CSCI 4831/5722 Computer Vision, Spring 2022
Instructor: Fleming
Homework 5, Due Friday, March 18th, by 11:55pm

Segmentation via Clustering

In this homework assignment, you will use clustering algorithms to segment images. You will then use these segmentations to identify foreground and background objects. And finally, you will transfer foreground objects from one image to another as shown in the figure below.

Provided files: (feel free to use Python; all code files can be converted to .py)
All of the image and Matlab files for this assignment are provided. Source: Stanford Vision Lab (Prof. Fei Fei Li). There are 4 folders:

1. **`code:`** contains Matlab scripts and functions. Some functions have headers but are waiting for your function definition.
2. **`imgs:`** contains 17 images of cats
3. **`gt:`** contains 17 binary images representing the foreground/background segmentation for the images in the **imgs** folder
4. **`test_data:`** synthetic data you can use to test your clustering algorithm solution, as well as the feature normalization solution.

Task 1: Clustering Algorithms: K-Means

As discussed in class, K-Means is one of the most popular clustering algorithms. We have provided skeleton code for K-Means clustering in the file `KMeansClustering.m`. Your first task is to finish implementing the K- Means algorithm using the provided file. You can use `KMeansClusteringTest.m` to test your implementation.

Task 2: Hierarchical Agglomerative Clustering*:*

Another simple clustering algorithm is Hierarchical Agglomerative Clustering, which is sometimes abbreviated as HAC. **Agglomerative** clustering uses a *bottom-up* approach, wherein each data point starts in its own cluster. These clusters are then joined greedily, by taking the two most similar clusters together and merging them.

Next, pairs of clusters are successively merged until we are left with the desired number of predetermined clusters. The result is a tree-based representation of the objects, named *dendrogram*.

Note 1: the Stanford implementation defines "most similar clusters" as the closest clusters by computing the distance between the cluster's centroids.

Note 2: *depending on the implementation, this algorithm is very memory intensive. Consider following the advice of only recomputing the centroid distance for the newly merged cluster.*

We have provided skeleton code for hierarchical agglomerative clustering in the file `HAClustering.m`. Please finish the implementation in this file. You can use `HAClusteringTest.m` to test your implementation.

Task 3: Hierarchical Agglomerative Clustering*: a) single link, b) complete link, c) average link*

There is no given function file and stub for Task 3. Your task is to implement HAC, similarly to Task 2. Here, the "most similar clusters" are defined by the *a) single link, b) complete link, c) average link* **inter-cluster distance***,* as outlined in lectures 16-17 and in the textbook (Ponce, Forsyth, Chapter 9.3.1). Here is an excerpt from the textbook:

What is a good inter-cluster distance?
Agglomerative clustering uses an inter- cluster distance to fuse nearby clusters; divisive clustering uses it to split insufficiently coherent clusters. Even if a natural distance between data points is available (which might not be the case for vision problems), there is no canonical inter-cluster distance. Generally, one chooses a distance that seems appropriate for the data set. For example, one might choose the distance between the closest elements as the inter-cluster distance, which tends to yield extended clusters (statisticians call this method *single-link clustering*). Another natural choice is the maximum distance between an element of the first cluster and one of the second, which tends to yield rounded clusters (statisticians call this method *complete-link clustering*). Finally, one could use an average of distances between elements in the cluster, which also tends to yield "rounded" clusters (statisticians call this method group *average clustering*).

Task 4: Pixel Feature Vectors

Before we can use a clustering algorithm to segment an image, we must compute some feature vector for each pixel. The feature vector for each pixel should encode the qualities that we care about in a good segmentation. More concretely, for a pair of pixels p_i and p_j with corresponding feature vectors f_i and f_j, the distance between f_i

and f_j should be small if we believe that p_i and p_j should be placed in the same segment and large otherwise.

4.1 Color Features

One of the simplest possible feature vectors for a pixel is simply the vector of colors for that pixel. This method has been implemented for you in the file `ComputeColorFeatures.m`.

4.2 Color and Position Features

Another simple feature vector for a pixel is to concatenate its color and its position within the image. In other words, for a pixel of color (r,g,b) located at position (x,y) in the image, its feature vector would be (r, g, b, x, y). Implement this method of computing feature vectors in the file `ComputePositionColorFeatures.m`. You can test your implementation by running `ComputePositionColorFeaturesTest.m`.

