Project03- Final

ECE558

Due: December 03, 2021

Teammates: Britanya Wright, Joshua Larsen, Zack Peppmeier

Table of Contents

Introduction

Blob detection is a common method used in computer vision to detect regions of a digital image that contain different properties, such as brightness or color, compared to the region's surroundings. In its simplest form, a blob is a region of an image in which properties are consistent. The most common method for blob detection is convolution and thus our algorithm begins with a convolution which will be described at a later point.

There are two different common classes of blob detectors, differential methods and methods based on local extrema. This project focuses on the second method based on local extrema and is based on finding the local maxima and minima of the function.

Blob detectors provide information about image regions not already provided by edge detection. When looking at the magnitude of the Laplacian signal response, an edge is shown by a ripple while a blob is represented by the superposition of two ripples, further expanding from edge detection. These regions of interest or blobs are also used to detect key points in an image and can be used with the application of object recognition.

Convolving an image with a "blob filter" at multiple scales provides a resulting scale space where you are able to look at the different scales and extrema of the filter response. When computing blob detection, it is important to find the characteristic scale of the blob by convolving it with Laplacians at several different scales and capturing the maximum response then normalizing the scale.

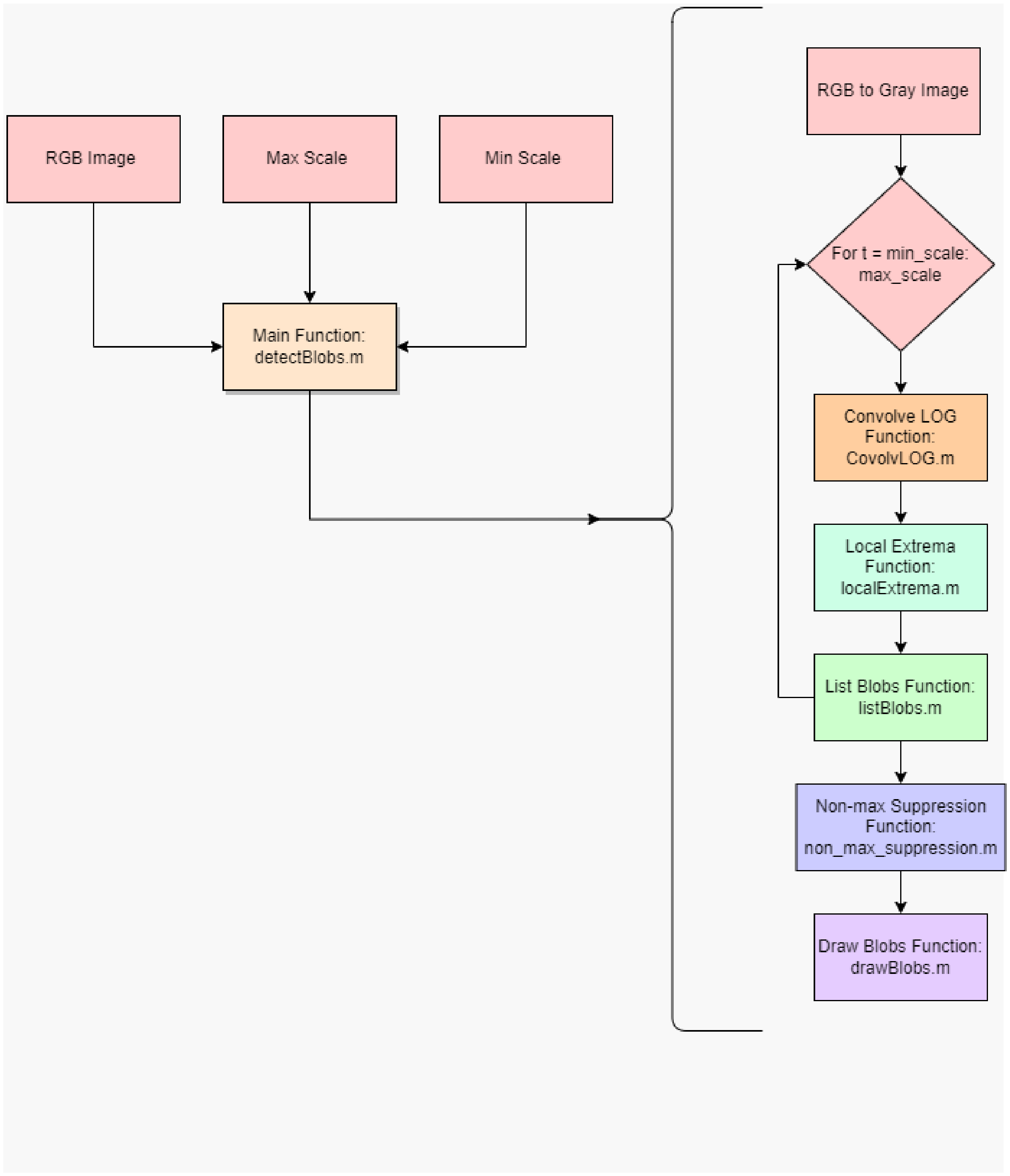
Methods and Algorithm

Overall Algorithm Approach

The main algorithm functions rather simply.

First, we convert the image to grayscale, making it a simple 2D matrix. Then, we create a loop with the minimum set to a minimum scale that was predefined and a maximum scale that was also predefined. At each scale, we convolve the 2D image with a Laplacian of Gaussian kernel that has been tailored to fit the scale we have chosen. From here, we identify local minimums and maximums of the array. These local extrema are then recorded in an array containing their locations and the scales at which they were found. This array then gets passed into a non-maximum suppression function that will remove duplicates and ensure that each blob only gets counted once across all scales. Finally, the image is passed into a final function that converts back to a color image where red circles are drawn with radius based on the scale that each blob was detected at.

