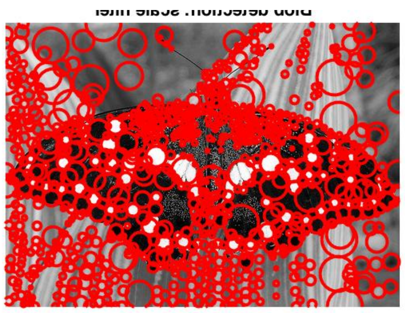
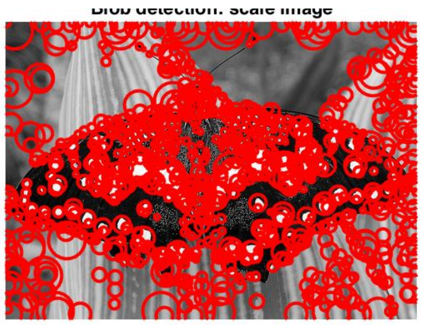
442 HW3

Part.1 Display

1.Butterfly

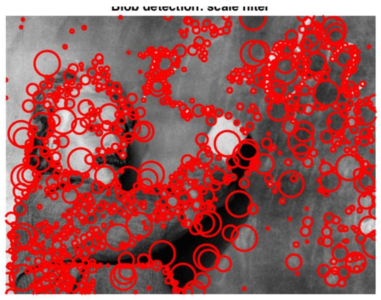


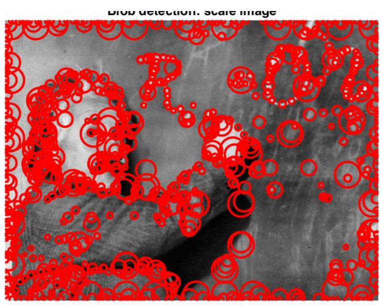


inefficient:1.983263s

efficient:1.10584s

2.Einstein

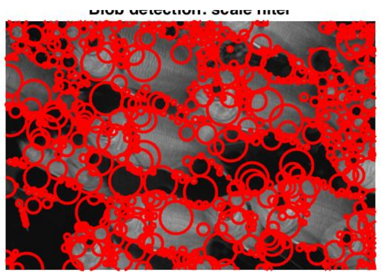


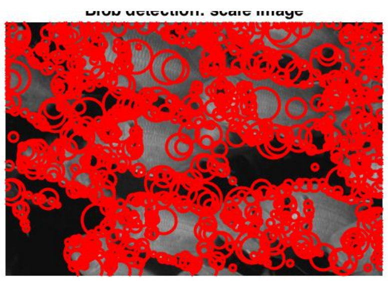


inefficient:3.132019s

efficient:1.801170s

3.Fish

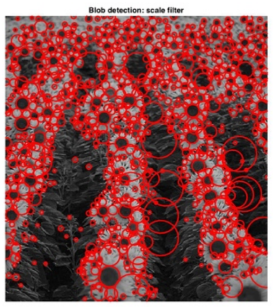


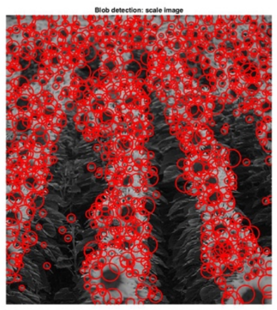


inefficient:1.851326s

efficient:1.005558s

4.Sunflowers:

