Computer Vision Assignment 1

Craioveanu Sergiu-Ionut, 407

The approach towards this task can be split into the following phases:

1. Warping perspective of all images to match template (robustly)
2. Detecting differences from one image to another (robustly)
3. Matching the cropped features of newly placed domino to board coordinates
4. Splitting the template into squares and positions
5. Performing the scoring rationale, given current predictions

I'll go over each of the sub-tasks and other steps required below.

1. Warping perspective of all images to match template (robustly)

In my opinion, this is the foundational step towards a reliable solution. Given a template, I wanted all images to be perfectly aligned with this template, so that I could use a grid of positions (which I will describe later on) in order to detect the location(s) of a domino.

I performed this warping of perspective and template matching using SIFT features, similar to what was shown in Lab 4, but the code was actually much closer to what is seen in the documentation.

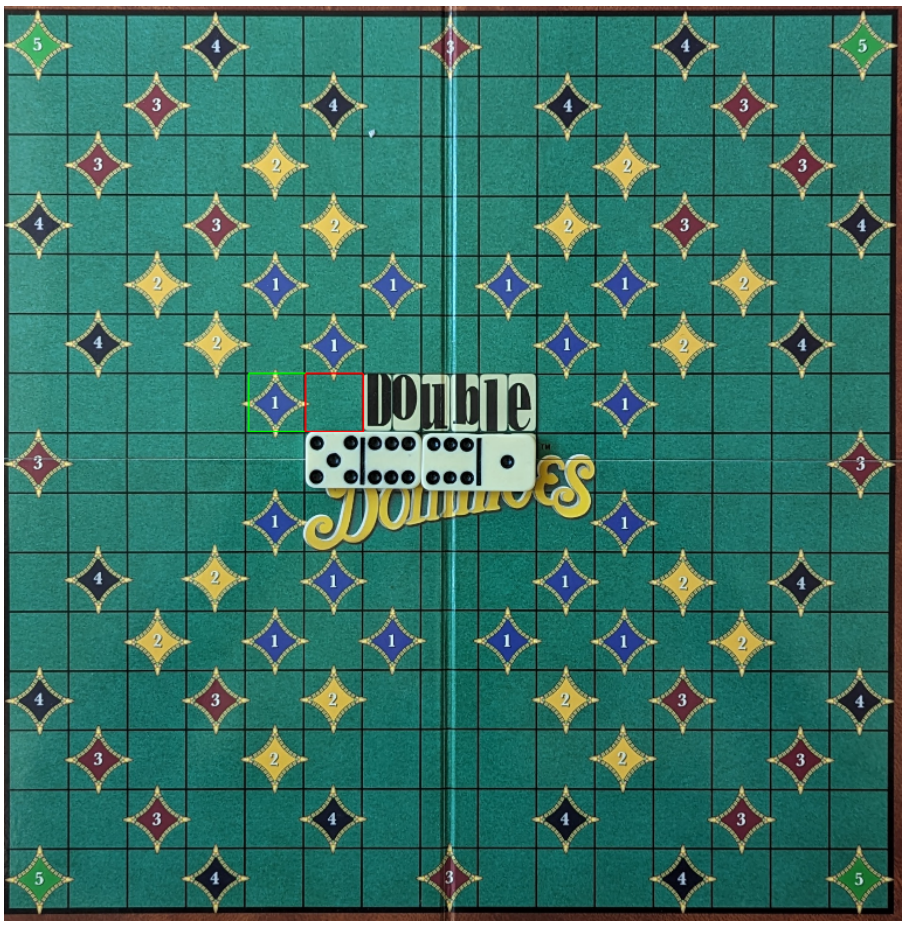
Given a template image (illustrated above), I wanted to match the features between the template and any given board image, and to find the corresponding points in the template and that image. I would then compute the homography matrix and apply it to the image in the (train) dataset. All warped images are stored in a list of images. All warped images are (perfectly) aligned with the template image, which allows me to go further in solving this task. The difficulty relied in finding the best way to reliably perform this operation - I first tried out ORB, which would run much faster (seconds as opposed to 20 minutes with SIFT), but would not give results of the same quality.

