10-601: Homework 6

Due: Sunday, 9 November 2014 11:59pm (Autolab)

TAs: Daniel Ribeiro Silva, Jingwei Shen

Name: Wenhan Lu

Andrew ID: wenhanl

Please answer to the point, and do not spend time/space giving irrelevant details. You should not require more space than is provided for each question. If you do, please think whether you can make your argument more pithy, an exercise that can often lead to more insight into the problem. Please state any additional assumptions you make while answering the questions. You need to submit a single PDF file on autolab. Please make sure you write legibly for grading.

You can work in groups. However, no written notes can be shared, or taken during group discussions. You may ask clarifying questions on Piazza. However, under no circumstances should you reveal any part of the answer publicly on Piazza or any other public website. The intention of this policy is to facilitate learning, not circumvent it. Any incidents of plagiarism will be handled in accordance with CMU's Policy on Academic Integrity.

⋆: Code of Conduct Declaration

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: No (e.g. *Jane explained to me what is asked in Question 3.4*)
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: No (e.g. *I pointed Joe to section 2.3 to help him with Question 2*).

⋆: Notification

If you have any questions, please post it on Piazza or email:

Daniel Ribeiro Silva: drsilva@andrew.cmu.edu

Jingwei Shen: js1@andrew.cmu.edu

1 Clustering in Computer Vision: Image Segmentation

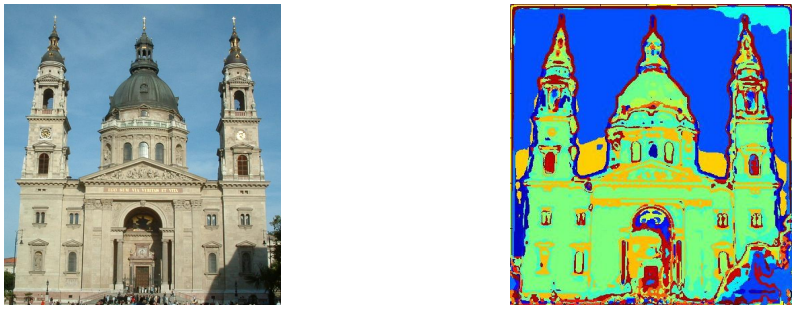
In this homework you will apply your knowledge of clustering to a common problem in computer vision: image segmentation. Don't worry, no computer vision knowledge is required to complete this assignment. All computer vision and image analysis code will be provided to you. You'll only need to focus on implementing the clustering algorithm and some simple code for creating and visualizing the segments.

1.1 Introduction

In computer vision, it is common to want to divide a given image into homogeneous regions. This task is commonly referred to as image segmentation. Image 1 below shows an example of such a task, where each segment is represented by a different color. It is very interesting to notice that this algorithm is completely unsupervised (we never provide any labels).

One of the many possible approaches to achieve such results is to do clustering and group similar pixels together. However, doing clustering directly on the pixels themselves tends to yield bad results, since individual pixels are unable to encode local features such as color gradients or edges/contours. We can solve this problem by applying filters to the image and extracting a feature vector (based on the filter responses) for each pixel. These filter responses will encode local features of the image and will result in a much better segmentation performance.

For this homework, you don't need to understand how such filters work, or how we obtain the feature vector. The code for that will be provided. All you need to keep in mind is that each pixel (x,y) of the image will be associated with a 99-dimensional feature vector, and your task will be to do clustering on these vectors.



(a) Original (b) Segmented

Figure 1: Image segmentation using visual words approach

1.2 Segmentation by Visual Words

In this homework, we'll use a simplified version of the visual word approach to do segmentation on images. This method consists of 3 steps:

1. **Extract feature vectors:**
   Each pixel of each image will be represented as a 99-dimensional feature vector (or point). This feature vector is a result of the responses of filters applied to the image. We will provide

a method to extract the feature vectors.

