16-720 Computer Vision: Homework 6
Superpixels for Image Segmentation

Instructor: Srinivasa Narasimhan
TAs: Arun Venkatraman, Aayush Bansal, Yiying Li

Due: Wednesday, April 22nd, 11:59:59.$\bar{9}$ p.m.

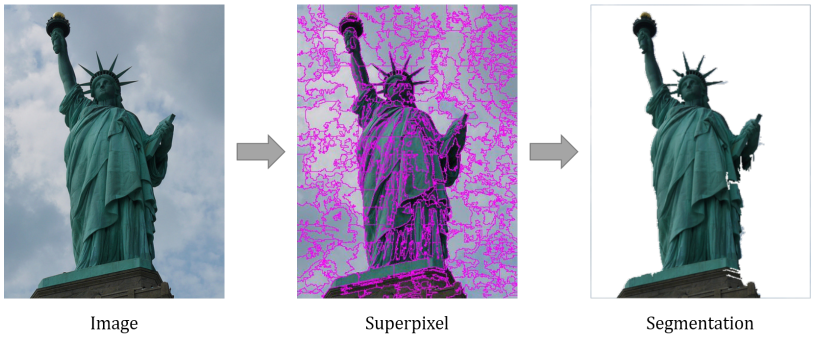

Figure 1: **Foreground/Background Segmentation:** Given an image, can a computer program automatically determine which set of pixels belong to the foreground and which set belong to the background? In this homework, you will build segmentation pipelines based on superpixels to perform this task, with or without training images.

Instructions/Hints

1. Please pack your system and write-up into a single compressed file named <**AndrewId**>**.zip**, see the complete submission checklist in Section 5.
2. All questions marked with a **Q** require a submission.
3. For the implementation part, please stick to the headers, variable names, and file conventions provided.
4. **Start early!** As always.
5. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge mess when you put everything together.

Overview

Segmentation is a fundamental problem in computer vision. The objective for most image segmentation algorithms is to divide the image into a set of contiguous regions. Nowadays the most popular ones include: normalized cut [7], graph-based method [2, 4], mean shift [3], *etc.* While these approaches worked on images with homogeneous foreground and non-cluttered background, for real-world images they often fail to produce consistent segmentation results. One major reason for this lies in the pure bottom-up idea of segmentation itself: Objects can be complex, and they often consist of multiple visually-consistent clusters. This turns out to be too difficult for the vanilla bottom-up segmentation algorithms.

To this end, the idea of superpixels is proposed. It differs from the traditional segmentation scenario in which the goal is to produce semantically meaningful regions. Instead, the primary goal for superpixels are to generate visually consistent regions regardless of the size and defer the semantics into later steps in the pipeline. Because superpixels capture the redundancy of an image, they can greatly reduce the computational complexities of processing, and a richer set of features can usually be computed from them [6]. Therefore, computing superpixels is often a crucial pre-processing step for higher level vision tasks such as object recognition. Besides the algorithms mentioned before, algorithms specifically designed for superpixel computing also exist [1, 5].

In this assignment you will explore the usage of superpixels by using them for the foreground-background segmentation task in both unsupervised (Section 2) and supervised (Section 3) settings. You will be working with the widely used segmentation dataset:

- MSRC [8]: 14 object classes with about 30 images per class;

In the course of the homework, you will do three things:

1. First, you will explore the **unsupervised** setting of segmentation by separating foreground from background using K-means and Spectral Clustering
2. Second, you will explore the **supervised** setting of segmentation by separating foreground from background in test images using a trained model.
3. Third, you will attempt to improve the results of any of the above methods by trying other superpixel segmentation algorithms and/or other features and/or modifying the clustering steps.

We provide you with a number of functions and scripts in the hopes of alleviating some tedious or error-prone sections of the implementation. You can find these in `matlab/provided/`.

1 Representing an Image with Superpixels

1.1 SLIC setup

As a building block for later recognition tasks, in this part, you are provided an algorithm that takes in an image and outputs superpixels, **SLIC**. Simple Linear Iterative Clustering (SLIC) [1] is a simple algorithm that performs a local clustering of pixels represented by a concatenation of values in color spaces and image coordinates. You may think about it as a local K-means algorithm.
The following function lets you call the SLIC lgorithm:

```
[labels, num_labels] = slicSuperPixels(image, num_clusters, compactness)
```

where `im` is the image to process, `num_clusters` is the number of super pixels we desire and `compactness` is now localized we want each superpixel to see. The output `num_labels` is the number of labels (super pixels) that were generated and may be less than the requested number due to collation of centers during the clustering. `labels` is the output segmentation (same width and height as input image) that marks each pixels with the index number of its cluster.

