I. Short answer problems [30 points]

1. Suppose we are using \( k \)-means clustering to group pixels in a (tiny) image based on their intensity. The image’s intensities are: 5, 10, 3, 20, 9, 0. We pick the initial centers randomly to be 0 and 9, and set the number of clusters \( k=2 \).

   a) Which points will be assigned to each center in the first iteration?
   b) What will be the values of the \( k \) new centers (means)?

2. In order to segment an image into regions with consistent texture, we could use clustering to group pixels that produce similar responses to a bank of linear filters. Say the filter bank consists of eight \( n \times n \) kernels, each of which is designed to get maximal responses for contrast at a particular orientation. After filtering an image with all eight filters, each pixel gets an 8-dimensional feature vector, corresponding to all of the filter responses at that position. For any two such vectors \( p_i \) and \( p_j \), their affinity is defined as:

   \[
   Affinity(p_i, p_j) = \exp\left(-\frac{1}{2\sigma^2}\|p_i - p_j\|^2\right).
   \]

   This procedure will have some problems around pixels that are positioned near a true region boundary in the image. Why?

3. The shapes below exhibit reflectional symmetry about some vertical axis. Suggest a voting-based (Hough transform-like) method that could detect such symmetry and estimate the axis placement, given some edge points on the object’s contour. You may assume we only care about objects that are symmetric about an axis parallel to the image’s y-axis.
II. Programming problem: texture-based image matching and segmentation [70 points]

The goal is to use texture features to segment or compare images. We'll experiment with images in which texture is a defining feature. In both cases, we'll compare against a color-only baseline. Write programs that can do the following:

A. **Global image comparisons using texture**: Use texture to compute a global comparison between two images, and then classify the image using nearest-neighbor classification. First, compute a texton codebook from a large (~100K) random sample of filter response vectors extracted from the provided images. Second, summarize each image by a single global (image-wide) histogram of its texton occurrences. Finally, given any image as a new "query", sort the remaining images based on their similarity according to the $\chi^2$ (chi-squared) distance between their histograms. Classify each image in turn as belonging to one of the two candidate categories (dalmatian or zebra), depending on which class its nearest neighbor belongs to. Compare the overall accuracies when using either global texture histograms or global color histograms computed with the "hue" channel.
B. **Texture-based image segmentation:** Given an image, segment it into \( R \) regions using local texton histograms computed within windows. First, compute a texton codebook (dictionary) from the input image using \( k \)-means on its filter response vectors, where \( k \) equals the number of desired textons. Second, construct a texton histogram for each pixel based on the texton counts within its neighborhood (as defined by a local window of fixed scale that you choose). Finally, use \( k \)-means again to cluster those texton histograms, grouping the image's pixels into \( R \) regions. Choose parameter values (\( k \), \( R \), and window size) that yield a reasonable looking segmentation (of course, it won't perfectly agree with the object boundaries). Compare the output to what you get when using \( k \)-means to cluster the pixels' RGB values into regions.

See the end of this document for provided code and data, and tips on where to get started coding.

**Answer each of the following**, and include image displays where appropriate.

1. **[10 pts]** Look at the raw filter responses (as images) for some examples from each class. Choose an image from each class (one zebra, one dalmatian) and, by visual inspection, select two of the 38 filters for which these images' responses illustrate the texture differences well. Display the images and the corresponding responses, and explain what we are seeing.

2. **[10 pts]** Compute a universal texton dictionary using filter responses sampled from the 26 provided images. For the same two selected images as above, display the “texton map”, where every pixel is shaded or colored according to which texton it was assigned. (i.e., if there are 100 total textons in the dictionary, you will have at most 100 unique values in the texton map for an image). Briefly explain.

3. **[10 pts]** Report the 1-nearest-neighbor classification accuracy using the texture representation (as described in part A). Classify each of the 26 examples based on how the other 25 are ranked. If the first-ranked neighbor has the same label as the input image, it is correct. The accuracy is the percentage correctly classified out of all 26. Briefly explain the results, including any interesting error cases.

4. **[10 pts]** Report the 1-nearest neighbor classification accuracy using color histograms formed with the hue channel of the HSV image (also with \( \chi^2 \) distance). How does it compare to the texture-based representation?

5. **[10 pts]** Choose two image examples from each class (2 zebras, 2 dalmations), and for each, display their five most similar examples according to (a) the texture representation, and (b) the color representation. Label the subplots clearly as to what each is showing.

6. **[20 pts]** Select one image and use its filter responses to compute an image-specific texton dictionary. Segment the image using texture features computed within windows (as described in part B), and display the original image and segmentation result. Then segment the same image by clustering pixels by their RGB values; display the result. Briefly explain.