2. **Cluster points (compute dictionary):**
We will use K-Means to cluster the points obtained in the previous step. A good choice of K for this problem is a number between 100 and 300.
At this step we'll obtain K clusters. Each cluster is represented by its center and is a **visual word**. I.e., upon clustering we'll obtain K visual words. The set of all K visual words is a **dictionary**.

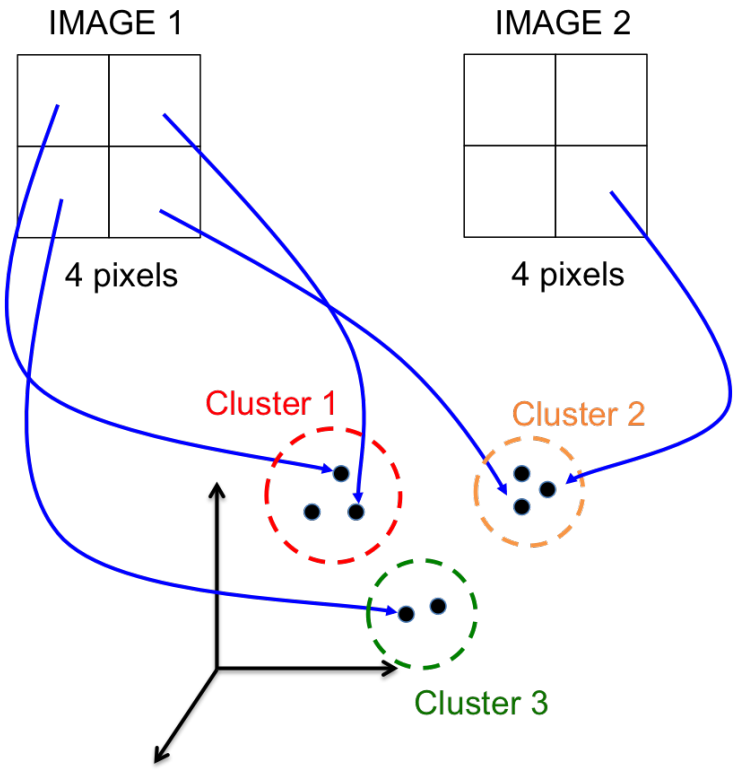


Figure 2: Feature vector extraction and clustering

3. **Segmentation:**
Once the dictionary is computed at step 2, we'll perform the segmentation. Basically, each visual word will correspond to a segment in the image. For any given image, we'll extract its feature vectors (one per pixel) and check what cluster it belongs to. In other words, for each feature vector of this new image we will find the cluster center (visual word) it is closest to and associate the corresponding pixel to that cluster (visual word). More specifically: we have K clusters $(c_1, \ldots, c_k)$. Each cluster is represented by a point in a 99-dimensional space (our feature space). For each pixel p_i in any given image, we extract its feature vector f_i (a point in this same 99-dimensional space) and we'll find its closest cluster. If the closest cluster to f_i is, say, c_5 then pixel p_i will belong to the cluster/visual word 5. Once we do that for all pixels, our image will be represented by the cluster each pixel belongs to. If we paint each cluster with a different color, then we'll get our segmentation (just as in Figure 1).

