k-Means Image Segmentation

COSC450 Assignment 2

Due: 28th September 2015, 5pm

This assignment is worth 20% of your final grade.

1 Overview

Images often contain many different objects. One common task in many image processing and computer vision systems is *segmenting* the image into different objects or components. This is a difficult task, and often relies on domain-specific knowledge. However, simple techniques can be quite effective.

In this assignment you will implement and experiment with the k-means algorithm. This is an algorithm for grouping data elements into a set of k partitions, where k is a parameter provided to the algorithm. While k-means is used in a wide variety of applications, for this assignment the data elements are pixels from an image, and the task is to assign a set of k labels to the pixels, dividing the image into k partitions.

Figure 1 shows a simple example of this. The input image contains a number of different colours of object. When k-means is applied on the basis of red-green-blue triples, these objects can be separated out. The choice of k is important however. If k is too small then some colours are grouped together, while if k is too large then some colours are assigned a mix of two labels.

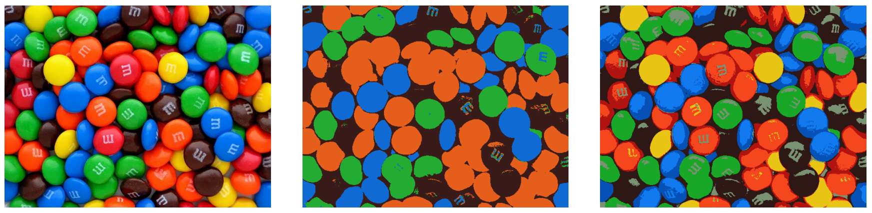

Figure 1: An example of k-means segmentation. Pixels from an input image (left) is divided into k clusters. When $k = 4$ (center), the red and yellow objects are grouped together. Increasing k to 8 resolves this, but the blue and red objects now have two distinct clusters each (a darker and lighter version of each colour).

2 Assignment Requirements

Note: OpenCV provides an implementation of k-means. For this assignment you are required to implement the algorithm yourself.

2.1 Basic k-Means

The basic k-means algorithm is fairly straightforward. Given a set of n data points, k initial cluster centres are selected. Each point is then assigned to the cluster centre that it is closest to. The cluster centres are then updated to be the average of all the points assigned to that cluster. This process repeats until the clusters are stable, and is summarised in Algorithm 1.

Algorithm 1 Basic k-means algorithm.

```
Input: A set of n data points, $p_1, p_2, \dots p_n$; The number of clusters, $k$.
Output: A set of n labels, $\ell_1, \ell_2, \dots \ell_n$, where $1 \le \ell_i \le k$.
 1: Initialise $k$ cluster centres, $c_1, c_2, \dots, c_k$.
 2: while not done do
 3:   for each data point, $p_i$ do
 4:     $\ell_i = 0$
 5:     $d_{\min} = \infty$
 6:     for each cluster centre, $c_j$ do
 7:       $d_j$ = distance from $p_i$ to $c_j$
 8:       if $d_j < d_{\min}$ then
 9:         $\ell_i = j$
10:         $d_{\min} = d_j$
11:       end if
12:     end for
13:   end for
14:   for each cluster centre, $c_j$ do
15:     $c_j = \text{average}\,(p_i,\ \text{such that } \ell_i = j)$
16:   end for
17: end while
```

There are a number of things here which need to be refined, including:

- How do you pick the initial cluster centres?
- What is the distance between a data point and a cluster centre?
- How do you tell when you have finished?

There are a number of different options for this, and they are discussed in the following sections.

2.1.1 Initialisation Methods

The choice of initial cluster centres can have a large effect on the result, and there are a number of different approaches. Some of the common ones include:

Random selection Choose k data elements at random to act as initial cluster centres

Random clustering Assign each data element to one of the k clusters at random. Use the average of these clusters as the initial cluster centres.

***k*-means++** Choose the first cluster centre at random, then choose the remainder so that points which are distant from the centres that have already been chosen are more likely. See https://en.wikipedia.org/wiki/K-means++ for more details.

You should implement at least two of these, and experiment with them to see what difference they make.

2.1.2 Distance Metrics

The distance between data points can be computed in a number of different ways. There most straightforward method for pixels is to treat the red, green, and blue values as vectors and to use the usual Euclidean distance. Given two pixels, p_1 and p_2, with RGB values $\{r_1, g_1, b_1\}$ and $\{r_2, g_2, b_2\}$ this distance is

$$d(p_1, p_2) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}.$$

This works well in some cases, but there are some potential issues with this approach.

