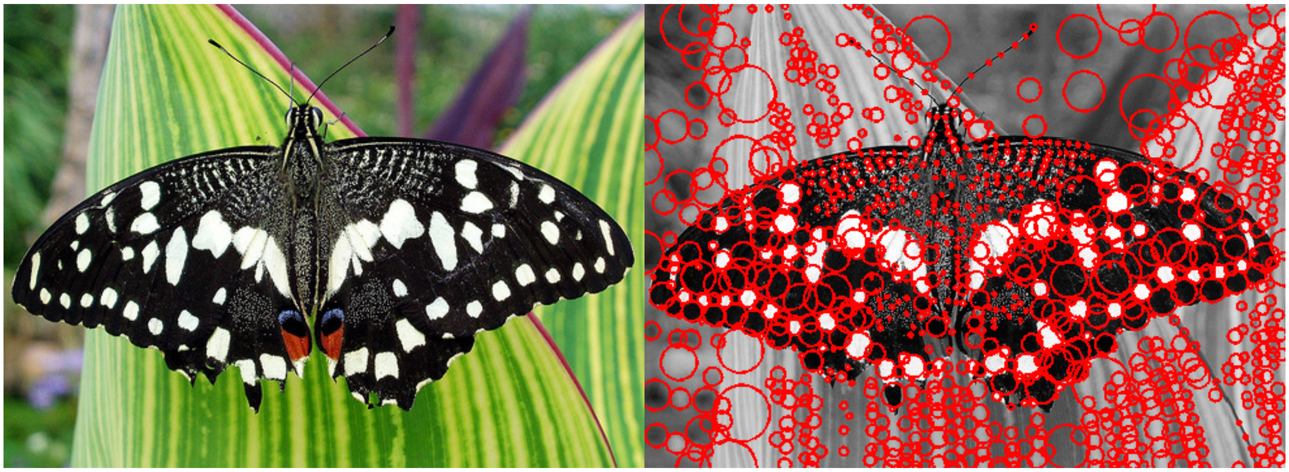
EECS 442: Computer Vision, Winter 2016

Homework 3: Scale-space blob detection

Due date: February 15 by 11:59PM



Basic project outline

1. Generate and apply a Laplacian of Gaussian filter at a variety of scales.
   You will develop two separate implementations.
   - First, try increasing the size of the filter to detect blobs at different sizes.
   - Next, write code to scale the image instead before applying the filter.
2. Perform nonmaximum suppression in each scale space.
3. Display resulting circles.

Test images

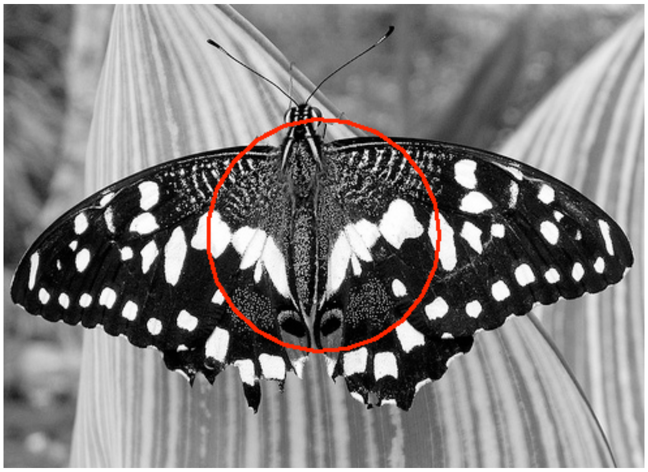
The data folder contains four images to test your code, and the output folder contains sample output images for your reference. Keep in mind, though, that your output may look different depending on your threshold, range of scales, and other implementation details. In addition to the images provided, also **run your code on at least four images of your own choosing**.

Running the code

Download the code and data and run `evalCode.m`. It runs a dummy implementation of the code and draws the blob (a circle in the center of the image). On running the code the ouput should be something like this (exact times will vary based on the machine):

```
>> evalCode
Elapsed time is 0.007286 seconds.
Elapsed time is 0.002676 seconds.
```

And two identical images:



Detailed instructions

Here are the key steps to implement the blob detector:

- Don't forget to convert images to grayscale (`rgb2gray` command) and double (`im2double`).

- For creating the Laplacian filter, use the `fspecial` function (check the options). Pay careful attention to setting the right filter mask size. **Hint:** Should the filter width be odd or even?

