CIS 4930/CIS 5930 Computer Vision
Homework 1, Due: October 4th
120 points

Important: If you haven't used Matlab or Python to work with matrices and images before, please review some tutorials online. Like the tutorial in the course.

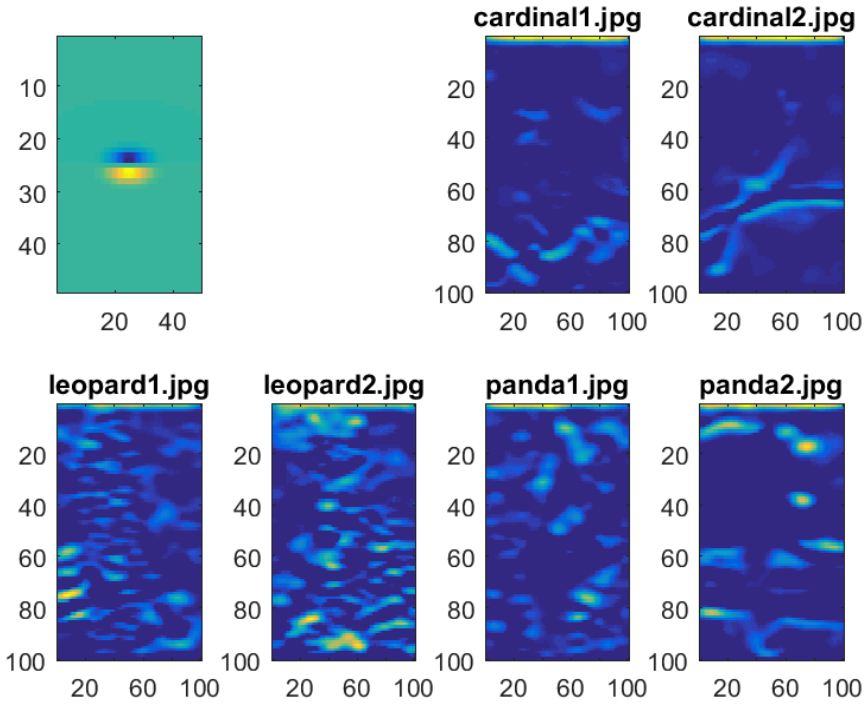
Part A: Image Responses with Filters (15 points)
In this problem, you will measure the responses of images to different filters. For each image, you will compute the response of each image pixel to the filters from a "filter bank" as discussed in class. In a later assignment, you will use these responses to filters to compute image representations.
1. Download these images from the homework directory: **`cardinal1.jpg, cardinal2.jpg, leopard1.jpg, leopard2.jpg, panda1.jpg, panda2.jpg`**

2. Download the Leung-Malik filter bank (named after **`filters.mat`**) from the homework directory (this is a Matlab file; in Matlab, you can read it with load, and in Python, you can read it with loadmat). Each filter **`F(:,:,i)`** is of size 49x49, and there are 48 filters.

3. [5 pts] Read all images, and convert all images to the same square size (e.g. 100x100), so that the visual map of responses can be more comparable across images. Also, convert the images to grayscale. Then convolve your image with each of the 48 filters. In Matlab, use imfilter; in Python, use convolve.

4. [5 pts] For each filter, generate a 2x4 subplot showing the following subplot rows: (1) the filter and a blank subplot, (2) the responses to the cardinal images, (3) the responses to the leopard images, and (4) the responses to the panda images.

5. [5 pts] Choose and include in your submission (1) one filter where the responses of images of the same animal are similar, while the responses of images of different animals are quite distinct; and (2) one filter where responses of different animals look fairly similar. Name the files **`same_animal_similar.png`** and **`different_animals_similar.png`**. See an example for one filter below.



Part B: Image Description with Texture (5 points)
In this problem, you will use texture to represent images. You will compute two image representations based on the filter responses. The first will simply be a concatenation of the filter responses for all pixels and all filters. The second will contain the mean of filter responses (averaged across all pixels) for each image.

