Felix Sun

9.77 Final Project – Assembling Puzzles and Detecting Discontinuities

In this project, we look at solving jigsaw puzzles using computer vision. The essential problem – detecting whether two pieces belong next to each other in the same image – is one of detecting continuity / discontinuity. I hope to devise a better way of solving the continuity problem, with puzzle assembly as the motivating application.

I was unable to develop a better measure of continuity. Instead, I found a way for continuity metrics to express how confident they are of a matching. I improved the greedy puzzle-solving algorithm by having it place the most confident piece first, which results in significantly better solutions compared to row-by-row placement.

Background / Prior Art

Wolfson et al. (1988, "Solving jigsaw puzzles by computer") attempted to solve jigsaw puzzles by matching the shape of the pieces. They picked out the curvature of the pieces from scans of the unassembled puzzle, and used this to calculate the degree to which pairs of pieces fit together. They showed that finding a configuration of the pieces that maximized total degree of fit (for some relevant metric) is an NP-complete problem, by reduction to the travelling salesman problem. Using approximation schemes, they managed to solve a 104-piece puzzle.

Instead of shape, Pomeranz, Shemesh, and Ben-Shahar (2011, "A fully automated greedy square jigsaw puzzle solver") actually used the images on the puzzle pieces. The dissimilarity between two pieces was defined as the squared difference between each edge pixel on one piece, and the corresponding edge pixel on the other piece. To solve a puzzle, pieces were first greedily assembled using local similarity; then, some global relaxation was applied in an attempt at finding the absolute minimum dissimilarity, over a local minimum. This algorithm could completely solve a majority of test puzzles with 432 pieces.

Experimental Setup

This project will focus on detecting continuity between images (puzzle pieces). The puzzle solver will be used to evaluate different continuity metrics (and to provide a nice visual indication of performance); it is not an end in and of itself. Therefore, we will use a purely greedy solver, and leave out the global relaxation done by Wolfson and by Pomeranz. Following the framework of both papers, we will first compute a neighbor likeliness measure between every pair of puzzle pieces, in both left-right and top-bottom directions. Then, we agglomerate a solution by adding one piece at a time to a seed piece. In every iteration, a hole is chosen next to an existing piece, and the most likely piece is found to fill that hole.

To simplify the puzzle solving process, the greedy algorithm is given the identity of the upper-left corner piece. This makes agglomeration a lot simpler, but results in slightly easier puzzles than the ones used by Pomeranz. Starting from the top left, the algorithm fills the rows one by one with puzzle pieces.

As test puzzles, 1200x800 grayscale images were cut into a variable number of pieces. The pieces are rectangular; no protrusions or indentations are added. The result is shown in Figure 1.

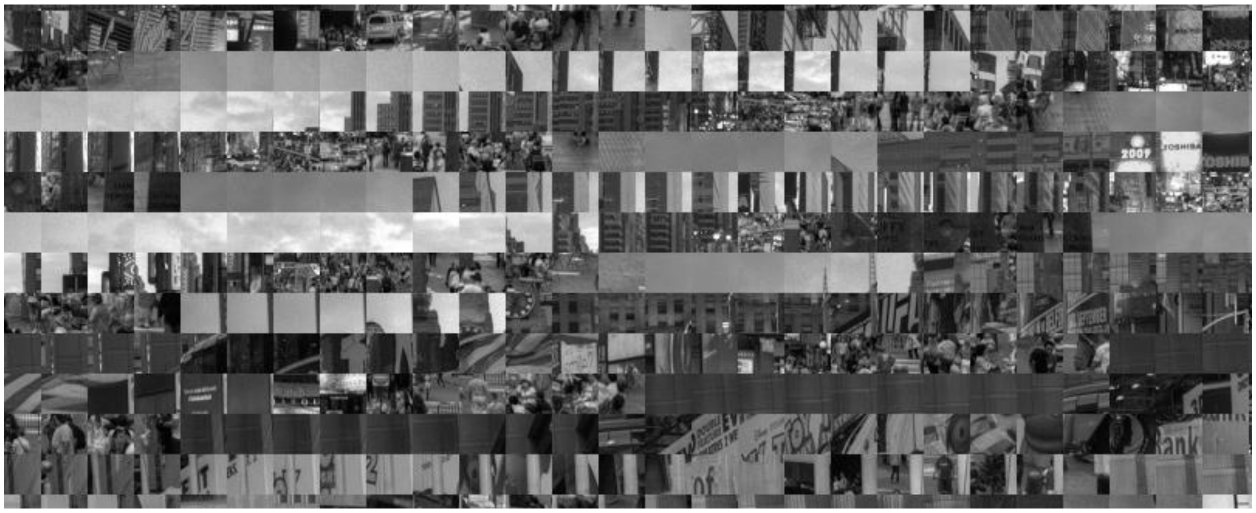


Figure 1: A sample of puzzle pieces from a 1200-piece puzzle we used in testing.

