CS131 Computer Vision: Foundations and Applications
(Autumn 2013)
Programming Assignment 2
GrabCat: Foreground-Background Segmentation via Clustering
Due 11/13/2013 5pm

1 Introduction

In this assigment you will use clustering algorithms to segment images. These segmentations will then be used to separate images into foreground objects and background objects. As an application of this technique, you will transfer foreground objects from one image to another as shown in Figure 1.
This will involve several subtasks:

- **Clustering algorithms:** Implement K-Means clustering and Hierarchical Agglomerative Clustering.
- **Pixel feature vectors:** Implement a feature vector that combines color and position information and implement feature normalization. Optionally, get creative and implement your own feature transform.
- **Image Segmentation:** Segment several images using a variety of parameters and qualitatively assess the results.
- **GrabCat: Transfer Segments Between Images:** Transfer objects from one image to another.
- **Quantitative Evaluation:** Quantitatively evaluate segmentations algorithms with a variety of parameters by comparing computed segmentations against a dataset of ground-truth segmentations.

Figure 1: Transfering an object from one image to another. Far Left: Original image. Center Left: Segmented cat. Center Right: Original background. Far Right: Composite image; we "grab" the cat from the first image and place it in the background image.

2 Clustering Algorithms

2.1 K-Means Clustering

As discussed in class, one of the most popular clustering algorithms is K-Means. We have provided skeleton code for K-Means clustering in the file `KMeansClustering.m`.

Finish implementing the K-Means algorithm in the file `KMeansClustering.m`. You can use `KMeansClusteringTest.m` to test your implementation.

2.2 Hierarchical Agglomerative Clustering

Another simple clustering algorithm is *Hierarchical Agglomerative Clustering*, which we will sometimes abbrevieate as HAC. In this algorithm, each point is initially assigned to its own cluster. Then pairs of clusters are merged until we are left with the desired number of clusters; see Algorithm 1.

We have provided skeleton code for hierarchical agglomorative clustring in the file `HAClustering.m`. Finish the implementation in this file. You can use `HAClusteringTest.m` to test your implementation.

Algorithm 1 Hierarchical Agglomerative Clustering

Require: Points $x_1, \ldots, x_m \in \mathbb{R}^n$, desired number of clusters $k \in \mathbb{Z}$
  Assign each point to its own cluster
  while There are more than k clusters **do**
    Compute the distance between all pairs of clusters
    Merge the two closest clusters
  end while

3 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.

3.1 Color Features

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

3.2 Color and Position Features

Another simple feature vector for a pixel is to concatentate its color with 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`.

3.3 Feature Normalization

Sometimes we want to combine different types of features (such as color and postion) 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 $f_i \in \mathbb{R}^m$ is the feature vector for a single pixel, and f_{ij} is the value of the jth feature for the ith pixel. We compute the *mean* μ_j and *variance* σ_j^2 of each feature by computing

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

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 $\tilde{f}_1, \ldots, \tilde{f}_n$ where

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

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

3.4 Get Creative!

For this programming assignment we have asked you 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 or edges or SIFT descriptors or other information to your feature vectors? Could a different type of normalization give better results?

Depending on the creativity of your approach and the quality of your writeup, implementing extra feature vectors can be worth extra credit. You can use the function `ComputeFeatures.m` as a starting point to implement your own feature transforms.

In Your Writeup

In your writeup you should describe all methods of computing feature vectors that you use in your project. For each method of computing feature vectors, explain why you expect that this feature vector will or will not produce a good segmentation for an image.

In addition you should describe all methods of feature normalization that you employ.

4 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 p_i and p_j in the image will be placed in the same segment if and only if their corresponding feature vectors f_i and f_j 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 (just color features or 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 the computed segmentation 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 Figure 2.

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.

In Your Writeup

In your writeup you should include visualizations of at least 4 different segmentations. At least 2 of these segmentations should be successful, and at least 2 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.

You should also answer the following questions in your writeup (a few sentences for each question is sufficient): 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? How do each of these parameters affect the speed of computing a segmentation? How do the properties of an image affect the difficulty of computing a good segmentation for that image?

5 GrabCat: Transfer Segments between Images

A successful segmentation of an image should separate the objects from the background. Assuming that we compute such a successful segmentation for an image, we can "transfer" objects from one image to another by transfering the segments that make up the object. An example image produced using this procedure is shown in Figure 1.

