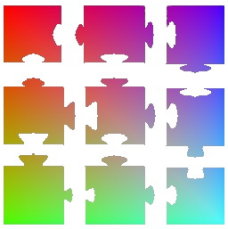
Matching puzzle pieces together



Description	Using generated puzzles to train a neural network.
Period	Summer 2021
Languages & Libs	Python, Keras
Tags	Computer Vision

Some people enjoy solving puzzles in the old fashioned way, but engineers would like to automate tedious tasks like that. This is a well-suited task for supervised learning, but naturally it requires training data. Gathering it from real-life puzzles would be time-consuming as well, so I opted for generating it instead. This gives a lot of control on the data, but the resulting system might not even work with real-life inputs. There are also several different styles of puzzles, but in this project each "base-piece" is a rectangle of identical size. An example 3×3 puzzle is shown in the thumbnail.

A "normalized" puzzle edge is defined as a set of Bezier curves, and is constrained to always start from coordinates $(0, 1)$ and to end to $(1, 0)$. In addition its slope is 100% horizontal at the beginning and at the end. This leaves them with four degrees of freedom. The curve consists of three Bezier curves, which are shown in different colors in Figure 1. They are specified by three control points, which are shown red. Black dots are either hard-coded beginning and ending coordinates, or interpolated as the midpoint between two red control points. For simplicity this code generates only symmetrical edges, but naturally it could be extended to more complex and asymmetric shapes.

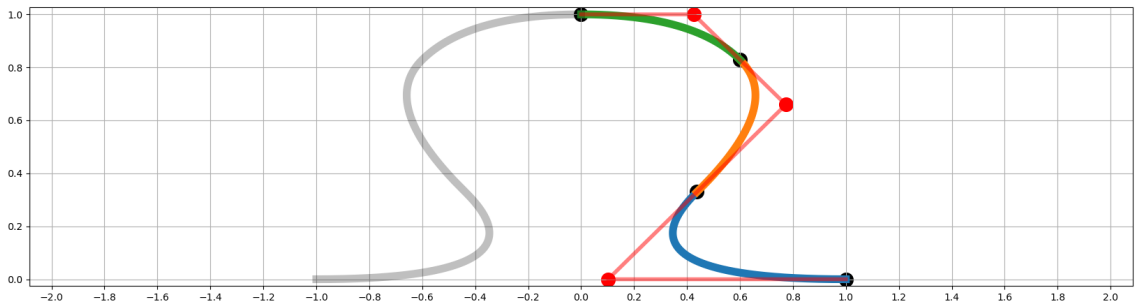


Figure 1: A "normalized" edge piece has four degrees of freedom, and it consists of three Bezier curves. When stitched together like this, they form a single continuous path.

In addition to the four "basic" degrees of freedom, each shape can also be scaled along the x and y axis, so there are actually six parameters to adjust. Randomly sampled curves in Figure 2 show the extend of variety in edges. Rasterized low-resolution examples are shown in Figure 3.

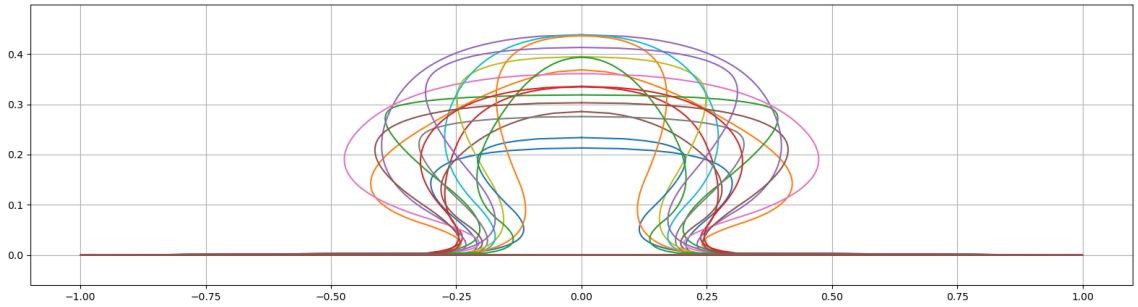


Figure 2: Examples of random edge shapes, showcasing all six degrees of freedom.