Here is an overall chart of the flow of all functions:


RGB Image
Max Scale
Min Scale
Main Function: detectBlobs.m
RGB to Gray Image
For t = min_scale: max_scale
Convolve LOG Function: CovolvLOG.m
Local Extrema Function: localExtrema.m
List Blobs Function: listBlobs.m
Non-max Suppression Function: non_max_suppression.m
Draw Blobs Function: drawBlobs.m

Convolve LOG Function

Purpose

The convolveLOG.m function will convolve a given 2D array with a Laplacian of Gaussian array with a specified scale.

Implementation

The convolveLOG function calls a subfunction, LOGKernel.m, which performs the subfunction of developing the Laplacian of Gaussian array. This function returns the normalized Laplacian of Gaussian kernel. This is generated with the mathematical function

$$LoG(x, y) \;=\; -\,1/(\pi\sigma^4) \,*\, [1 - \;(x^2 \,+ y^2)/(2\sigma^2)] \,*\, e^{(-(x^2+y^2)/(2\sigma^2))}$$

In our implementation, we base the sigma value on the scale that was chosen.

The convolveLOG function also calls a subfunction, Convolution2.m, which performs the convolution on the given image, using the kernel function, LOGKernel.m, and one of four padding types. The four padding options given with the function are as follows: clip/zero padding, wrap around, copy edge and reflect across edge. The padding types are represented by 0, 1, 2, 3, respectively within the pad input value for the function, Convolution2.m.

Extrema Locating Function

Purpose

Generation of “truth table”. The truth table is an array of equal size to the original array passed to the function. It contains a +1 where a local maximum was found and a -1 if a local minimum was found at the specified scale.

Example) If the original array were passed in with a scale of one, it would return the following truth table.

Original Array

0.5	0.5	0.5	0.5	0.5
0.5	0.75	0.5	0.5	0.5
0.5	0.25	0.5	0.5	0.5
0.5	0.5	0.25	0.5	0.5

0.5	0.5	0.5	0.5	0.5

Truth Table

0	0	0	0	0
0	1	0	0	0
0	-1	0	0	0
0	0	-1	0	0
0	0	0	0	0

List Of Blobs Function

Purpose

The listBlobs.m function will take a truth table, and extract the locations of all the extrema. From this, it will append to an array of blobs. Blobs are defined as a simple struct containing blob.x,blob.y, blob.k, and blob.magnitude. The variables within the blob struct are defined as follows

blob.x	blob.y	blob.k	blob.magnitude
The x coordinate of the blob	They coordinate of the blob	The scale where the blob was found	A +1 or -1 signifying if the blob was a local maximum or local minimum

Non-Max-Suppression Function

Purpose

The non_max_suppression.m function performs two tasks.

