Your Name: Yen-Ting Liu
Your NetId: ytliu2

Part 1 Hybrid Images:

You will provide the following for ***3 different examples*** (1 provided pair, 2 pairs of your own):

- two input images
- two *filtered* input images
- two generated hybrid image (at different resolutions, similar to images C and D above)
- two σ values (one for each filter)

You will provide the following as further discussion overall:

- Explanation of how you chose the σ values
- Discussion of how successful your examples are + any interesting observations

Example 1:

Input (Hi-Freq)
$\sigma = 15$

Hybrid

Input (Lo-Freq)
$\sigma = 3$

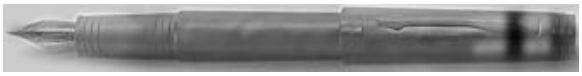
1/5 Hybrid

Example 2:

Input (Hi-Freq)

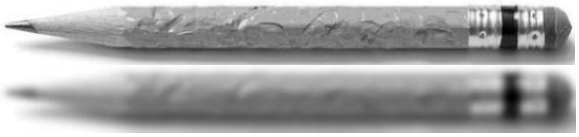
$\sigma = 5$

Hybrid



Input (Lo-Freq)

$\sigma = 7$



1/3 Hybrid

Example 3:

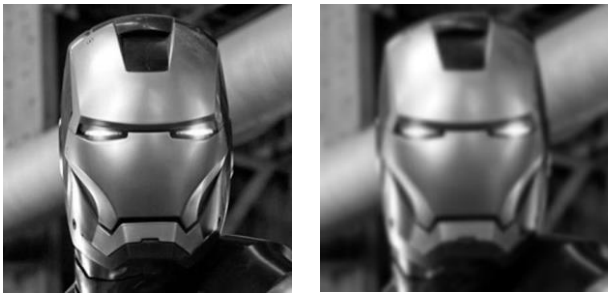
Input (Hi-Freq)

$\sigma = 11$

Hybrid

Input (Lo-Freq)

$\sigma = 3$



1/3 Hybrid

Discussion:

Since the problem statement writes optimal (sigma) values can vary from image to image, above examples uses empirically determined sigma for hi/lo-pass filter. Detailed comparison is outlined here.

After creating filtered images using different sigma $\sigma = (3,5,7,9,11)$,

High pass filtered

Low pass filtered

We can trivially observe that information gradually degrades for low pass filtered image, where as for high pass filtered images, edges are gradually enhanced. Therefore, when combining images that go through high pass and low pass filter independently,

- When standing far away, or zoomed out, high pass information is missing, leaving only low pass filtered image in view.
- In close up view, low pass filtered image is now a background for the high pass filtered edges, therefore, we need high pass image to remain high SNR to perform well.

I conclude that

- low pass filter sigma should keep as low as possible (without having significant feature that blocks edges from high pass filtered image)
  Example 2 violates this, causing the dark band to remain visually significant.
- while high pass filter sigma should keep as large as possible to enhance edges (without having significant wide-spread contrast differences that leave marks in zoom out).
  Example 3 violates this, Stark's hair is still visible in close up view.

Part 2 Scale-Space Blob Detection:

You will provide the following for ***8 different examples*** (4 provided, 4 of your own):

- original image
- output of your circle detector on the image
- running time for the "efficient" implementation on this image
- running time for the "inefficient" implementation on this image

You will provide the following as further discussion overall:

- Explanation of any "interesting" implementation choices that you made.
- Discussion of optimal parameter values or ones you have tried

In following examples, blob detections all use

- Minimum sigma 2
- Maximum sigma 50
- 32 pyramid levels
- Threshold 0.01
- Execute 10 times and take the runtime average of function `blob_detection`

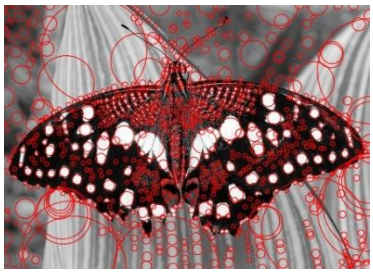
The inefficient implementation is named as *sigma*, the efficient one is named as *resize*. Input image resolution is listed below the header of each example.

Parameters for these examples are not tuned, in order to have a general idea how the detector perform across different input. Tuned results are provided in the discussion.