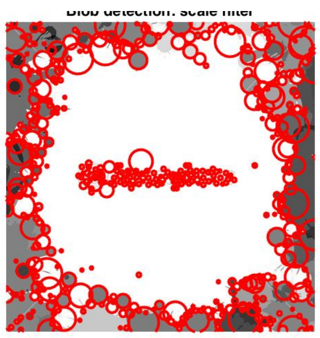




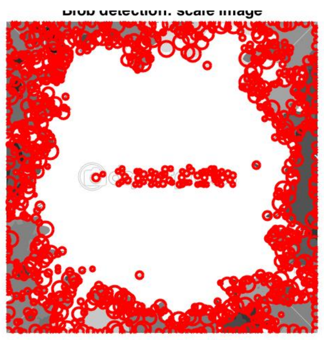
inefficient:1.269051s

efficient:1.171156s

5.Colorful

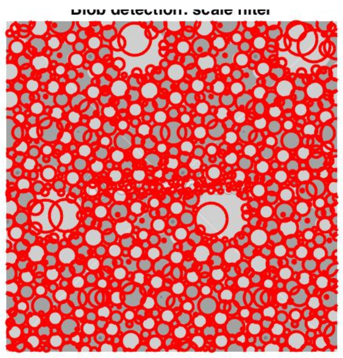


inefficient:2.165769s

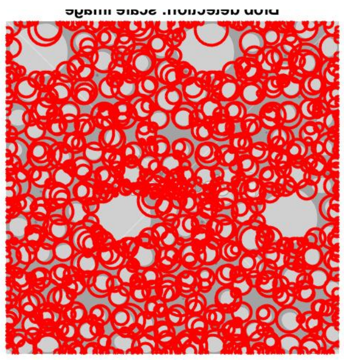


efficient:1.71156s

6.Yellow

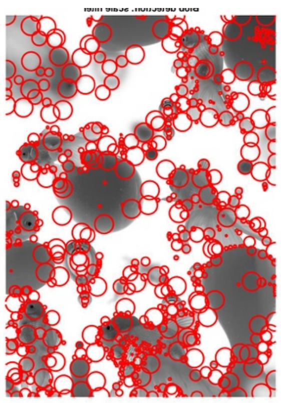


inefficient:2.150592s

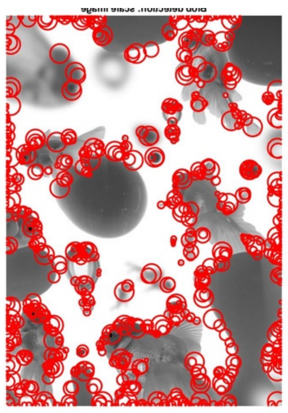


efficient:1.202828s

7.Goldfish

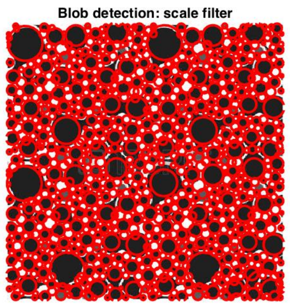


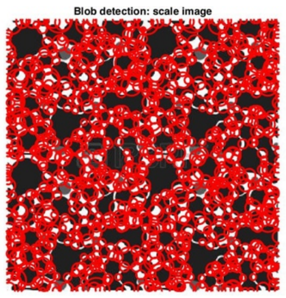
inefficient:6.159862s



efficient:3.170951s

8.Distort





inefficient:1.1391628s efficient:0.804955s

Part2 Implementation

1. For the filterScale method, I developed a for loop to implement different LoG to filter the picture. I used 3*sigma as the filter's half size. After that, I used abs to get the absolute value for the scale space. The next step is to implement the non-maximum suppression. I tried nlfilter and orfilter2 and decided on ortfilter2 because it performs better and faster than nlfilter. In that way, I get the correct result of space scale. The next thing is to find the max value for the same point in different levels. I first made a 2-Dimension matrix to record the max score for each point then I implement the ordfilter2 to this matrix in order to eliminate the blobs that are too close. And I used a somehow stupid for loop at each point in scale space to get the sigma and redius for each point. Finally I pushed them in to blobs to print out the result.
2. For the imageScale, I used resize method in matlab to generate a series of different size figures. And implementing the same LoG for them. After used nonmaximum suppression for the result, I rescaled them in to the same size of original pictures. The following steps are the same as filterScale method.

Part 3 Parameters

1. The first parameter selection is that which filter function we should use when we implement the nonmaximum suppression. I tried both the nlfilter and ordfilter2 method. For nlfilter, you need to develop a function that get the max element in the matrix for you. However ordfilter2 has its own parameter to do this for you. For efficiency, the time using nlfilter to calculate the butterfly.jpg takes me 20~30s per pass, but the ordfilter2 only costs me about 2 seconds. The efficiency is obvious thus I chosed ordfilter2 for my implementation.
2. Then you need to choose the parameter for ordfilter2, which represents from which range are you telling the local maximum of a point. And that's the exact parameter effects your efficiency, since most of the time will be spent here. For filterScale method, I used a dynamic range which is equal to sqrt(2)*sigma, the radius of the blobs we are detecting. I believe that makes sense since when we change our sigma ,the blobs are also changing. For imageScale, I chose a constant 5 for my implementation, which works best after several tests of different number.

Also, I need to choose the parameter for ordfilter2 after I done the nonmaximum suppression. We need to eliminate the blobs that are too close. I used 7 here thus you will never see to blobs which are in the distance of 7 pixels.

3. The third parameter you need to choose is the scale factor, which decides your blobs radius. If you choose it too big, then the small blobs will disappear and vice versa. I test different initial sigma and factor for these two implementations. Finally I used sigma as 1.5 and factor as 1.3, which displays the details best.

Part4. Bonus
No implementation.