Ca' Foscari University of Venice

Department of Environmental Sciences, Informatics and Statistics

Master's Degree in Computer Science

Final Thesis

Using residual networks for Jigsaw puzzle solving

Supervisor
Ch. Prof. Marcello Pelillo
Co-supervisor
Dr. Alessandro Torcinovich

Graduand
Mattia Zonelli
Matriculation Number 870038

Academic Year
2021/2022

Abstract. In this work, we address the jigsaw puzzle solving task, proposing an automated pipeline to assess the adjacency relationship among tiles and order them. In particular, we compare two approaches Relaxation Labeling (ReLab) and Puzzle Solving by Quadratic Programming (PSQP). We train convolutional neural networks (CNNs), trying different methods to extract compatibility between tiles of images, first by approaching the task as a supervised learning problem and then by using self-supervised learning, a variation of the unsupervised learning theme. We build a CNN trained for a pretext task, which can later be repurposed to extract tiles compatibility. Finally, we test different combinations of CNNs – as automatic feature extractors – and puzzle solving methods on publicly available datasets, providing the feasibility of our proposed method.

Contents

1 Introduction

Jigsaw Puzzles were originally proposed in the 18th century by John Spilsbury for educational purposes. The problem of puzzle solving consists of combining small pieces of an image to produce a coherent picture when they are reordered. The rearrangement of a set of non-overlapping square pieces on a 2D grid in a way to produce a coherent image can be formally described as, the search for a particular permutation of such pieces.
Puzzle solving can become notably challenging when we have to manage picture that represents nature, building, or, more generally repetitive design and patterns.
If the affinity between tiles is uncertain, the task of puzzle solving is considered NP-complete [6], anyway, in the last few years such kind of tasks gained much attention and their application can be seen in several problems like: reconstruction of archaeological paintings [17] [2] , shredded documents [34] [13], or in speech recognition [33], image editing [4], DNA modelling [16] and some more. Moreover, also several approaches have been employed to solve jigsaw puzzles. They either use deep neural networks to predict feasible positions for the tiles or hand-crafted compatibility measures combined with algorithms for puzzle reordering [10, 20, 22, 31]. Each approach tackles puzzle solving with different constraints.
In our work, we address the problem of reassembling pictures from small square non-overlapping tiles with identical dimensions, to be positioned in a rectangular grid of the same shape and size as the original picture. Also, we assume that each tile has a specific orientation. In contrast to physical tiles we work with pieces that have linear boundaries, so they do not retain additional geometric information, which makes the rearrangement even more challenging.
Jigsaw puzzle solving can be viewed as the combination of two smaller tasks, compatibility extraction, and tiles reordering. Compatibility extraction consists of finding for each tile which other pieces are likely to be its neighbor relying on their color information. When puzzle pieces bring equivalent color information, compatibility extraction can become pretty complex. Tiles reordering is the sub-task of finding the best permutation of tiles such that the

result is as similar as possible to the original image. The main difficulty in tile reordering is the exponential increase of feasible solutions as the number of tiles grows.
For compatibility extraction, we propose a self-supervised deep neural network that should classify neighboring pieces similarly. Then, through a correlation measure, we find the compatibility of tiles and build a compatibility matrix. The proposed metric does not use any hand-crafted measure that evaluates the compatibility of adjacent pieces. Our deep neural network is trained first on the pretext task of detecting rotation of tiles, and then, on the actual task of learning compatibility. For puzzle reassembling, we use two different solvers that have already been exploited in literature, PSQP from Andalo et al.[1] and relaxation labeling from Khoroshiltseva et al.[14].
In the end, we test our solution on some publicly available datasets (MIT and McGill), and we compare it with some methods already present in the literature. We provide the feasibility of our solution by showing that our CNN, which works in an unsupervised domain, achieves slightly lower results compared to some of the best hand-crafted compatibility metrics.