Evaluating Similarity Metrics

Pomeranz used an edge pixel similarity measure: the degree of (mis-)fit between two pieces was defined as the sum of the squares of the pairwise difference between pixel values along their shared edge. Formally, for a left-right boundary between img1 and img2:

$$S(img1,\ img2) = -\sum_{i=1}^{width} \left(img1[i, end] - img2[i,\ 1]\right)^2$$

We are interested in metrics in the frequency domain. In the simplest metric (which we call "Fourier spectra similarity"), we compare the one-dimensional Fourier transforms of the edges, and define the dissimilarity as the sum of the squared differences of the power at each frequency. In addition, we can apply a low-pass filter to each frequency series, giving more weight to disparities in the lower frequencies. Or, we can average data from a wider strip of edge pixels.

To measure the accuracy of these similarity metrics, we set up a classification task. A test image was divided into 1200 puzzle pieces (in a 40x30 cut). For each piece A, we calculate the left-right similarity between A and every other piece. The metric classifies A successfully if the best-matching piece is indeed the piece that was to the right of A in the original image. This process is repeated for top-bottom similarity, and the total fraction of correctly-matched pieces is recorded. This is a 1200-way classification task, so the baseline probability is 1/1200.

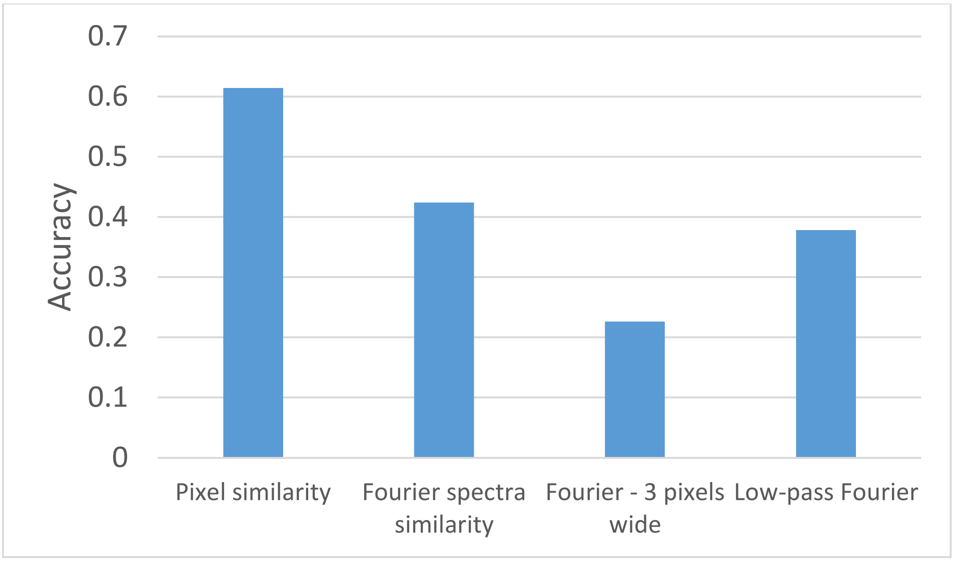


Figure 2: Performance of different similarity metrics in a task requiring identification of neighbor puzzle pieces.

As shown in Figure 2, none of the proposed similarity metrics performed as well as pixel similarity, though all of them were well above the baseline probability. This is disappointing, but perhaps not surprising in retrospect. Frequency domain representations discard phase data, which determine how various objects in the strip of pixels are positioned. This position information is very important in detecting whether two images are continuous.

Match Confidence

We now turn to extracting an additional piece of information from the pixel similarity metric: the confidence of a match between two pieces. We may not be able to increase the matching accuracy, but we would like to know how confident we are in our choice of best match. Given a piece A looking for a partner, we can find both the best match B and the second-best match C. We can then look at $S(A,B) - S(A,C)$, the gap in score between the best match and the second-best match. If this gap is large, we expect that B is correctly matched to A, because there were no other plausible competitors. On the other hand, if the gap is small, B may not be the correct match (even though it is our best guess), because there exist alternatives that are nearly as good. Therefore, the larger the gap, the more confident we are in the match we found.