4.3 Feature Normalization

Sometimes we want to combine different types of features (such as color and position) into a single feature vector. Features from different sources may have drastically different ranges; for example each color channel of an image may be in the range [0, 1) while the position of each pixel may have a much wider range. Uneven scaling between different features in the feature vector may cause clustering algorithms to behave poorly.

One way to correct for uneven scaling between different features is to apply some sort of normalization to the feature vector. One of the simplest types of normalization is to force each feature to have zero mean and unit variance.

Suppose that we have a set of feature vectors $f_1, \ldots, f_n$ where each $f_i \in R^m$ is the feature vector for a single pixel, and f_{ij} is the value of the j^{th} feature for the i^{th} pixel. We can then compute the mean μ_j and variance σ_j^2 of each feature as follows

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} f_{ij} \qquad\qquad \sigma_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(f_{ij} - \mu_j)^2.$$

To force each feature to have zero mean and unit variance, we replace our feature vectors $f_1, \ldots, f_n$ with a modified set of feature vectors

$$\tilde{f}_{ij} = \frac{f_{ij} - \mu_j}{\sigma_j}.$$

Implement this method of feature vector normalization in the file `NormalizeFeatures.m`. You can test your implementation by running `NormalizeFeaturesTest.m` .

Task 5: Optional Extra Credit for UG students, mandatory for Graduate students

For this programming assignment so far you were asked to implement a very simple feature transform for each pixel. While it is not required, you should feel free to experiment with other feature transforms. Could your final segmentations be improved by adding gradients, edges, SIFT or SURF descriptors, or other information to your feature vectors? Could a different type of normalization give better results? *Add a paragraph to your write up documenting the features vectures you chose and the reason for which you choose them.*

You can use the function `ComputeFeatures.m` as a starting point to implement your own feature transforms.

Task 6: Image Segmentations

After computing a feature vector for each pixel, we can compute a segmentation for the original image by applying a clustering algorithm to the computed feature vectors. Each cluster of feature vectors corresponds to a segment in the image, and each pair of pixels pi and pj in the image will be placed in the same segment if and only if their corresponding feature vectors fi and fj are located in the same cluster.

You can compute a segmentation for an image using the function `ComputeSegmentation.m`. This function allows you to specify the function used to compute features for each pixel, whether the features should be normalized, and the clustering method used to cluster the feature vectors.

For example, to compute a segmentation for the image `img` with 5 segments using K-Means clustering and using `ComputeColorFeatures` to compute pixel features with feature normalization you would write:

```
segments = ComputeSegmentation(img, 5, 'kmeans', @ComputeColorFeatures, true);
```

You can read the full documentation for `ComputeSegmentation` by typing `help ComputeSegmentation`.

If you find that your segmentations take a long time to compute, you can set the optional `resize` argument of `ComputeSegmentation`. If this argument is set then the image will be shrunk before being segmented, and the segmentation will then be upsampled to the size of the original image. You will probably need to use this feature when segmenting images with hierarchical agglomerative clustering.

The syntax `@ComputeColorFeatures` creates a handle to the function `ComputeColorFeatures`; this mechanism allows functions to be passed as arguments to MATLAB functions.

At this point you have a lot of options for computing segmentations: you have two different clustering algorithms (K-Means and HAC), two choices of pixel features (1-just color features and 2- color and position features), and the choice to either normalize or not normalize the features. You can also vary the number of segments that are computed.

Once you have computed a segmentation for an image, you can visualize it using the functions `ShowSegmentation` and `ShowMeanColorImage`. Read the documentation for these functions (using the help command) to see how they are used. Example output from these visualization tools can be seen in the second figure.

You can use the script `RunComputeSegmentation` as a starting point to compute segmentations for images. Choose a few images (either your own or images from the imgs folder) and compute segmentations for these images using different combinations of segmentation parameters (feature transform, normalization, clustering method, number of clusters). A successful segmentation will cleanly separate the objects in the image from each other, while an unsuccessful separation will not.

Task 7: GrabCat: Transfer segments between images Optional Extra Credit for UG students, mandatory for Graduate students

A successful segmentation of an image should separate the objects from the background. Assuming that we compute such a successful segmentation, we can "transfer" objects from one image to another by transferring the segments that make up the object. An example image produced using this procedure is shown in the first figure.

Once you have computed a successful segmentation for an image, you can use the `ChooseSegments` function to choose a subset of these segments to transfer to a background image. The documentation for the `ChooseSegments` function contains more details on how to use it.

Use some of your successful segmentations from the previous section to transfer objects from one image to another. You can use the provided foreground images of cats in the imgs directory and the provided background images in the imgs/backgrounds; also feel free to use your own images. The function `ChooseSegments.m` implements a graphical interface which should aid in selecting a set of segments; use it as a starting point for this section.

