CS1674: Homework 3 - Programming

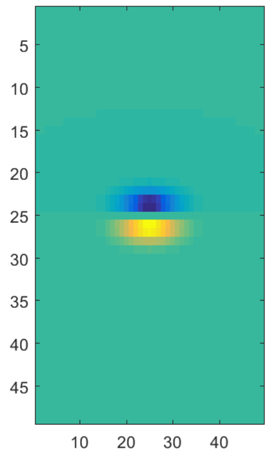
Due: 9/21/2016, 11:59pm

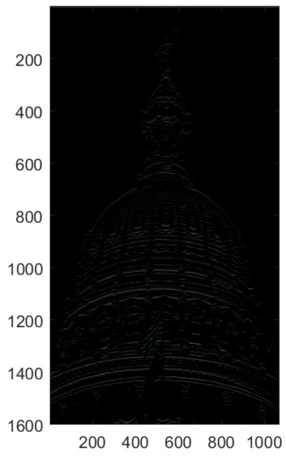
This assignment is worth 50 points. I estimate it would take you about 2-4 hours to complete if you understood the past few lectures.

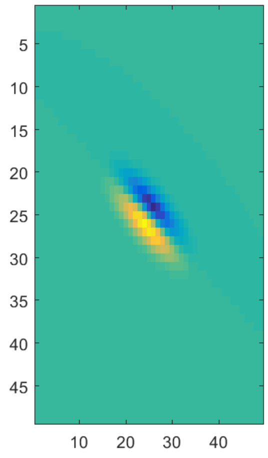
Part I: Image Representation with Filters (25 points) [should take you 1-3hrs]

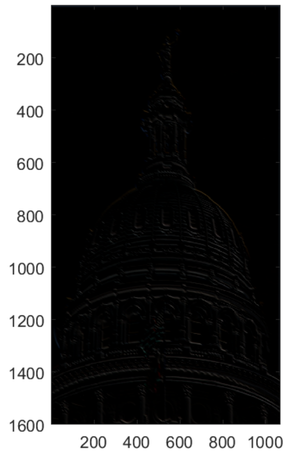
In this problem, you will use texture to represent images. For each image, you will compute the response of each image pixel to the filters from a "filter bank" as discussed in class. You will then compute a histogram and mean of these values for each image, and compare the histograms or means across images of the same or different categories.

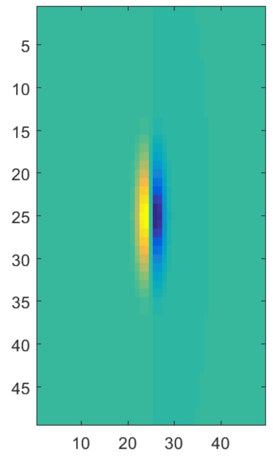
1. Download these images: cardinal1, cardinal2, leopard1, leopard2, panda1, panda2. As you can see, there are two images for each of three animal categories. Use a cell array to store the image filenames in Matlab, so you can use matrix/vector indices to refer to the k-th image. For simplicity, convert all images to the same square size (e.g. 512x512).
2. Download the Leung-Malik filter bank code (one function) and read the description at the top of how to run it. Run the code to obtain the filter bank `F`. You only need to do this once.
3. For each image, you need to do the following. First, read in the image, convert it to grayscale, and reduce its size by 0.5 or 0.25 (the smaller the faster it will be to convolve) using Matlab's `imresize`. Then convolve each pixel in your image with each of the 48 filters. i.e. with each `F(:, :, i)`. This would allow you to generate images like the responses to the Capitol image shown in class. Use the function `imfilter(image, filter)`. For three of the six images, show the response of the image to each filter, using subplots and imagesc. An example is shown below.

