Submission Requirements and Description (Very Important !)

Format requirements

(i) Please use the provided *Latex template* to write your report, and the report should contain your name, student ID, and e-mail address;

(ii) You should choose between Matlab and python to write your code, and provide a README file to describe how to execute the code;

(iii) Pack your **report**.pdf, **code** and **README** into a zip file, named with your student ID, like MG1833001.zip. If you have an improved version, add an extra '_' with a number, like MG1833001_1.zip. We will take the final submitted version as your results.

Submission Way

(i) Please submit your results to email nju.cvcourse@gmail.com , the email subject is "Assignment 2";

(ii) The deadline is 23:59 on May 14, 2022. No submission after this deadline is acceptable.

About Plagiarize

DO NOT PLAGIARIZE! We have no tolerance for plagiarizing and will penalize it with giving zero score. You may refer to some others' materials, please make citations such that one can tell which part is actually yours.

Evaluation Criterion

We mainly evaluate your submission according to your code and report. Efficient implementation, elegant code style, concise and logical report are all important factors towards a high score.

1 Edge Detection (50%)

Overview: The main steps of edge detection are: (1) assign a score to each pixel; (2) find local maxima along the direction perpendicular to the edge. Sometimes a third step is performed where local evidence is propagated so that long contours are more confident or strong edges boost the confidence of nearby weak edges. Optionally, a thresholding step can then convert from soft boundaries to hard binary boundaries.

We have provided 50 test images with ground truth from BSDS, along with some code for evaluation. Your job is to build a simple gradient-based edge detector, extend it using multiple oriented filters, and then describe other possible improvements. Details of your write-up are towards the end.

(a) Build a simple gradient-based edge detector that includes the following functions (25%)

$$function[mag, theta] = gradientMagnitude(im, sigma) \tag{1.1}$$

This function should take an RGB image as input, smooth the image with Gaussian std=sigma, compute the x and y gradient values of the smoothed image, and output image maps of the gradient magnitude and orientation at each pixel. You can compute the gradient magnitude of an RGB image by taking the L2-norm of the R, G, and B gradients. The orientation can be computed from the channel corresponding to the largest gradient magnitude. The overall gradient magnitude is the L2-norm of the x and y gradients. mag and $theta$ should be the same size as im.

$$function\ bmap = edgeGradient(im) \tag{1.2}$$

This function should use gradientMagnitude to compute a soft boundary map and then perform non-maxima suppression. For this assignment, it is acceptable to perform non-maxima suppression by retaining only the magnitudes along the binary edges produce by the Canny edge detector: $edge(im,' canny')$. Alternatively, you could use the provided $nonmax.m$ (be careful about the way orientation is defined if you do). You may obtain better results by writing a non-maxima suppression algorithm that uses your own estimates of the magnitude and orientation. If desired, the boundary scores can be rescaled, e.g., by raising to an exponent: $mag2 = mag^{0.7}$, which is primarily useful for visualization.

Evaluate using $evaluateSegmentation.m$ and record the overall and average F-scores.

(b) Try to improve your results using a set of oriented filters, rather than the simple derivative of Gaussian approach above, including the following functions: (15%)

$$function[mag, theta] = orientedFilterMagnitude(im) \tag{1.3}$$

Computes the boundary magnitude and orientation using a set of oriented filters, such as elongated Gaussian derivative filters. Explain your choice of filters. Use at least four orientations. One way to combine filter responses is to compute a boundary score for each filter (simply by filtering with it) and then use the max and argmax over filter responses to compute the magnitude and orientation for each pixel.

$$function\ bmap = edgeOrientedFilters(im) \tag{1.4}$$

Similar to part (a), this should call $orientedFilterMagnitude$, perform the non-maxima suppression, and output the final soft edge map.

Evaluation: The provided $evaluateSegmentation.m$ will evaluate your boundary detectors against the ground truth segmentations and summarize the performance. You will need to edit to put in your own directories and edge detection functions. Note that I modified the evaluation function from the original BSDS criteria, so the numbers are not comparable to the BSDS web page. The overall and average F-score for my implementation were (0.57, 0.62) for part (a) and (0.58, 0.63) for part (b).

Write-up: Include your code with your electronic submission. In your write-up, include:

- Description of any design choices and parameters
- The bank of filters used for part (b) (imagesc or mat2gray may help with visualization)
- Qualitative results: choose two example images; show input images and outputs of each edge detector
- Quantitative results: precision-recall plots (see "pr_full.jpg" after running evaluation) and tables showing the overall F-score (the number shown on the plot) and the average F-score (which is outputted as text after running the evaluation script).

(c) Ideas for improvement (10%): Describe at least one idea for improving the results. Sketch an algorithm for the proposed idea and explain why it might yield improvement. Improvements could provide, for example, a better boundary pixel score or a better suppression technique. Your idea could come from a paper you read, but cite any sources of ideas.

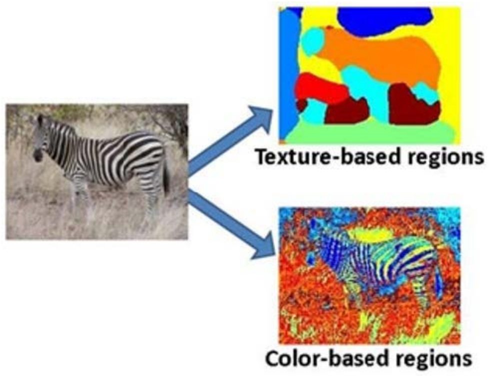
Extra credit (up to 10%): Try to improve the results that you get in (b), e.g., using your idea from (c). Include code, explain what you did, and show any resulting improvement. Note: this is not an easy way to get points; 10 pts would be reserved for big improvements.

Related papers: Reading these will aid understanding and may help with the assignment.

Berkeley Pb Detector: `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/papers/mfm-pamiboundary.pdf`

2 Image segmentation with k-means (50%)

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%) Given an $h \times w \times d$ 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 $k \times d$ 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 $h \times w$ matrix of integers indicating the cluster membership (1…k) for each pixel. Please use the following form:

$$function[labelIm] = quantizeFeats(featIm, meanFeats) \tag{2.1}$$

(b) (5%) 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:

$$function[textons] = createTextons(imStack, bank, k) \tag{2.2}$$

where bank is an $m \times m \times d$ matrix containing d total filters, each of size $m \times m$, and $textons$ is a $k \times d$ matrix in which each row is a texton, i.e., one quantized filter bank response. See provided code and data below for $\backslash filterBank.mat$" 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%) 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:

$$function[featIm] = extractTextonHists(origIm, bank, textons, winSize) \tag{2.3}$$

where $textons$ is a $k \times d$ matrix.

(d) (5%) Given an $h \times w \times 3$ 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:

$$\begin{aligned} function[colorLabelIm, textureLabelIm] = compareSegmentations(origIm, bank,\\ textons, winSize, numColorRegions, numTextureRegions) \end{aligned} \tag{2.4}$$

where $colorLabelIm$ and $textureLabelIm$ are $h \times w$ 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%)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%) 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 $49 \times 49 \times 38$ matrix. It contains 38 total filters, where each filter is 49×49. (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