Dave Durbin

Dell Zhang

Intelligence Based Image Processing

24 March 2012

Image Segmentation using K-Means and Normalized Cut algorithms

Introduction

Human beings' primary source of information about the world around them is generated through their senses and chief amongst these is vision. The ability to comprehend the stream of incoming visual data is one of the most important and equally one of the most difficult to replicate with computers.

Amongst the tasks required is the ability to divide an image into components parts which can then be used to determine what is actually happening in the image. Humans are very good at this problem but as gestalt theorists point out, there is no categorically 'correct' segmentation of an image.

This aim of this project was to build and analyse implementations of K-Means and Normalised Cut image segmentation algorithms. Algorithms were constructed in Matlab and then tested against images selected from the Berkeley Image Segmentation Corpus which has the advantage of providing ground truth data based on human interpretation.

This report is organised as follows. Section 2 describes the segmentation problem and discusses the algorithms considered by this project. Section 3 describes the approaches adopted to developing the code and presents experimental results.

2 Image Segmentation

Image segmentation refers to the task of dividing an image into multiple segments or groups of pixels so that it becomes more tractable to analysis or description. Segmentation algorithms apply labels to each pixel in an image where the labels represent discrete areas that share some common feature or visual characteristic.

K-Means Clustering

The k-means clustering is a data mining algorithm that subdivides a set of data into a number of clusters such that each element of a cluster is closer to the mean of that cluster than to the mean of any other cluster. i.e. it minimises the within cluster sum of square distances from the cluster mean.

$$\arg_s \min \sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2$$
$$where\ \mu_i\ is\ the\ mean\ of\ the\ points\ in\ S_i \tag{1}$$

Determining the clusters is an NP-hard problem however there are a number of heuristic algorithms which converge rapidly to a local minimum. For this project we implement the k-means algorithm using Lloyd's algorithm [1]. This is a 3 step iterative process as follows:

1. Assign the initial data points into k categories using either random selection or some heuristic
2. Calculate the centroids of the k clusters via some metric, usually a Euclidean measure of distance between points
3. Reassign the points to the clusters with the nearest centroid, using the same metric.

Steps 2 and 3 are repeated until the algorithm converges i.e. there is no change in the clusters.

Because this algorithm may not converge, real world implementations may terminate after some other criteria is met, e.g. the furthest distance moved by a point is less than some threshold.

Matlab's implementation of k-means uses a second phase which according to the documentation:

"...uses online updates, where points are individually reassigned if doing so will reduce the sum of distances, and cluster centroids are recomputed after each reassignment. Each iteration during the second phase consists of one pass though all the points."

K-means clustering is very sensitive to the initial choice of centroids and better results can be obtained by careful selection of these seeds or by running the algorithm several times with different randomly selected starting points which can lead to finding a global minimum.

The application of k-means clustering to image segmentation requires the use of the k-means algorithm to separate the pixels in an image into 'like' clusters. This means selecting some metric which can be used for the within cluster distances. Candidates are image colour or brightness values and pixel spatial data.

This project implemented two main components:

•An image segmentation algorithm based on the k-means algorithm

•An implementation of the k-means algorithm

Normalised Cut

In [2]Jianbo Shi proposed a graph theoretic criterion for measuring the goodness of an image partition. An image is represented as a graph where each pixel is a node

in the graph and the edges connecting these nodes are weighted with a measure of the similarity of the pixels.

