CSCI 5561 HW 3 An An Yu

1.

a. Algorithm

n - level Pyramid construction:

Build Laplacian pyramid by recursively blurring the image with a fixed gaussian filter, record the prediction error between the blurred result and the image, then downsample the blurred result by a factor of 2 and set it as the starting image for the next iteration. The n recorded prediction error matrices and the remaining image makes up our Laplacian pyramid. Each level contains the frequencies not captured in the Gaussian pyramid.

reconstruction:

We take the top level of the pyramid (the remaining image) as our first image, then iteratively upsample the image by a factor of 2, apply sinc interpolation, add the result with the next level of the pyramid and start the next iteration with it.

b. Results

Figure 1: Original image

Figure 2: Reconstructed image

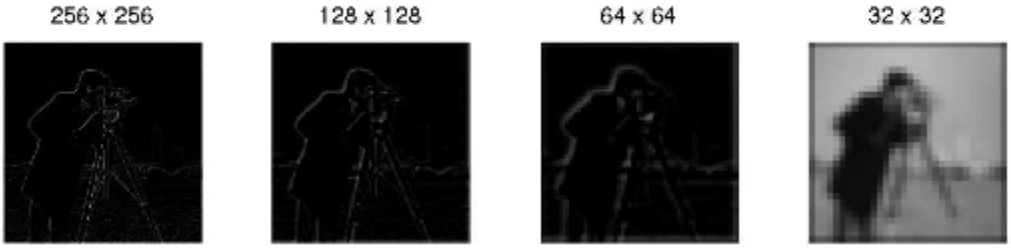


Figure 3: Laplacian pyramid, levels 3 (leftmost) to 1(second on the right)

Reconstruction:

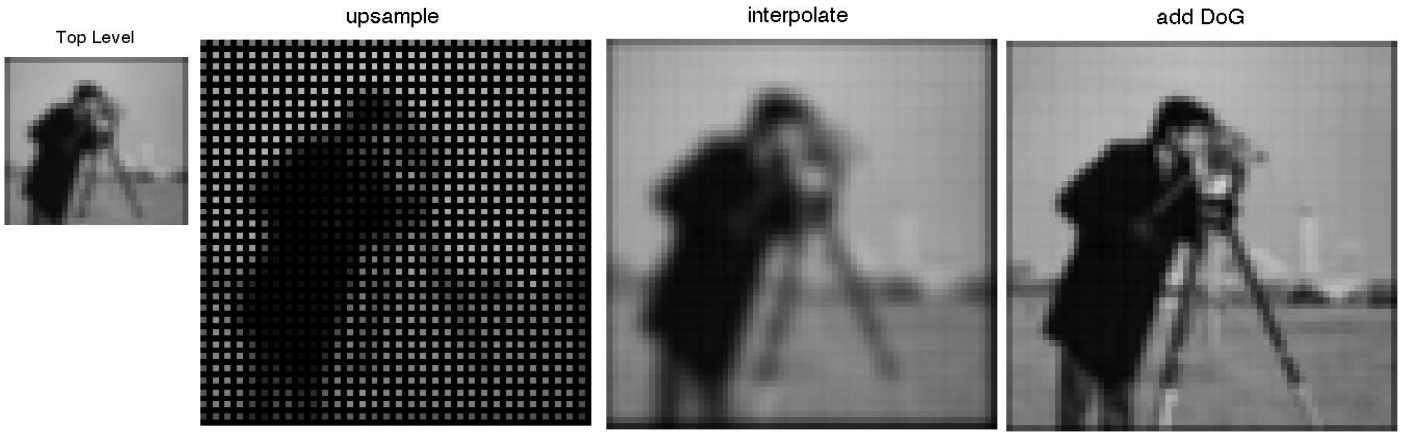


Figure 4 : first iteration of reconstruction (from left to right)

2.

a. Algorithm

First, we smooth the image with a Gaussian filter to eliminate noise. Then, we calculate gradient intensity with Sobel operators, this would give us thick edges. Once the gradient magnitude and directions of the image have been computed, we perform non-maximum suppression on each gradient in four directions (0, 45 , 90 and 135 degrees), in which gradients will be suppressed if they are not the local maxima in their direction.

