Image Segmentation

Richard Collins

***Abstract*—This report details the design and implementation of an algorithm that segments images based on colour. A number of functions have been built into the algorithm that give the user control over number of colours detected, adding Gaussian noise, and separating objects into separate images.**

Overview

THE basic principle behind this algorithm is to use k-means clustering to identify similarly coloured pixels. Then, using the pixels in each cluster to create a binary mask to select only the desired colours from the original image. Added functionality of the algorithm is controlled by user input in the command window of Matlab. Added functions include being able to apply Gaussian noise at varying signal-to-noise ratios, specifying the number of clusters into which pixels are classified, and being able to separate objects and paste them into a separate figure.

Explanation of Code

The main segmentation algorithm can be found in the file named **LabSpaceClassify.m**.

Clear all variables and images: Lines 4 - 7

The head of the code simply clears the workspace of all variables and closes all existing figures. This was implemented due to the potentially large number of variables and figures generated by the code. The command window is also cleared to avoid clutter.

Prompts: Lines 15 - 41

The program displays prompts in the command window at different stages to indicate to the user to input values. All of the prompts are kept at the top of the code for easy referencing.

Read image file: Lines 24 - 25

The first such example of a user prompt is the one asking to input a file name. The user need not wrap the file name in quotes as the `'s'` argument of the `input` function converts the user input to a string. The image is then saved to the variable `I`.

Add Gaussian noise: Lines 30 - 41

The first input from the user is whether to apply Gaussian noise or not. If the variable `noiseCondition` is *'y'* then noise is added to the original image `I`. Any other value for `noiseCondition` results in no noise added and a message displayed in the command window.

If noise is added, the user is then asked to provide the signal-to-noise ratio, which is saved as the variable `snr`. Noise is added using the function `imnoise(I, type, parameters)`, where `type` is set to `'gaussian'` and the `parameters` are the mean and variance of the applied noise. The variance of the applied noise σ^2_{noise} (written `sigma_noise^2` in the code) is given by the following equation:

$$\sigma^2_{\text{noise}} = \frac{\text{varI}}{\text{snr}}$$

where varI is the variance of pixel values in `I` (after having been converted to a double), and snr is defined by the user. Line 36 shows that `I` is redefined with the noise for the remainder of the program.

Converting image to Lab space: Lines 47 - 48

The image `I` is converted to Lab colour space (Luminosity and two colour components a and b for green-red and blue-yellow respectively). The effect this has is shown in Fig. 1.

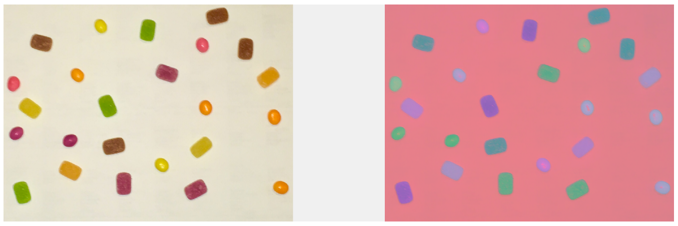

Fig. 1. Left: rgb colour space `I`. Right: Lab colour space `lab_I`.

The transformation is achieved via the `makecform` and `applycform` functions, which create and apply colour transformation structures.

Cluster by k-means: Lines 53 - 65

Initially, the two colour components a and b are isolated from `lab_I` and reshaped into effectively a double column vector with length equal to the number of pixels in the image. This is saved as the variable `ab`. `ab` is then fed into the `kmeans` function to determine which cluster each pixel belongs to. The number of clusters is defined by user input and saved as the variable `nColours`. From the other arguments we can see that the squared Euclidean distance is chosen to calculate pixel distances. The clustering is repeated three times in order to avoid local minima that may not be representative of the actual pixel clusters.

As a check for the user, an image of the pixel clusters is displayed. Line 64 returns the cluster indices to a matrix with the same shape as `I`. In effect, what has happened is that the pixel values of `I` have been converted to a number between 1 and `nColours`. Line 65 then displays a greyscale image scaling the display based on the range of pixel values in `pixel_labels`. An example is shown in Fig. 2.

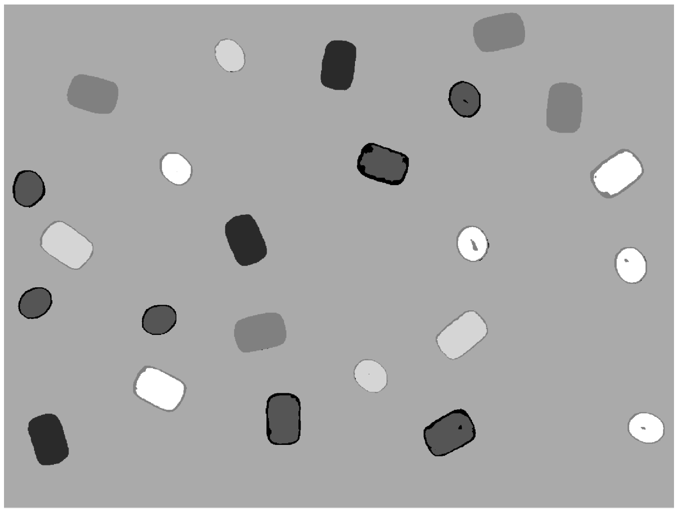

Fig. 2. Image showing colours segmented into 7 clusters.