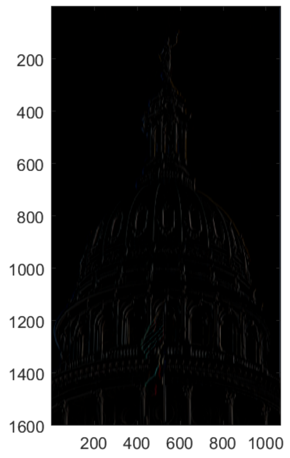


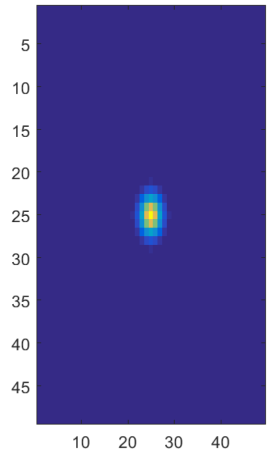


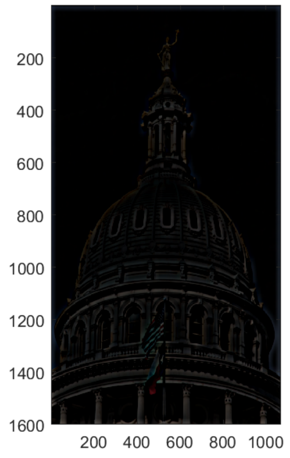


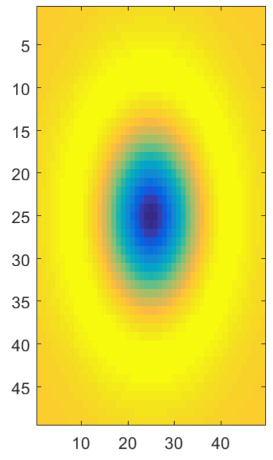


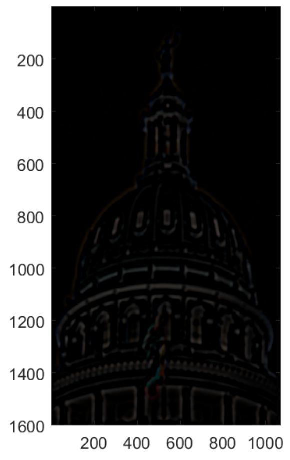












4. For each image, compute a histogram over the filter responses. Let `filt_im = imfilter(im, F(:, :, j)); filt_im = filt_im(:);` be the vector containing the responses to the i-th filter. Use `histc` to compute a histogram over the responses, after fixing the bin edges/ranges to `2.^(0:0.5:7)`. Then concatenate the histograms for all filters, resulting in a 1x720 vector. This vector is now the descriptor for the image. You will have one descriptor per image.
5. Now compute the Euclidean distance between pairs of images. As discussed in class, Euclidean distance measures the square root of the sum of the squared element-wise distances between the image descriptors. Initialize (set to `[]`) one variable to store within-category distances (distances between images that are of the same animal category), and another to store between-category distances (distance between images showing different animal categories). Then iterate over the image pairs and concatenate the distance for that image pair to the correct (within_ or between_) variable.
6. Print the mean of the within-category and between-category distances. Which one is smaller? By how much? Is this what you would expect? Why or why

not? Answer these questions in a text file called `responses.txt` which you include in your submission zip file.

7. Now let's compute the image represenation in a different way, again using filters. However, rather than computing a histogram, each image's representation will be the mean response across all pixels to each of the filters, resulting in one mean value per filter and an overall image representation of size 1x48. Repeat the process above to compute within-category and between-category distances. Now how does average within-category and between-category distance compare? Is this more in line with what you would expect?
8. Finally, let's just use the image pixels i.e. `im(:)` as the representation/description. Compute a histogram as for the texture histogram representation (the first one you tried), and use `0:5:255` as the bin edges. Compute within-/between-category distances again.
9. Which of the three types of descriptors that you used is the best one? How can you tell? Include your reasoning in your response. Don't worry about whether you get the answer that I expect, I care more about your reasoning than the actual answer, and your response in the text file mentioned above.
10. Include all your code in a script `compare_filter_responses.m`.

Part II: Hybrid Images (10 points) [should take you 5-15 min if you understood the hybrid image example]