Write a function **`computeTextureReprs`** with inputs image, **`F`** where image is an RGB image and **`F`** is the **`49 x 49 x num_filters`** matrix of filters you used before. The function should output two arguments **`texture_repr_concat`**, **`texture_repr_mean`** defined below.

1. [1 pts] First, create a new variable **`responses`** of size **`num_filters x num_rows x num_cols`**, where **`num_rows x num_cols`** is the size of the image. Convert the input image to grayscale.

2. [2 pts] Compute the responses of the image to each of the filters, and store the results in **`responses`**.

3. [1 pts] Create the first image representation **`texture_repr_concat`** by simply converting **`responses`** to a vector, i.e. concatenating all pixels in the response images for all filters.

4. [1 pts] Now let's compute the image representation in a different way. This time, the representation **`texture_repr_mean`** will be of size **`num_filters x 1`**. Compute each entry in **`texture_repr_mean`** as the mean response across all pixels to the corresponding filter. In other words, rather than keeping information about how each pixel responded to the filter, we are collapsing that information to a single value: the mean across all pixels.

Part C: Hybrid Images (10 points)
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 (**`im1`** and **`im2`**) from the homework directory:
`woman_happy.jpg` and **`woman_neutral.jpg`**, or
`baby_happy.jpg` and **`baby_weird.jpg`**

2. [3 pts] Read in the first image in the pair as im1 and the second as im2. Resize both images to the same square size (e.g. 512x512), and convert them to grayscale.

3. [2 pts] Create and apply a Gaussian filter; see gaussian filter. Save the results as **`im1_blur`**, **`im2_blur`**.

4. [2 pts] Obtain the detail image by subtracting **`im2_blur`** from **`im2`**, and save the result as **`im2_detail`**.

5. [3 pts] Now add **`im1_blur`** and **`im2_detail`**, show the image, save it as **`'hybrid.png'`**, and include it with your submission. Play with scaling it up and down to see the "hybrid" effect.

Part D: Canny Edge Detector (20 points)

In this problem, you will implement an edge detection pipeline, as discussed in class. This link also gives you many useful information.

1. [4 pts] Load the butterfly.jpg from the homework directory. Resize it to 256x256, and create a Gaussian filter to perform smoothing. Calculate the gradients from horizontal and vertical directions using the Sobel operators as we discussed in the class. Save these two gradient images as **`Ix`** and **`Iy`**. Calculate the magnitude of the image gradients as we discussed in class. Save the gradient image as a variable **`grad_img`** and also export it as **`grad_img.jpg`**.

2. [8 pts] Given **`Ix`** and **`Iy`**, calculate the angle for each point. Quantize the angle to 4 sections, and use **`[0, 1, 2, 3]`** to represent each section and save the quantized variable as **`q_theta`**. Use **`grad_img`** and **`q_theta`** to perform non-maxima suppression for the 3x3 regions of each pixel in **`grad_img`**. Export the result as **`non_maxima_supp.jpg`**.

3. [8 pts] Perform double thresholding on **`non_maxima_supp.jpg`**. Set **`T_1`** as the lower threshold and **`T_2`** as the higher threshold. After thresholding with **`T_1`** and **`T_2`**, perform edge linking with remaining pixels between values **`T_1`** and **`T_2`**. For each pixel between **`T_1`** and **`T_2`**, if the 3x3 neighborhood has an edge along the direction determined by **`q_theta(i,j)`**, then connect the current pixel with the edge else set it to 0. Set the magnitude of the reminned edges to 255 and save the resulting image to **`butterfly_edges.png`**.

Part E: Feature Detection (30 points)

In this problem, you will implement feature extraction using the Harris corner detector, as discussed in class. Write a function **`extract_keypoints`** with the following inputs/outputs:

Input:

- **`image`** is a color image which you should convert to grayscale in your function.