After non-maximum suppression, our image contains thinner but broken edges, so we do hysteresis to track down and highlight all important edges (points whose gradient magnitude are larger than the low threshold), then kill the rest. We first set two thresholds H and L. For each pixel, if its gradient magnitude is larger than H, then it is marked a strong point and is an edge; if it is between H and L, then it is marked a weak point, weak points need to be connected to a strong point in order to be counted as an edge; if it is lower than L, then it is set to zero.

I implemented hysteresis by doing BFS from each strong point (setting one them as root for each run), so that all weak points that are connected to a strong point will be reached, and all weak points that are not connected to strong points will be killed.

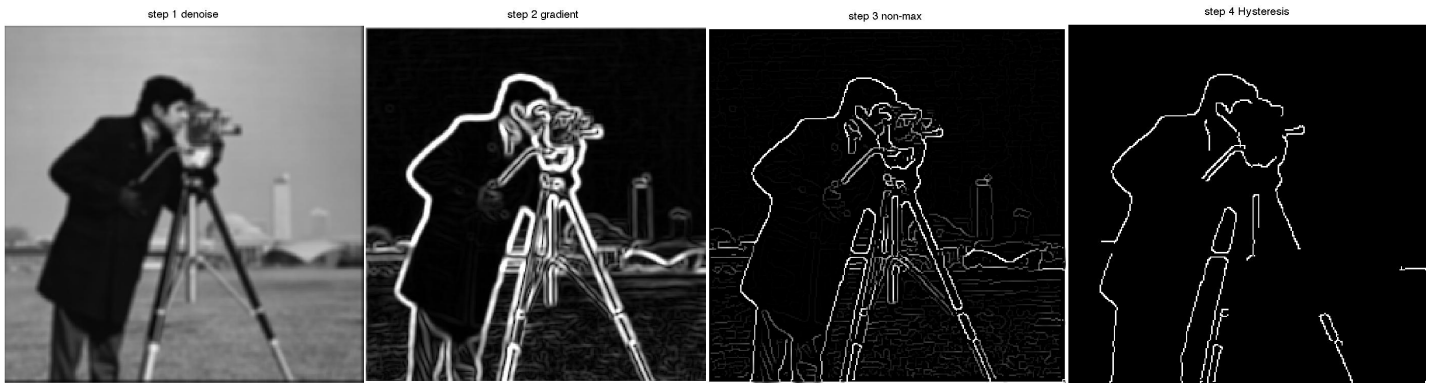


Figure 5: Intermediate results

b. Compare result with *edge()*

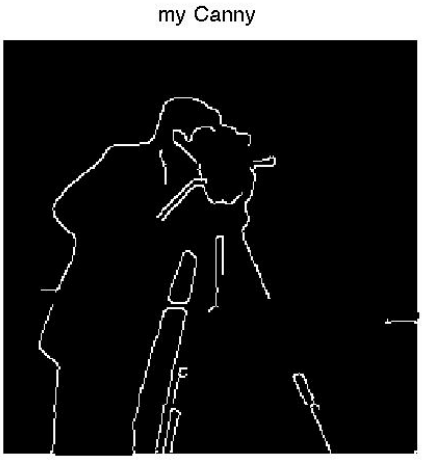


Figure 6 : T_high = 0.8

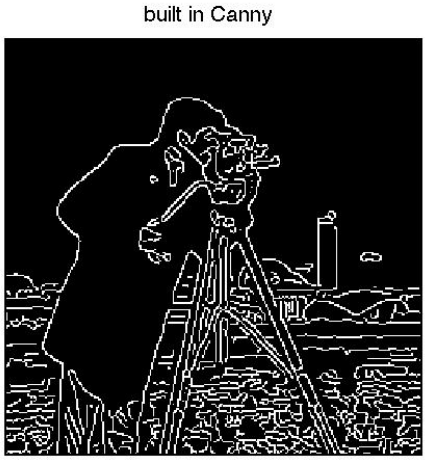


Figure 7: built-in

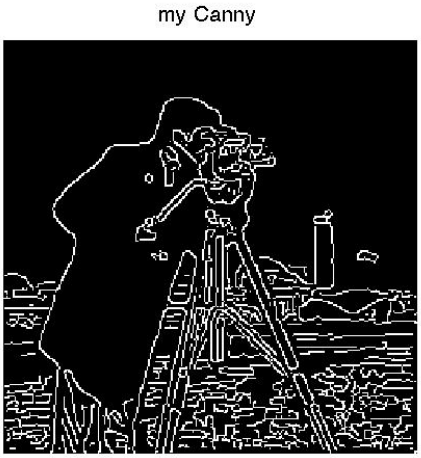


Figure 8: T_high = 0.08

The threshold values determine how much detail we get from the edge detection. Lower threshold values give us more details but may contain noise and unimportant information.

The built-in implementation is more detailed than T_high=0.8, but it is similar to our implementation with T_high=0.08.

3.

a. Algorithm

The main idea is to utilize LoG pyramid (detects step-like patterns) and its circular shape property to detect blobs. When we overlay a LoG filter of the right size on a circle, we would maximize the response because it essentially takes difference between the border of the

circle and its interior. We first generate a list of *n* LoG filters of various scales, then we apply each of them on to the image to get *n* response images. For each location on the image, we take its max among the *n* response images (compressing n images into one), this operation corresponds to finding the right size of the circle centered at that location. Then, we perform non-maximum suppression on the result image in a 15x15 window and thresholding such that only the centers remain in our result.

b.