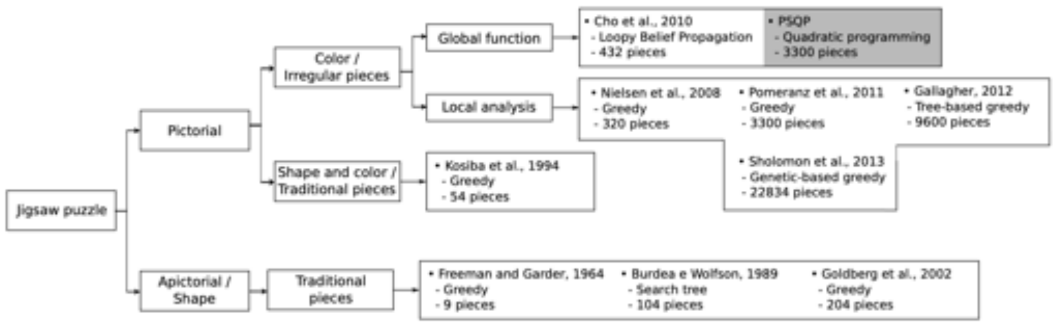

Jigsaw puzzle
Pictorial
Color / Irregular pieces
Global function
• Cho et al., 2010 - Loopy Belief Propagation - 432 pieces
• PSQP - Quadratic programming - 3300 pieces
Local analysis
• Nielsen et al., 2008 - Greedy - 320 pieces
• Pomeranz et al., 2011 - Greedy - 3300 pieces
• Gallagher, 2012 - Tree-based greedy - 9600 pieces
• Sholomon et al., 2013 - Genetic-based greedy - 22834 pieces
Shape and color / Traditional pieces
• Kosiba et al., 1994 - Greedy - 54 pieces
Apictorial / Shape
Traditional pieces
• Freeman and Garder, 1964 - Greedy - 9 pieces
• Burdea e Wolfson, 1989 - Search tree - 104 pieces
• Goldberg et al., 2002 - Greedy - 204 pieces

Fig. 1: Classification of some puzzle solving methods (from [1]).

2 Related Work

In Figure 1 we sum up some of the methods that we are mentioning below.

In 1760, John Spilsbury created the first jigsaw puzzle. He attached one of his world maps to wood and sliced each country out, giving it to children for geography education [36]. Later, in 1964, Freeman and Garder [7] presented the first attempt to computationally solve jigsaw puzzles. In the later years, other methods concerning solving apictorial puzzles were published [3, 9, 27]. In apictorial puzzles, the pieces should be matched only by their shape, instead, in pictorial puzzles, identical rectangular pieces, traditional pieces, and also irregularly shaped pieces are considered and the matching characteristic is mainly the chromatic information of the pieces. Figure 2 shows some examples of different types of puzzles. The first to propose a method to use both jigsaw shape and image information was Kosiba et al.[15], they were able to successfully solve puzzles of 54 pieces with a greedy strategy. Their tile matching process was the first to compute adjacent tiles compatibility by taking into account color samples along the edges. Since there, a few other similar methods have been proposed [18, 26, 32], then the research focus shifted to merely color-based square-tiles puzzle solvers.
In 2010, Cho et al.[5] presented a probabilistic pictorial puzzle solver which obtained an approximate reconstruction of the original image using graphical