Note that before using this code, you may need to compile the source[1] with the command:

```
mex slicmex.cpp
```

[1]Created by: `http://ivrl.epfl.ch/research/superpixels`

to get the proper mex file. For 64bit Mac OS X machine (Mavericks or Yosemite), we already have a compiled file `slicwrapper.mexmaci64` for you. For 64bit Linux, we have a compiled file `slicmex.mexa64`. For 64-bit Windows, we have compiled it to `slicmex.mexw64`. Check if these work for you. If not, delete the file and recompile it on your machine.

1.2 Dataset & Other Tools

We will be using the MSRC data set[2] in this assignment. We have a subset of it parsed and ready to go inthe `data/MSRC/` directory. You will notice that there is a `.mat` file for every class that we will we looking at. Each `mat` file consists of a struct array. Each element in the struct array is a struct corresponding to a single image and relevant meta-data.

To get started using the provided code, `cd` into the `matlab/` directory and add the `provided/` directory to the path (see Matlab's `addpath` command or right-click on the folder in the "current folder" bar). Now we have access to all the functions in the `provided` directory.

If we run the below couple lines of code, we can see what the data structure looks like:

```
% loads in struct array variable called images
load('../data/MSRC/tree.mat')
images
    1x30 struct array with fields:
images(1)
         im: [320x213x3 uint8]
      class: 'tree'
        num: '10'
         gt: [320x213 logical]
       name: '2_10_s.bmp'
```

We have also provided some useful visualization functions:

```
im_num = 1
im = images(im_num).im;

[labels, num_labels] = slicSuperPixels(im, 200, 30);

% Create a binary image with just the boundaries of the segments
bmap = seg2Bmap(labels);
% Show boundary overlayed on the original image
bmap = bMapOverlay(im, bmap);
% Create image with different color for each segment
color_labels = randomColor(labels);

% Create image with overlaying colored segments on original image
gt = images(im_num).gt
overlay_labels = overlayClasses(im, gt)
```

Play around with these commands to get a feel for how they work. Especially, try the `slicSuperPixels` function to get a feel for how the number of clusters and the compactness parameters affect the output superpixels clusters.

2 Unsupervised FG and BG segmentation

2.1 Bag of Words Revisited

Our end goal in the assignment is to segment the foreground and background from an image. In the context of our superpixel representation, we want to assign each superpixel a foreground or back-

[2] `http://research.microsoft.com/en-us/projects/objectclassrecognition/`

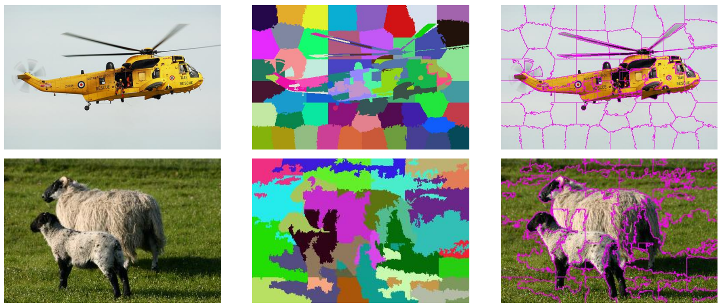
Figure 2: Two input images and outputs with the SLIC algorithm.

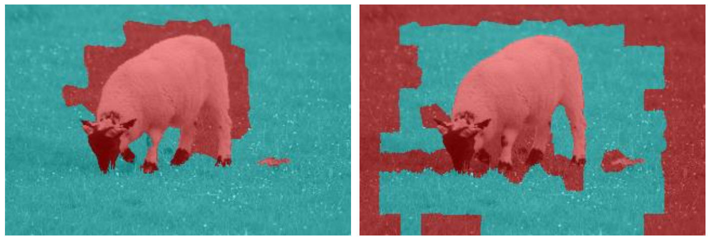
Figure 3: Example results with K-means on a sheep image. Notice how two different runs can produce dramatically different results.

ground label. To do so, we need to first assign a feature representation that captures information across the whole superpixel. This should sound familiar – in a prior assignment, we had to capture a representation of a whole image for scene classification! We accomplished this previously with Bag of Words.