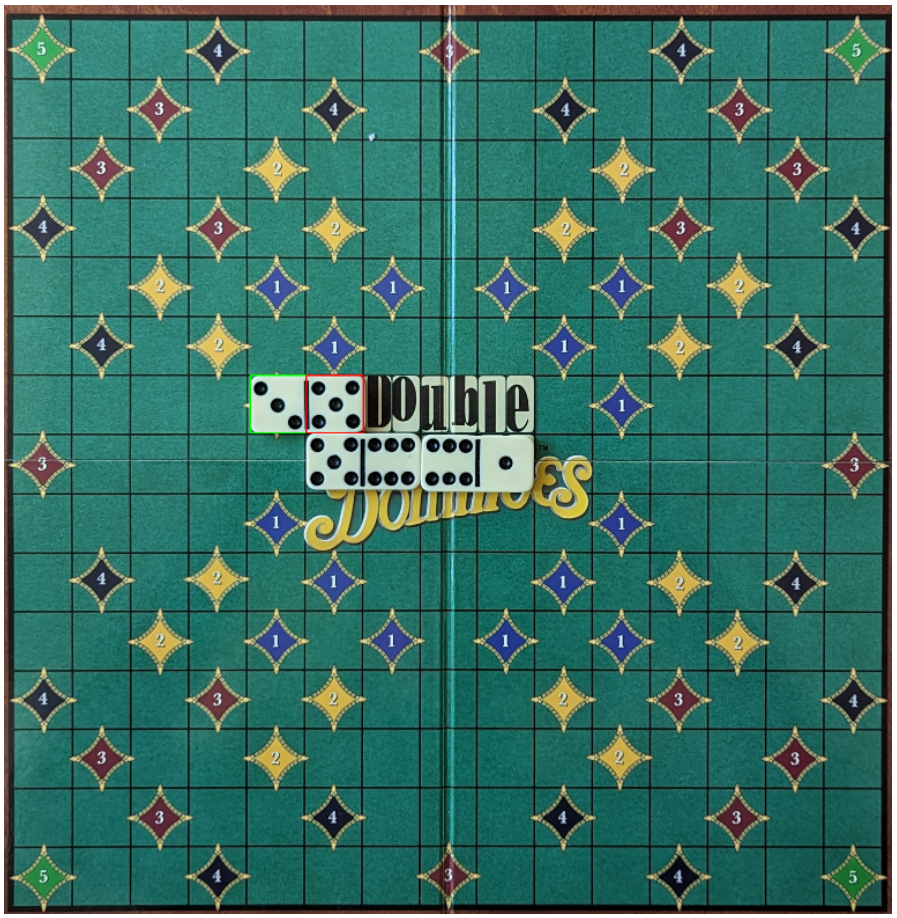
2. **Detecting differences from one image to another (robustly)**

This aspect of the code is not the most complex - essentially, after perform image pre-processing (grayscale, gaussian blur), I compute an absolute difference between the 2 processed images, and draw a contour of the difference (which given the image has been properly aligned, that difference will equate to the new domino being placed). I create a rectangle shape (with OpenCV) that indicates the placement of the new domino. I'll return the largest contour and the images containing that highlighted area, with and without the placed domino.

I mention this step separately, because it's slightly more non-trivial than I expected for 2 reasons: it matters how the domino is placed in order to correctly split the rectangle contour on the largest side, and **WHAT** do you do with the 2 regions of interest (2 boxes) after plotting and extracting their contour?

There is 1 "before image" and 1 "after image". The "before image" contains the board boxes as small square images, and the "after image" contains the domino faces. These are all features of interest, but how can I extract that information reliably and programmatically in order to further leverage it for score computation? That's what I'll address in the next sub-tasks. Below is an example of a before image and an after image, with the contours for box-shaped regions of interest.


Double
Dominoes


Double
Dominoes

3. Matching the cropped features of newly placed domino to board coordinates

This was the sub-task that took most of my time, at the order of 20h (probably). I actually tried a lot of variants of comparing images, some very complex (involving Deep Learning Convolutional Networks for feature extractions), and others very simple (computing mean squared error). I first tried creating domino features and domino features - and by that I mean storing small square images of each board face and each domino/diamond face. I then tried to compare extracted region of interest (ROI) boxes with my features and try to create some similarity score - and this was the biggest obstacle ever - because it was very hard for an algorithm (deep learning or not) to distinguish between a diamond with a 3 on it or a 4; or a domino face with 3 dots or 4 (or any close vicinity of dots). After constructing many SSIM metrics scores, template matches, and so forth, I came to the following solution:

- I would make sure the contour is robustly detected, and extract the 2 squares for each new domino (phase 1 was key here).
- I would then create a grid of the board (next sub-task), and compare coordinates.
- I would then use the coordinates to see the closest mapped box to my detected box and check if the domino was placed over a diamond (or not).