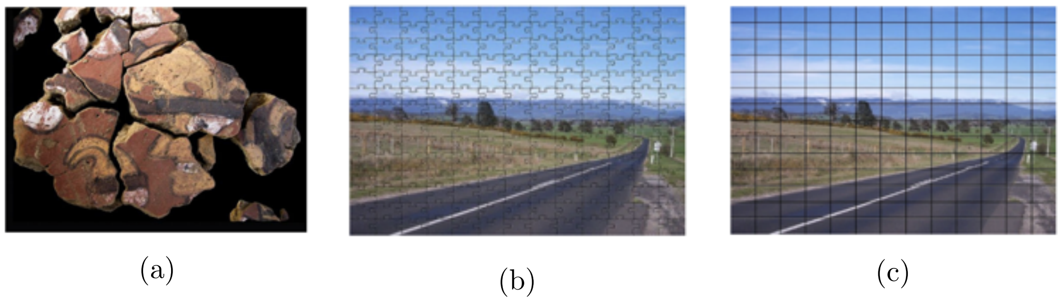


Fig. 2: Different design of puzzle problems. (a) irregular shaped pieces, (b) traditional shaped tiles, (c) linear boundary tiles.

models and a probabilistic function. The method needs prior knowledge about the layout of the original image, to achieve that they exploited two strategies: estimation of the image in low resolution from a few tiles, to serve as local evidence in the graphical model; and the correct fixation of some tiles by the user, called anchors. This method was capable to solve puzzles up to 432 pieces, but it requires some user intervention.
Furthermore, Pomeranz et al.[22] introduced a fully automated square jigsaw puzzle solver based on a greedy method able to handle puzzles of up to 3000 pieces. Their strategy consists of, first, a compatibility function to measure affinity between every pair of tiles. Then there are three modules, positioning, segmentation, and translation. The positioning module places all the tiles on the grid according to predetermined logic and takes into account the seeds selected at random; the segmentation module uses the best buddies metric to find regions that are assembled correctly; and the translation module which relocates both regions and tiles on the board such that a better solution is reconstructed. A more general solution was proposed by Gallagher et al.[8], the method employs neither piece orientation nor puzzle dimension but it works on square pieces and it uses a new compatibility measure for quantifying the compatibility of possible jigsaw piece matches based on expecting smoothness in gradient distributions across boundaries and a tree-based greedy approach. This strategy allows for reassembly puzzles of up to 9600 pieces.
Sholomon et al. [29] introduced a genetic algorithm that with knowledge of

tile rotation and puzzle dimensions can solve puzzles up to 22834 puzzle pieces. In literature other strategies of puzzle solving have been studied, some of that is the methods suggested by Andalo et al.[1] and Khoroshiltseva et al.[14]. Andalo et al.[1] reduced the formulation of the puzzle solving problem to the maximization of a constrained quadratic function, which can be solved by the gradient ascent approach [24]. The proposed deterministic method can solve puzzles of up to 3300 pieces and even with arbitrary identical rectangular pieces. Khoroshiltseva et al.[14] also provided a novel method that uses some of the previously cited compatibility measures but abstracts the puzzle solving task as a consistent labeling problem which amounts to maximizing a quadratic function over a probability space that can be solved using standard relaxation labeling algorithms.
Sholomon et al.[30] proposed a different way of extracting compatibility, in their paper is presented the first deep neural network capable to predict with high precision whether two tiles of a puzzle are neighboring or not. But they stated that they trained their network in a supervised manner. In our work, we try to find a model capable of extracting compatibility by exploring the unsupervised learning theme.

3 Puzzle Solving

3.1 Relaxation Labeling

First of all, we formulate the task of puzzle solving as a consistent labeling problem that needs to satisfy particular compatibility relations and with a one-to-one correspondence between the puzzle's tiles and their position. We can solve this kind of problem using the relaxation labeling algorithm which has the advantage of avoiding the choice of a step size [21].
A consistent labeling problem can be defined as follow, suppose to have: a set of n objects $B = \{1, \dots, n\}$; a set of m labels $\Lambda = \{1, ..., m\}$. The goal of these kinds of problems is to assign a label of Λ to each object of B.
In the beginning , for each object $i \in B$, the algorithm starts with an initial m-dimensional probability vector:

$$p_{i\lambda}^{(0)} = (p_{i1}^{(0)}, ..., p_{im}^{(0)})^T$$

with $p_{i\lambda}^{(0)} \geq 0$ and $\sum_\lambda p_{i\lambda}^{(0)} = 1$.
Each $p_{i\lambda}^{(0)}$ represents the probability distribution that the object b_i at time 0 is labeled with λ. There is one probability distribution associated with each object in B and if we concatenate all of these probabilities vectors $p_1^{(0)}, ..., p_n^{(0)}$, we get the initial weighted labeling assignments $p^{(0)} \in \mathbb{R}^{nm}$. Otherwise, the labeling assignments can be handled as a matrix p that belongs to the space IK defined as:

$$IK = \Delta^m = \Delta \times \cdots \times \Delta \tag{3.1.1}$$

where Δ is the standard simplex of $\mathbb{R}^n$.

$$\Delta^m = \left\{ p \in \mathbb{R}^m \mid p_{i\lambda} \geq 0, \lambda \in \Lambda \quad \wedge \quad \sum_{\lambda=1}^{m} p_{i\lambda} = 1, i = 1, ..n \right\} \tag{3.1.2}$$

each vertex of IK is called unambiguous labeling assignment [21].
An initial labeling assignment may be based on local measures that capture the relevant features of individual isolated objects. Another kind of information is

used, contextual information helps to enhance the weak labeling assignments given by the local measurements. Contextual information is expressed in terms of $n^2 \times m^2$ matrix of compatibility coefficients:

$$R = r_{ij\lambda,\mu}, \tag{3.1.3}$$

where $r_{ij\lambda,\mu}$ reflects the compatibility between the hypothesis b_i has label λ and b_i has label μ.
We can measure the support given by the context to the hypothesis b_i has label λ at time t as [12]:

$$q_{i\lambda}(t) = \sum_j \sum_\mu r_{ij\lambda\mu} p_{j\mu}(t) \tag{3.1.4}$$

By properly weighting and combining the support of all labels at all objects, we can also quantify the average support of the assignment, or the so-called average local consistency [12][14]

$$A(p) = \sum_{i,j} \sum_{\lambda,\mu} r_{ij\lambda\mu} p_{i\lambda} p_{j\mu} \tag{3.1.5}$$

A labeling assignment p is consistent if for all $v \in \Delta^{n\times m}$

$$\sum_\lambda^m p_{i\lambda} q_{i\lambda} \geq \sum_\lambda^m v_{i\lambda} q_{i\lambda} \qquad \forall i = 1, ...n \tag{3.1.6}$$

and if the matrix R is symmetric, then any local maximizer $p \in \Delta^{n\times m}$ of $A(p)$ is consistent [12].
At each iteration step, the algorithm updates the probability vectors using the following heuristic formula, provided by Rosenfeld, Hummel, and Zucker in

1976 [25, 21]:[1]

$$p_{i\lambda}(t+1) = \frac{p_{i\lambda}(t)q_{i\lambda}(t)}{\sum_{\mu} p_{i\mu}(t)q_{i\mu}(t)} \qquad \forall i, \lambda \tag{3.1.7}$$

Advantages of this update rule are that it does not require the choice of step size, and, that the non-negativity and symmetry conditions on the matrix R, guarantee convergence to a consistent labeling [25].

To formulate the task of puzzle solving as a consistent labeling problem, we define the set of objects B as the puzzle tiles, the labels Λ as all the possible positions in the grid, and the goal is for each tile from B to find a different position from Λ such that the average local consistency is maximized. Unless we are under the case of "ideal" compatibility, the algorithm might converge to an incorrect permutation matrix. Even if the update rule guarantees that p is a stochastic matrix, rows sum to 1, and we do not have the same constraints for its columns. Thereby, the process can converge to a solution where several tiles are assigned to the same position and other positions have no tiles at all assigned to them. Hence, to help the relaxation algorithm to converge into a permutation matrix, we performed the Alternating projection [14]. This procedure starts after t steps, we choose $t = 10$, so first, it uses the update rule (3.1.7) for t steps, then it switches alternatively between the update rule in equation (3.1.7) and the following:

$$p_{i\lambda}(t+1) = \frac{p_{i\lambda}(t)q_{i\lambda}(t)}{\sum_{j} p_{j\lambda}(t)q_{j\lambda}(t)} \qquad \forall i, \lambda \tag{3.1.8}$$

Use of this procedure is possible since objects and labels in the puzzle solving abstraction are interchangeable.