Q1 (10 points): Generate a texton dictionary for the provided MSRC dataset. Then generate wordmaps for every image.

For the dictionary creation, you should adapt your code from Homework 1. For generating the wordmaps, you can use the script `provided/batchToVisualWords.m` to generate the wordmaps. It works in a similar way to Homework 1 by calling your `getVisualWords()` function. The wordmaps are stored in `../data/wordmaps/` in the same struct format to the original data except with an additional `wordmap` field.

Output: Submit a `dictionary.mat` with the filter bank and the dictionary. Do *not* submit the wordmaps directory.

2.2 Simple Clustering with K-means

One of the simplest ways to cluster the superpixels into foreground vs background is to use K-means. K-means will try to group superpixels based on how close they are in their feature space with respect to some distance metric.

Q2 (15 pts): Implement a function that takes a wordmap and SLIC superpixel segmentation and returns a k clustering of the superpixels. The function should have the prototype:

```
function labels_out = wordmapKmeans(wordmap, num_words, labels, k)
```

where `wordmap` is the wordmap image ($w \times h$), `num_words` is the size of the dictionary, `labels` is an image ($w \times h$) where each pixel value represents its current segmentation assignment (e.g. SLIC segmentation labels), and `k` is the number of clusters we want K-means to find. `labels_out` is an image ($w \times h$) with each pixel labeled as an integer from $1 - k$ (or $0 - (k - 1)$).

- First, we need to aggregate the wordmap values for each label in the image. We have provided a function `provided/valuesPerLabel.m` to help with this.
- We must aggregate some statistics on the wordmap values for each segment label from SLIC. Construct a histogram of the wordmap values (as we did with bag-of-words in a previous assignment). (*Hint:* see Matlab's `arrayfun()`).
- Note that to get foreground and background classes, we will use $k = 2$.
- It may be useful to consider distance metrics other than the default Euclidean distance (see the help for `kmeans`). You may even also consider implementing your own k-means using a histogram similarity score like we used in the previous assignment.

Q3 (15 pts): Write a script `matlab/evaluateKmeans.m` that will evaluate the quality of k-means unsupervised clustering on the foreground background segmentation task.

- You may have noticed that running `wordmapKmeans()` can generate very variable output depending on the k-means' randomized initialization (see Figure 3). To properly evaluate the performance, we thus have to run each image many times to see the average performance.
- To assist you in generating the performance metric, we have provided the function `provided/evaluateImFGBG.m`. This function will compute the accuracy and F1 score[3] given the ground truth labels and the predicted labeling. It will account for the fact that we don't know from k-means which cluster number is the foreground and which is the background label.
- For each class (bike, sheep, tree, etc.), you will run `wordmapKmeans()` on each image multiple times and store the average performance for that image. We will also store the performance separately for all images in the class so we can compute a final average and standard deviation for the accuracy and F1 score for each class.
- The names of all the semantic classes can be found in `provided/class_names.mat`
- Your implementation should achieve an accuracy and F1-score average per class similar to the baseline results in Table 1
- **Output:** Generate a bar graph showing both the accuracy and F1 scores for all classes (label the x-axis with each class name – bike, tree, etc.). Include error bars with the standard deviation across images for that the class. Include the bar graph in your writeup along with a table (for LaTeX users see `provided/matrix2latex.m`) with numerical results for all classes along with example images. (*Hint:* see Matlab's `bar` and `errorbar` functions) Also in your writeup include example images and describe features used, parameters chosen, etc. and any effects you saw on your results. We want to see some analysis of what you believe are success and failure cases of this method.

	`bike`	`bird`	`car`	`cat`	`chair`	`cow`	`dog`	`face`	`flower`	`house`	`plane`	`sheep`	`sign`	`tree`
Acc.	0.64	0.76	0.68	0.65	0.67	0.75	0.74	0.67	0.81	0.72	0.6	0.8	0.77	0.7
F1	0.61	0.5	0.64	0.51	0.56	0.65	0.58	0.62	0.74	0.69	0.44	0.7	0.7	0.71

Table 1: Baseline results for K-means foreground vs background segmentation

[3] `http://en.wikipedia.org/wiki/F1_score`

2.3 Better Clustering with Spectral Clustering

A more complex way than k-means that isn't dependent on initialization is spectral clustering. Spectral clustering requires analyzing the eigenvalues of the affinity matrix between every superpixel. An affinity matrix is the similarity between each pair of superpixels. Some examples of valid similarity measurements are correlation, cosine similarity, as well as the inverse of any distance metric such as hamming distance or euclidean distance.