1. ***Removal of non-max k*:** Remove all instances of k values less than the maximum value at locations Xi and Yi

   Example) In the table below, columns with the same Xi and Yi are colored in the same color. This function identifies these values. After identification, the K values of this Xi and Yi are compared. The largest K value is kept and all others deleted. Locations with the largest K are bolded below.

Original Blob array:

i	1	2	3	4	5	6
Xi	1	1	1	5	6	6
Yi	6	6	7	8	9	9
K	8	9	7	10	9	2

Updated Blob Array after removal of non-max k values:

i	1	2	3	4
Xi	1	1	5	6
Yi	6	7	8	9
K	9	7	10	9

2. ***Removal of overlapping neighboring blobs:*** A pixel location's (Xi, Yi) neighbor is defined in this function as a pixel location (Xj, Yj, Kj). The ranges of the neighboring X and Y values are defined by this: Xi - Ki/2 < Xj < Xi + Ki/2 and Yi - Ki/2 < Yj < Yi + Ki/2. The first thing this function does is search the blob list for neighbors of (Xi, Yi). If Ki/2 is greater than Kj/2, the neighboring location is deleted from the blob array.

Example) Let the location be Xi = 10, Yi = 5, Ki = 4.

Grid with all (Xi, Yi, Ki) and (Xj, Yj, Kj):

(Xj,Yj,Kj) = (8,3,Kj)	(Xj,Yj,Kj) = (8,4,Kj)	(Xj,Yj,Kj) = (8,5,Kj)	(Xj,Yj,Kj) = (8,6,Kj)	(Xj,Yj,Kj) = (8,7,Kj)
(Xj,Yj,Kj) = (9,3,Kj)	(Xj,Yj,Kj) = (9,4,Kj)	(Xj,Yj,Kj) = (9,5,Kj)	(Xj,Yj,Kj) = (9,6,Kj)	(Xj,Yj,Kj) = (9,7,Kj)
(Xj,Yj,Kj) = (10,3,Kj)	(Xj,Yj,Kj) = (10,4,Kj)	(Xi,Yi,Ki) = (10,5,4)	(Xj,Yj,Kj) = (10,6,Kj)	(Xj,Yj,Kj) = (10,7,Kj)
(Xj,Yj,Kj) = (11,3,Kj)	(Xj,Yj,Kj) = (11,4,Kj)	(Xj,Yj,Kj) = (11,5,Kj)	(Xj,Yj,Kj) = (11,6,Kj)	(Xj,Yj,Kj) = (11,7,Kj)
(Xj,Yj,Kj) = (12,3,Kj)	(Xj,Yj,Kj) = (12,4,Kj)	(Xj,Yj,Kj) = (12,5,Kj)	(Xj,Yj,Kj) = (12,6,Kj)	(Xj,Yj,Kj) = (12,7,Kj)

An example blob array is shown below. (Xi, Yi, Ki) is highlighted in green. All its neighbors are highlighted in orange. The neighbors that meet the requirement of Kj/2 <= Ki/2 are deleted from the blob array (bolded below).

Original Blob array:

i	1	2	3	4	5	6
X	Xi = 10	10	5	6	1	12
Y	Yi = 5	7	8	9	7	5
K	Ki = 4	8	10	9	7	2

Updated Blob Array after removal of overlapping neighbors:

i	1	2	3	4	5
X	Xi = 10	10	5	6	1
Y	Yi = 5	7	8	9	7
K	Ki = 4	8	10	9	7

Input to function: Output from the previous function is in the form: struct blob[X, Y, K] with 3 fields. Where X and Y are coordinates and K is the size of the blob to be drawn

Output to function: The function outputs struct blobF[X, Y, K] to be used to draw the circle on the final image

Implementation

Step 1: Turn struct to an array to make it easier for manipulation
Step 2: Loop to iterate through each column of the blob array

a. Find all duplicates of Xi andYi
b. Remove all k values that are not the maximum at that location

Step 3: Iterate through the neighboring x and y values to see which ones overlap

a. The ranges of the neighboring X and Y values are defined by this: Xi - Ki/2 < Xj < Xi + Ki/2 and Yi - Ki/2 < Yj < Yi + Ki/2
b. If Ki/2 is greater than Kj/2, the neighboring location is deleted from the blob array.
c. Adjust locations so the blobs are not self-deleting