To sum up, we initialize the initial label assignment $p(0)$ to the barycenter of the multidimensional simplex, $p_{i\lambda}(0) = \frac{1}{m}$ for all i and all λ and we update

[1]Rosenfeld, Hummel, and Zucker in their original paper [25] used $1 + q_{i\lambda}(t)$ instead of $q_{i\lambda}(t)$, because their qs were correlation measures. Overall, their formulation is interchangeable with what is stated here [21]

permutation matrix according to the alternating projection method until convergence. We reach convergence when the euclidean difference between $p(t)$ and $p(t+1)$ is lower than a given threshold or when we reach the maximum number of iterations that we set at 200.

3.2 PSQP

In this section, we explain the functioning of Puzzle Solving by Quadratic Programming (PSQP). Also this method, as the ReLab, is based on the maximization of a global compatibility function, but it requires additional constraints and uses the gradient projection method proposed by Rosen to solve the maximization problem. [24]. Maximizing the global compatibility function should bring us to a solution where we find the best permutation among tiles, and a local compatibility measure of tiles being placed in neighboring locations.
PSQP does not need any hyperparameter, we give it only the compatibility matrix and the number of tiles.
To explain the PSQP, first, we will illustrate the compatibility function and show its reformulation as a quadratic homogeneous function. Then, we are going to describe the method used to solve the optimization problem.

3.2.1 Global compatibility function

As done for the Relaxation Labeling, we have a 2D grid with N locations labeled $1, ...N$ and N tiles, $t_1, ..., t_N$. And we have to find a rearrangement for each tile in a different location. The biunivocal correspondence between locations and tiles can be described by a permutation π of N tiles.
We arrange this formulation in a directed graph $G = \{V, E = E_H \bigcup E_V\}$, where the vertices represent the locations, $V = \{1, \ldots, N\}$, and the edge set E include all pairs of neighboring locations. Sets E_H and E_V represent horizontal and vertical neighboring locations, respectively. Graph G is directed because a switch of two tiles results in a global compatibility value change.
For each pair of tiles (t_i, t_j), for $1 \leq i, j \leq N$ and $i \neq j$, we define two local compatibility measures, $C_{H_{i,j}} \geq 0$ and $C_{Vi,j} \geq 0$, which correspond to the compatibility of the two tiles being associated with locations connected by any horizontal edge $e \in E_H$ or vertical edge $e \in E_V$. Finally, the global

compatibility function of a permutation π is

$$\varepsilon(\pi) = \sum_{(i,j)\in E_H} C_{H_{\pi(i)\pi(j)}} + \sum_{(i,j)\in E_V} C_{V_{\pi(i)\pi(j)}} \tag{3.2.1}$$

where (i, j) is the edge that connects the adjacent locations i and j, and $\pi(i)$ is the tile permuted to location i [1].
Taking in account all permutations π of N elements, the goal is to maximize this function (eq 3.2.1).

3.2.2 Reformulation of global compatibility function

Next, we reformulate the global compatibility function (eq. 3.2.1) as a homogeneous quadratic function of a square matrix.
A permutation matrix can be used to represent each permutation π of N elements:

$$P_{ik} = \begin{cases} 1, & \text{if } k = \pi(i) \\ 0, & \text{if } k \neq \pi(i) \end{cases} \tag{3.2.2}$$

