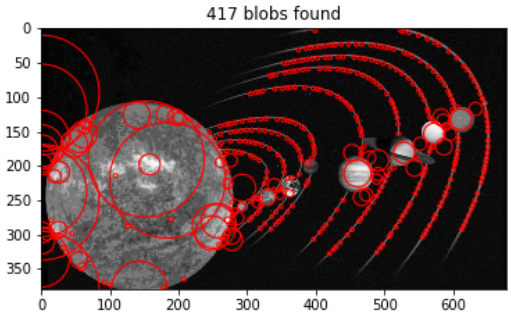


(g) Original Image

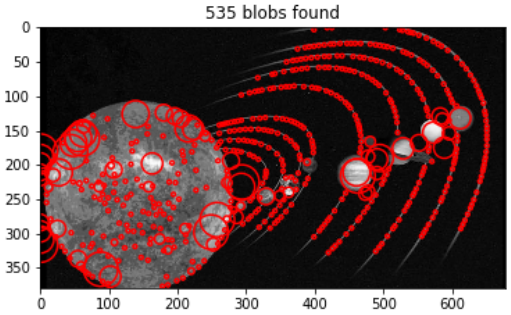
Figure 12: Student Image: 'Hunt-2'.



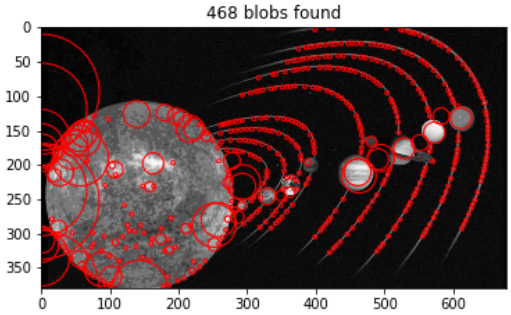
(a) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



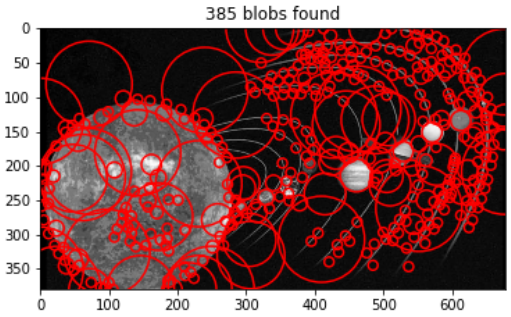
(b) σ= 2, threshold= 0.01, # of scales= 15, Filter= DoG



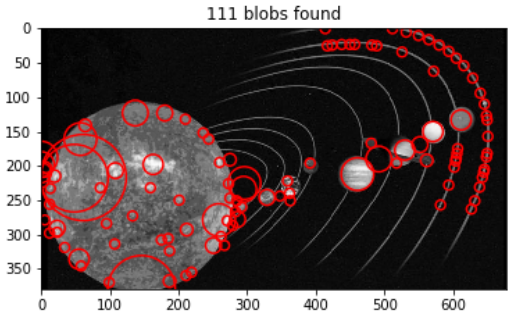
(c) σ= 2, threshold= 0.01, # of scales= 10, Filter= LoG



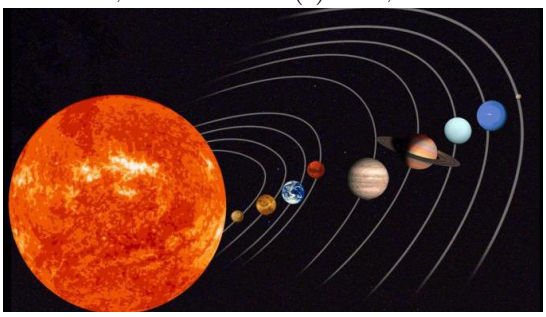
(d) σ= 2, threshold= 0.01, # of scales= 15, Filter= LoG



(e) σ= 5, threshold= 0.01, # of scales= 10, Filter= DoG



(f) σ= 5, threshold= 0.01, # of scales= 10, Filter= LoG



(g) Original Image

Figure 13: Student Image: 'solar system'.