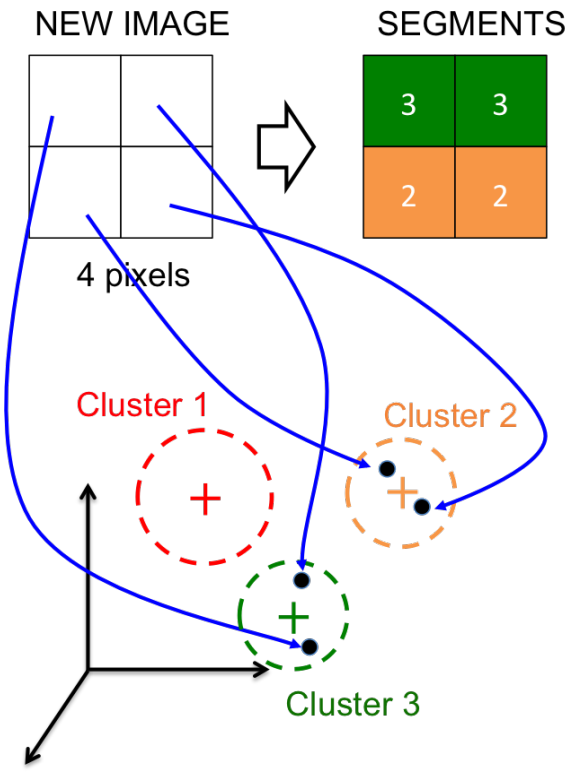


Figure 3: Segmentation of images based on clusters

1.3 Dataset

In this homework, you will use a dataset containing 173 different images of basilicas. They are part of the SUN dataset (http://groups.csail.mit.edu/vision/SUN/).

1.4 Provided Files

Since we are dealing with images, we will use some image analysis libraries, which have different behaviors in Matlab and Octave. For this homework, you can choose to develop on Matlab or on Octave. In the provided files you'll find two folders: *matlab* and *octave*. If you are developing in Matlab, please use the files from the *matlab* folder and if you are developing in Octave, use the files from the *octave* folder. If you choose to develop in Matlab, please just make sure that your K-Means method also runs in Octave without any problems. Further instructions about the provided code can be found in the sections below.

1.5 Getting Your Hands Dirty

Now it's time for you to start doing some work. We are providing all the image analysis and computer vision methods, but it will be up to you to implement the clustering algorithm and to generate the final segmentation based on the result of your clustering.

This homework consists of 3 parts, and you must do them in this order, since each step depends on the previous one:

1. Implement K-Means
2. Compute dictionary (by completing and running *computeDictionary.m*)
3. Visualize segmentation (by completing and running *segmentation.m*)

1.5.1 K-Means Clustering [60 Points]

Your first task is to implement the K-Means algorithm. Just as in previous homeworks, you'll have to implement it in Matlab/Octave. We have already created a file with a method having the signature below. Your job is to write the implementation of this method (open the existing file and write your code in there). We provide a script for testing tour code (*testKMeans.m*). Make sure you run this test and get a coherent result.

```
[clusterCenters, clusterBelonging] = k_means(data, k, startPoints)
```

where

- *data*: the points to be clustered. It is a $N \times F$ matrix containing N points of dimension F.
- K: the number of clusters to be generated by K-Means.
- *startPoints*: the starting points of your K-Means algorithm. It is a $K \times F$ matrix containing K points of dimension F. This is an optional argument, which means that your code should be able to handle the case where no start points are provided. I.e., it should be able to generate such points. There are many ways of doing that. A simple option is to select K random (and distinct!) points from *data*.
- *clusterCenters*: the centers of the clusters found by K-Means. It is a $K \times F$ matrix containing the K cluster centers obtained.
- *clusterBelonging*: the cluster each data point belongs to. Each cluster is represented by an integer from 1 to K and respects the order of the *clusterCenters* matrix. In other words, the first cluster in *clusterCenters* is represented by 1, and so on. The output variable *clusterBelonging* is a $N \times 1$ matrix.

Make sure that this particular file runs without errors on Octave (even if you are developing in Matlab!).

1.5.2 Computing the Dictionary [20 Points]

Now that you have implemented the K-Means clustering algorithm, we will use it to compute the dictionary of visual words. in this part, you will implement steps 1 and 2 described in Section 1.2.

We are providing you a file called *computeDictionary.m* as a base for this part. You must implement your code in that file. The file contains some base structural code to guide you.

For this part, we are giving you a file called *data.mat* which contains two objects:

- *imagePaths*: a cell array containing the path to (i.e. file name of) the images on the dataset. The images are located in the folder called *basilica*. Make sure you don't change the structure of the provided folders and files.

- $filterBank$: list of filters used to generate the feature vectors. Don't worry too much about this object. Just use it as specified below.

In this part, you will read all images from $imagePaths$, extract the feature vectors for each pixel and then perform clustering to obtain the dictionary.