III. **[OPTIONAL] Extra credit [up to 10 points each, max 20 points extra credit]**

1. Implement basic background subtraction. Use the squared difference at each pixel between the current frame and the “background” frame to determine where the largest changes occur. Create the background frame itself by taking the median of all intensities along the provided sequence (*pset2_extra_credit_images.tar.gz*). Let the foreground pixels be those that survive some selected threshold on the squared differences. Extract the connected components, and
clean up the foreground regions with morphological operators.  (useful Matlab functions: median, bwlabel, imdilate, imclose, imerode, imopen). Show the results.

2. Using a universal texton dictionary, first segment two images from the same class (e.g., two zebras) into regions based on their texton histograms computed from local windows. Then, use the mean texton histogram associated with each resulting region to describe it. Let this be the region-level descriptor. For each region in the first image, find the region in the second image whose descriptor is closest (using L2). Display the region-to-region matching results. Then add the region’s centroid position to the descriptors, and compare the region-to-region matching results. Briefly explain.

3. Expand on #6 in Section II above: examine and show the impact of the neighborhood window scale parameter on the resulting segmentations, for one of the images.

Provided code and data:

- `dist2.m`: This function does fast computation of the squared Euclidean distances between two lists of vectors. This will be helpful when mapping the per-pixel vectors of filter responses to cluster centers to assign a texton to each pixel using the universal texton dictionary (part A). See the specifications at the top of the file.

- `filterBank.mat`: A .mat data file containing the filter bank as a single variable, \( F \). The filter bank is stored as a 49 x 49 x 38 matrix. It contains 38 total filters, where each filter is 49 x 49. (Load into memory with `load`).

- `makeRFSfilters.m`: For your reference, this is the Matlab code used to generate the provided filter bank. \( F = \text{makeRFSfilters;} \)  (Code by Manik Varma et al., http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html)

- `displayFilterBank.m`: Simple function to display the individual filters in the filter bank.

- `pset2images.tar.gz`: Images for the experiments. There are two classes, with 13 color images in each class.

Tips: Where to start? Your code will need to be able to perform the following tasks:

- Apply the filter bank to an image to produce a vector of responses at each pixel. For \( N \) filters, you will have \( N \) values computed for each pixel.
- Collect a random subsample of filter response vectors from the images.
- Given a set of filter responses and the desired number of textons, form a texton dictionary (codebook) using `kmeans` clustering.
- Given an image’s filter responses and a texton dictionary, map each pixel to its associated texton (e.g., an index, or cluster ID).
- Given an image, window scale, and texton dictionary, at each pixel compute the local texton histogram using the pixels falling within its local neighborhood window. (B)
- Cluster a single image’s texton histograms with \( k \)-means to form a texture-based region segmentation. (B)
- Given an image and texton dictionary, compute the image’s single global texton histogram. (A)
- Given an image, segment its pixels with \( k \)-means on the RGB (3-dimensional) values. (B)
• Compute an image’s color histogram using the hue channel (see rgb2hsv). The number of bins is a parameter; a value around 32 should be fine. (A)
• Given two histograms, compute the $\chi^2$ distance between them. For histograms $h_i$ and $h_j$, each with $K$ total bins, the distance is:

$$\chi^2(h_i, h_j) = \frac{1}{2} \sum_{k=1}^{K} \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

where $h_i(k)$ denotes the count in the $k$-th bin of histogram $h_i$.
• Given a query image, sort all other images relative to it using the $\chi^2$ distance.
• Compute the accuracy of the 1-nearest neighbor classifier (when using either texture or color features).

**Useful Matlab functions:** conv2, fspecial, imread, rgb2gray, imagesc, imshow, subplot, rgb2hsv, kmeans, reshape, dir, min, randperm, sort, load, im2double, axis equal, label2rgb, histc

• Filtering should be done with doubles, and on grayscale images.

**Relevant papers** for additional background reading (see class website for links):


**Submission instructions: what to hand in**

**Electronically:**

• Your well-documented Matlab code .m files.
• A pdf file containing the following:
  o Your name and CS login ID at the top.
  o Your answers to Section I, numbered.
  o Your responses and image results Section II, numbered. Insert image figures in the appropriate places for these questions, and label clearly.
  o (optional): any results and descriptions for extra credit portions in Section III.

Submit all the above with one call to turnin:

```bash
>> turnin --submit jaechul pset2 pset2.pdf codeFileXYZ.m codeFileABC.m dataFile.mat etc.
```

**Hardcopy:** Print out the pdf file, and bring it to class on Tuesday 10/6/09. Do not print out code. The hardcopy must be identical to what is submitted electronically by Monday night.