Q4 (20 pts): Write a script `matlab/evaluateSpectral.m` that will evaluate the quality of unsupervised spectral clustering on the foreground background segmentation task.

- To assist you (since spectral theory is not covered in this class) the provided function `provided/spectralCluster.m` will compute the spectral clustering given an affinity matrix. The function has the following prototype:

```
function C = spectralCluster(A, k, T)
```

  where `A` is the affinity matrix, `k` is the number of clusters (2 in this case for fg and bg), and `T` is the type of normalization used (either `Shi` or `Jordan`). The type of normalization is optional. The output `C` is the for each superpixel the cluster id that it corresponds to.

- For each class you will first need to generate an affinity matrix from the histograms (hint: try multiple similarity metrics, not all of them perform well)
- You should achieve accuracy and F1-score average per class similar to the baseline results in Table 2 Note: Spectral Clustering doesn't necessarily outperform k-means in certain classes, it is however more immune to initialization problems, and doesn't have to be ran multiple times.
- **Output:** Generate a bar graph showing both the accuracy and F1 scores for all classes (label the x-axis with each class name – bike, tree, etc.). Include error bars with the standard deviation across images for that the class. Include the bar graph in your writeup along with a table (for LaTeX users see `provided/matrix2latex.m`) with numerical results for all classes along with example images. (*Hint:* see Matlab's `bar` and `errorbar` functions) Also in your writeup include example images and describe features used, parameters chosen, etc. and any effects you saw on your results. We want to see some analysis of what you believe are success and failure cases of this method.

	`bike`	`bird`	`car`	`cat`	`chair`	`cow`	`dog`	`face`	`flower`	`house`	`plane`	`sheep`	`sign`	`tree`
Acc.	0.63	0.86	0.71	0.67	0.69	0.75	0.76	0.68	0.88	0.73	0.56	0.88	0.83	0.73
F1	0.61	0.61	0.69	0.5	0.56	0.63	0.56	0.63	0.84	0.7	0.4	0.77	0.78	0.75

Table 2: Sample results for Spectral Clustering foreground vs background segmentation

3 Supervised: Semantic Segmentation

In the above, we saw the difficulty in detecting foreground vs background in the unsupervised setting. What does it mean for something to be foreground or background? In the section we look to compare against a supervised approach where we know what 'foreground' means.

3.1 Simple case: Known semantic class

To simply matters, we will consider the problem of foreground or background as single-class classification, learning a separate model for each semantic class (bike, bird, tree, etc.). We will then assume that we know which class we're looking for given a test image - we just need to segment out where that semantic class is in the image.

Q5 (20 pts): Implement a set of functions/scripts that implement supervised segmentation with a separate model per semantic class. Create a script called `evaluateSupervisedKnown.m` that will run and get the per-semantic class accuracy and F1 scores.

- We have included a train/test split for each class in `provided/supervised_splits.mat`. This is a map object that you can access the split structs using commands like `splits_map('bike')`
- For training, superpixels with a 90% overlap with the foreground are included as positive examples, and 90% overlap with the background are included as negative examples. You may find the provided function `valuesPerLabel` to be useful in extracting from the ground truth if a superpixel is foreground or background.
- You can use any supervised learning algorithm that you want. We achieved the below baseline improvements (Table 3) using Matlab's `fitcknn()`. Make sure you describe in your writeup the learner used and how you decided on the relevant parameters for it.
- When using `provided/evaluateImFGBG()`, you need to pass a third argument of `true` to evaluate the results for the supervised setting. In this setting, the label of '1' corresponds to foreground and '0' to background.
- Remember that at test time, we know which semantic class the image is from. Using the model trained for that semantic class, predict which superpixels are foreground or background.
- **Output:** Generate a bar graph comparing the per-class performance of this in comparison to the unsupervised baseline (you can ignore the error bars if you want). In your writeup, along with including the bar graph, include a table with the per-class *performance change* along with the average performance change across classes. Also in your writeup, include example pictures showing segmentation results and how the results differ qualitatively from the unsupervised results. Discuss failure modes and successes of the supervised approach.