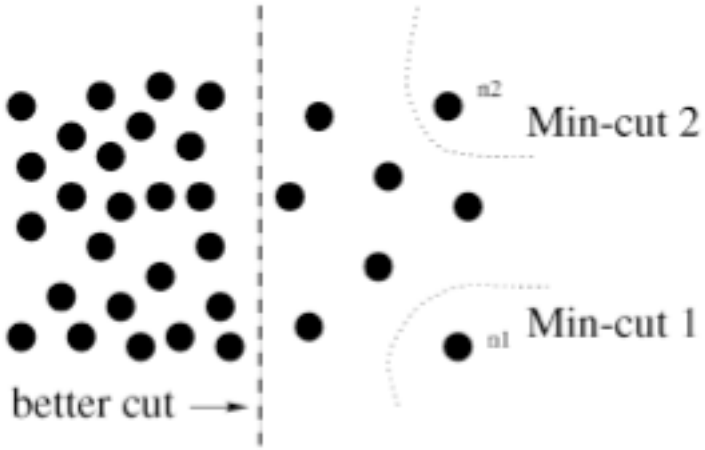
We can then partition the graph by removing edges, into two disjoint sets A and B such that

$$A \cap B = V \; and \; A \cup B = \varnothing \tag{2}$$

Since the edges in the graph represent the similarity of connected nodes (pixels), then the sum of the weights of the edges removed from the graph form a representation of the dissimilarity of the two partitions. In graph theoretic terms this is called a *cut.*

$$cut(A,B) = \sum_{u \in A, v \in B} w(u,v) \tag{3}$$

The task of making a minimum cut is a well studied problem and Wu and Leahy in [3]propose an image segmentation solution based on this approach. While it works well for some images, an issue noted by Wu and Healy is that it tends to favour cutting out small sets of isolated nodes form the graph.



Shi and Malik proposed an alternative measure for the cost of a cut which overcomes this problem. The normalised cut computes the cost of the cut as a fraction of the total connections from that node to all other nodes in the graph.

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}, \tag{4}$$

This new measure overcomes the issue with isolated points since although the weight of the edges connecting them to the remainder of the graph may be low, it is still likely to be a high proportion of the total weights of edges connected to this node.

In his paper, Shi showed that the problem of minimising Ncut(x) is that of minimising the objective function:

$$min_x Ncut(\boldsymbol{x}) = min_{\boldsymbol{y}} \frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}, \tag{5}$$

with the constraints that

$$y_{(i)} \in \{1, -b\}\, and\, y^{\mathrm{T}}\boldsymbol{D}\mathbf{1} = \mathbf{0} \tag{6}$$

Shi showed that the task of minimising this normalised cut cost exactly is an NP-complete problem. He went on to show that if the problem domain was relaxed to real valued solutions, then an approximate discrete valued solution could be found efficiently.

This is an NP-complete problem however if the problem domain was relaxed to real valued solutions, then an approximate discrete valued solution could be found efficiently. The approach to this is to minimise

$$(\boldsymbol{D}-\boldsymbol{W})\,\boldsymbol{y}=\lambda\boldsymbol{D}\boldsymbol{y} \tag{7}$$

This is a generalised eigensystem and if we solve for y and extract the second smallest eigenvector then this provides a real valued solution to our problem. We can discretize this solution by thresholding the eigenvector such.