Outputs:

- Each of **`x,y`** is an **`nx1`** vector that denotes the x and y locations, respectively, of each of the *n* detected keypoints (i.e. points with "cornerness" **`R`** scores greater than a threshold that survive the non-maximum suppression). Keep in mind that `x` denotes the horizontal direction, hence *columns* of the image, and `y` denotes the vertical direction, hence *rows*, counting from the top-left of the image.
- **`scores`** is an **`nx1`** vector that contains the **`R`** score for each detected keypoint.
- **`Ix,Iy`** are matrices with the same number of rows and columns as your input image, and store the gradients in the x and y directions at each pixel.

Apply the function to **`cardinal1.jpg, leopard1.jpg, panda1.jpg`** from the homework directory.

Step-by-step instructions:

1. [5 pts] Let's do some preprocessing. First, set some parameters for use in your functions, at the beginning of your function: set the value of **`k`** (from the "Harris Detector: Algorithm" slide) to 0.05, and

use a window size of 5. Second, read in the image, and convert it to grayscale. Compute the horizontal image gradient **`Ix`** and the vertical image gradient **`Iy`**. Finally, initialize with zeros a matrix `R` of the same size as the image that will store the "cornerness" scores for each pixel.

2. [10 pts] Use a double loop to compute the cornerness score **`R(i, j)`** at each pixel **`i, j`**. This score depends on a 2x2 matrix `M` computed for each pixel, as shown in the slides. Use a window function of 1 inside, 0 outside, i.e. all neighbors of **`i,j`** that are less than **`half_window_size`** away from it have the same weight, and other neighbors don't contribute. Thus, the matrix `M` for a given pixel is a summation of **`window_size^2`** matrices, each of size 2x2. Each of the 2x2 entries is the product of gradient image values at that particular pixel. After computing **`M`**, use the formula from class to compute the **`R(i, j)`** score for that pixel. If a pixel is less than 2 pixels from the top/left or 2 pixels from the bottom/right of the image, set its R score to 0.

3. [5 pts] After computing all **`R(i, j)`** scores, it is time to threshold them in order to find which pixels correspond to keypoints. You can set the threshold for the "cornerness" score *R* however you like, e.g. (a) you can set it to 5 times the average *R* score, or (b) you can output the top **`n`** keypoints (e.g. top 1%).

4. [5 pts] Perform non-maximum suppression by removing those keypoints whose `R` score is not larger than all of their 8 neighbors; if a keypoint does not have 8 neighbors, remove it. The **`scores/x/y`** that you output should correspond to the final set of keypoints, after non-max suppression.

5. [5 pts] Display the input image, and visualize the keypoints you have detected, for example by drawing circles over them. Use the **`scores`** variable and make keypoints with higher scores correspond to larger circles. Include the visualization for three images in your submission (named **`cardinal.png, leopard.png, panda.png`**).

Part F: Feature Description (20 points)

In this problem, you will implement a feature description pipeline, as discussed in class. While you will not exactly implement that, the SIFT paper by David Lowe is a useful resource, in addition to Section 4.1 of the Szeliski textbook.

Write a function **`compute_features`** with inputs **`x, y, scores, Ix, Iy`** defined as above. The output **`features`** is an **`n x d`** matrix where each row contains the *d*-dimensional descriptor for the **`n`**-th keypoint. We'll simplify the histogram creation procedure a bit, compared to the original implementation presented in class. In particular, we'll compute a descriptor with dimensionality **`d=8`** (rather than 4x4x8), which contains an 8-dimensional histogram of gradients computed from an 11x11 grid centered around each detected keypoint (i.e. -5:+5 neighborhood horizontally and vertically).

1. If any of your detected keypoints are less than 5 pixels from the top/left or 5 pixels from the bottom/right of the image, i.e. pixels lacking 5+5 neighbors in either the horizontal or vertical direction, set its descriptor to be a vector of zeros.