	Avg.	bike	bird	car	cat	chair	cow	dog	face	flower	house	plane	sheep	sign	tree
ΔAcc.	**7.59%**	0.065	-0.14	0.1	0.15	0.085	0.14	0.032	0.09	0.014	0.068	0.22	0.13	0.052	0.046
ΔF1	**0.043**	4.4e-4	-0.15	0.09	0.11	-0.012	0.11	-0.073	0.042	0.045	0.044	0.18	0.18	0.023	0.022

Table 3: Improvement of baseline results for supervised segmentation with separate models per semantic class compared to unsupervised setting. As we know the semantic class of the test image, we use the appropriate semantic class's trained model.

3.2 Harder case: Unknown semantic class

Q6 (20 pts): Let's now consider the harder scenario where we don't know the semantic class of the test image. Create a script called `evaluateSupervisedUnknown.m` that will run and get the per-semantic class accuracy and F1 scores.

- You have a couple of options in how to approach this problem. The easiest is to aggregate training data across all the semantic classes and train one model to predict foreground or background. This is the method used to produce the baseline results below (Table 4)

- You may alternatively consider using each of your individually trained semantic models to predict for each superpixel the class. In the previous question you only had a single prediction per superpixel, but in this setup you would have a prediction for each semantic class for each superpixel. You would then have to figure out a way to combine these to predict foreground or background.
- The baseline results in Table 4 show the performance to be worse in F1 but better in accuracy. How does your method stack up?
- **Output:** Generate a bar graph comparing the per-class performance of this in comparison to the *unsupervised* scenario (you can ignore the error bars if you want). Also include a bar graph showing a comparison to the *known* semantic class setting. In your writeup, along with including the bar graphs, include a table with the per-class *performance change* with respect to the along with the average performance change across classes. Also in your writeup, include example pictures showing segmentation results and how the results differ qualitatively from the unsupervised and known setting results. Discuss why you believe the results you are seeing are better or worse. What makes this problem harder or easier than before? How many you improve the results? Discuss failure modes and successes of this supervised approach.

	Avg.	bike	bird	car	cat	chair	cow	dog	face	flower	house	plane	sheep	sign	tree
ΔAcc.	**3.42%**	0.03	0.033	0.018	0.094	0.044	0.12	-0.043	0.012	-0.021	0.038	0.2	0.024	-5.6e-3	-0.063
ΔF1	**-0.022**	0.03	-0.079	0.017	0.055	0.018	0.047	-0.16	0.019	-0.011	-0.091	0.11	-0.071	-0.049	-0.14

Table 4: Improvement of baseline results for supervised segmentation with a single model for all semantic class compared to unsupervised setting.

4 Extending the pipeline: Extra Credit

QX (20 pts): We have explored unsupervised and supervised segmentation with SLIC and bag-of-words on textons. As we've seen in this course, image understanding is difficult! By trying out new things we can see new failure modes or success in our methods and algorithms. In this problem, you will be doing some 'research' into trying something different or improving the previous results. You can choose from both or either of the below options:

1 Propose ways to improve the unsupervised or supervised results. Then, choose some of your improvements and implement them.

2 Alternatively, you may try out algorithmic changes and see how they effect results (positively or negatively).

Make any of these changes in a `matalb/custom/` directory. You may want to try out a different superpixel algorithm like F-H or mean-shift. You may also want to try another clustering algorithm such as normalized cuts. Can we improve our feature representation? You are free (and encouraged) to download code and use it as long as you mention it in your writeup. Include in your writeup a discussion on the new things you tried, the results they produced, and insight on why the results got better or worse. Extra credit will be awarded based on performance boosts and/or interestingness of tried extensions to the pipeline.

5 What to Submit

Your submission should consist of only 1 file, a zip file named `<AndrewId>.zip`. This zip file should contain

- A folder `matlab` containing all the `.m` and `.mat` files you were asked to write and generate
- A folder `matlab/custom` containing improvements or extensions to the pipeline from **Q6**.
- a `pdf` named `writeup.pdf` containing the results, explanations and images asked for in the assignment. Make sure to include images and discussion showcasing your results for each section.
- Do not include segmentation results for all your images - choose ones that you want to showcase for positive and negative results and weave them into your discussion in your writeup.
- Do not include the `data` directory in your submission or any other temporary files you create in the process of completing this assignment.

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. *EPFL Technical Report*, 2010.

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *TPAMI*, 23(11):1222–1239, 2001.

[3] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *TPAMI*, 24(5):603–619, 2002.

[4] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004.

[5] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *TPAMI*, 31(12):2290–2297, 2009.

[6] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *CVPR*, 2003.

[7] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *TPAMI*, 22(8):888–905, 2000.

[8] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV*. 2006.