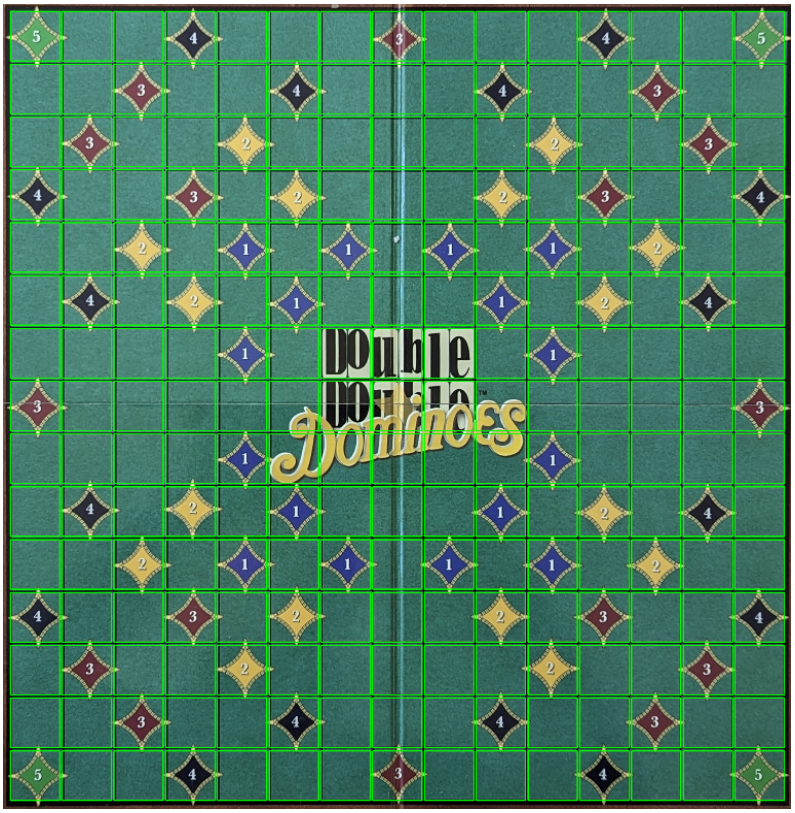
So this solved my diamond detection problem, without having to rely on crazy computer vision and deep learning. I would just use the existing contour and perform a simple euclidean distance computation for (x,y) positions of the rectangle (but close to square) contour.

But how about detecting the number of dominos on a domino? This was actually done by counting the circles in the image (with certain constraints regarding radius and size, of course). After it dawned on me that I could actually count the circles of a certain size, I felt dumb for even thinking of Deep Learning. The function that does this is quite simple and is actually partially given as an example in a documentation example, I just played around a bit with the parameters. This also solves my rotation problem, as the number of circles remains invariant of the rotation - feature embeddings mattered when computing similarity, and not all methods of computing domino face features differences wereinvariant to rotation.

I actually still use some SSIM (similarity score) computation for the first move, because tracing the contour on the first move of a game is impossible as there's no 2 images to compare with (only one). I tried using the template as the first image, and the first image of a game as the second, but contour detection is unreliable here. As such, I also use some SSIM as follows: we know what the starting position is, (and count the domino dots belonging to that square) but we're trying to find out where the second face of the domino is. It can only be above, below, left or right, and I compare those board faces with any of the dominos - for the box that obtains the highest similarity score, I count the circles there. This seems to work on all games, and I'm happy I managed to come up with a non-compute-heavy robust solution.

4. Splitting the template into squares and positions

This is actually quite trivial due to the first phase being extremely reliable - we can just correctly create the grid for the template, and it will apply to all images in our dataset. We know (after trial and error) where the first square is (top left corner), and I use increments of about 126 pixel increments to the right and 128 pixel increments below to compute locations of all squares. I take these as the ground truth for further comparison with detected ROIs when computing the contour of game images.



5. Performing the scoring rationale, given current predictions

To recap, we have the following:

- Phase 1 helps us correctly bring all images to the form of the template
- Phase 4 helps us correctly map a grid on our template, and extract the contours and coordinates (positions).
- Phase 2 and 3 help us detect changes in images (through a contour), and given this largest contour, we can split it and see if there's a diamond underneath and check the number of dots on each domino face.

So we have positions and domino phases, we would need the score of each image. We need to hard-code the sequence of dominos on the side of the board, and always remember the face on which a player currently is. One way of doing this is my keeping count of the score, as the score indicates their position. Computing the score is quite trivial, there are these 3 main rules to keep in mind:

- If you cover a diamond with a domino, you score that many points
- If the domino is a double domino, you score double the points
- If you are on the track (domino face) and a domino with that number is played, every player on that face gets 3 points

I implemented this using a few if-else clauses, stored the position of each player, read the game moves file (so I know which is the current player), computed the score for both players, but only outputted the score of the current player. I take all 5 elements of interest: position 1, domino face 1, position 2, domino face 2, current player score and create a text file given the current image's name.

Current train accuracy is 493/500 correct predictions - I make a contour detection mistake in game 5 and it messes up a few scores for a player later on. I would hope (and expect) the accuracy of the test set to be somewhat similar.