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

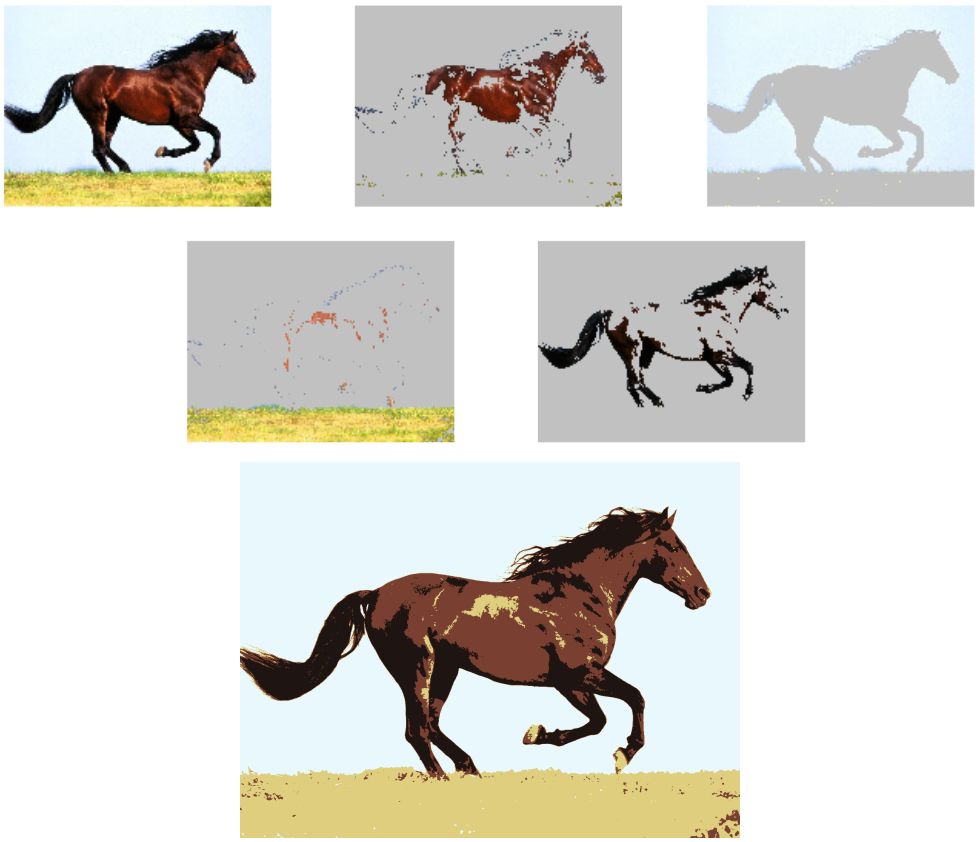

Figure 2: Visualizing a computed segmentation. Top: Visualize individual segments using `ShowSegmentation` Bottom: Visualize the segmented image as a whole using `MakeMeanColorImage`; the pixels in each cluster have been replaced with the average color of all pixels in that cluster.

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 foreground and background images.

You can use the script `GrabCat.m` as a starting point for this section.

In Your Writeup

Include at least 2 examples of composite images produced by transfering segments from one image to another. For each composite image explain how you produced it: what were the input images and what segmentation parameters were used?

6 Quantitative Evaluation

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 project we have supplied a small dataset of cat images and ground truth segmentations of these images into foreground (cats) and background (everything else). We will 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 fuction `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.

In Your Writeup

Include a detailed evaluation of the effect of the 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 10 combinations of parameters. To present your results, you might consider making a table like Table 1.

You should expand upon the qualitative assessment of Section 4 and try to answer the following question: based on your quantitative experiments, how do each of the segmentation parameters affect the quality of the final foreground-background segmentation? 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? Also feel free to point out or discuss any other interesting observations that you made.

You may combine your discussion in this section with your discussion from Section 4 if you like. However, your grade for both sections will depend on the thoroughness and correctness of your discussion. Just as a point of reference, we anticipate that none of the questions above can be answered in a single sentence; you should back up all answers with either experimental data or a convincing argument. Overall for this assignment, we care more about your explanation and discussion than about your actual code or results (although those are important too!).

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.

References

[1] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[2] Richard Szeliski. *Computer vision: algorithms and applications*, pages 284–286. Springer, 2011.