Permutation matrices are a special case of doubly stochastic matrices [28], non-negative matrices in which each row and each column sum up to 1.
With this notation, we can rewrite the equation 3.2.1 as:

$$\varepsilon(\pi) = \sum_{(i,j)\in E_H} (P^T C_H P)_{ij} + \sum_{(i,j)\in E_V} (P^T C_V P)_{ij} \tag{3.2.3}$$

where $(P^T CP)_{ij}$, corresponding to edge $e = (i, j)$, is the element i, j of the square matrix $(P^T CP)$. Note that, for each edge $e = (i, j)$, the term $(P^T CP)$ is a homogeneous non-negative quadratic function of elements of matrix P [1].
If we stack up the columns $p_1, ..., p_N$ of matrix P of dimensions $N \times N$ in a

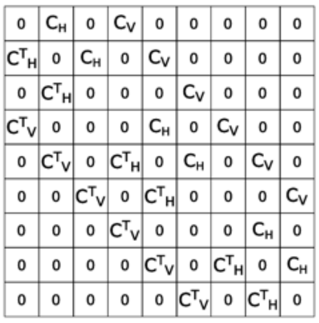
column-vector p of dimension N^2, we get

$$\varepsilon(\pi) = \sum_{(i,j)\in E_H} (p_i^T C_H p_j) + \sum_{(i,j)\in E_V} (p_i^T C_V p_j)_{ij} \tag{3.2.4}$$

Last, equation (3.2.4) can be rewritten in the canonical quadratic form $p^T Ap$, where A is a symmetric non-negative block matrix of dimension $N^2 \times N^2$ that corresponds to the Hessian of $\varepsilon(P)$

$$\varepsilon(P) = p^T Ap \tag{3.2.5}$$

For example, if we have a 3x3 puzzle, the block matrix A have dimensions 81x81 and each block is a 9x9 null matrix, C_H, C_V, or their transposes.



0	C_H	0	C_V	0	0	0	0	0
C^T_H	0	C_H	0	C_V	0	0	0	0
0	C^T_H	0	0	0	C_V	0	0	0
C^T_V	0	0	0	C_H	0	C_V	0	0
0	C^T_V	0	C^T_H	0	C_H	0	C_V	0
0	0	C^T_V	0	C^T_H	0	0	0	C_V
0	0	0	C^T_V	0	0	0	C_H	0
0	0	0	0	C^T_V	0	C^T_H	0	C_H
0	0	0	0	0	C^T_V	0	C^T_H	0

Fig. 3: An example of the symmetric block matrix A of dimensions 81x81, where each block is a 9x9 matrix.

3.2.3 Constrained Gradient Ascent

An important point is that each doubly stochastic matrix satisfies N^2 inequality constraints, which specify that the elements of p are non-negative, and $2N$ equality constraints, which specify that the sum of elements of each row and each column of P is equal to 1. Now, we have all the ingredients to introduce the formulation of the problem as a quadratic optimization problem and the algorithm to solve it.
If we extend the domain of $\varepsilon(P)$ for all doubly stochastic matrices, we can reduce the problem to finding a solution for the quadratic optimization problem:

$$\begin{aligned} \max \quad & f(p) = p^T A p, \\ \text{s.t.} \quad & P\overline{1} = \overline{1}, \\ & P^T\overline{1} = \overline{1}, \\ & p_k \geq 0, \qquad\qquad \forall k = 1, ..., N^2 \end{aligned} \tag{3.2.6}$$

