COMP 9517
S2, 2018
Assignment 1: Specification
Maximum marks achievable: 10 marks

This assignment is **worth 10% of the total course marks**. The optional part of the assignment will attract additional marks as specified and will count towards the final course mark.

The assignment files should be submitted online.

Instructions for submission will be posted closer to the deadline.

Deadline for submission is week 4 Thursday August 16th, 23:59:59

Preliminaries: The assessment environment

Please ensure that your code compiles and runs on the generic CSE (linux) vlab environment. Your submission will be tested and marked in the CSE (linux) lab environment only. You can find details about remote access to vlab here: https://www.cse.unsw.edu.au/~cs1511/17s2/home-computing/vlab/

The following software will be used in the testing environment, and it is recommended that you use the same to test before final submission:

Python 2.7+
gcc (Debian 4.9.2-10+deb8u1) 4.9.2 (supports C++11)

The following libraries are also available:
OpenCV 3.0+
Numpy, Numba

Do not assume availability of other libraries.

For convenience, these may be installed in a virtual environment as follows:

```
virtualenv venv
source venv/bin/activate
pip install opencv-python opencv-contrib-python numba
```

If you are using the most recent release of OpenCV to develop with, note there are a variety of differences between old and new summarized here:
https://docs.opencv.org/master/db/dfa/tutorial_transition_guide.html

It is recommended that you use Python for this assignment, but C++ is also acceptable. **Only Python and C++ are allowed.**

You should submit 4 (optional 5) files, and an additional makefile if using C++:

1. k_means_thresholding.{py,cpp}
2. filtered_k_means_thresholding. {py,cpp}
3. step_2.txt
4. count_eggs_4. {py,cpp}
5. (optional) 2D_k_means_thresholding. {py,cpp}
6. makefile

The maximum runtime allowed per image for all functions during testing is **2 minutes**.

You should NOT use any of the following OpenCV library functions:

```
threshold, adaptiveThreshold, watershed, findContours,
contourArea, drawContours, connectedComponents
```

You should NOT use any inbuilt python k-means clustering algorithm (e.g.: `sklearn.cluster.KMeans or equivalent in OpenCV`)

Each subsection of the code should be able to be run as a command line run as follows:

```
python step1.py my_input.jpg my_output.jpg
OR
./step1  my_input.jpg my_output.jpg optional_argument a
```

The problem: *Segmentation of frog eggs by thresholding*

Image segmentation is the task of partitioning a digital image by grouping pixels by similar features of interest, and very useful in practice in many domains.

For this assignment, an application from biological sciences is considered, where we can help scientists by automating the tedious task of counting frog eggs in a laboratory sample. Black circles in the image are frog eggs. You can download the following sample image from the class website:

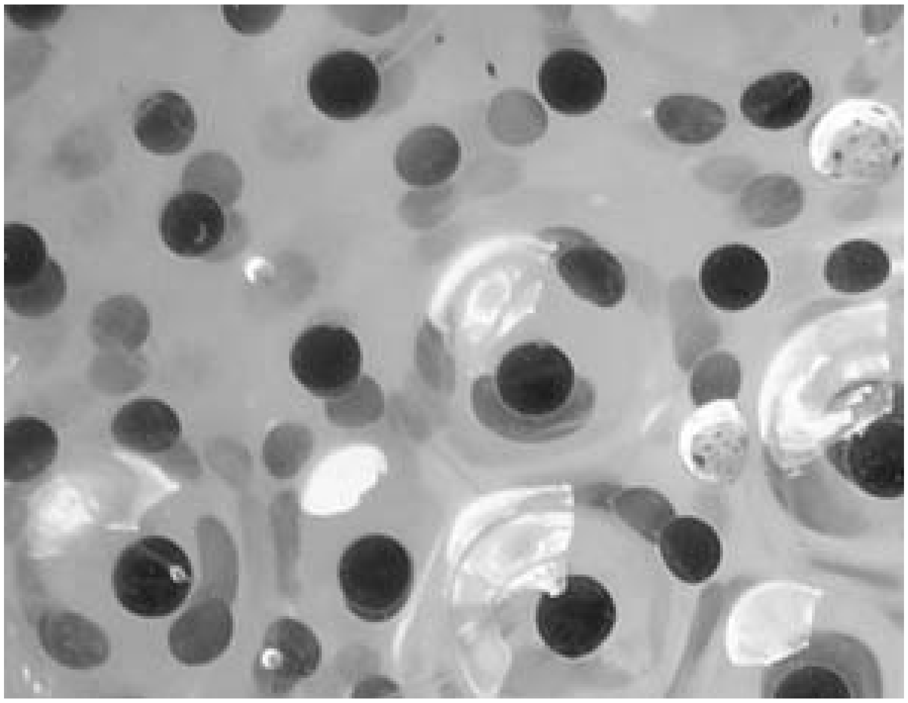


Figure 1: https://en.wikipedia.org/wiki/File:Frogspawn_closeup.jpg

In this assignment, the segmentation of a greyscale image will be carried out in 3 steps:

Step 1: thresholding of the greyscale image using K-means algorithm
Step 2: filtering and thresholding of the greyscale image
Step 3: partitioning similar neighbourhood pixels in output of Step 2 using connected components

There is an optional step you may attempt, which carries 2 bonus marks for this assignment.

Bonus Task: 2D K-means thresholding of the greyscale image

Step 1: K-means thresholding (5 marks)

K-means algorithm optimally partitions a data array into k classes by minimising the net intra-class variation. This technique can be applied to digital image segmentation. In this assignment, let us consider the case of k=2 (bi-level thresholding), in the range [0, 255]. K-means clustering can segment a greyscale image into two optimal partitions $\{C_1, C_2\}$ by minimising:

$$J = \sum_{j=1}^{2} \sum_{x,y \in C_j} (I(x,y) - U_j)^2$$

where U_j is the mean of the cluster C_j, $I(x,y)$ is the greyscale intensity of the pixel (x,y) that belongs to the cluster C_j.

The k-means clustering algorithm for bi-level thresholding of a greyscale image can be described as follows:

```
1. Select 2 values in the range [0, 255] as the initial
   cluster means
2. Assign each pixel to the cluster where the cluster mean is
   closest to the pixel intensity
3. When all the pixels have been assigned to two clusters,
   compute the mean of pixel intensities in each cluster, and
   assign this computed value as the new cluster mean
4. Repeat steps 2 and 3 until the cluster means are no longer
   changed
5. Compute the thresholding point by averaging the final
   values of the two cluster means
```

(i) Implement the K-means clustering method to find a good threshold for the input image and generate a binary image as output.

For convenience of display, ensure the eggs are encoded as [0] (i.e. black) and the background as [255] (i.e. white).

- **Input:** a greyscale image
- **Output:** a binary image
- **Testing sequence:**

```
python k_means_thresholding.py input.jpg output_binary.jpg
OR
./k_means_thresholding input.jpg output_binary.jpg
```

Step 2: Improving K-means thresholding (2 marks)

To obtain better bi-level thresholding of a noisy image, appropriate morphological filtering may be required prior to and / or following the K-means thresholding.

(ii) Improve the K-means thresholding by applying appropriate filter(s) to the image, before and / or after applying K-means thresholding. In this step, you can use inbuilt OpenCV filters.

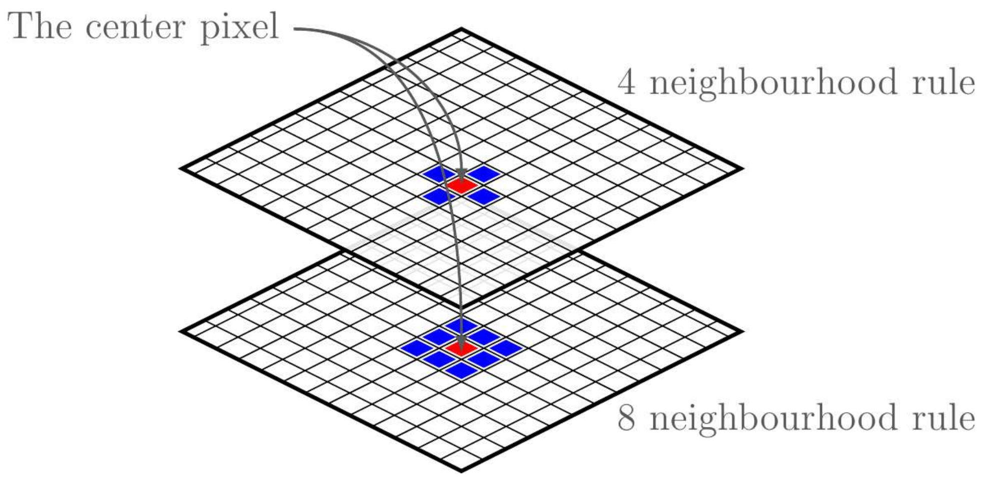
- **Input:** a greyscale image
- **Output:** a binary image
- **Testing sequence:**

```
python filtered_k_means_thresholding.py input.jpg output_binary.jpg
OR
./filtered_k_means_thresholding input.jpg output_binary.jpg
```