(1) Circle.jpg

Figure 9: LoG Pyramid

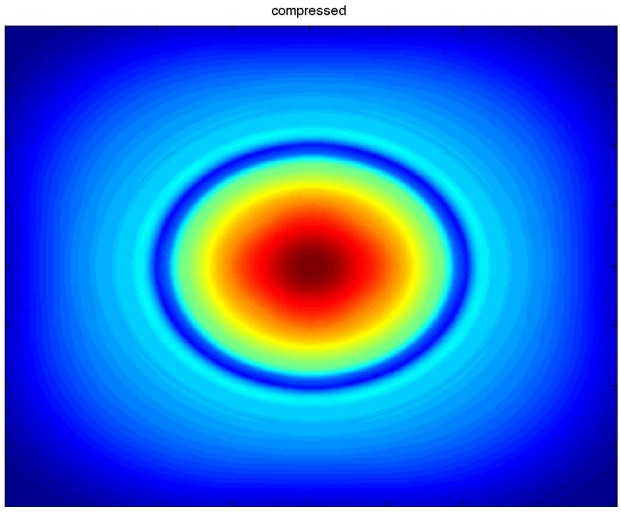


Figure 10 : First step of Non-maximum suppression

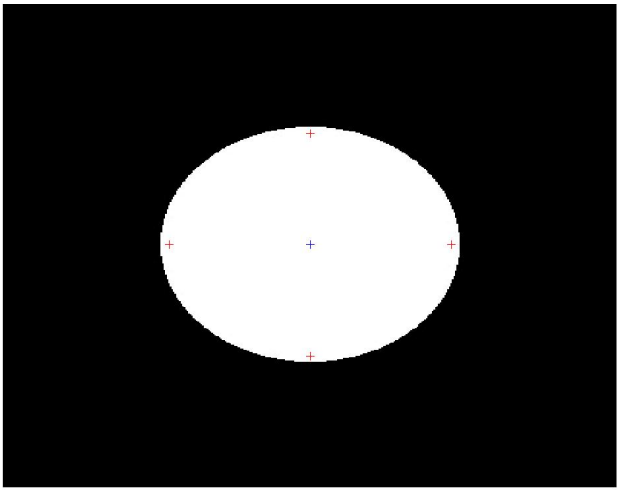