where $\overline{1}$ is an N-column vector of ones.
Since all diagonal values of A are zeros, we have that the matrix is not positive definite nor positive semi-definite. This means that even if $f(p)$, the objective function, is positive on the feasible set, it is not necessarily concave. So, there is no guarantee to reach the global maximum, but working with the constraints, we can search for a local maximum of Equation (3.2.6) that represent a close solution to the global maximum.
The chosen algorithm is a constrained gradient ascent method with gradient projection [24].
The algorithm has a set of active variables, that are used to log the ones that are still in the feasible set. To avoid further updates on the variables, this active set is used to deactivate the variables that reach the boundary of the set. In this way, every time a variable is deactivated, we reduce the dimensionality of the problem by one. In the begin all the variables are initialized as $p_{kl} = \frac{1}{N}$, for $1 \leq k, l \leq N$ and the whole set to active, $active_{kl}$ tell if p_{kl} is active or not. The ascent direction is computed as $d = \nabla f(p) = 2 * A * p$ at current p. But, it may be that the ascent direction does not remain in the space defined by the linear equality constraints. Hence, there is the need to project the ascent direction d into the space defined by the linear equality constraints. To derive the constrained ascent direction s we first need to find the projection matrix as done in equation (3.2.29) in section 3.2.4 [24].
Then we normalize the ascent direction s as in equation (3.2.30). Thus, p is computed as $p_{kl} \leftarrow p_{kl} + step * s$, for $1 \leq k, l \leq N$ and $active_{kl} = true$, to estimate the best value for $step$ such that $0 \leq p_{kl} \leq 1$, we used the line search function provided by scipy [35].
Whenever a variable reaches the boundary of the feasible region, to keep it there and not update it anymore, the constraints should be modified and

p must be re-initialized. For the purpose of reinitialize p, first the variables equal to 0 or 1 are deactivated, then we initialize $p_{kl} \leftarrow \frac{1}{N-N_a}$, for $1 \leq k, l \leq N$ *and* $active_{kl} = true$, where N_a is the number of variables that have already reached the upper limit. A variable reaches the upper limit when it is equal to 1, which means that the variable has been assigned to a location. The algorithm ends when all locations have a variable assigned.
The following pseudo-code gives a rough idea of the used algorithm [1].

Algorithm 1: Constrained Gradient Ascent

```
Data: block matrix A, number of tiles N.
Result: permutation $\pi$ of tiles.
$N_a \leftarrow 0$
$active_{kl} \leftarrow true$, for $1 \leq k, l \leq N$
while $N_a \leq N$ do
    $p_{kl} \leftarrow \frac{1}{N-N_a}$, for $1 \leq k, l \leq N$ and $active_{kl} = true$
    $d \leftarrow \nabla f(p) \leftarrow 2 * A * p$
    $s \leftarrow Kd$, where K is the projection matrix
    $p_{kl} \leftarrow p_{kl} + step * s$, for $1 \leq k, l \leq N$ and $active_{kl} = true$
    for $1 \leq k, l \leq N$ and $active_{kl} = true$ do
        if $p_{kl} = 0$ then
            $active_{kl} \leftarrow false$
        end
        if $p_{kl} = 1$ then
            $active_{kl} \leftarrow true$
            $\pi(l) \leftarrow k$
            $N_a \leftarrow N_a + 1$
        end
    end
end
```

3.2.4 Rosen Gradient Projection Method

In the previous subsection, we have seen that there is the need to project the ascent direction into the space defined by the linear equality constraints, the computation of the search direction is then defined in the following lines.
A problem with m variables, x_i, $i = 1, ..., m$ is considered. Any set of values for the x_i can be represented by an m-dimensional column vector,

$$x = \{x_1, x_2, ..., x_m\} \tag{3.2.7}$$

where x_i represents a point in a Euclidean m-dimensional space E_m.
By assumption, these variables are constrained by a set of k linear inequalities, which form a convex region R in E_m. A point in the convex region R is called a feasible point. The constraints are in the form

$$\sum_{j=1}^{m} n_{ij} x_j - b_i \geq 0 \qquad (i = 1, 2, ..., k) \tag{3.2.8}$$

where the n_{ij} have been normalized, so that

$$\sum_{j=1}^{m} n_{ij}^2 = 1 \qquad (i = 1, 2, ..., k) \tag{3.2.9}$$

To each of the k constraints, there is a corresponding vector n_i defined as

$$n_i = \{n_{i1}, n_{i2}, ..., n_{im}\} \tag{3.2.10}$$

These are unit vectors, since $|n_i|^2 = n_i^T n_i = \sum_{j=1}^{m} (n_{ij})^2 = 1$, by equation (3.2.9). Thus, the inequalities in equation (3.2.8) can be written as