Detect Blob Function

Purpose

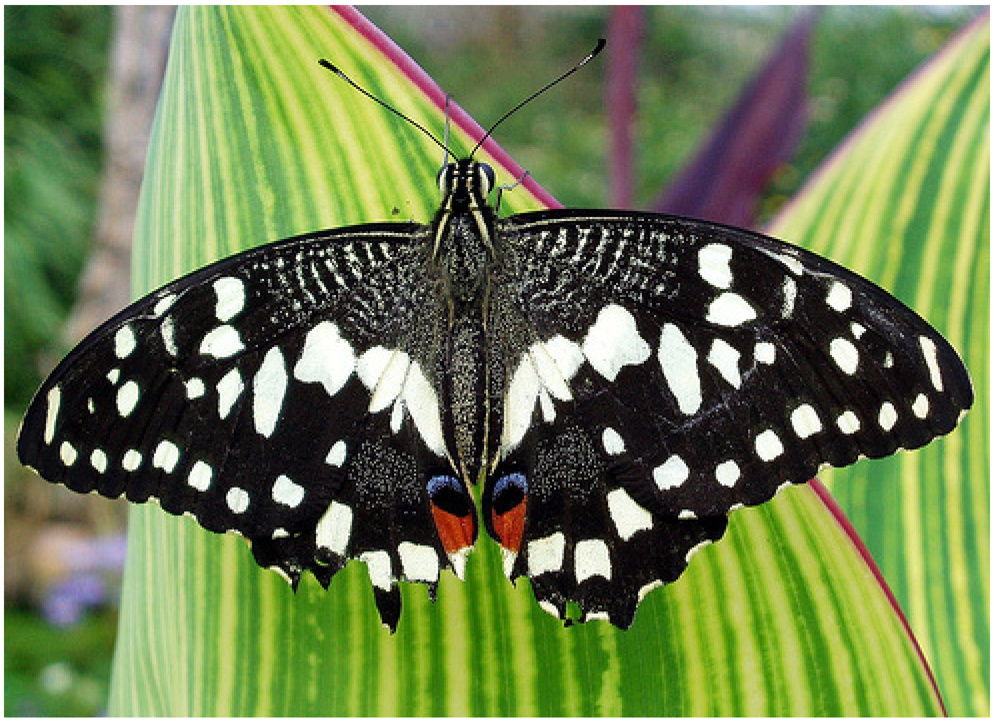
The detectBlobs.m function performs the overall algorithm as described earlier. It uses the constructed functions as building blocks to perform blob detection.

Implementation

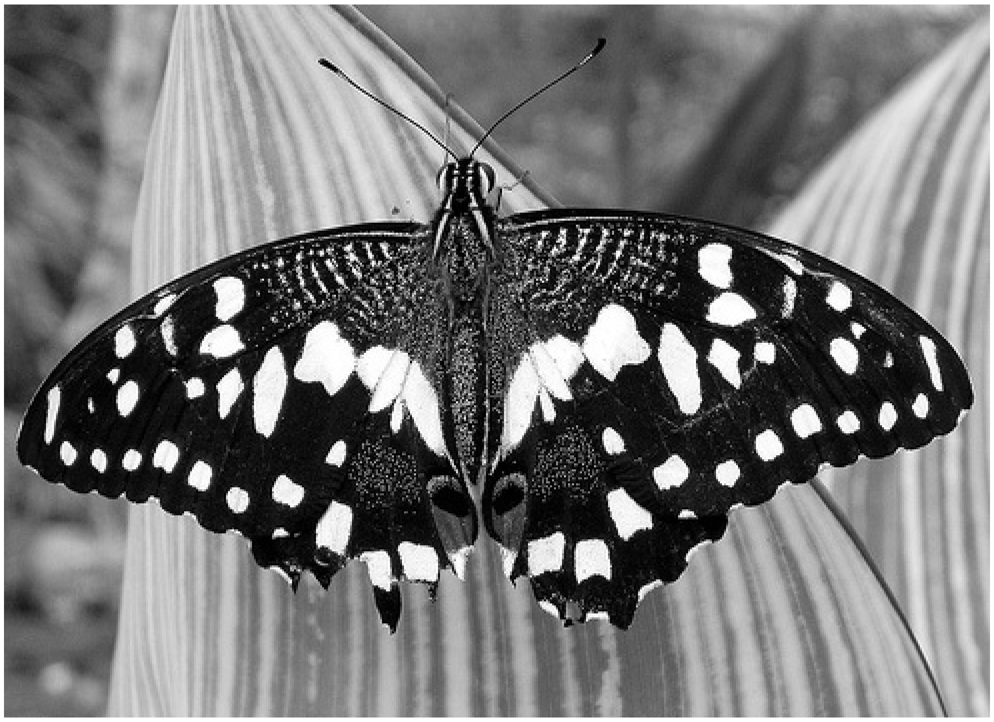
The function will take in an array, and repeatedly pass it into the convolveLOG function and the Extrema locating function from some minimum scale to a maximum scale. While this happens, it passes the truth tables to the list blobs generator, so as to keep a running list of all the blobs that have been found and every scale at which they have been found.
Finally, we perform non-max-suppression and draw the circles on the blobs that remain. This gives the final image that can be seen.