Table 1: Blob detection runs and their results for Test Images.

run	image name	initial scale	scale size	threshold	Filter	time	blobs found
1	butterfly	2	10	0.01	DoG	43.7463	584
2	butterfly	2	15	0.01	DoG	61.4173	573
3	butterfly	2	10	0.01	LoG	43.9239	716
4	butterfly	2	15	0.01	LoG	61.1940	573
5	butterfly	5	10	0.01	DoG	49.2448	483
6	butterfly	5	10	0.01	LoG	49.2765	350
7	einstein	2	10	0.01	DoG	48.1930	329
8	einstein	2	15	0.01	DoG	106.6267	329
9	einstein	2	10	0.01	LoG	48.9386	484
10	einstein	2	15	0.01	LoG	110.7511	483
11	einstein	5	10	0.01	DoG	58.4803	484
12	einstein	5	10	0.01	LoG	58.5772	258
13	fishes	2	10	0.01	DoG	26.0474	280
14	fishes	2	15	0.01	DoG	57.7511	268
15	fishes	2	10	0.01	LoG	26.4342	394
16	fishes	2	15	0.01	LoG	58.0712	335
17	fishes	5	10	0.01	DoG	31.3585	271
18	fishes	5	10	0.01	LoG	31.6313	172
19	sunflowers	2	10	0.01	DoG	19.1962	280
20	sunflowers	2	15	0.01	DoG	41.3874	268
21	sunflowers	2	10	0.01	LoG	19.1193	394
22	sunflowers	2	15	0.01	LoG	42.4214	335
23	sunflowers	5	10	0.01	DoG	22.3885	271
24	sunflowers	5	10	0.01	LoG	22.1346	172

Table 2: Blob detection runs and their results for Student Images.

run	image name	initial scale	scale size	threshold	Filter	time	blobs found
1	rocks	2	10	0.01	DoG	9.8595	502
2	rocks	2	15	0.01	DoG	21.1215	502
3	rocks	2	10	0.01	LoG	9.9684	501
4	rocks	2	15	0.01	LoG	21.1365	501
5	rocks	5	10	0.01	DoG	11.7534	264
6	rocks	5	10	0.01	LoG	11.6995	260
7	hunt	2	10	0.01	DoG	9.6626	334
8	hunt	2	15	0.01	DoG	21.2580	333
9	hunt	2	10	0.01	LoG	9.8065	426
10	hunt	2	15	0.01	LoG	21.0052	385
11	hunt	5	10	0.01	DoG	11.8004	222
12	hunt	5	10	0.01	LoG	11.5975	131
13	ece	2	10	0.01	DoG	13.1386	350
14	ece	2	15	0.01	DoG	28.3683	356
15	ece	2	10	0.01	LoG	13.8723	947
16	ece	2	15	0.01	LoG	29.3962	932
17	ece	5	10	0.01	DoG	16.1879	381
18	ece	5	10	0.01	LoG	31.6313	529
19	circles	2	10	0.01	DoG	13.9422	182
20	circles	2	15	0.01	DoG	30.6848	183
21	circles	2	10	0.01	LoG	14.1321	234
22	circles	2	15	0.01	LoG	31.0029	258
23	circles	5	10	0.01	DoG	17.1214	166
24	circles	5	10	0.01	LoG	17.0934	157
25	hunt2	2	10	0.01	DoG	15.1515	351
26	hunt2	2	15	0.01	DoG	32.7918	356
27	hunt2	2	10	0.01	LoG	15.1435	384
28	hunt2	2	15	0.01	LoG	32.7505	372
29	hunt2	5	10	0.01	DoG	18.0589	263
30	hunt2	5	10	0.01	LoG	17.9040	177
31	solar	2	10	0.01	DoG	41.3488	429
32	solar	2	15	0.01	DoG	89.5480	417
33	solar	2	10	0.01	LoG	41.1300	735
34	solar	2	15	0.01	LoG	90.9282	468
35	solar	5	10	0.01	DoG	48.5028	385
36	solar	5	10	0.01	LoG	46.5999	111

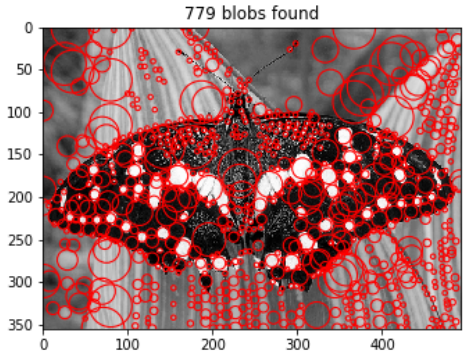
4 Analysis

From the conducted experiments we can observe that the blob detector is capable of detecting blobs correctly. Also, we can observe that the DoG filter can closely approximate the LoG filter since the results of using either one is close to each other. Moreover, we observe that using DoG filter can reduce the computation time in most cases. Another point is the scale size (number of scales to iterate in scale-space) increasing this number will generally increase the number of blobs detected and some larger details in the image will be detected as a result. The other factor that we studied is the initial scale (σ). A good initial scale value turned out to be equal to 2, However, increasing initial scale value will generally result in detecting larger details in the image and larger blobs are returned from the detector.