Task 8: Quantitative Evaluation. Report

Looking at images is a good way to get an idea for how well an algorithm is working, but the best way to evaluate an algorithm is to have some quantitative measure of its performance.

For this assignment we have supplied a small dataset of cat images and ground truth segmentations of these images into foreground (cats) and background (everything else). You can quantitatively evaluate your segmentations by evaluating their performance on this dataset.

To achieve good performance, you will probably need to divide each image into more than two segments. This means that you will need to combine multiple segments in order to reconstruct the foreground and the background. You can manually choose the foreground segments using `ChooseSegments.m` as in the previous sections. Alternatively, you can have the evaluation function `EvaluateSegmentation.m` automatically choose which segments should be in the foreground and which segments should be in the background.

You can use the script `EvaluateAllSegmentations.m` to evaluate a segmentation method's ability to separate foreground from background on the entire provided dataset. Use this script as a starting point to evaluate a variety of segmentation parameters. Note that you can toggle the `chooseSegmentsManually` variable to either choose foreground segments yourself or allow `EvaluateSegmentation.m` to automatically choose foreground segments for you.

What to include in your report:

a) You should describe all methods of computing feature vectors that you used in the assignment. UG students should use any combinations of color and position data (from Task 2), or add more feature data if they wish to. Graduate students can add gradients, edges, SIFT or SURF features, or any new feature vectors they came up with for Task 3. For each method of computing feature vectors, explain the reason you expect that this feature vector will or will not produce a good segmentation for an image.

Think about it as a hypothesis for an experiment: "I believe this combination will give good results because …". Your report will be graded as much on the results as on the reasoning you provide.

In addition, you should describe all methods of feature normalization that your clustering method employs (task 2).

b) You should include visualizations of at least 6 different segmentations (task 6). At least 3 of these segmentations should be successful, and at least 3 of these segmentations should be unsuccessful. Each of your examples should use different parameters for segmentation, and the parameters for each of your examples should be different.

c) You should also answer the following questions (a few sentences for each question is sufficient):
   1. What effect do each of the segmentation parameters (feature transform, feature normalization, number of clusters, clustering method, resize) have on the quality of the final segmentation?
   2. How do each of these parameters affect the speed of computing a segmentation?
   3. How do the properties of an image affect the difficulty of computing a good segmentation for that image?

d) (Extra Credit for Graduate students) Include at least 2 examples of composite images produced by transferring segments from one image to another (task 7). For each composite image explain how you produced it (i.e. describe what the input images were and what segmentation parameters were used).

e) Include a detailed evaluation of the effect of varying segmentation parameters (feature transform, feature normalization, clustering method, number of clusters, resize) on the mean accuracy of foreground-background segmentations on the provided dataset. You should test **a minimum of 30** combinations of parameters. To present your results, you might consider making a table similar to Table 1.

Feature Transform	Feature Normalization	Clustering Method	Number of clusters	Resize (or max pixels)	Mean accuracy
Position	Yes	K-Means	10	None	0.91
Position + Color	No	HAC	5	0.2	0.58
⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Example table of results; the numbers are made up. You might want to make a table like this in your report.

f) Expand upon the qualitative assessment of Section c) by answering the following questions:

1. Based on your quantitative experiments, how do each of the segmentation parameters affect the quality of the final foreground-background segmentation?

2. Are some images simply more difficult to segment correctly than others? If so, what are the qualities of these images that cause the segmentation algorithms to perform poorly?

3. Also feel free to point out or discuss any other interesting observations that you made.

Note: Overall for this assignment, the focus is more on your choices, the reasoning you provide, and the explanation & discussion, than on the actual code or results (although these are important too!).

Submitting the assignment:
Make sure each script or function file is well commented and it includes a block comment with your name, course number, assignment number and instructor name. Zip all the **.m** and the **write-up** file together and submit the resulting *.zip* file through Canvas as Homework 5 by Friday, March 18th, by 11:55pm.

Rubric:

	Undergraduate	**Graduate**
Q1	15	10
Q2	20	15
Q3	30 (10+10+10)	30 (10+10+10)
Q4.2	10	5
Q4.3	15	10
Q5	10 EC	15
Q6	10	5
Q7	10 EC	10
Q8.		
a)	30	30
b)	10	10
c)	20	20
d)	-	20EC
e)	30	30
f)	10	10
TOTAL	**200**	**200**
EC	20	20