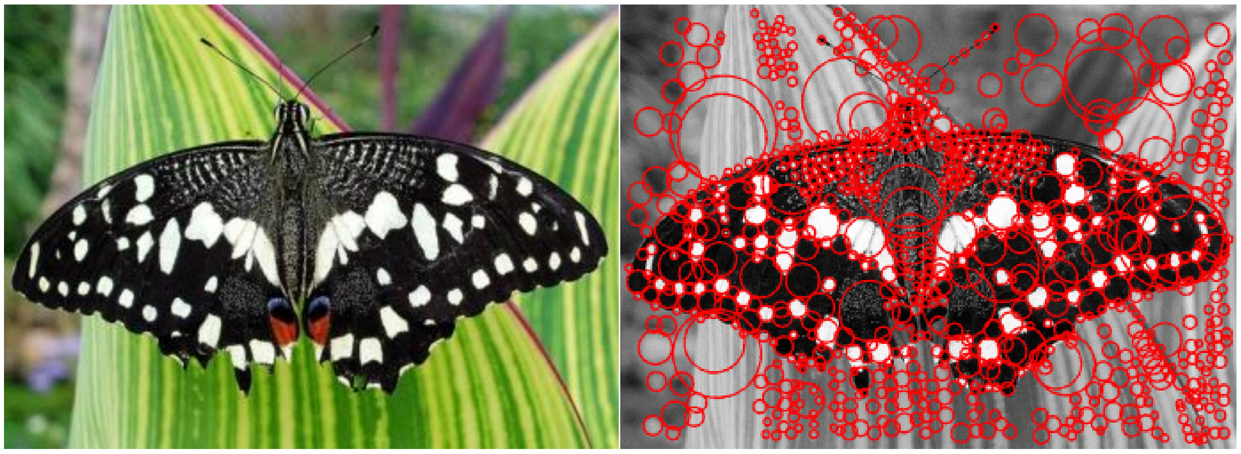
Mini Project 2: Scale-space blob detection

The goal of the assignment is to implement a Laplacian blob detector as discussed in the lecture 6.



Algorithm outline

1. Generate a Laplacian of Gaussian filter.
2. Build a Laplacian scale space, starting with some initial scale and going for n iterations:
    1. Filter image with scale-normalized Laplacian at current scale.
    2. Save square of Laplacian response for current level of scale space.
    3. Increase scale by a factor k.
3. Perform nonmaximum suppression in scale space.
4. Display resulting circles at their characteristic scales.

Test images

Here are four images to test your code, and 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**.

Detailed instructions

- 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, and one that downsamples the image.** 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:

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

  Then `scale_space(:,:,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:

  ```
  scale_space = cell(n,1); %creates a cell array with n "slots"
  scale_space{i} = my_matrix; % 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 on the squared Laplacian response above which to report region detections. You should play around with different values and choose one you like best.

- To display the detected regions as circles, you can use this function (or feel free to search for a suitable MATLAB function or write your own). **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 my 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, recall from the class lecture that 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

- Sample Harris detector code.
- 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.
- Some notes on scale selection and affine adaptation from Svetlana Lazebnik's Ph.D. thesis. You may find these helpful if you are implementing the bonus affine adaptation step.
- 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.

What to hand in

You should prepare a (very brief) report including the following:

- The output of your circle detector on all the images, together with running times for the "efficient" and the "inefficient" implementation.
- An explanation of any "interesting" implementation choices that you made.
- An explanation of parameter values you have tried and which ones you found to be optimal.
- Discussion and results of any extensions or bonus features you have implemented.

Similarly to last time, your submission should consist of a PDF report file entitled **lastname_firstname_a2.pdf** and a zip file containing your MATLAB code and additional images, entitled **lastname_firstname_a2.zip**. As last time, the lateness penalty is 25% per day. **Late submissions will incur a penalty of 25% per day. You will likewise be penalized for not following directions.** If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must let me know **ahead of the due date**.

Academic integrity policy

Feel free to discuss the assignment with each other and to consult the Web, but do all your own coding and writing. Be sure to acknowledge all outside sources in your report.