Firstly, the colour values depend on lighting as well as the colour of the object. Parts of an object in shadow are darker, so may appear quite different. One way to overcome this is to convert from RGB colour to another representation such as HSV. HSV divides colour into its hue (basic colour, such as green or yellow), saturation (how intense the colour is) and value (how light or dark the colour is). Value then could be ignored, and just the hue and saturation values used to determine the distance between pixels.

A second issue is that this metric doesn't take location into account. Pixels on the same object are generally close to each other in the image, and this information is ignored when just colour values are used. A direct way to include location information is to include the pixel's co-ordinates to the distance function. This could lead to

$$d(p_1, p_2) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2},$$

where p_1's image location is (x_1, y_1) and p_2's location is (x_2, y_2). This helps in some cases, but in others (like the example from Figure 1) it may not. It also raises questions about comparing colour to pixel distance. Is a difference of 10 between two red values more or less significant than a distance of 10 pixels in the image?

Finally, information from a single pixel may not be sufficient to segment an image. Consider the picture in Figure 2 for example. Most people would consider this image to have four distinct regions, but there are only two colours. In order to overcome this, a pixel's description for clustering could be comptued from a small window around it. This could be the average and standard deviation of the neighbouring pixels, a histogram of pixel values, or based on gradient computations.

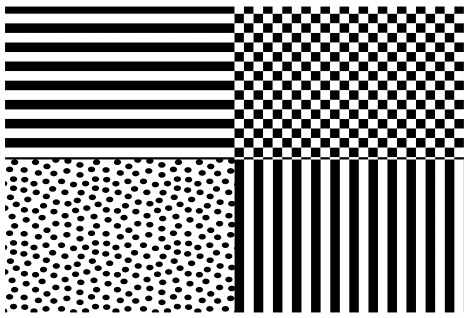

Figure 2: An image which has four distinct regions but only two colours.

You should implement at least two different distance measures and compare them in your assignment.

2.2 Termination Conditions

The k-means algorithm iteratively updates the cluster centres and segmentation, and you need to determine when to stop this process. There are several criteria which can be used to determine when the process has converged:

- Run for some (large) number of iterations. This is easy but if the number of iterations is too small the segmentation might not be optimal, and if it is too large then the later iterations might not change the result at all.
- Check to see if the cluster centres are changing between iterations. This often checks to see how much change there is, and stops if the change is very small.
- Check how many data points change labels between iterations. Again, you could wait until no points change labels, or until very few points are updated.

You should implement some reasonable termination criterion and discuss it in your report. You should be able to guarantee that your program will terminate in reasonable time (no infinite loops, please).

2.3 Experiments

Computer vision algorithms deal with measurements (images) of the real world. As a result, experiments are of particular importance when evaluating computer vision algorithms. Designing good experiments can be difficult, but here are some things to keep in mind:

- A good experiment should answer a question. You should have a clear question in mind and design an experiment that you expect to have different outcomes based on the possible answers to the question. This is not always easy, because often our questions are not clear, but trying to clarify your questions can make it much easier to decide what experiments you need to do.

- Your experiments should enable you to tell what is the cause of any changes in observations. For example, suppose you have two initialisation methods, Init1 and Init2, and two distance metrics, Dist1 and Dist2. You find that when you run your program with Init1 and Dist1 it performs much better than with Init2 and Dist2. What does this tell you? There are a lot of possibilities:

  - Init1 might be much better than Init2
  - Dist1 might be much better than Dist2
  - Init1 and Dist1 might work particularly well together
  - Init2 and Dist2 might not be well suited to each other
  - And so on. . .

  A better design could be to fix the initialisation method and compare distance metrics, or to compare all possible combinations of methods.

- The choice of images is important. The image in Figure 1 is very well suited to segmentation based on colour information, but that in Figure 2 is not. Experiments on a single image rarely tell you anything for sure, and there isn't a 'typical' image, or even set of images that can be used. A few things that might help for your assignment are:

  - There are a lot of image data sets that are commonly used for different applications. These are scattered around the internet, and you might want to make use of some of them.
  - You might want to restrict your experiments to a specific problem. For example you might want to consider a task such as segmenting faces from the background. Your test images then would be a set of images with faces in them (and some without), and you could evaluate how well your method works on them.
  - There are a lot of different aspects to 'good' performance. The quality of the output is obviously important, but can be hard to judge. For this assignment it is OK to describe the differences qualitatively (e.g.: Method X works well on brightly coloured objects, but not for more subtle differences). Other differences, such as average run time or memory usage can be more objectively measured.