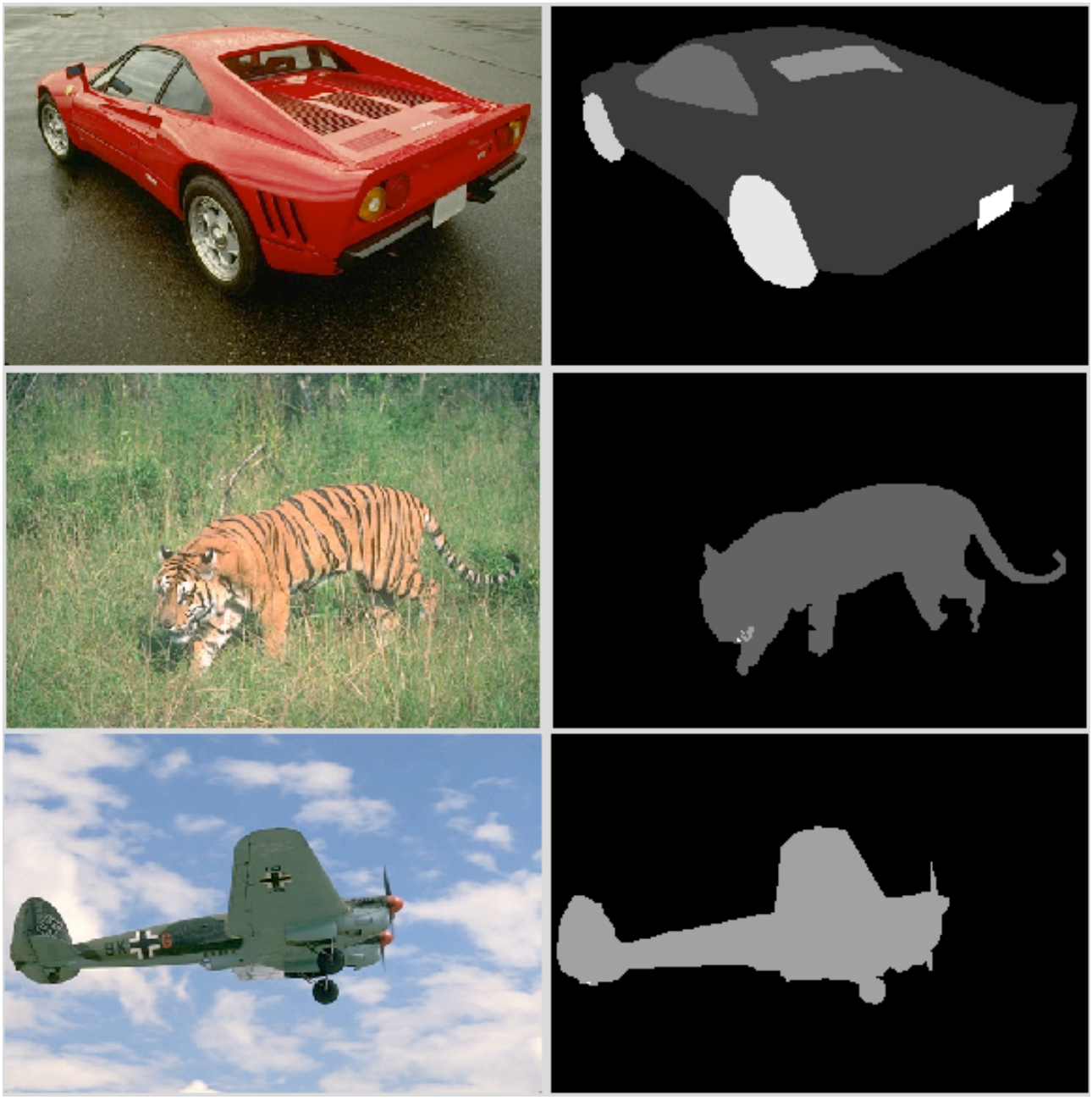
Given that the eigenvector has N elements, there are at most N+1 possible threshold values. Finding the optimal threshold is simply a question of evaluating (5) for each [4].

3 Approach

K-Means

The general approach to this problem was to attempt to develop image segmentation code using Matlab's built-in `kmeans` function. Once this achieved a reasonable quality of segmentation, a k-means implementation was built from scratch using Lloyd's algorithm as described above. The interface to this function was made common with that provided by Matlab so that the two algorithms could be directly compared.

The images used for experimentation were taken from the Berkeley Image Database. They're shown below with their ground truth data.



Images are 29030 (car), 3063 (plane), 108004 tiger. Ground truths are 8 segs, 5 segs, 3 segs

Segmentation with k-means

The two principle problems to be solved when using k-means for image segmentation are:

- What is the metric to be minimised
- What starting seeds should be used

RGB Segmentation

An initial naive implementation was built which used vectors of RGB values. A segmentation of the car image gave the following.

This is a fairly poor segmentation. The grey road and silver wheels have been included with the pale red body work in the right segment while in the central segment again, grey and red pixels are mixed.