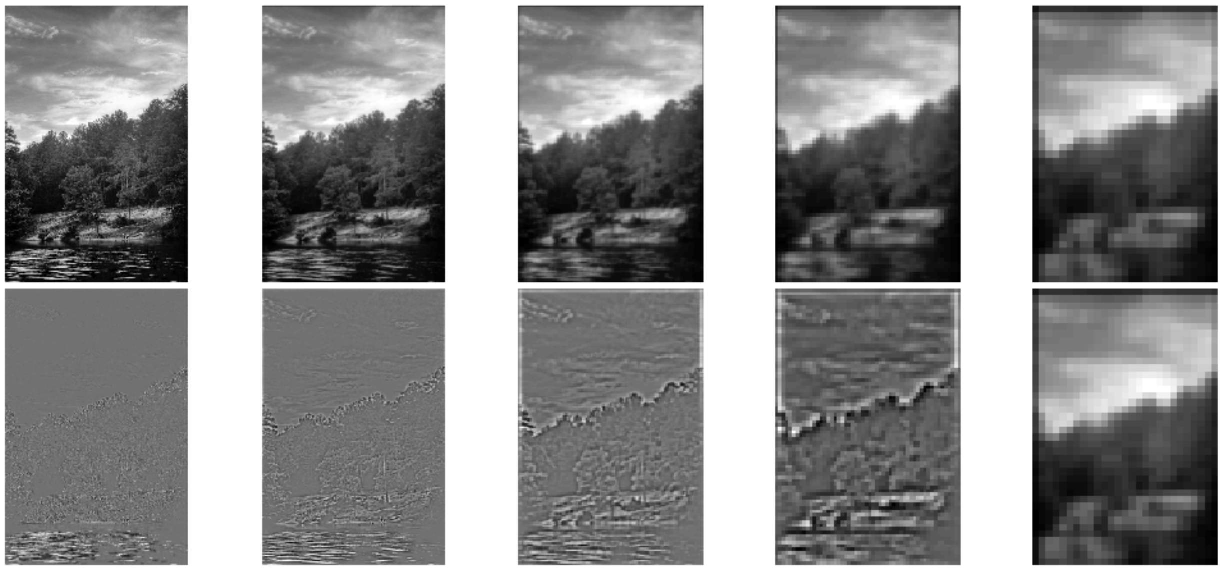
In this problem, you will create a hybrid image (which looks like one thing zoomed in and another zoomed out) like the one shown in class.

1. Download one pair of images: woman_happy and woman_neutral, or baby_happy and baby_weird.
2. In your code, read in both images and resize them to the same square size (e.g. 512x512). Also convert both of them to grayscale. Name your script `hybrid_image.m`.
3. Create a Gaussian filter with `fspecial('gaussian', hsize, sigma);`
4. Apply the filter to both, saving the results as `im1_blur, im2_blur`.
5. For the second image, subtract the blur of the image from the image (as we did with the Pittsburgh image in class), and save the result as `im2_detail`.
6. Now add `im1_blur` and `im2_detail`, show the image, save it, and include it with your submission. Play with scaling it up and down (by dragging in Matlab) to see the "hybrid" effect.

Part III Image Pyramids (15 points) [should take you 30min-1hr if you understood image pyramids]

In this problem, you will illustrate different levels of detail in an image, using image pyramids, as discussed in class.

1. Choose an image that has an interesting variety of texture (from Flickr/Google/Bing or your own images). The image should be at least 640x480 pixels and converted to grayscale. Use the Matlab function `rgb2gray`.
2. Write code for a Gaussian and Laplacian pyramids of level N (use `for` loops). In each level, the resolution should be reduced by a factor of 2. As discussed in class, before each subsampling, the image should be blurred; use the Matlab function `imfilter`.
3. To compute the Laplacian at each stage, remember that your images (pre- and post-blurring) should be of the same size, but the image you store in the actual pyramid should be scaled down by a factor of 2 compared to the image at the previous level (except of course the first image in the Gaussian pyramid, which is just the original image).
4. To scale an image down, simply ignore every other pixel. For example, you can retrieve only pixels in increments of 2. In Matlab, the way to do that is e.g. `im = im(1:2:end, 1:2:end);`
5. Use the same filter at each step.
6. Your code should include a function with signature `[G, L] = pyramids(im, fil);`, where `G` is the Gaussian pyramid and `L` is the Laplacian pyramid, while `im` is a grayscale image and `fil` is the output of `fspecial('gaussian', hsize, sigma)`. `G` should be a cell array such that `G{i}` returns the i-th level of the Gaussian pyramid. Similarly `L{i}` should return the i-th level of the Laplacian pyramid.
7. Set `L{n} = G{n};`
8. Your code should also include a script titled `image_pyramids.m` that loads the image, runs the `pyramids` code, and shows the pyramids. Show a Gaussian and Laplacian pyramid of level 5 for your chosen image using your code. You can show each `G{i}` and `L{i}` separately, resulting in 10 separate figures which you label appropriately using `title`, or you can show them together in 1 subplot with 10 figures or 2 subplots with 5 figures. You can (but don't have to) also use the `tight_subplot` function to format your plot. Your displayed images for the Gaussian and Laplacian pyramids should look something like the image below. The image at the bottom-right can be skipped.
9. Also include the original image you chose and the image pyramid figure(s) in your submission.



Acknowledgement: Part III is adapted from Derek Hoiem.