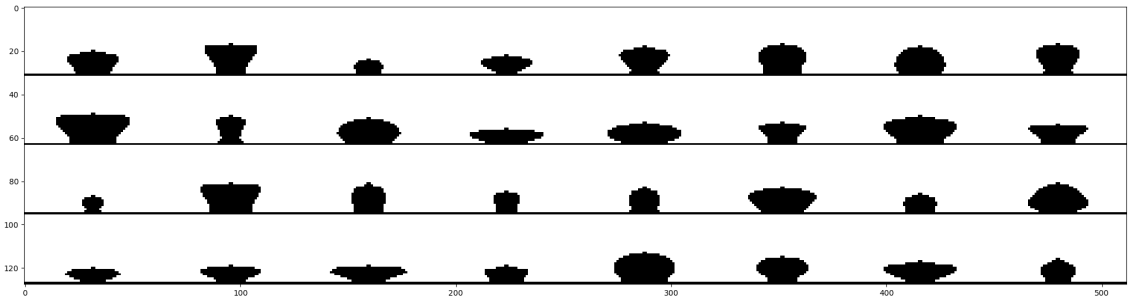


Figure 3: Rasterized examples of random piece edges.

The amount of variation within pixels is visualized in Figure 4. Images' top corners are always white and the bottom middle pixel is always black, and on average the middle area is between these two extremes. Various generated full puzzle pieces are shown in Figure 5. Actually each piece matches the neighboring four other pieces, so the puzzle is in a solved state. In addition to choosing a random edge shape to each four sides, there is also an option whether the piece has a "male" or "female" part of the shape. A more detailed view of the pieces is shown in Fiugre 6.

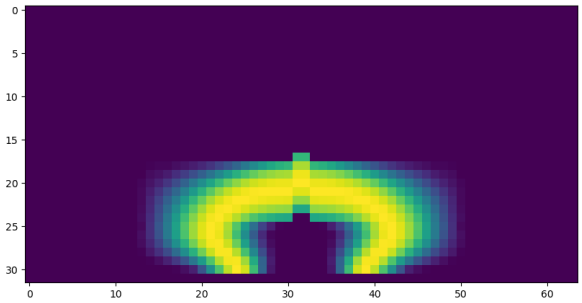


Figure 4: Edge pixel's variance is largest on the region which has a 50% chance of being either black or white. Many pixels have a variance of zero.

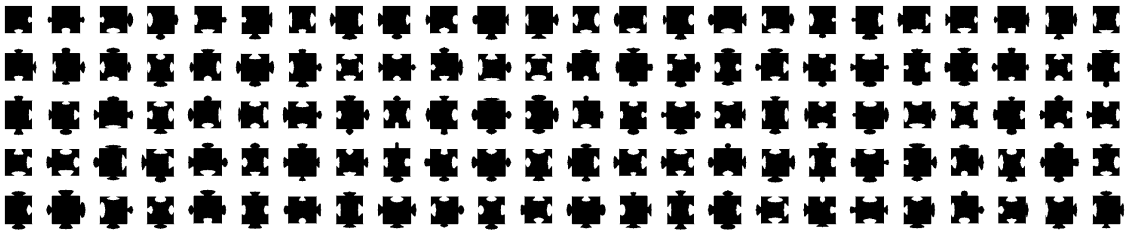


Figure 5: An example random puzzle, only the top left corner is shown.

The network consists of two parts: a shared feature extraction step using 2D convolutions, and a fully connected decision-making step. This architecture is commonly known as a Siamese neural network. The feature extraction compresses the input image of size $64 \times 64 \times 1$ to an output of $12 \times 12 \times 8$, having 1152 elements. All activations here are relus. This tensor is then flattened, and a linear layer projects it down to just 64 dimensions. This final output size is a freely adjustible hyperparameter. A smaller number of outputs is less prone for overfitting, but naturally at some point the model will only underfit to the data. Feature extraction steps' tensor dimensions and the number of layer parameters is listed below.