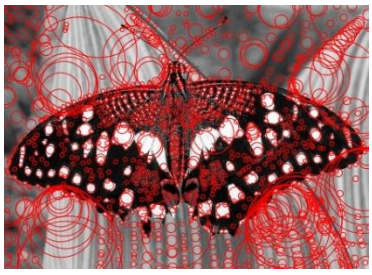
Example 1:

Original
493 x 356

Circles (sigma)
2.25 s

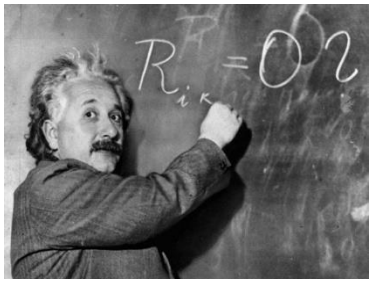


Circles (resize)
0.39 s



Example 2:

Original
640 x 480



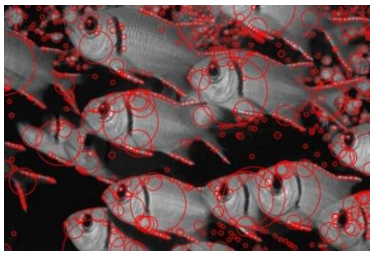
Circles (sigma)
4.17 s

Circles (resize)
1.05 s

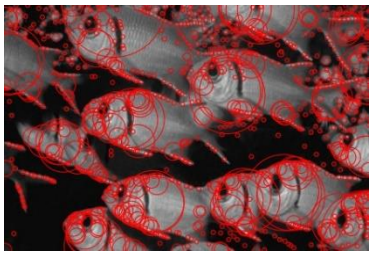
Example 3:

Original
500 x 335

Circles (sigma)
2.40 s



Circles (resize)
0.49 s

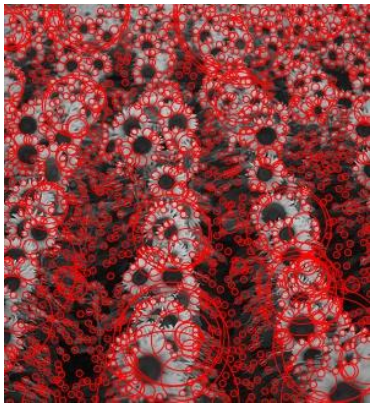


Example 4:

Original
328 x 357

Circles (sigma)
1.51 s

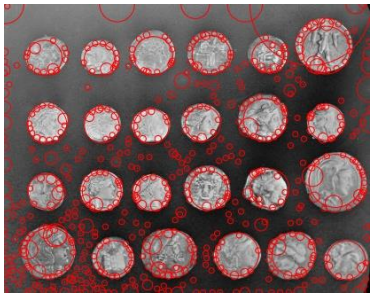
Circles (resize)
0.28 s



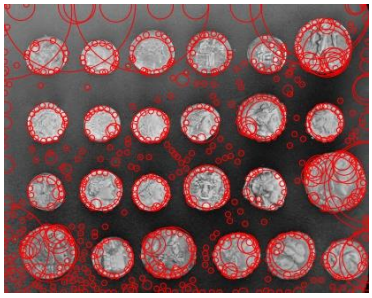
Example 5:

Original
384 x 303

Circles (sigma)
1.78 s



Circles (resize)
0.31 s

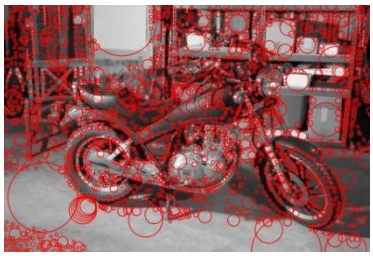


“British Museum, London, England”. Brooklyn Museum, Goodyear [from Pompeii].

Example 6:

Original
741 x 500

Circles (sigma)
4.96 s



Circles (resize)
0.90 s

Stereo motorcycle. Middlebury 2014 stereo benchmark.

Example 7:

Original
512 x 512

Circles (sigma)
3.53 s

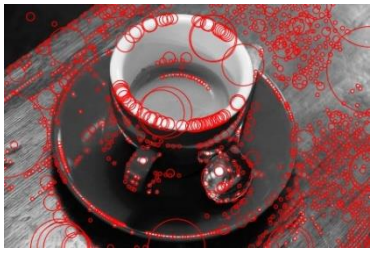
Circles (resize)
0.79 s

Photograph of Eileen Collins. From the NASA Great Images database.

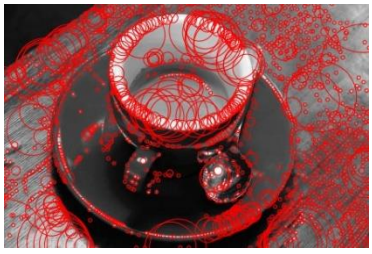
Example 8:

Original
600 x 400

Circles (sigma)
3.19 s



Circles (resize)
0.54 s



Pikolo Espresso Bar. CC0 by photographer Rachel Michetti.

