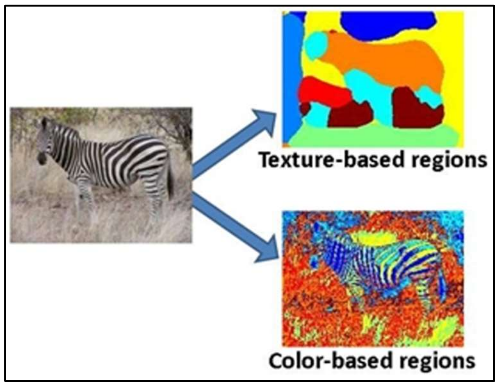
CS 378H Honors Machine Vision
Spring 2017

Assignment 2
Out: Friday Feb 10
Due: Friday Feb 24 (11:59 PM)



1. Image segmentation with k-means [50 points total]

For this problem you will write code to segment an image into regions using k-means clustering to group pixels. You will experiment with two different feature spaces---color and texture---and play with some design choices to understand their impact. Include each of the following, writing any additional functions as needed.

a) [5 pts] Given an `h x w x d` matrix `featIm`, where `h` and `w` are the height and width of the original image and `d` denotes the dimensionality of the feature vector already computed for each of its pixels, and given a `k x d` matrix `meanFeats` of `k` cluster centers, each of which is a d-dimensional vector (a row in the matrix), map each pixel in the input image to its appropriate k-means center. Return `labelIm`, an `h x w` matrix of integers indicating the cluster membership (1…k) for each pixel. Please use the following form:

```
function [labelIm] = quantizeFeats(featIm, meanFeats)
```

b) [5 pts] Given a cell array imStack of length `n` containing a series of `n` grayscale images and a filter bank `bank`, compute a texton "codebook" (i.e., set of quantized filter bank responses) based on a sample of filter responses from all `n` images. Note that a cell array can index matrices of different sizes, so each image may have a different width and height. Please include a function with this form:

```
function [textons] = createTextons(imStack, bank, k)
```

where `bank` is an `m x m x d` matrix containing `d` total filters, each of size `m x m`, and `textons` is a `k x d` matrix in which each row is a texton, i.e., one quantized filter bank response. See provided code and data below for "`filterBank.mat`" when applying this function, i.e., to populate `bank` with some common filters. Note that to reduce complexity you may randomly sample a *subset* of the pixels' filter responses to be clustered. That is, not every pixel need be used.

c) [5 pts] Given a grayscale image, filter bank, and texton codebook, construct a texton histogram for each pixel based on the frequency of each texton within its neighborhood (as defined by a local window of fixed scale `winSize`). Please include a function with this form:

```
function [featIm] = extractTextonHists(origIm, bank, textons, winSize)
```

where `textons` is a `k x d` matrix.

d) [5 pts] Given an `h x w x 3` RGB color image `origIm`, compute two segmentations: one based on color features and one based on texture features. The color segmentation should be based on k-means clustering of the colors appearing in the given image. The texture segmentation should be based on k-means clustering of the image's texton histograms. Please include a function:

```
function [colorLabelIm, textureLabelIm] =
      compareSegmentations(origIm, bank, textons, winSize,
      numColorRegions, numTextureRegions)
```

where `colorLabelIm` and `textureLabelIm` are `h x w` matrices recording segment/region labels, `numColorRegions` and `numTextureRegions` specify the number of desired segments for the two feature types, and the others are defined as above.

e) [15 pts] Now for the results. Write a script `segmentMain.m` that calls the above functions appropriately using the provided images (`dress.jpg`, `butterfly.jpg`, and `gumballs.jpg`) and, if you like, an image of your choosing and displays the results to compare segmentations with color and texture. Include the following variants:

- Choose parameter values (`k, numRegions, winSize, etc`) that yield a reasonable looking segmentations for each feature type and display results for the provided images. Of course, they *won't* perfectly agree with the object boundaries. We'll evaluate your assignment for correctness and understanding of the concepts, not the precise alignment of your regions.