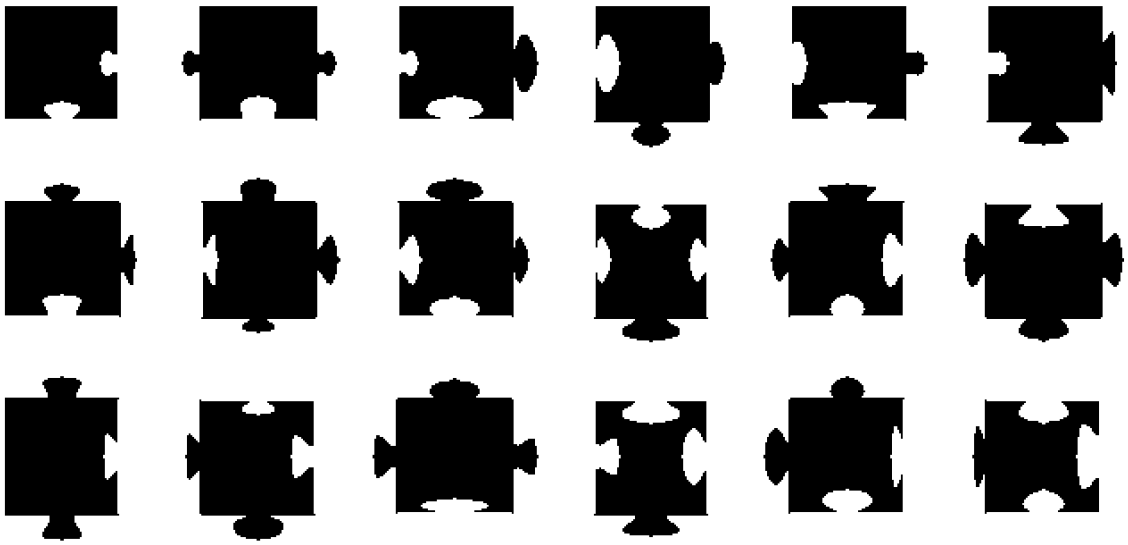


Figure 6: A more zoomed in view of generated pieces, from the top left corner of the puzzle.

```
Layer (type)                 Output Shape              Param #
=================================================================
                             (None, 64, 64,  1)        0
conv2d_338 (Conv2D)          (None, 62, 62, 16)        160
batch_normalization_1049 (Ba (None, 62, 62, 16)        64
conv2d_339 (Conv2D)          (None, 60, 60, 32)        4640
batch_normalization_1050 (Ba (None, 60, 60, 32)        128
max_pooling2d (MaxPooling2D) (None, 30, 30, 32)        0
conv2d_340 (Conv2D)          (None, 28, 28, 64)        18496
batch_normalization_1051 (Ba (None, 28, 28, 64)        256
conv2d_341 (Conv2D)          (None, 26, 26, 64)        36928
batch_normalization_1052 (Ba (None, 26, 26, 64)        256
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
conv2d_342 (Conv2D)          (None, 12, 12, 8)         2056
flatten_163 (Flatten)        (None, 1152)              0
dense_5 (Dense)              (None, 64)                73728
=================================================================
Total params: 136,744
Trainable params: 136,376
Non-trainable params: 368
```

The "decision" network is trained to predict one of the five cases from two input pieces: either the pieces do not match, or they match right-to-left, left-to-right, bottom-to-top or top-to-bottom. Examples of these are shown in Figure 7. This means that the network doesn't need to consider whether to rotate one or both of the pieces by 90, 180 or 270 degrees. It was assumed that this makes the network easier to train and less prone to overfitting. If also rotations were taken into account, the network would need 17 outputs: one for a mismatch and 16 for each of the ways the four edges from piece A can be paired with the edges of piece B. In this solution each piece pair has to be fed to the network four times. Piece A is used as-is, but the piece B has its image rotated by 0, 90, 180 and 270 degrees. Since `softmax` normalization is applied separately for each of the inputs, the concatenated output doesn't add up to 100%. Hopefully the networks' output is "consistent", meaning that it doesn't claim that the pieces fit together in two different ways. This hasn't been checked in the current prototype.