OBSERVATION 1: You will perform K-Means only once, with the feature vectors of all the images in the dataset - there is not a separate clustering process for each image.

OBSERVATION 2: The number of feature vectors obtained from all images is huge (many hundreds of thousands of them). Running K-Means on all these points would be too slow. We highly recommend you to run it only for a sub-sample of them. For example, you could randomly select 1% of the points and then perform clustering on them to obtain your dictionary of visual words.

Your dictionary is simply the set of the cluster centers obtained in K-Means (i.e. the $clusterCenters$ output variable). It will be a matrix of size $K \times F$, where K is the number of clusters/visual words and F is the dimension of the feature space. In this case, $F = 99$ and you are free to choose K. We strongly suggest an integer between 100 and 300.

Create a variable called $dictionary$ that contains that result (you can simply do $dictionary = clusterCenters$;) and save it to a file:

```
save('dictionary.mat', 'dictionary');
```

Below you will find the instructions to get the feature vectors for each pixel and some other notes (most of this code is already provided anyways).

NOTE 1: In order to get the path of the $i-th$ image (and store it in a variable called img_path), type:

```
img_path = imagePaths{i};
```

NOTE 2: In order to read the image with the path stored in img_path (and store it in a variable called I) type:

```
I = imread(img_path);
```

NOTE 3: In order to get the feature vectors for the pixels in an image I type:

```
featurePoints = extractFilterResponses(I, filterBank);
```

The variable $featurePoints$ in the example above will be of size $N \times 99$ where N is the number of pixels in the image I.

1.5.3 Doing the Segmentation [20 Points]

Now that you have computed the dictionary of visual words, we will perform segmentation in a new image. You will select your own image to perform segmentation on. In order to do so, find your own JPG image of a basilica (use Google, Bing, Baidu, ...) and save it as *myBasilica.jpg* in your root folder (together with your other scripts).

NOTE: If the size if your selected image is huge, we suggest resizing it to something like 400 pixels of width or height. Also, try to find an image that is not too small.

We are providing you a file called *segmentation.m* as a base for this part. You must implement your code in that file. The file contains some base code to guide you.

In this script, you will read your selected image *myBasilica.jpg*, extract the feature vectors for each of its pixels and find their corresponding clusters based on the dictionary you computed in the previous section. Basically, what you'll be doing here is finding, for each feature vector, its closest cluster. The function *pdist2* can be particularly helpful in this task.

If your image has dimension $w \times h$ then your final result should be a matrix of size $w \times h$ with the cluster belonging for each pixel. This matrix must be named *imageSegments*. In other words, if the pixel (i, j) has a feature vector that belongs to cluster c then $imageSegments(i, j) = c$.

HINT: If you want to reshape a list of elements into a matrix, you can use the method *reshape*. In this particular case, your code would look something like:

```
imageSegments = reshape(closestCluster,[size(I,1) size(I,2)]);
```

Once you obtained your *imageSegments*, it is very easy to visualize it. All you have to do is use the *imagesc* method, as shown below:

```
imagesc(imageSegments);
```

The result should be something like the one in Figure 1.

Save the image you generate of the segments as *myBasilicaSegmented.jpg* and submit it together with your original *myBasilica.jpg* image.

1.6 Submission

The submission should be a tgz file containing ONLY "report.pdf", "k_means.m", "computeDictionary.m", "dictionary.mat", "segmentation.m", "myBasilica.jpg", and "myBasilicaSegmented.jpg". **Please do not change file names**. Use the following command to create the tar file:

```
tar -cvf hw6.tgz report.pdf k_means.m computeDictionary.m dictionary.mat segmentation.m
myBasilica.jpg myBasilicaSegmented.jpg
```

Your *report.pdf* file should contain your Code of Conduct Declaration, your choice for the value of K, and the image you selected with the segmentation result (something like Figure 1).

Autolab will only grade your K-Means implementation. The rest of the submission will be hand-graded.

Good luck and have fun!