Segment colours: Lines 70 - 123

This section of the code consists of a for-loop that loops through the clusters found by the k-means. The main variable here is `segmented_images{k}` that displays the pixels in the kth cluster (blacking out any other pixels). Lines 83 - 86 represent an improvement which removes groups of pixels that are smaller than 1400 pixels. This removes noise and shadows that don't represent complete objects. An example of this effect is shown in Fig. 3.

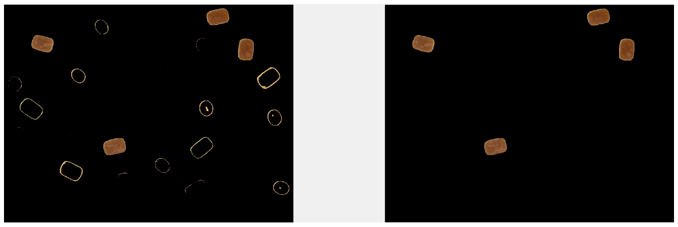

Fig. 3. Effect of removing pixel groups smaller than 1400 pixels.

The user is able to specify whether the identified objects are displayed *in situ* (as in the right side of Fig. 3), or separated onto a different figure. If the variable `cropCondition` is *'y'* then lines 92 - 102 are executed and objects in each cluster are neatly organised into rows and columns. Otherwise, lines 105 - 121 are executed and the clustered pixels are displayed *in situ* similar to the right side of Fig. 3. The function `bwconncomp` is able to count the number of connected objects in each cluster. This is shown in the title of each figure generated.

INSTRUCTION

Instructions for Part One

Run program and follow the prompts in the command window with the following inputs:
File name = lego-bricks-1.jpg / lego-bricks-2.jpg / lego-bricks-3.jpg
Gaussian noise = "n"
Number of colours = 5 - 9
Organise objects = "n" × Number of colours

Instructions for Part Two

File name = lego-bricks-1.jpg / lego-bricks-2.jpg / lego-bricks-3.jpg
Gaussian noise = "y"
Signal-to-noise ratio = 1 - 20
Number of colours = 5 - 9
Organise objects = "n'" × Number of colours

Instructions for Part Three

File name = lego-bricks-1.jpg / lego-bricks-2.jpg / lego-bricks-3.jpg
Gaussian noise = "n"
Number of colours = 5 - 9
Organise objects = "y" × Number of colours
Note: figures are replaced upon each generation.

Instructions for Part Four

File name = sweets-white-background.jpg / sweets-busy-background.jpg
Gaussian noise = "n"
Number of colours = 5 - 9
Organise objects = "n" × Number of colours

RESULTS AND CONCLUSIONS

Part one

	# of objects	visual	automatic
lego-bricks-1	116	89	2027
lego-bricks-2	116	103	3860
lego-bricks-3	116	ambiguous	1762

Fig. 4. Test results for part one ($k = 8$).

Visually inspecting the images, the algorithm is able to segment the bricks. Depending on the number of clusters, some colours are grouped together; the numbers quoted in Fig. 4 were generated with 8 clusters. For example, white bricks are often regarded as the background. The choice of cluster number changes the performance of the algorithm; a high k risks splitting objects with different colours, a low k groups colours that are visually distinct. $k = 8$ was seen as the best compromise.

The algorithm fails to adequately count individual objects (Lines 107 and 116 in the code). This is due to the function `bwconncomp`'s limited ability to recognise what we would perceive as an object. Detecting edges

Part two

The application of Gaussian noise is applied from the user input. Further statistical tests were, however, not implemented; an adequate method for calculating pixel classification error could not be devised.

Part three

The separating of object into a different figure works adequately for the first two test images. The function `regionprops` seems to be able to identify objects better than `bwconncomp`. Orientating the objects was not implemented in this version of the algorithm, but by using the function `imrotate` and appropriate user input the objects could be rotated to the correct orientation.

Part four

	# of objects	visual	automatic
sweets-white-background	24	24	25
sweets-busy-background	24	24	27

Fig. 5. Test results for part 2 ($k = 7$).

Performance is much better with the two other test images. It is easy to visually confirm all the sweets are in the correct colour category, and the `bwconncomp` function is able to identify objects much more accurately.

Acknowledgment

I would like to acknowledge the mathworks website [1] for providing detailed documentation of all Matlab features used in this algorithm. Particularly for providing worked examples, upon which this work was based.

References

[1] https://uk.mathworks.com