We can validate this proposal by tallying the gap sizes for correct matches and incorrect matches. These histograms are shown in Figure 3. There is indeed a sizeable difference between the gap sizes of correct and incorrect matches. Correct matches had a median gap size of 0.048; incorrect matches had a median gap size of 0.007. Of course, we also note that this is not a perfect predictor of classification correctness: many correct matches have very small gaps.

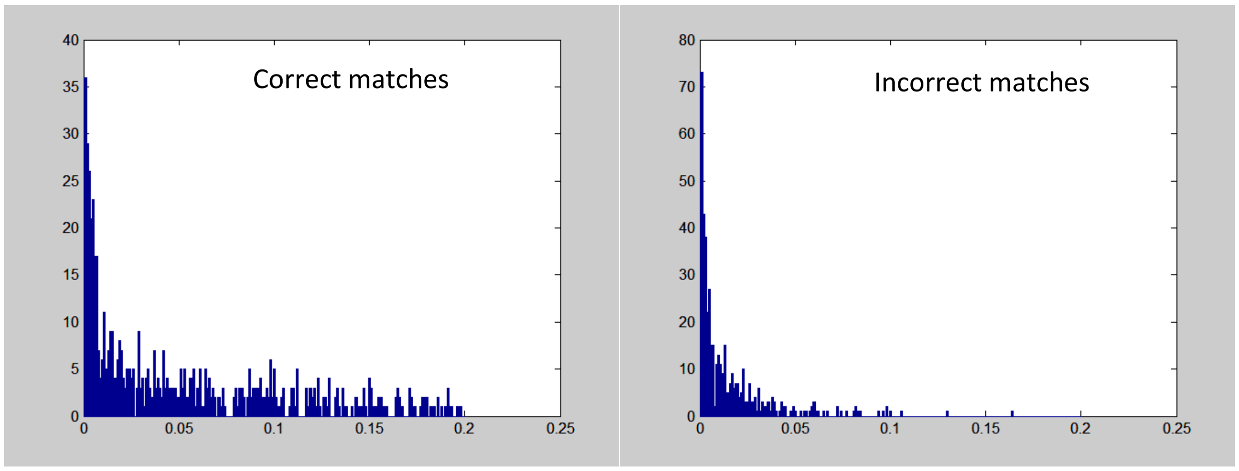


Figure 3: Histogram of gap sizes in the 1200-way classification task. The gaps from correct matches are shown on the left; the gaps from incorrect matches are shown to the right.

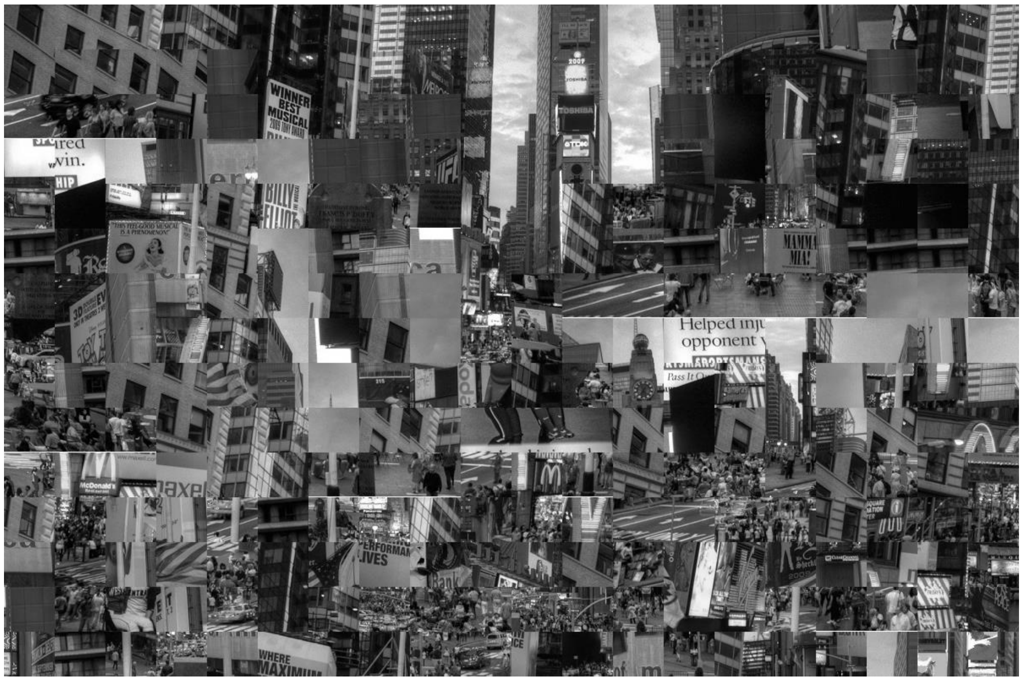
Using this measure of match confidence, we can build a better greedy puzzle solver. Instead of filling the puzzle one row at a time, we can add the most confident piece in each iteration. More specifically, we track a list of "frontier holes", which are unfilled locations that border a filled location. In each iteration of the algorithm, we look for the frontier hole that we can fill with the highest confidence. We fill that hole with a puzzle piece. The hope is, if we are initially uncertain about a hole, we will become more certain if more pieces near that hole are placed, so we can only do better by waiting.