This was expected since the RGB space is not uniform or perceptually linear. For example, consider the image below. Here all of the colours on the top row are equidistant (using a Euclidean measure of distance) from the colour on the bottom row.

RGB colours are from left to right and top to bottom: (128,192,128), (192,128,128), (128,128,192), (64,128,128), (128,64,128), (128,128,64) and (128,128,128)

L*a*b*

As a second experiment and in an attempt to improve on this, the RGB image was converted into the L*a*b* colour space. The rationale here is that colour is an important determinant in human segmentation of images and by using L*a*b* all colour information is stored in the a* and b* elements so the Euclidean distance of the 2D a*b* vectors could be used as a metric. Perceptually this is a more pleasing segmentation.

*Segmentation using k=3, a*b* data only*

Aircraft segmentation with k=3.

With the aircraft image, the algorithm was often unable to separate the aircraft from some of the cloud formations. This did vary with the random seed start point and sometimes the segmentation was a lot cleaner.

The tiger image segmented very well with k=3.

HSV Based Segmentation

The final colour segmentation experiment was based on converting the RGB image into the HSV model. In this model the focus was again on segmentation using colour and all three components were used. Results were similar to segmenting using the a* and b* components of the L*a*b* colour space though arguably were less successful; note the noise on the central image of the tiger below as compared to the L*a*b* version above.

Segmentation using HSV and k=3. notice the noise in the middle image around the tiger

Including Spatial Data

The next set of experiments was aimed at using spatial data as well as colour data. Spatial data is encoded as X and Y coordinates within the image. In order to balance the effect of spatial vs colour data, it was necessary normalise the spatial data.

In the L*a*b* colour space, the a* and b* values can range from -100 to +100 although they may come nowhere near than for practical purposes. Spatial data was normalised to be in the same range using the following code:

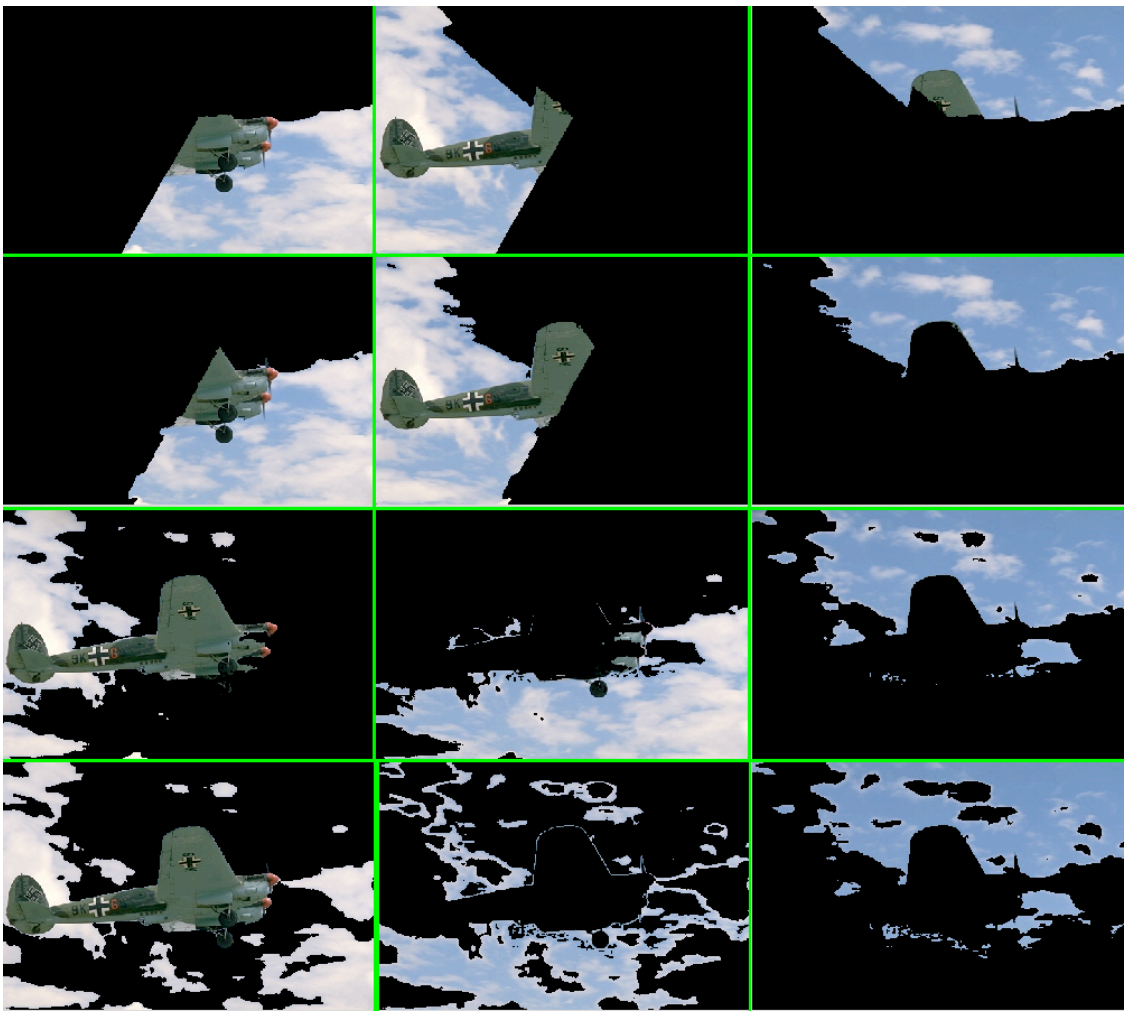
```
% Construct spatial vectors
yvec = repmat( (1:height)',width,1);
xvec = reshape(repmat( (1:width),height,1),width*height,1);

% Normalise the spatial data to be in the range -100 to +100
% which is in line with the a* and b* ranges
midX = width / 2;
midY = height / 2;
yvec = ((yvec - midY) / midY) * 100;
xvec = ((xvec - midX) / midX) * 100;
```

The first attempt at segmenting the plane picture using three segments and spatial data produced the following.

Here it is apparent that the spatial weighting is too high, sufficiently so that little significance is being given to the colour data. A weighting factor was then applied to the segmentation algorithm to allow the overall weight of the spatial data to be varied.

The following shows the result of using a range of values for the weight.



From top to bottom, values of weighting are 07., 0.4, 0.2, 0.1.

At 0.1, the weight of the spatial data is only slightly apparent. The 0.3 setting provides a better segmentation of this image than using the colour data alone.

Improving seed selection

K-means has a sensitivity on initial seed values and as such some time was spent investigating how these may best be selected to converge on a good solution.

Intuitively, a good segmentation will cluster similar coloured objects together. This leads to the view that providing seeds based on the most common, distinct colours may be a good approach. It is important that the colours be sufficiently distinct that they are unlikely to be collected into the same cluster.

In [5], Ilea and Whelan propose a solution to selecting initial colour seeds this but there was insufficient time to implement the algorithm.

Normalised Cut

The algorithm for normalised cut breaks down into four key steps.

1. Construct a weighted graph **G=(V,E)**

As per Shi's paper, we elect to construct the weight of each node from two key measure, a distance measure to reflect the fact that closer pixels are more likely to belong to the same object and a measure of 'colour' reflecting the fact that similarly coloured pixels are also likely to be parts of the same object.

$$w_{i,j} = e^{\frac{-\| F(i) - F(j) \|_2^2}{\sigma_I^2}} * \begin{cases} e^{\frac{-\| X(i) - X(j) \|_2^2}{\sigma_X^2}} & if \| X_{(i)} - X_{(j)} \|_2 < r \\ 0 & otherwise \end{cases} \quad (8)$$

Two versions of the weighting algorithm were implemented. The first for greylevel images and the second for colour images assuming that the images were represented as HSV rather than RGB format.