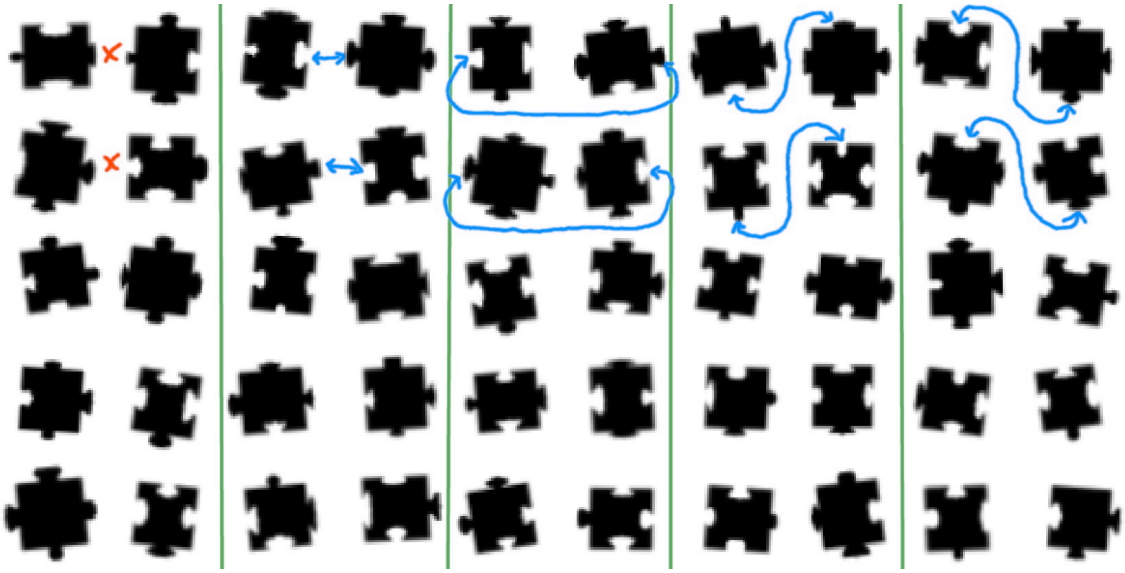


Figure 7: Example input image pairs from the five distinct classes: no match (left) or one pair of the four edges match without having to rotate either piece. The image also shows that the input resolution is rather small (64 pixels), causing some of the details be lost. Input images are augmented with random zoom, translation and rotation to midigate overfitting and having the network to be tolerant of such errors in the input images.

The decision network's tensor sizes and the number of layer parameters is shown below. 136744 (96.4%) of the parameters come from the feature extraction step. The network achieves an accuracy of about 97%.

```
Layer (type)                 Output Shape              Param #
=================================================================
                             (None, 64, 64,  2)        0
model_1 (Functional)         (None, 64, 2)             136744
flatten_2 (Flatten)          (None, 128)               0
dense_7 (Dense)              (None, 32)                4128
batch_normalization_1089 (Ba (None, 32)                128
dense_1240 (Dense)           (None, 16)                528
batch_normalization_1090 (Ba (None, 16)                64
dense_1241 (Dense)           (None, 8)                 136
batch_normalization_1091 (Ba (None, 8)                 32
dense_1242 (Dense)           (None, 5)                 45
=================================================================
Total params: 141,805
Trainable params: 141,325
Non-trainable params: 480
```

The fully implemented system would receive a photo of an unsolved puzzle, detect its pieces, rotate them into "orthogonal" north-south & east-west orientation, feed each pair to the network and solve which pairs of pieces belong together. This might take significant amount of time, since even a small-ish puzzle of 500 pieces has about 125k pairs to be checked, and remember that each pair has to be run through the network four times (the other part is rotated in 90 degree increments). However this can be optimized by running the network in two separate steps: feature extraction and decision making. One only needs to run the feature extraction for 500×4 images, and this accounts for almost 100% of the calculation requirements. The decision step is very light weight, consisting only of a sequence of dense layers with small inputs.

The last step would be to inspect the predicted puzzle piece matches, and construct the most likely globally consistent solution to the puzzle. If the false positive rate isn't too high, there aren't that many plausible solutions. Puzzle construcion is naturally very tolerant of false negatives, since it is sufficient that the piece is correctly matched to two or three of its four neighboring pieces.