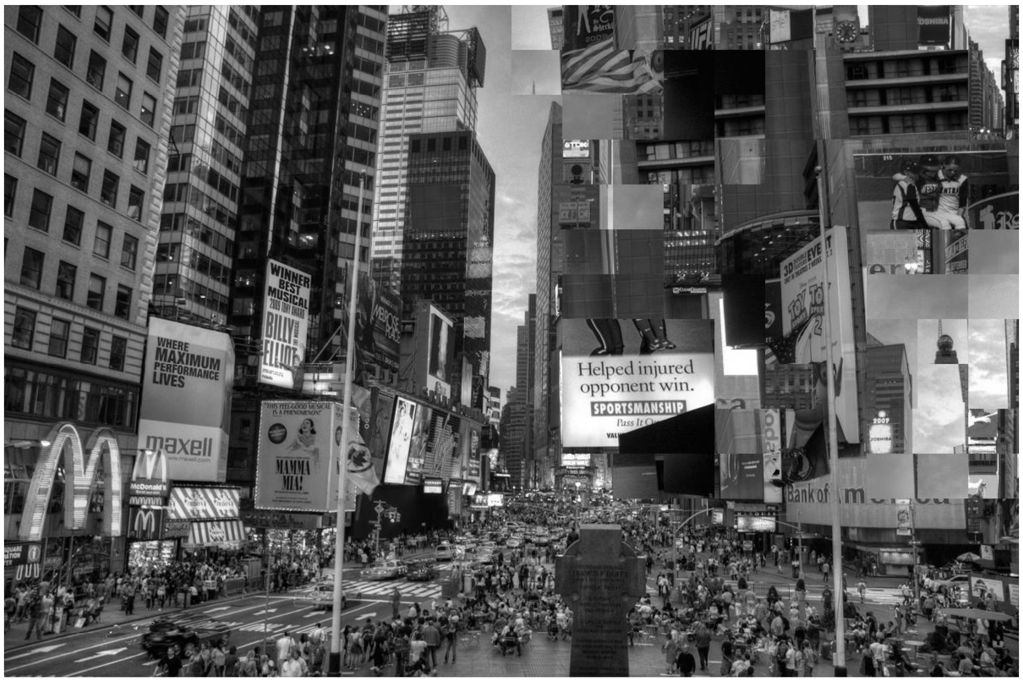
To evaluate the performance of the greedy algorithm with and without confidence-based filling, we count the fraction of neighbor pieces in the ground truth solution that are also neighbors in the algorithm's solution. (This method was used by Pomeranz.) With 15x20 puzzles, filling the most confident pieces first results in large performance gains, as shown in Figure 4.

Image #	Row-by-row	Most confident first
1	0.211	0.758
2	0.460	0.455
3	0.278	0.515
4	0.136	0.568
5	0.333	0.786
Average	**0.284**	**0.616**

