Winter 2018/19 COSC 4P80: Assignment 2

Due: Fri March 15th

Task: Use Self-Organizing Feature Maps for colour clustering and image compression

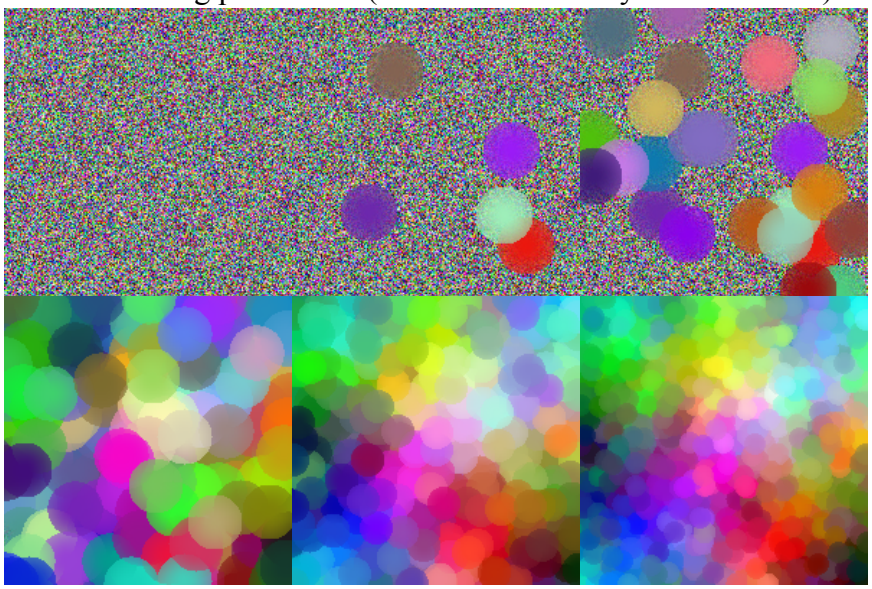
This assignment is comprised of two parts; both of which requiring the application of clustering. The first part will consist of training 3D vectors to cluster random colours. The second part will deal with the creation of a compressed image format that creates (and uses) a codebook.
This will include experimenting to determine appropriate parameters and writing your SOFM engine sufficiently abstract so as to make it applicable to both parts of the assignment.

For this assignment, you are no longer required to use LaTeX, but please still write in a formal style.

Part 1 – Colour Organization

For the first part, write an application to train your Kohonen network on random colours.
Specifically, create an N×N grid of vectors, each with an appropriate range (I'd suggest [0..1]), initially randomized. You'll then be training it on random colour vectors.

- The ~~vector~~grid will be analogous to (and convertible into) an image. Pick an appropriate size – e.g. 200×200
- Train the network for a suitable number of epochs
- Show a colour visualization of the map at some regular interval across execution
  - For full marks: show an animated window that updates very frequently: every 1-5 epochs
  - For some marks: export (i.e. save) an image of the network every 50 or so epochs
    - If you go this route, at least let the user choose a filename *pre*fix, and save them in a specified folder
- Allow for different training parameters (command-line entry is fine for this)



As part of your submission, include basic usage instructions and screenshots of the result after two different training parameters (include labels indicating the parameters used).

Part 2 – Image Compression

Consider the photograph at the end of this assignment.

The original image was 3648×2736 pixels, which would mean 9,980,928 bytes uncompressed (assuming 8-bit greyscale). In practice, it can actually be saved perfectly in 3.8MB as a PNG. However, we'll be trying something different.

You'll be creating a cluster-based approach to compressing the image. Specifically:

- Divide the greyscale image up into 'frames' of size 8×8 (or 16×16) each
  - Yes, you can assume widths and heights divisible by 16
  - Yes, you can assume greyscale
  - Allow the user to decide whether to use 8×8 or 16×16
- Create a 16×16 SOFM, with unit vectors of size 64 or 256 (depending on the frame size)
  - Train the network on the image frames
  - Allow the user to choose the training parameters (neighbourhood size, learning rate, etc.)
  - Command-line is perfectly fine for the interface
- Cluster the image frames to determine their network indices
  - Irrespective of whether the network is 1D or 2D, ensure that the index is still a single value
  - e.g. for a 16×16 network (with 256 units), that means an index of [0..255]
- Save your results into a binary file
  - You'll be creating your own image format, but here's what I used:
    - Width of the image (measured in frames)
    - Height of the image (measured in frames)
    - Width of a single frame
    - Height of a single frame
    - Width of the network (by default, 16)
    - Height of the network (by default, 16)
    - Codebook
      - For network-width×network-height tiles:
        - frame-width×frameheight floating-point values
    - Codebook indices for frames
      - For width×height frames:
        - A single *byte* denoting that frame's index
        - Because our goal is to compress data, this part's necessary – you need to use *byte*
- Conduct suitable experiments to gauge the accuracy of the compression by comparing a reconstructed version from your compressed file against the original image
  - The easiest test would be to sum the absolute values of the difference in intensities between each pixel in the original against the compressed, but if you can think of a suitable alternative, then by all means use it
  - Do this for more than one image

- Additionally, implement one of two options:
  - Allow for a network of 16 units *total*
    - Either create a 4×4 or a 1×16
    - If using this option, since only 4-bits are needed to represent the indices, you *must* save *two* indices per byte! – i.e. one nybble per frame
  - Repeat the above, but with K-Means clustering instead of SOFM
    - Include tests on error/accuracy for this approach as well and compare against SOFM
- Finally, create a simple program to view your compressed images
  - It doesn't matter whether it's a `live-viewer', or simply exports to .png
- Demonstrate on different images
  - Pick different styles – e.g. photos and illustrations (maybe geometric shapes?)
  - Pick enough images to adequately demonstrate the method's strengths and/or weaknesses
- Write a report documenting your work and experiments
  - It can be far simpler this time
  - Write it formally, but you don't need to use LaTeX or any particular publication style
  - Still include the brief basics: what clustering algorithm(s) you used, how you tested the compression, experimental results, conclusions, etc.
- Make sure to remember to include basic instructions (particularly if your program isn't intuitive)
- There will be a very small portion of marks devoted to including additional work or features not explicitly required. I'd like you to use your imagination here, but if you just want suggestions instead, then **one** of:
  - A second set of tests, wherein you compare the compressed images to equivalent *mosaic* versions of the originals
  - A single GUI-driven program that can convert .png to cluster-compressed, cluster-compressed to .png, and view either directly
  - Create an *animated* cluster-compressed image (with live-viewer)
    - Presumably, each frame would use the same codebook
  - Create a colour version instead of being limited to greyscale
  - If you choose something else, then please explicitly note it on your writeup so the marker will definitely spot it

Note: Even though we're *compressing*, that doesn't mean the final file will necessarily actually be smaller. In particular, though each frame will substantially reduce the size used, the codebook at the beginning may be substantial.
For example, if you used 16×16 frames, the codebook is 256 values, and each double-precision floating-point value is 8 bytes, then that means 512KB of header! You may wish to try using single-precision values instead (i.e. `floats`).

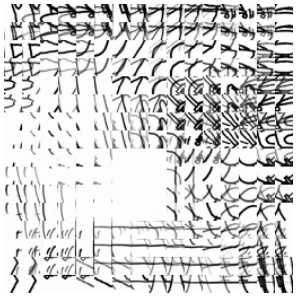
Submission:
Submission is electronic-only. Include your code, a .pdf of a sample execution, and a .pdf of your writeup. Submit by uploading to sandcastle and then running `submit4p80 .`
As always, plagiarism is a serious offense, and will result in both charges of academic misconduct and severe grumpiness.

Examples

Original

Using 8×8 tiles

Using 16×16 tiles



16×16 Codebook

Original

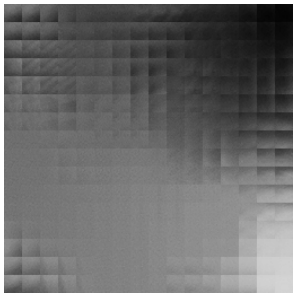
Using 8×8 tiles

Using 16×16 tiles

Original

Using 8×8 tiles

Using 16×16 tiles



16×16 Codebook

First image copyright Randall Munroe, taken from xkcd comic #303.
http://xkcd.com/303/