- It is relatively inefficient to repeatedly filter the image with a kernel of increasing size. Instead of increasing the kernel size by a factor of k, you should downsample the image by a factor 1/k. In that case, you will have to upsample the result or do some interpolation in order to find maxima in scale space. **For full credit, you should turn in both implementations: one that increases filter size `detectBlobsScaleFilter.m`, and one that downsamples the image `detectBlobsScaleImage.m`.** In your report, list the running times for both versions of the algorithm and discuss differences (if any) in the detector output. For timing, use `tic` and `toc` commands.

  Hint 1: think about whether you still need scale normalization when you downsample the image instead of increasing the scale of the filter.

  Hint 2: For the efficient implementation, pay attention to the interpolation method you're using to upsample the filtered images (see the options of the `imresize` function). What kind of interpolation works best?

- You have to choose the initial scale, the factor k by which the scale is multiplied each time, and the number of levels in the scale space. I typically set the initial scale to 2, and use 10 to 15 levels in the scale pyramid. The multiplication factor should depend on the largest scale at which you want regions to be detected.

- You may want to use a three-dimensional array to represent your scale space. It would be declared as follows:

  ```
  scaleSpace = zeros(h,w,n); % [h,w] - dimensions of image, n - number of levels in scale space
  ```

  Then `scaleSpace(:,:,i)` would give you the ith level of the scale space. Alternatively, if you are storing different levels of the scale pyramid at different resolutions, you may want to use a cell array, where each "slot" can accommodate a different data type or a matrix of different dimensions. Here is how you would use it:

```
scaleSpace = cell(n,1); %creates a cell array with n "slots"
scaleSpace{i} = myMatrix; % store a matrix at level i
```

- To perform nonmaximum suppression in scale space, you should first do nonmaximum suppression in each 2D slice separately. For this, you may find functions `nlfilter`, `colfilt` or `ordfilt2` useful. Play around with these functions, and try to find the one that works the fastest. To extract the final nonzero values (corresponding to detected regions), you may want to use the `find` function.

- You also have to set a threshold above which the Laplacian response will count as a detection. You should play around with different values and choose one you like best.

- To display the detected regions as circles, you can use the `drawBlobs.m`. You should display the highest scoring 1000 detected blobs for each image. If your code returns fewer than 1000 blobs then you may have to lower your threshold. **Hint:** Don't forget that there is a multiplication factor that relates the scale at which a region is detected to the radius of the circle that most closely "approximates" the region.

For extra credit

- Implement the difference-of-Gaussian pyramid as mentioned in class and described in David Lowe's paper. Compare the results and the running time to the direct Laplacian implementation.

- Implement the affine adaptation step to turn circular blobs into ellipses as shown in the lecture (just one iteration is sufficient). The selection of the correct window function is essential here. You should use a Gaussian window that is a factor of 1.5 or 2 larger than the characteristic scale of the blob. Note that the lecture slides show how to find the relative shape of the second moment ellipse, but not the absolute scale (i.e., the axis lengths are defined up to some arbitrary constant multiplier). A good choice for the absolute scale is to set the sum of the major and minor axis half-lengths to the diameter of the corresponding Laplacian circle. To display the resulting ellipses, you should modify the circle-drawing function or look for a better function in the MATLAB documentation or on the Internet.

- The Laplacian has a strong response not only at blobs, but also along edges. However, edge points are not "repeatable". So, implement an additional thresholding step that computes the Harris response at each detected Laplacian region and rejects the regions that have only one dominant gradient orientation (i.e., regions along edges). If you have implemented the affine adaptation step, these would be the regions whose characteristic ellipses are close to being degenerate (i.e., one of the eigenvalues is close to zero). Show both "before" and "after" detection results.

Helpful resources

- Blob detection on Wikipedia.
- D. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60 (2), pp. 91-110, 2004. This paper contains details about efficient implementation of a Difference-of-Gaussians scale space.

- T. Lindeberg, "Feature detection with automatic scale selection," International Journal of Computer Vision 30 (2), pp. 77-116, 1998. This is advanced reading for those of you who are *really* interested in the gory mathematical details.

Grading checklist

As before, you must turn in both your report and your code. Your report will be graded based on the following items:

1. The output of your circle detector on all the images (four provided and four of your own choice), together with running times for both the "efficient" and the "inefficient" implementation.
2. An explanation of any "interesting" implementation choices that you made.
3. An explanation of parameter values you have tried and which ones you found to be optimal.
4. Discussion and results of any extensions or bonus features you have implemented.

Instructions for submitting the homework

On CTools submit the following files:

- detectBlobsScaleFilter.m
- detectBlobsScaleImage.m
- report.pdf

Also include additional code (e.g. for extra credit) and explain it in the report what each file does.

Acknowledgements

This homework was developed by Subhransu Maji at UMass, Amherst and is based on a similar one made by Lana Lazebnik at UIUC.