Figure 11 : Final result using scatter

(2) Circles.jpg

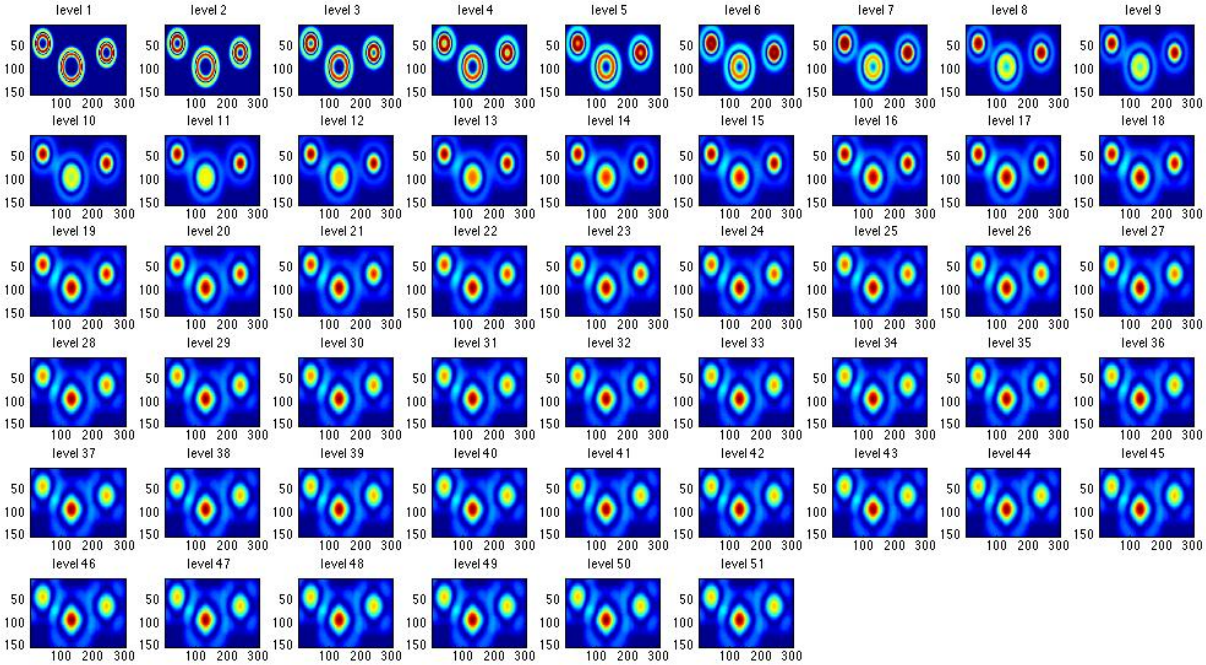
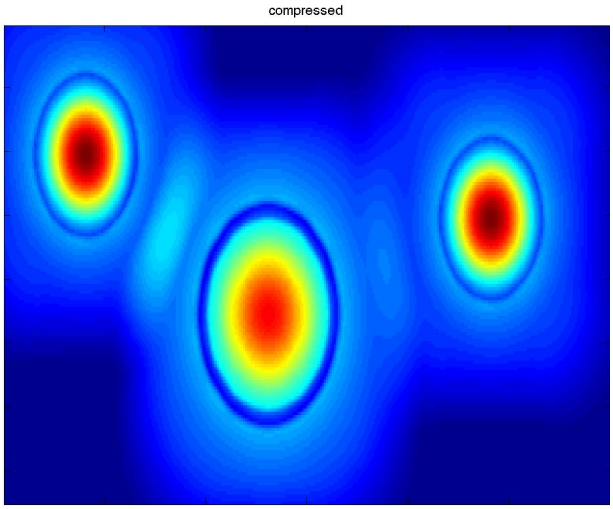