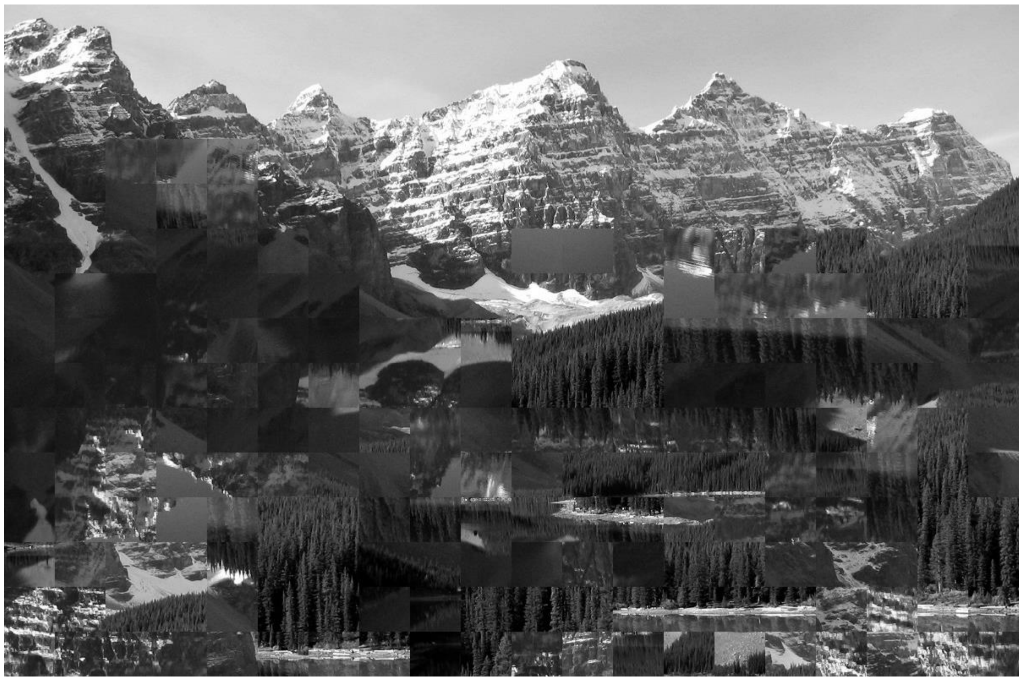
Figure 4: Neighbor accuracy of the greedy puzzle solver with row-by-row filling, and with confidence-based filling. Five different 15x20 puzzles were tested.

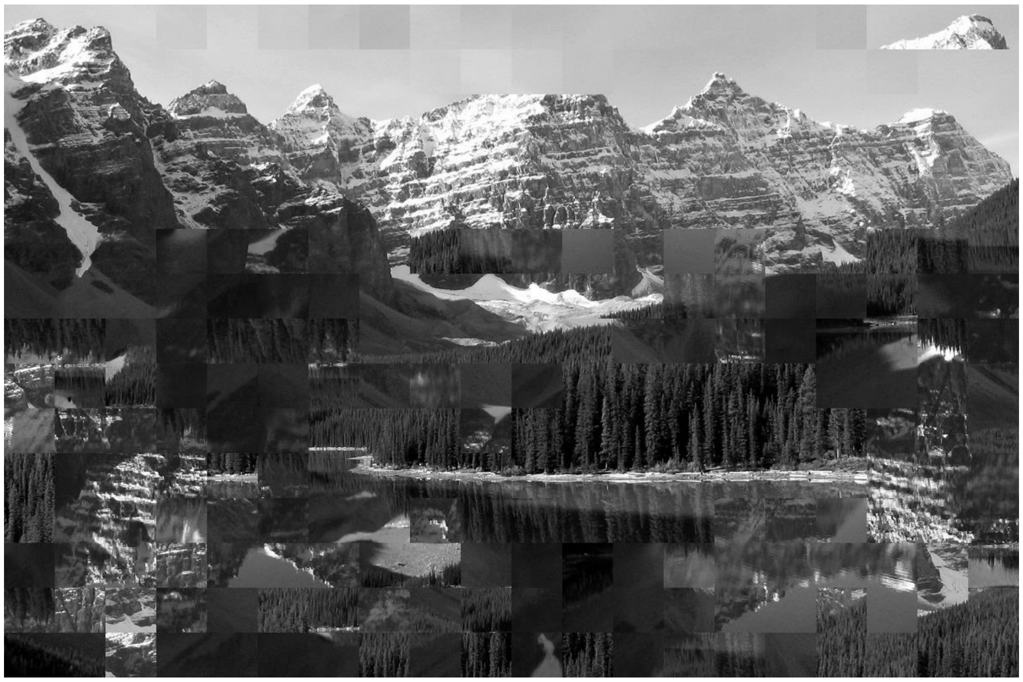
Below, we look at some solutions generated by the greedy solver.


Row-by-row, 21% accurate


Most confident first, 76% accurate

This is puzzle 1, for which the inclusion of confidence information dramatically improved performance. An animation of how the algorithm picked pieces is attached to this write-up.


Row-by-row, 46% accurate


Most confident first, 46% accurate

This is puzzle 2, which was a failure case for the confidence algorithm.

Conclusions

Solving jigsaw puzzles by placing the most probably correct pieces first is a good strategy, resulting in large accuracy improvements for most puzzles. The next logical step would be to introduce this technique to a state-of-the-art puzzle solver, with a global relaxation procedure after initial greedy matching. A better greedy solver will certainly make relaxation easier runtime-wise. It may also increase the maximum puzzle size that can be solved.