•F(i) = I(i), the intensity value for segmenting grey level images

•F(i) = [v,v * s * sin(h), v * s * cos(h)]

In both cases, the colour values were normalised to be in the range 0...1 before using them as weights.

Experiments were tried with normalising distances too; dividing x coordinates by the width and y coordinates by the image height. In the end it was felt that the scale factor was a better mechanism for managing weighting of distance.

We also construct D, the degree matrix representing the total cost of the connections from each node.

Both graphs are represented using a sparse matrix in Matlab since due to the radius component of the affinity measure, the vast majority of nodes in the graph are zero.

D can be constructed as follows:

```
d = sum(W, 2);
D = spdiags(d, 0, numNodes, numNodes );
```

2. Solve for the eigenvectors with the smallest eigenvalues of the system

$$(\boldsymbol{D}-\boldsymbol{W})\,\boldsymbol{y}=\lambda\boldsymbol{D}\boldsymbol{y} \tag{9}$$

This is easily solved using MATLAB's `eigs` command

```
[values, vectors] = eigs(D-W, D, 2, 'sm')
```

3. Threshold the eigenvector and discretize

Matlab again makes this easier. Using `fminsearch`. We define a function to calculate the cost of the cut, `costOfNormalisedCut` and then use an anonymous function handle to wrap it for calling in `fminsearch`.

The seed value for threshold is set as the mean of the eigenvector following the suggestion set out in [6].

```
costFunction = @(threshold) costOfNormalisedCut(Wij, D, v, threshold);
threshold = fminsearch( costFunction, threshold);
```

Once the threshold is established we can partition the eigenvector. Segment membership is managed by maintaining a vector of node/pixel indices

```
cluster1 = find( v > threshold );
cluster2 = find( v <= threshold );
```

4. Recurse

   Once we have two partitions, we continue to recursively subdivide them until a maximum cost parameter is exceeded.

   There is no need to recalculate the weights matrix as the weights do not change. The Degree matrix will need to be recalculated since it depends on the weights of all connections from each node and some have been removed.

   Recursion is managed by maintaining a vector of pixel/node indices representing the current cluster.

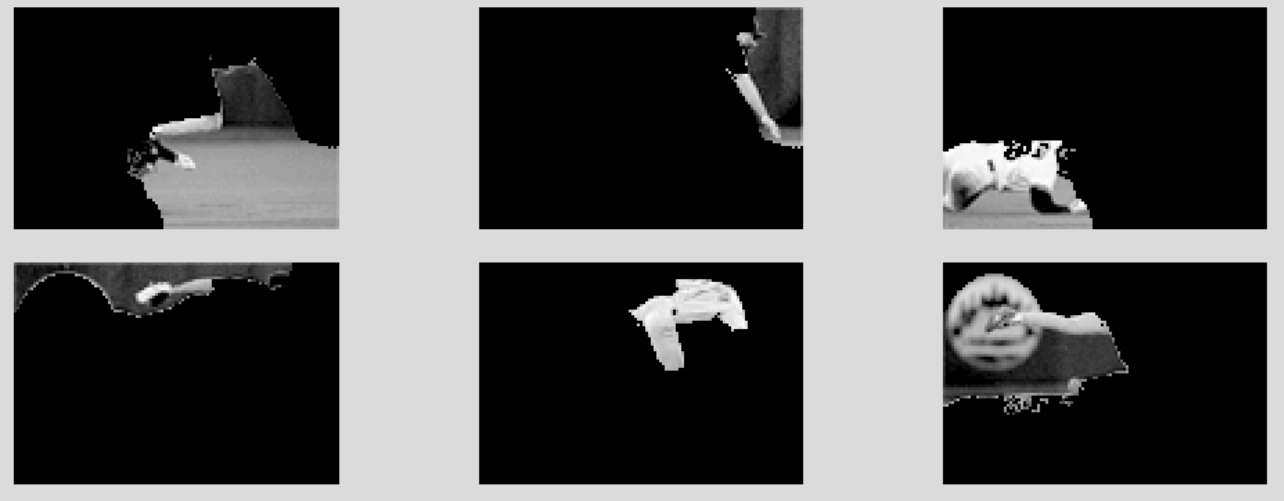
```
        cluster1Nodes = nodes(cluster1);
        cluster1Weights = Wij(cluster1, cluster1);
        d = sum( cluster1Weights, 2 );
        cluster1Degree = spdiags(d, 0, length( cluster1 ), length( cluster1 ) );
        cluster1Segments = partition( cluster1Nodes, cluster1Weights, clus-
ter1Degree, maxCost);
```