2. [5 pts] To compute the gradient magnitude **`m(x,y)`** and gradient angle **`θ(x,y)`** at point **`(x,y)`**, use the Sobel operators shown in the class. If the gradient magnitude is 0, then both the **`x`** and **`y`** gradients are 0, and you should ignore the orientation for that pixel (since it won't contribute to the histogram).

3. [5 pts] Quantize the gradient orientations in 8 bins (so put values between -90 and -67.5 degrees in one bin, the -67.5 to -45 degree angles in another bin, etc.). For example, you can have a variable with the same size as the image, that says to which bin (1 through 8) the gradient at that pixel belongs.

4. [5 pts] To populate the SIFT histogram, consider each of the 8 bins. To populate the first bin, sum the gradient magnitudes that are between -90 and -67.5 degrees. Repeat analogously for all bins.

5. [5 pts] Finally, you should clip all values to 0.2 as discussed in class, and normalize each descriptor to be of unit length. Normalize both before and after the clipping. You do not have to implement any more sophisticated detail from the Lowe paper.

Part G: Image Description with SIFT Bag-of-Words (10 points)

In this part, you will compute a bag-of-words histogram representation of an image. The histogram for image $\mathbf{I_j}$ is a **k**-dimensional vector: $\mathbf{F(I_j) = [freq_{1,j}\ freq_{2,j}\ ...\ freq_{k,j}]}$, where each entry $\mathbf{freq_{i,j}}$ counts the number of occurrences of the **i**-th visual word in image **j**, and **k** is the number of total words in the vocabulary.

Write a function **computeBOWRepr** with inputs **features, means** where **features** is the **nx8** set of descriptors computed for the image and **means** is the **k x 8** set of cluster means, which is provided for you as **mean.mat** in the homework directory. The output should be a normalized bag-of-words histogram **bow_repr**.

1. [2 pt] A bag-of-words histogram has as many dimensions as the number of clusters *k*, so initialize the `bow` variable accordingly.

2. [4 pts] Next, for each feature (i.e. each row in **features**), compute its distance to each of the cluster means, and find the closest mean. A feature is thus conceptually "mapped" to the closest cluster.

3. [2 pts] To compute the bag-of-words histogram, count how many features are mapped to each cluster.

4. [2 pts] Finally, normalize the histogram by dividing each entry by the sum of the entries.

Part H: Comparison of Image Descriptions (10 points)

In this part, we will test the quality of the different representations. A good representation is one that retains some of the semantics of the image; oftentimes by "semantics" we mean object class label. In other words, a good representation should be one such that two images of the same object have similar representations, and images of different objects have different representations.

To test the quality of the representations, we will compare two averages: the average *within-class distance* and the average *between-class distance*. A representation is a vector, and "distance" is the Euclidean distance between two vectors (i.e. the representations of two images). "Within-class distances" are distances computed between the vectors for images of the same class (i.e. cardinal-cardinal, panda-panda). "Between-class distances" are those computed between images of

different classes, i.e. cardinal-panda, panda-leopard, etc. If you have a good image representation, should the average within-class or the average between-class distance be smaller?

1. [2 pts] Load the **cardinal**, **leopard**, and **panda** images again, and resize them to 100x100.

2. [2 pts] Use the code you wrote above to compute three image representations (**`bow_repr`**, **`texture_repr_concat`**, and **`texture_repr_mean`**) for each image.

3. [6 pts] Compute and print the ratio **`average_within_class_distance / average_between_class_distance`** for each representation. To do so, use one vector 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). Compute the mean of each of the two vectors, then compute the ratio of the means. Create a Word file named 'results.doc' and put **`bow_repr`**, **`texture_repr_concat`**, **`texture_repr_mean,`** and the ratio '**`average_within_class_distance / average_between_class_distance`**' into the Word file**.**

Acknowledgment: most of this assignment is adopted from the computer vision courses by Prof. Adriana Kovashka.