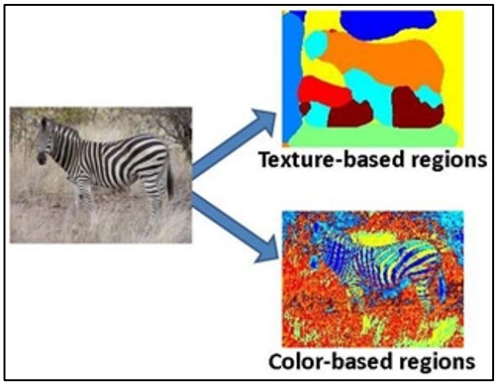
CS 376 Computer Vision
Spring 2018

Assignment 2
Out: Friday Feb 16
Due: Friday Mar 2 (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`). Note that textons are discrete. A pixel is mapped to

one discrete texton based on its distance to each texton. (see `quantizeFeats` above). 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 (`gumballs.jpg`, `snake.jpg`, and `twins.jpg`) and one other image of your choosing, and then 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 window sizes for the texture representation, a smaller and larger one, with sizes chosen to illustrate some visible and explainable 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. You might choose to make the subset you use quite limited to make the visual difference dramatic.

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.

Important: We are expecting only *reasonable* segmentation results, not perfect results. Part of this exercise is to experience firsthand the challenge of unsupervised segmentation. We will not deduct any points from you if (1) your code is correct, and (2) you show effort in selecting a reasonable parameter setting, and (3) you explain your results clearly.

Potentially useful Matlab functions: `kmeans`, `rgb2hsv`, `hsv2rgb`, `imshow`, `imagesc`, `label2rgb`, `im2double`, `reshape`, `subplot`, `title`, `hist`. Also, `dist2` (for fast Euclidean distances) 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.

- Provided test images

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

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

Include a function with the following form:

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

where `im` is the input image, `radius` specifies the size of circle we are looking for. Your detector should *not* exploit the gradient direction, though this is an option for extra credit (see below). 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) [20 pts] Demonstrate the function applied to the provided images `coins.jpg` and `planets.jpg` and one image of your choosing. Display the images with detected circle(s), labeling the figure with the radius. You can use the Matlab function `impixelinfo` or `imdistline` to estimate the radius of interest manually. Note: you only need to select *one reasonable radius* and display all detected circles (i.e., those with highest votes) under that radius. You are not required to consider circles with a center off

the image. Explain how your implementation post-processes the accumulator array to determine automatically how many circles are present.

c) [10 pts] For one of the images, display and briefly comment on the Hough space accumulator array.

d) [10 pts] For one of the images, demonstrate and explain the impact of the vote space quantization (bin size). In other words, alter the bin size and compare and contrast with a brief explanation why what happened makes sense.

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

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 max of one item]

Please note that only assignments submitted by the deadline above are eligible for extra credit.

- Extend your Hough circle to exploit the gradient direction at edge points when voting. Discuss the result. (useful function `atan2`)
- 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. Explain briefly.
- Combine color and texture for segmentation and show some result where their combination appears to help. Explain briefly.
- Compare the k-means segmentation results to those of another clustering algorithm: mean shift, normalized cuts. Explain the differences.

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.