Results on Test Images

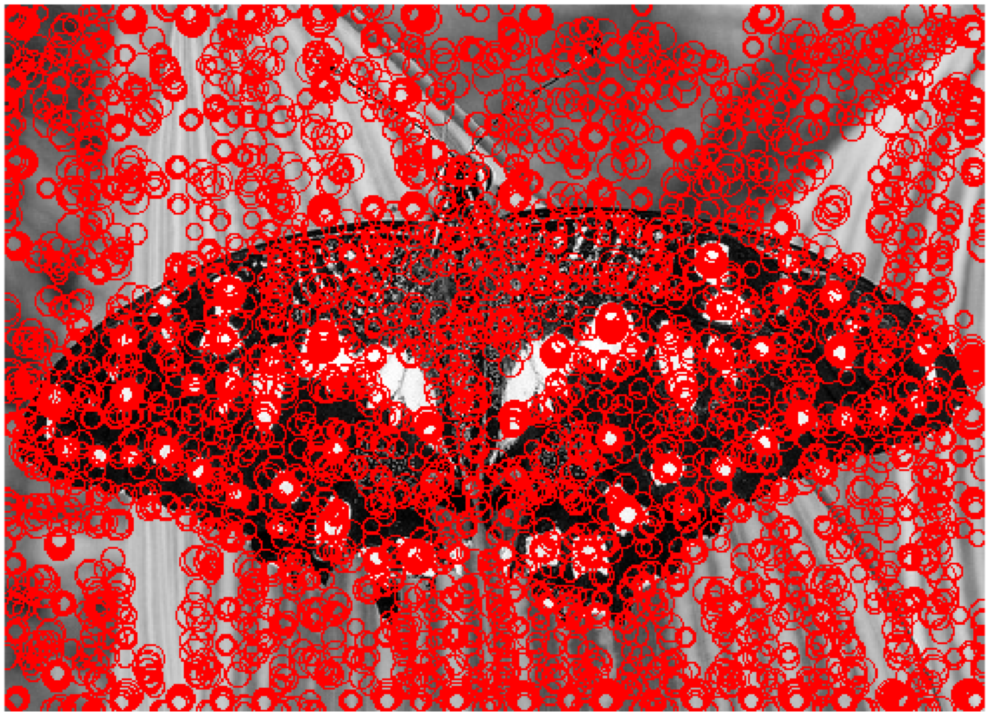
Butterfly Example



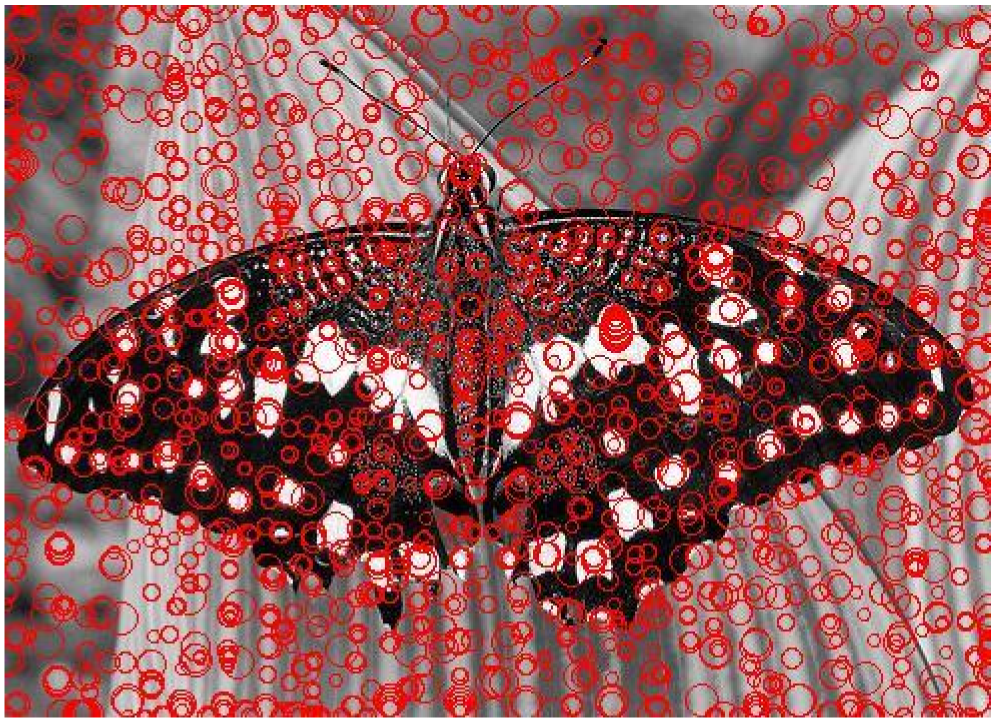
Original



Grayscale

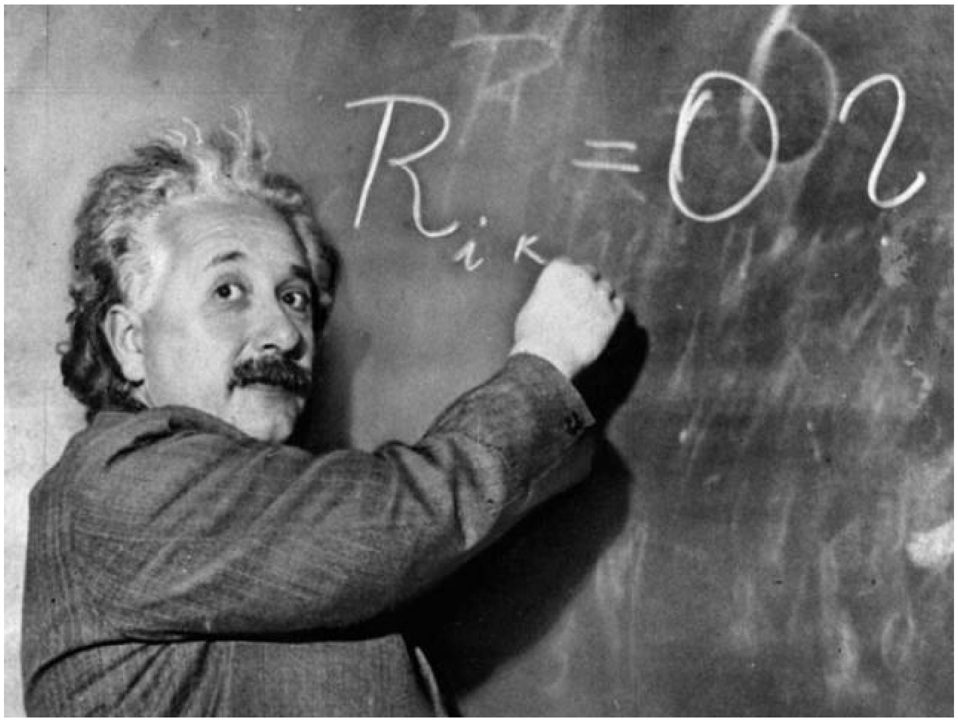


Before non-max Suppression

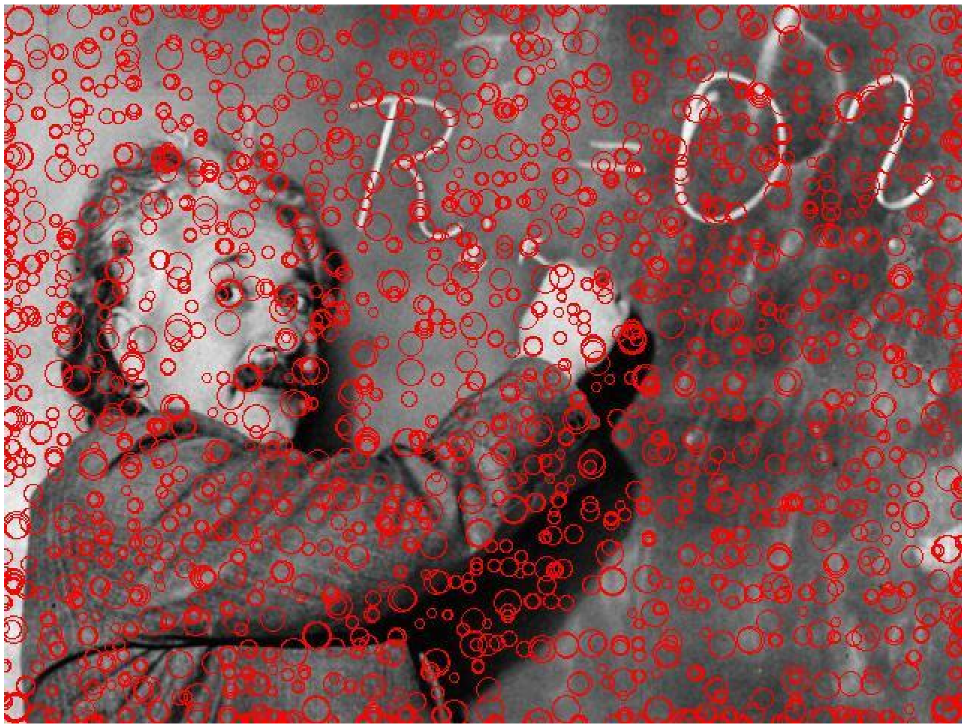


After non-max Suppression

