Machine Learning, Spring 2016

Homework 3: a picturesque case of combinatorial optimization

Summary. The objective is to reconstruct an image from a heap of sniplets, which I prepared with a cruel smile, electronically ripping apart nice pictures into giant puzzles.

Data. The data comes in the form of a 4-dimensional Matlab array `RGBtilesShuffled`, of dimension *l* x *l* x 3 x N, which should be understood as a collection of N image files (in Matlab's `unit8` based image format), each image file (call it "tile") being a size *l* x *l* pixel array with three color channels for R, G, B, with 8-bit color resolution, that is, every color intensity is coded by integers from 0 to 255. I prepared two such files, an "easy" one where the tiles have side length *l* = 25, and a difficult one where the tiles have *l* = 20. The easy one was created by dividing an image file (which shows a photograph) into 12 x 18 (width x height) = 216 tiles, which were randomly shuffled to give an array `RGBtilesShuffled` of dimension 25 x 25 x 3 x 216. The difficult one was created by dividing an image file into 40 x 25 (width x heigth) = 1000 tiles respectively, which were randomly shuffled to give an array `RGBtilesShuffled` of dimension 20 x 20 x 3 x 1000. The tiles on the image margins were not shuffled – this will help a lot in later reconstruction. After rearranging the shuffled tiles in the original format, one gets the following two "pictures":

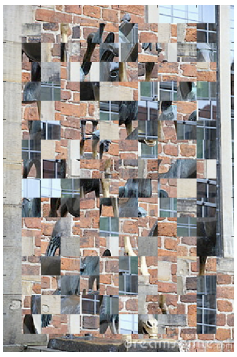

Figure: the tiled-&-shuffled „easy" (left) and difficult (right) images. Notice that the tiles on the margins are in their original order.

Downloads. In the zip-file on the course homepage you find the two shuffled image files in Matlab's `.mat` format, and a matlab script which I used to create these files from the original images. Consult the last line in the matlab script to learn what Matlab variables are contained in the `.mat` file. This script will also help you get a quick start with Matlab's image processing features.

A Matlab special remark. Matlab image arrays index the horizontal image axis in the second array dimension, the vertical image axis in the first array dimension. This is a bit counterintuitive (it results from thinking of images as matrices), but that's how it is.

Challenge and task. The task is simple: undo the shuffling, and re-create the original pictures, or something close... or not so close... Note that I was kind inasmuch as the original format (tileDimension x tileDimension) is given, and the tiles are not rotated, and no missing or corrupted tiles are introduced, and each set of tiles is complete and comes from exactly one picture... (all of this could be different for, say, a police engineer who needs to reconstruct a photo from a heap of sniplets found at a crime site...). In addition, all tiles that lie on the edge of the image will not be shuffled. That is tiles in the first and last row and column of `RGBtilesShuffled` will be in correct positions (look at the supplied Matlab script for details). *You must use simulated annealing as the main method*. When a similar miniproject was done some years ago, some students went to great lengths designing image-specific combinatorial search methods. However, that is not the purpose of this exercise. -- The tricky parts are (i) to find a good definition of a microstate (it is not necessarily the best thing to use shuffled images as microstates!), (ii) finding a good cost (energy) function, and (iii) a good proposal distribution. I would think that for the easy image, a cost function that uses only local (cut edge similarity) information will do the job, but for the difficult image, I would predict that a good cost

function needs to have both local and global components. A good energy function should identify and „reward“ larger chunks of already well-matched pieces, and a good proposal distribution should take care not to disrupt such good chunks too frequently. The difficult picture will probably require substantial thought on the energy function; long-range interactions between the puzzle pieces may turn out to be a key for success. If you find that you can't come to terms with the difficult image, just leave it out.

What ML alumni achieved. When this same project was thrown at ML students some years ago, many solved the easy image reconstruction problem to perfection, but the difficult image was never perfectly reconstructed.

Deliverables. The final result of your efforts should be Matlab image files, in the same format as the two shown above, that is one array `easyImage.mat` of size 450 x 300 x 3 (`uint8` format), and (optionally) another array `difficultImage.mat` of size 500 x 800 x 3. Plus, deliver your Matlab (or Python) code and a typeset report. The target size for a very good report is 4-5 pages. As usual, please send these items by email both to x.he and h.jaeger.

Grading. The project will be graded with percentage points, based (i) on the formal qualities of the report as a piece of scientific writing and (b) the depth of understanding, sweat, ingenuity, and results – in short, scientific quality. Both components count equally. For full marks it is enough to treat only the easy picture. Submissions that demonstrate extra good ideas and/or tackle the difficult image may be considered for bonus points.

Teaming. You may discuss problems and tricks with your course colleagues, but every participant has to program, document and submit his/her own solution. If you collaborate in discussions, please indicate your collaboration partners in the report.

Timeline. The deadline for email delivery is Friday April 8, 23:59 hrs. As usual, each started day of delay = minus 10 percentage pts.