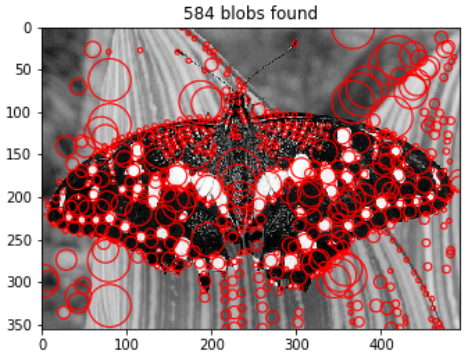
In order to study the effect of threshold on blob detection, another experiment is done on the 'butterfly' image. Same blob detector is applied on this image for different threshold values ([0.005, 0.01, 0.025, 0.05, 0.1, 0.5]). The results are depicted in figure 14. We can observe that increasing the threshold will decrease the number of blobs detected and only retain the ones with very high response to the filter. These blobs will be corresponding to most important ones of the image. As we increase the threshold value we can see that there is no detection when threshold = 0.5.

5 Conclusions

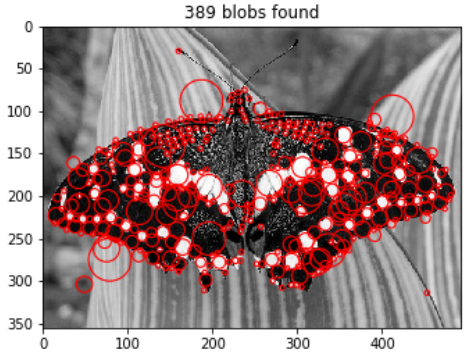
In this project we implemented blob detector in Python without using the in-built function. In order to create the blob detector, we defined multiple functions to handle the tasks of convolution, scale space creation, interpolation and image re-sizing, non-maximum suppression, etc. We applied the developed blob detector and tested that on the given test images as well as student provided images. Analysis of the results and comparison between different configurations are given.



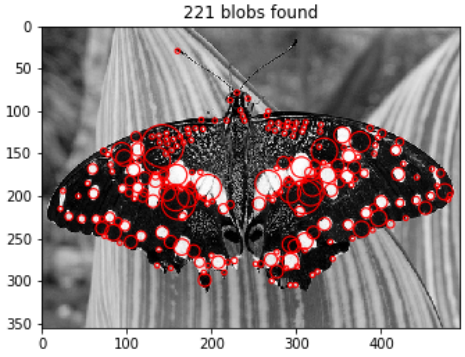
(a) σ= 2, threshold= 0.005, # of scales= 10, Filter= DoG



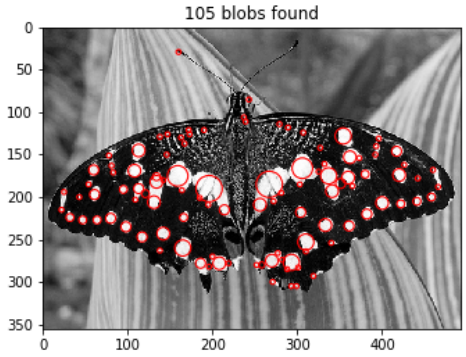
(b) σ= 2, threshold= 0.01, # of scales= 10, Filter= DoG



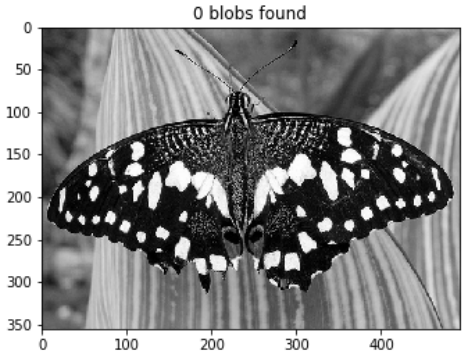
(c) σ= 2, threshold= 0.025, # of scales= 10, Filter= DoG



(d) σ= 2, threshold= 0.05, # of scales= 10, Filter= DoG



(e) σ= 2, threshold= 0.1, # of scales= 10, Filter= DoG



(f) σ= 2, threshold= 0.5, # of scales= 10, Filter= DoG

Figure 14: Effect of different threshold values on blob detector.

References

[1] *Blob Detector*, class notes for `ECE 558 NCSU fall 2018`.