- Consider two versions of the texture codebook: one where the texton codebook is computed from samples from all provided images, and one where the codebook is computed from only the individual image to be segmented.

- Consider two window sizes for the texture representation, a smaller and larger one, with sizes chosen to illustrate some visible tradeoff on one of the example images.

- Run the texture segmentation results with two different filter banks. One that uses all the provided filters, and one that uses a subset of the filters (of your choosing) so as to illustrate a visible difference in the resulting segmentations that you can explain for one of the example images. Note that the filter banks are organized by scale, orientation, and type.

f) [15 pts] In your writeup, explain all the results. Embed figures to illustrate, and label all figures clearly. We'd like to see one color/texture segmentation for each image. Then for each of the variations above, just choose one image to do the analysis.

Potentially useful Matlab functions: `kmeans, rgb2hsv, hsv2rgb, imshow, imagesc, label2rgb, im2double, reshape, subplot, title, hist.` Also, `dist2` and `displayFilterBank` below.

Matlab tip: if the variable `im` is a 3d matrix containing a color image with `numpixels` pixels, `X = reshape(im, numpixels, 3);` will yield a matrix with the RGB features as its rows.

Provided code and data:

- `dist2.m`: This function does fast computation of the squared Euclidean distances between two lists of vectors. This may be helpful when mapping the per-pixel vectors of filter responses to cluster centers to assign a texton to each pixel. 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 = makeRFSfilters;`) (Code by Manik Varma et al., `http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html`) You need not use this function.

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

- Images `dress.jpg`, `butterfly.jpg`, and `gumballs.jpg`

2. Circle detection with the Hough Transform [50 points total]

Implement a Hough Transform circle detector that takes an input image and a fixed radius, and returns the centers of any detected circles of about that size.

Include a function with the following form:

```
[centers] = detectCircles(im, radius, usegrad)
```

where `im` is the input image, `radius` specifies the size of circle we are looking for, and `usegrad` is a flag that allows the user to optionally exploit the gradient direction measured at the edge points. The output `centers` is an `N x 2` matrix in which each row lists the x,y position of a detected circle's center. Write whatever helper functions are useful.

Then experiment with the basic framework, and in your writeup analyze the following:

a) [10 pts] Explain your implementation in concise steps (English, not code).

b) [15 pts] Demonstrate the function applied to the provided images `circle-im-1.jpg` and `circle-im-2.jpg` and, if you like, an image of your choosing. Display the images with detected circle(s), labeling the figure with the radius. You can use the Matlab function `impixelinfo` to estimate the radius of interest manually.

c) [5 pts] For one of the images, display and briefly comment on the Hough space accumulator array.

d) [10 pts] For one of the images, experiment with ways to determine how many circles are present by post-processing the accumulator array.

e) [10 pts] For one of the images, demonstrate and explain the impact of the vote space quantization (bin size).

Useful Matlab functions: `atan2, hold on; plot, fspecial, conv2, im2double, sin, cos, axis equal; edge, impixelinfo.`

Matlab tip: note that the row number ("y" value) for an image position is flipped from Cartesian coordinates, increasing as we move down.

III. [OPTIONAL] Extra credit [up to 10 points for one item]

- Extend your Hough circle detector implementation to detect circles of any radius. Demonstrate the method applied to the test images.
- Implement a Generalized Hough Transform to detect a template shape. Demonstrate it on an image and template you choose.
- Combine color and texture for segmentation and show some result where their combination appears to help.
- Compare the k-means segmentation results to those of another clustering algorithm: mean shift, normalized cuts. Explain the differences.
- ***Have another extra credit idea you'd like to try? Run it by us.***

Submission instructions:

Create a single zip file to submit on Canvas that includes
- Your well-commented code, including the files and functions named as specified above.
- A pdf writeup of your results with embedded figures where relevant.

Please do *not* include any saved matrices or images etc. within your zip file.