Discussion:

- Discussion of optimal parameter values or ones you have tried

Implementation choices

- To maximize the benefit from downscaling the image, I repeatedly resize instead of starting from the original size, see the following snippet (`shape` is the next-smaller shape in each layer)

```
shape0 = im.shape
for _ in range(max_layer):
    im = resize(im, shape0)
    yield im
    …
    im = resize(im, shape)
```

  This may cause sampling error to accumulate with each layer.

- When downsizing the image, the image is not isotropically downsized,

```
np.ceil(np.array(im.shape) / downscale)
```

  This causes the effective downscale to change slightly across each layer. Therefore, effective sigma is slightly different.

- Since the 6σ requirement for kernel is repeatedly mentioned in Piazza, to avoid penalty, I explicitly uses the truncate parameter

```
gaussian_laplace(…, truncate=6)
```

  to make sure the kernel is indeed cutoff at 6σ.

- I output each Laplacian pyramid layer with sigma multiplied,

```
yield im * sigma**2
```

  so each scale is directly comparable during blob detection.

- This won't directly impact time complexity, but consider that we only work with each layer independently, I utilize yield keyword

```
for _ in range(max_layer):
    …
    yield im * sigma**2, sigma
```

  to have some flexibility at optimizing memory usage downstream.

- From Piazza discussion thread @135, it is my understanding that reference implementation prefers create scale threshold (`scale_max`) by non-maximal suppressed value (`pyramid`). Therefore,

```
pyramid = rank_filter(
    pyramid0,
    -1,
    footprint=np.ones((3, ) * (pyramid0.ndim-1) + (1, )
)
scale_max = np.max(pyramid, axis=-1)
```

  is later compare with the original pyramid value (`pyramid0`),

```
pyramid[…, i] = pyramid0 * (pyramid0 == scale_max[…, np.newaxis])
```

- *(Cont.)* It is my understanding that reference design uses *non-NMS* value to threshold over NMS'ed max , therefore

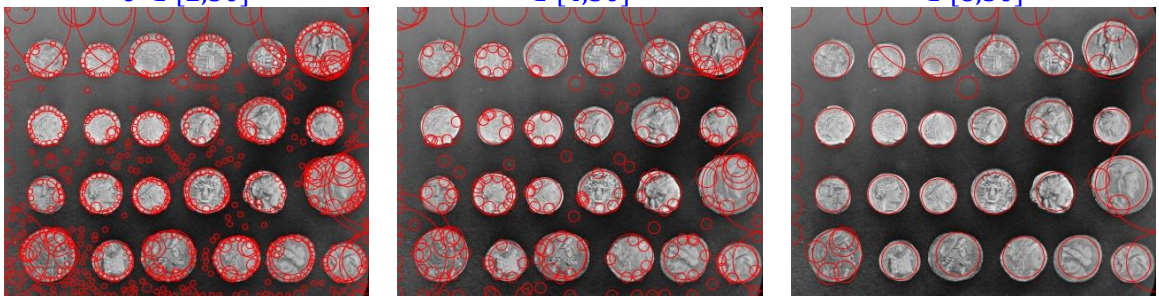
```
pyramid[…, i] = pyramid0 * (pyramid0 == scale_max[..., np.newaxis])
maxima = np.where(pyramid > threshold)
```

  aka, we only output detected circles for (sufficiently) significant scale.

Optimal parameters

Without additional filtering strategy to remove spurious edge responses (and to prove that I really have implemented NMS), we simply adjust the minimum sigma

$\sigma \in [2,50]$ $\in [4,50]$ $\in [8,50]$



From here, we can see that choosing a sigma range close to feature we want is important in this naïve implementation.

Bonus:

Hybrid Images Extra Credit

- Discussion and results of any extensions or bonus features you have implemented for Hybrid Images

Color image fusion

Sigmas are kept the same with previous example to allow comparison.