Figure 12 : LoG Pyramid



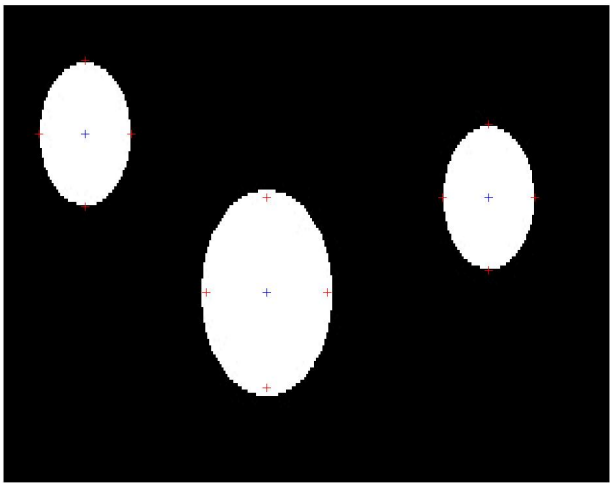


Figure 13: First step of Non-maximum suppression Figure 14: Final result

(3) Sunflower.jpg

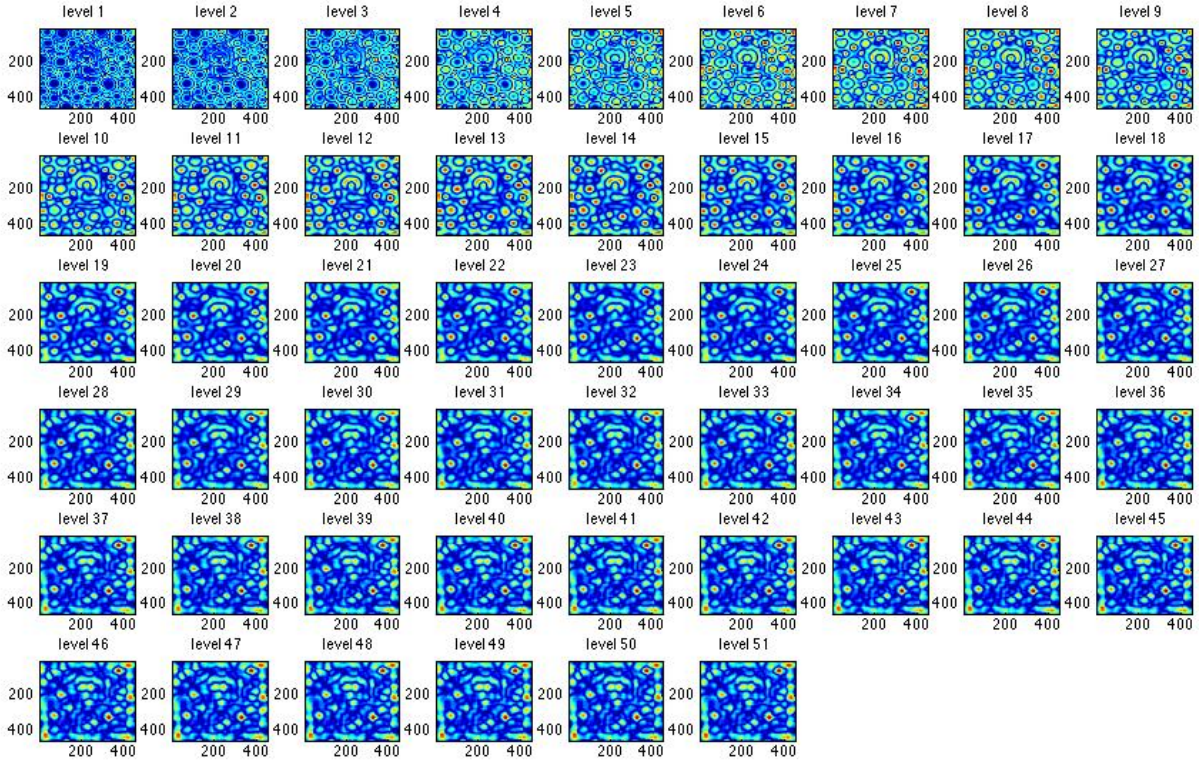


Figure 15 : LoG Pyramid

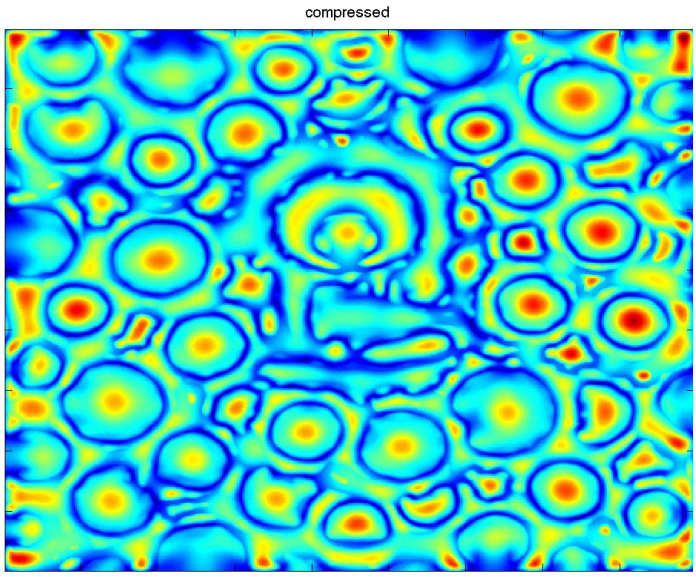


Figure 16 : First step of Non-maximum suppression

Figure 17 : Final result

4.

a. Algorithm

In Laplacian blending, we first build difference of Gaussians (DoG) pyramids for both source images. Then, we create a binary mask as a top level and derive a Gaussian pyramid from it. To blend, we use pyramid levels of the mask to combine the two Laplacian pyramids. In other words, we create a blended DoG pyramid by taking a linear combination of both pyramids at each level using the same level in the Gaussian pyramid as weights (using sigma = 1.4, window size = 5). Therefore, the lower frequencies of the images will be blended (locally, near the mask boundary) but their higher frequency components will be selectively preserved.
Lastly, we construct the blended image by calling pyramid reconstruction on the blended pyramid. Repeat all the steps above for R, G, and B channels individually, then concat the three result images along z-axis to get the final result (color image).

b. Results

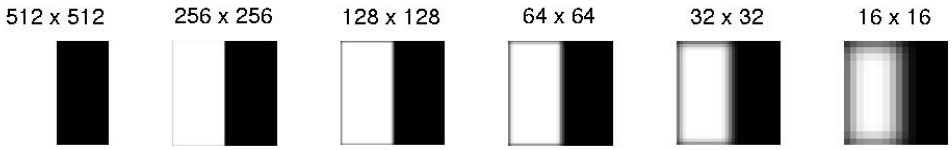


Figure 18 : Gaussian pyramid (masks)