Experiments

Shi documented some of his results in his paper including his Fig 4 which showed a series of image segments derived from a grey level baseball image with specified settings of sigmaI, sigmaD and radius.

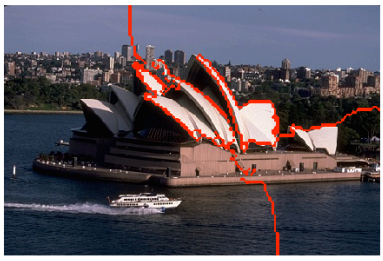
The first experiment attempted was to reproduce these segments using the same settings for the free parameters. Unfortunately the original image was not available and the copy grabbed form the paper was not dimensioned as the text described and presumably the grey levels varied from the original image.

Nevertheless the results obtained, shown below, were similar though only six segments were recovered rather than the eight.



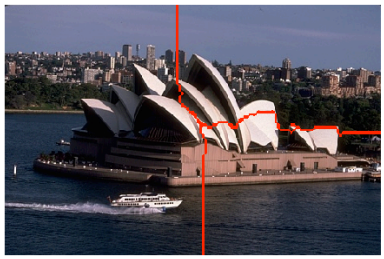
Further experiments were performed using images form the Berkeley Segmentation Database. Examples are shown below. Images from this dataset were reduced by 1/3 of their size (to 107 x 161 pixels) and converted to grayscale to improve performance.

The next set of tests were designed to determine the effect of varying the free parameters. While not perfectly clear some general trends were discovered.

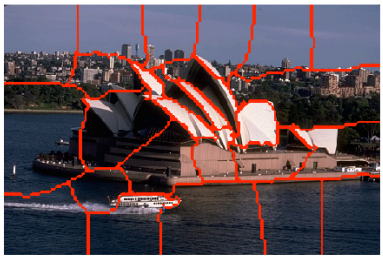


r=6, sigmaI=0.1, sigmaD=6.0, maxCutCost=0.04

r=5, sigmaI=0.05, sigmaD = 4.0, maxCutCost = 0.04

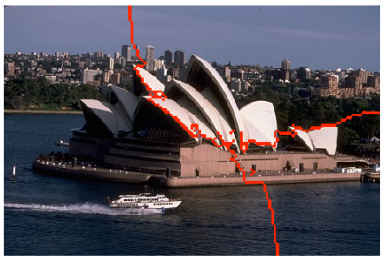


r=5, sigmaI=0.25, sigmaD = 4.0, maxCutCost=0.04



r=2, sigmaI=0.1, sigmaD=4.0, maxCutCost=0.04

r=5, sigmaI=0.1, sigmaD=6.0, maxCutCost=0.04



r=5, sigmaI=0.1, sigmaD=5, maxCutCost = 0.03

Radius

Varying the radius affects the construction of the graph by allowing more pixels further away to be considered for affinity with a given pixel in the image. This works in conjunction with sigmaD in the sense that if the radius is small for a large sigmaD, the distance factor

is small and may tail off to negligible levels before the radius is reached. In general increasing the radius allows larger features to be discovered and is likely to lead to a smaller number of segments.