Input (Hi-Freq)

$\sigma = 15$

Hybrid

Input (Lo-Freq)

$\sigma = 3$

1/5 Hybrid

The algorithm is briefly described as

1. Convert from RGB to CIE Lab
2. Perform the same hi/lo pass filter on brightness parameter of these images
3. The actual merge
    1) Merge brightness parameter directly

$$Y = Y_1 + Y_2$$

    2) Merge chromaticity coordinates (hue H and saturation S) using normalized brightness parameters ($\overline{Y}_*$), using hue as an example

$$H = H_1 \frac{\overline{Y_1}}{\overline{Y_1} + \overline{Y_2}} + H_2 \frac{\overline{Y_2}}{\overline{Y_1} + \overline{Y_2}}$$

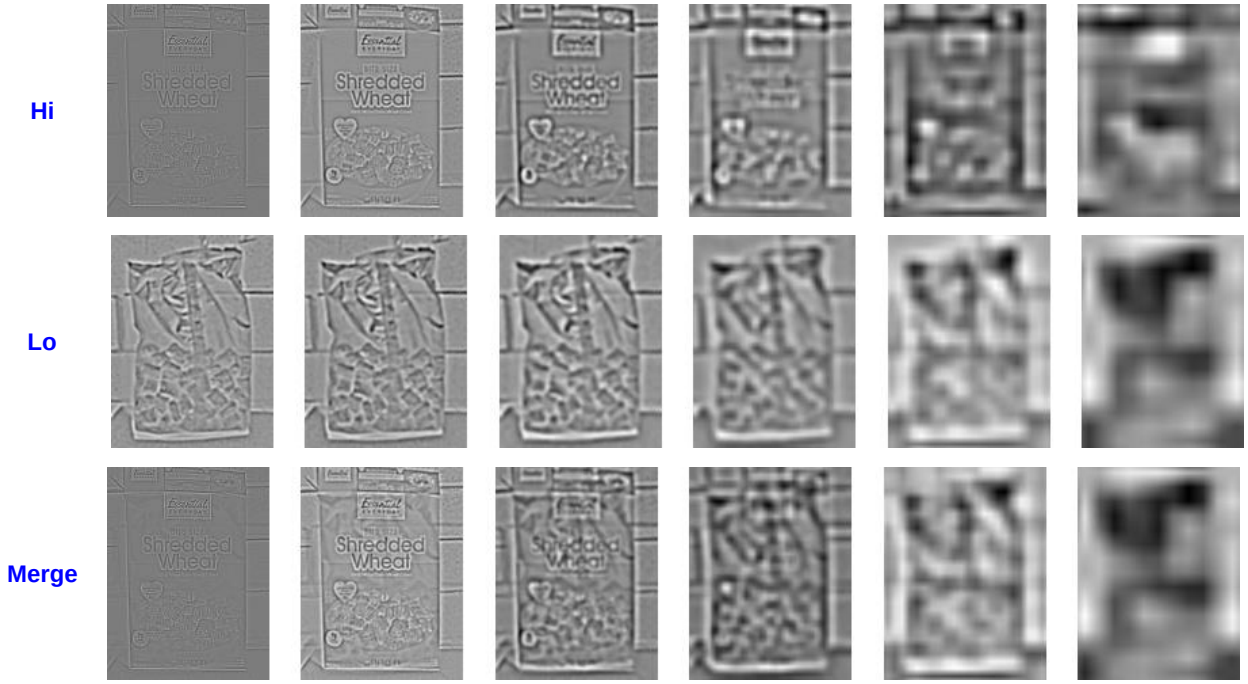
where

$$\overline{Y_*} = \frac{Y_* - \min Y_*}{\max Y_* - \min Y_*}$$

The Lab color space allows linear mixing of colors according to their corresponding brightness parameter.

Laplacian pyramid of fused image

Since part 2 already build a Laplacian pyramid generator, this is another low hanging fruit to write up. Continue with the same cereal box example,



In above observation, sigma goes from 1 to 32, from left to right. We can easily see that high frequency component is more dominant at high resolution, as resolution degrade (higher sigma), the pyramid level resembles more as the level from the low pass filtered image.

Blob-Detection Extra Credit

- Discussion and results of any extensions or bonus features you have implemented for Blob-Detection

Using Harris response to filter out false positive

Using the same set of sigma range and threshold, I calculate Harris response by first calculate the structure tensor for each scale

$$S_w[p] = \begin{bmatrix} \sum_r w[r](I_x[p-r])^2 & \sum_r w[r]\, I_x[p-r] I_y[p-r] \\ \sum_r w[r]\, I_x[p-r] I_y[p-r] & \sum_r w[r]\big(I_y[p-r]\big)^2 \end{bmatrix}$$

where w is the window. This gives Harris response

$$R = \det S_w - \alpha\, (\mathrm{tr} S_w)^2$$

Since edges would have $R < 0$ in this formulation, we can easily mask the Laplacian pyramid (`pyramid`) again with the Harris response (`harris`)

```
pyramid *= harris > 0
```

to remove false positives around edges. In the following example, we can see that small blobs close to the body are still there, but blobs of same size around wing edges are removed.

Naïve | **after Harris response filter**