Einstein Example



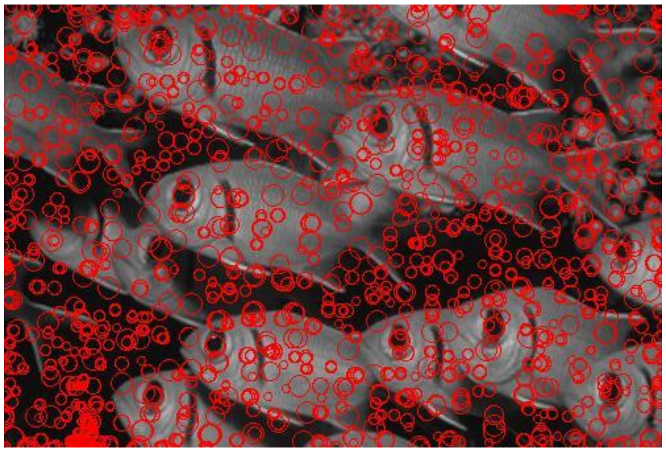
Original



After non-max Suppression

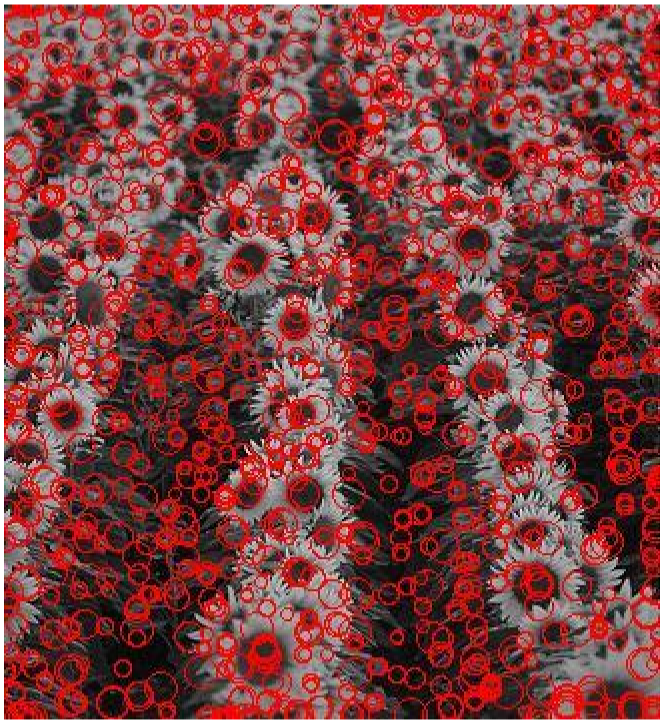
Fish Example

Original

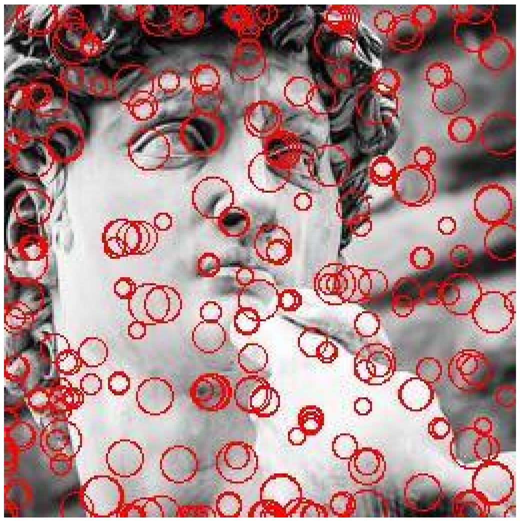

After non-max Suppression

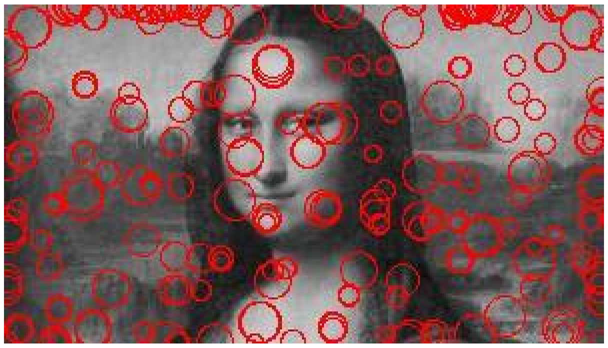