Many of these questions are particularly difficult in the case of image segmentation, since there is no agreement as to what a 'correct' segmentation is. I *do not* expect you to solve this problem – if you have a good idea how to, then come and talk to me about doing a PhD!

What I am looking for is some thought given to how to evaluate different variants of the k-means algorithm. For example, suppose you have implemented RGB-based and HSV-based methods. HSV is supposed to work better when there is a range of lighting across areas which have the same underlying colour. A good experiment would use a set of images where this is the case, and see how the two methods compare. You would expect that the HSV-based method would group similar colours regardless of light and shadow, while the RGB-based algorithm would separate the brightly lit areas from the shadows.

3 Resources

OpenCV is available on the lab machines in ∼`steven/Public/OpenCV`. You don't need to take a copy, you can just link to that from your code. A sample program (C++ source, Makefile, etc.) is in `~steven/Public/OpenCVTest`. To compile and run that, copy the contents of that directory somewhere then use the commands

```
make
source setPath.sh
./ocvTest
```

This builds the program, makes sure that the OpenCV libraries are in the library search path, and then runs the program. This should pop up a window with a bulls-eye pattern in it. Selecting that window and pressing any key will quit the program.

Another example is given in ∼`steven/Public/Segmentation`. This example loads an image, waits for a key to be pressed, and then segments the image by dividing the pixels into brightness. It can be built and run with the following commands:

```
make
source setPath.sh
./segmentation <input image> <k> <output image>
```

Where the parameters are the image to be segmented, the number of divisions, and a file in which to save the result. This code may be used as a starting point for this assignment, and the segmentation is done in a function called `kMeans` (although it does not do k-means yet).

The version of OpenCV provided is 2.4.9. Online documentation is available, including tutorials and a reference manual.

4 Deliverables

In summary your program should:

- Implement the k-means algorithm.
- Implement the two initialisation methods, as discussed in Section 2.1.1.
- Implement at least two distance metrics, as discussed in Section 2.1.2.
- Implement a sensible termination criterion as discussed in Section 2.2

The user should be able to choose between different methods with command line arguments.

Your report should:

- Give clear instructions for building and running your program on the lab machines.
- Discuss the methods you have implemented, how they work, and why you chose them.
- Present clearly experiments comparing the different methods you have implemented.
- Discuss the relative strengths and weaknesses of different approaches, using the results of your experiments as evidence.

You should send your report (PDFs preferred), source code (including a `Makefile`), and any supporting files required to steven@cs.otago.ac.nz. Your code should build and run on the lab machines. You may find it convenient to archive your files as a `.tar` or `.zip`, but be aware that archives sent from outside the department often get caught in the University's spam filters. This can be easily circumvented by renaming your files to remove the archive extension. If your submission is zipped up as `astudent450ass2.zip` then just rename it to `astudent450ass2.piz` or something and tell me what you've done in the body of your email. I will reply to acknowledge receipt of your assignment in any case.

4.1 Marking Scheme

Marks for the assignment will come from an examination of your code, the performance of your program, and your report. Marks will be divided as follows:

- 20% for your code. As with Assignment 1 clarity and correctness are the main concerns here. Comments, naming conventions, and appropriate division of code into functions and modules all help with this. While efficient methods are preferred, you should worry primarily about making sure that your program does what it should, and that this is clear to people reading your code.

- 20% of the marks will come from your choice, implementation, and discussion of different initialisation methods. For high marks you should implement relatively complex methods, and may wish to implement more than two.
- 20% of the marks will come from your choice, implementation, and discussion of different distance metrics. Again, for high marks you should implement relatively complex methods, and may wish to implement more than two.
- 40% of the marks will come from the design of your experiments and your discussion and analysis of the experimental results. I will be looking for experiments which are designed to highlight the difference between the methods you have chosen to implement, results on a range of different types of images, and clear presentation and analysis of the results of these experiments.

Late assignment submissions will be penalised at the rate of 10% per working day. Extensions to the deadline may be granted where appropriate, but should be sought well in advance. The usual university regulations relating to plagiarism apply (http://www.otago.ac.nz/study/plagiarism/index.html), and any work you submit must be your own, or be clearly attributed to the original author.