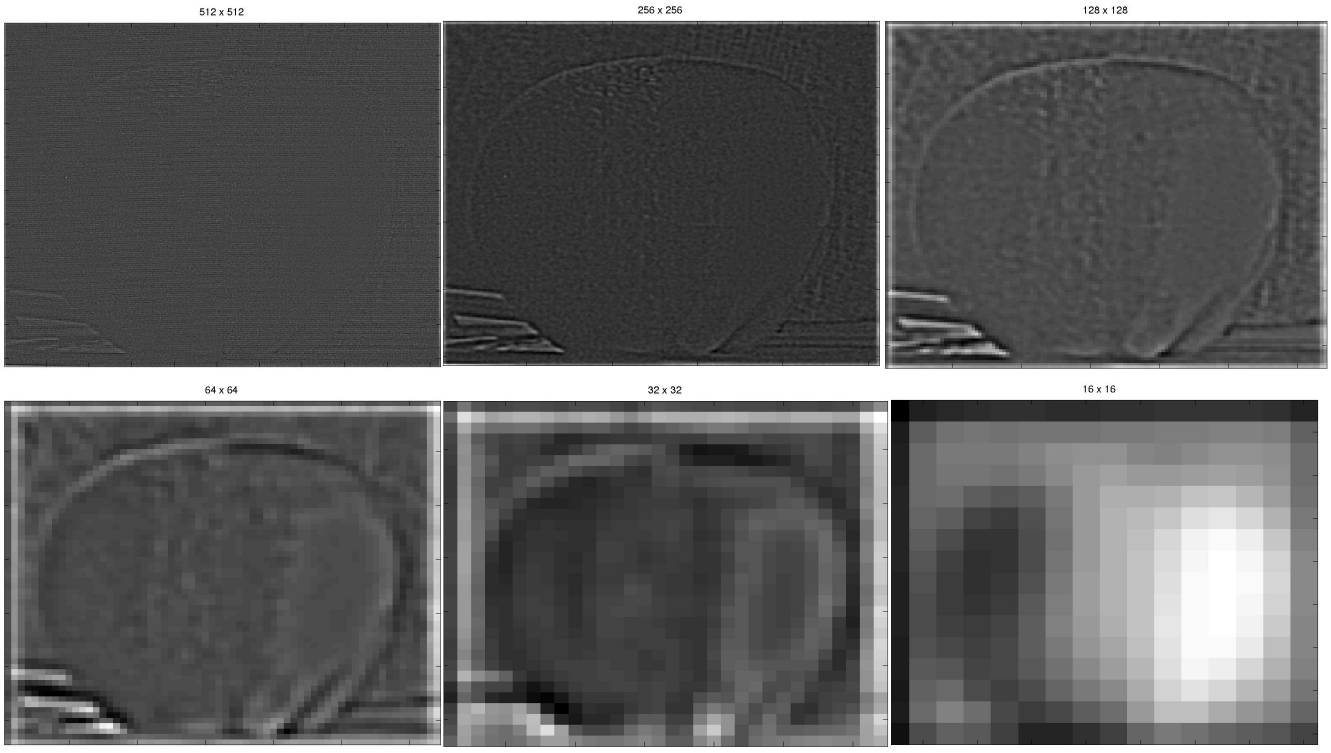


Figure 19: blended pyramid

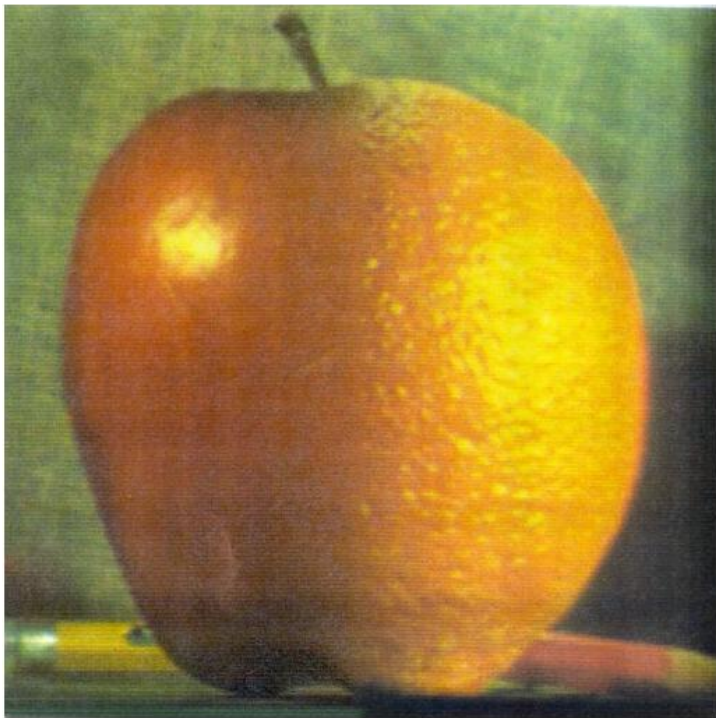


Figure 20 : blended image