Flower Example

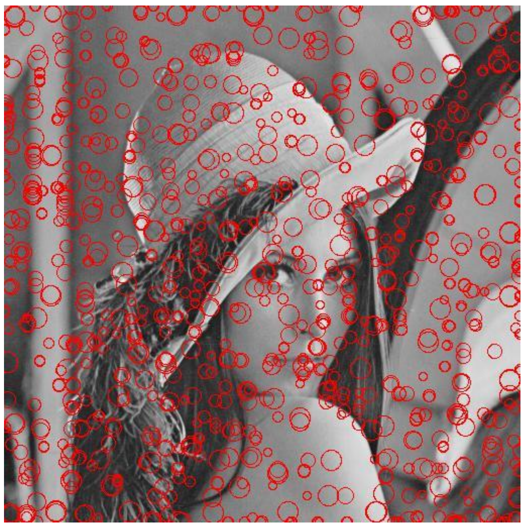
Original

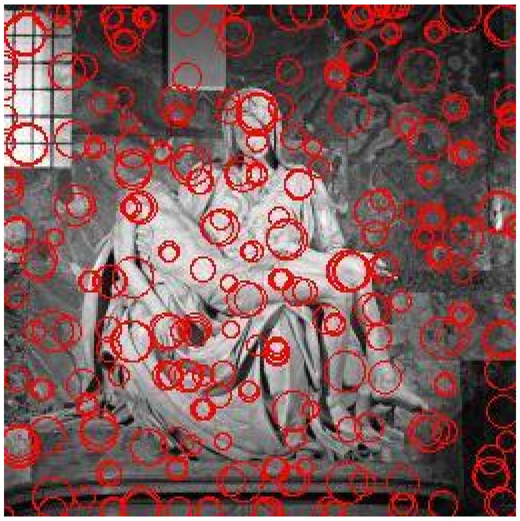

After non-max Suppression

Team’s Chosen Additional Test Images









Code

Here is a link to the Github file with the code.

References

[1] V. Mueller, “Non-maximum suppression,” *Medium*, 26-Oct-2021. [Online]. Available: https://towardsdatascience.com/non-maxima-suppression-139f7e00f0b5.

[2] S. K, “Non-maximum suppression (NMS),” *Medium*, 30-Apr-2021. [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c. [Accessed: 02-Dec-2021].