Additionally, for this step, you should submit a text file (step_2.txt), listing the filters you have used, and stating whether they are applied before or after K-Means thresholding.

Step 3: Eggs counting using 4-neighbourhood (3 marks)

In this section we shall count the number of separable eggs in each image by counting the number of connected components in the underlying graph. For the purposes of eggs counting, consider two pixels to be adjacent if they share the same colour/intensity and they belong to the four-neighbourhood of each other, as illustrated below:



Find the connected components in the graph generated by this procedure to isolate the eggs in the original image. Thus, count the number of separable eggs (same as the number of connected components) in the original image.

You should apply the two-pass connected component algorithm. In the first pass:

```
Iterate through each pixel in the image matrix:
     If the pixel is not the background:
          Get the neighbouring elements of the current
     pixel If there are no neighbours:
          uniquely label the current pixel and continue
     else:
          find the neighbour with the smallest label and assign it
          to the current pixel
     Store the equivalence between neighbouring labels
```

In the second pass:

```
Iterate through each element of the data by column, then by
     row
          If the element is not the background:
              Relabel the element with the lowest equivalent label
```

(source: https://en.wikipedia.org/wiki/Connected-component_labeling)

- **Inputs**: a binary image, a positive integer n which is the minimum number of pixels in the component
- **Outputs:** the number of eggs with area larger than n pixels in the image written to stdout AND write a colour image to ./output_eggs.jpg where each egg found is a different colour.
- **Testing sequence:**

```
python count_eggs_4.py binary_image.jpg output_eggs.jpg
OR
./count_eggs_4 binary_image.jpg output_eggs.jpg
```

Bonus task: 2D K-Means thresholding (2 bonus marks)

To further improve K-means thresholding, both pixel intensity values as well as the local average of grey levels within an $n \times n$ neighbourhood can be combined into a single data structure, and then K-means thresholding can be applied. In this case, each pixel carries two values, therefore this technique is known as 2D K-means thresholding.

The combined data structure of the size $h \times w \times 2$ can be represented as the following matrix:

$$[F(x,y)]_{h\times w\times 2} = [I(x,y), a(x,y)]$$

where $h \; and \; w$ are the height and the width of the image, $I(x,y)$ pixel intensity and $a(x,y)$ is the average intensities of an $n \times n$ neighbourhood around the pixel.

In 2D K-means thresholding, each cluster mean U_j consists of 2 values:

1. mean of the intensities of pixels that belong to the cluster: $\bar{I}_j$
2. mean of the $n \times n$ local neighbourhood averages of pixels that belong to the cluster: $\bar{a}_j$

and mean of each cluster C_j can be represented as:

$$[U_j]_{2\times 1} = [\bar{I}_j, \bar{a}_j]$$

K-Means clustering can then be applied to obtain the final cluster means $[U_1]_{2\times 1}$ and $[U_2]_{2\times 1}$. Implement 2D K-means thresholding. Implement an appropriate way of deriving a single threshold value.

- **Inputs:** a greyscale image, a positive integer n which is the size of the local filter
- **Output:** a binary image (jpg image)
- **Testing sequence:**

```
python 2D_k_means_thresholding.py input.jpg n output.jpg
OR
./2D_k_means_thresholding input.jpg n output.jpg
```