Increasing radius also has the effect of creating more non-zero entries in the weight matrix which leads to increased processing time for calculating eigenvalues and vectors.

MaxCost

maxCost determines the maximum cost of the cut that can be made. It works in conjunction with the weights and radius since decreasing these values leads to larger weights and therefore less capability to make cuts. The net effect of increasing this in isolation is to reduce the number of segments found, often quite dramatically.

Intensity and Distance weights

Note that both the original 1997 paper and the subsequent 2000 paper vary the scaling parameters between sigma, $sigma^2$ and $2sigma^2$. Since the weights are constant the precise value used shouldn't make a difference however when trying to recreate the results in the paper it is difficult to determine which variant should be used.

In general the distance weight should be selected to ensure that the effect of distance close to the limit of the radius is close to zero.

sigmaI	sigmaD	radius	maxNcuts	Time	Number of segments
0.1	4	5	0.04	18.93s	15
0.1	4	5	0.02	13.64	1
0.1	4	5	0.03	16.4	3
0.1	4	2	0.04	14.84	26

sigmaI	sigmaD	radius	maxNcuts	Time	Number of segments
0.1	4	10	0.04	19.9	1
0.1	4	7	0.04	20.64	4
0.1	2	5	0.04	19.75	15
0.1	6	5	0.04	19.57	15
0.5	4	5	0.04	16.77	3*
0.25	4	5	0.04	16.8	3*
0.05	4	5	0.04	22.13	15
0.1	6	6	0.04	19.02	4

4 Summary and conclusions

Implementations of k-means clustering and normalised cut based image segmentation algorithms were built and run against assorted sample data.

The k-means clustering algorithm can provide the basis for a robust image segmentation algorithm however great care must be taken to determine precisely what metric is to be minimised and this should be reflective of how the segmented image components are to be used. The use of spatial data can improve the quality of the segmentation however care must be taken to weight it appropriately.

Although there was no time to test the theory it is believed that careful selection of seed values for initial centroids could make a significant difference to the quality of segmentation.

Normalised cut provides a powerful but slow segmentation algorithm. The output is critically and sensitively dependent on the various parameters which control segmentation and a high degree of experimentation is required to obtain solid results.

Files

File	Description
kmeans_d.m	An implementation of the k-means clustering algorithm
kmeansSegHSV.m	HSV based segmentation algorithm using k-means clustering
kmeansHSVSpatial.m	As above but including spatial data
kmeansSegLab.m	L*a*b* colour space segmentation algorithm using k-means clustering.
kmeansSegLabSpatial.m	As above including spatial information
kmeansSegLabSeeds.m	L*a*b* colour space segmentation algorithm using k-means clustering and allowing seeds to be provided
kmeansSegRGB.m	RGB segmentation algorithm based on k-means clustering
normalizedCut.m	Implementation of the normalized cut algorithm. Top level function file.
partition.m	Partition the cluster of graph nodes provided into two sub partitions using the normalized cut algorithm. This function recursively calls itself until the cost of the cut made exceeds some threshold
costOfNormalisedCut.m	Calculates Ncut cost as per Shi's paper
constructGraphForImage.m	Builds Wij and D sparse matrices for normalizedCut

References

[1] S. Lloyd, “Least squares quantization in PCM,” *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.
[2] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000.
[3] Z. Wu and R. Leahy, “An optimal graph theoretic approach to data clustering: theory and its application to image segmentation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, no. 11, pp. 1101–1113, 1993.
[4] ForsythPonce, *Computer Vision A Modern Approach*. Prentice Hall.
[5] D. E. Ilea and P. F. Whelan, “Color image segmentation using a spatial k-means clustering algorithm,” 2006.
[6] Y. N. L. C. H. Huang and W. L. Chao, “DIP: Final project report Image segmentation based on the normalized cut framework.”