$$x^T n_i - b_i = g_i(x) \geq 0 \qquad (i = 1, 2, ..., k) \tag{3.2.11}$$

The $(m-1)$-dimensional manifold defined by $g_i(x) = 0$ is a hyperplane which will be denoted by H_i

$$H_i : \quad g_i(x) = 0 \qquad (i = 1, 2, ..., k) \tag{3.2.12}$$

All the points for which $g_i(x) \geq 0 \quad i = 1, 2, ..., k$ form the closed convex region R and all the points for which $g_i(x) = 0$ for at least one i form the boundary B of R. The unit vector n_i is orthogonal to H_i and is directed so that if it originates at a point x in H_i it points "into" the region R.

Now, consider a problem with linear constraints

$$\text{Maximize } f(x)$$

subject to

$$g_i(x) = x^T n_i - b_i = \sum_{j=1}^{m} n_{ij} x_j - b_i \geq 0 \qquad (i = 1, 2, ..., k) \tag{3.2.13}$$

Define the $m \times k$ matrix:

$$N = [n_1 \; n_2 \; \dots \; n_k] \tag{3.2.14}$$

and the k-dimensional vector

$$b = [b_1, \; b_2, \; \dots \; b_k] \tag{3.2.15}$$

Then the system of inequalities 3.2.13 can be written conveniently as

$$N^T x - b \geq 0 \tag{3.2.16}$$

The direction-finding problem for obtaining a usable feasible direction S can be formulated as

$$\text{Find S which maximize } S^T \nabla f(x)$$

subject to

$$N^T S = 0 \tag{3.2.17}$$

$$S^T S - 1 = 0 \tag{3.2.18}$$

where the equation (3.2.18) denotes the normalization of the vector S. To solve this equality constrained problem, we construct the Lagrangian function as

$$L(S, \lambda, \beta) = S^T \nabla f(x) + \lambda^T N^T S + \beta(S^T S - 1) \tag{3.2.19}$$

where

$$\lambda = \begin{Bmatrix} \lambda_1 \\ \lambda_2 \\ \dots \\ \lambda_k \end{Bmatrix} \tag{3.2.20}$$

is the vector of Lagrange multipliers associated with equation (3.2.17) and β is the Lagrange multiplier associated with equation (3.2.18). The necessary conditions for the maximum are given by

$$\frac{\partial L}{\partial S} = \nabla f(x) + N\lambda - 2\beta S = 0 \tag{3.2.21}$$

$$\frac{\partial L}{\partial \lambda} = N^T S = 0 \tag{3.2.22}$$

$$\frac{\partial L}{\partial \beta} = S^T S - 1 = 0 \tag{3.2.23}$$

Equation (3.2.21) gives

$$S = \frac{1}{2\beta}(\nabla f + N\lambda) \tag{3.2.24}$$

Then, substitution of equation (3.2.24) into equation (3.2.22) gives

$$N^T S = \frac{1}{2\beta}(N^T \nabla f + N^T N\lambda) = 0 \tag{3.2.25}$$

If S is normalized according to (3.2.23), β will not be zero, thus equation (3.2.25) gives

$$N^T \nabla f + N^T N \lambda = 0 \tag{3.2.26}$$

from which we can extract λ as

$$\lambda = (N^T N)^{-1} N^T \nabla f \tag{3.2.27}$$

from this last equation, if we substitute λ, equation (3.2.24) gives

$$S = \frac{1}{2\beta}(I - N(N^T N)^{-1} N^T) \nabla f = \frac{1}{2\beta} P \nabla f \tag{3.2.28}$$

where P is the so called projection matrix

$$P = I - N(N^T N)^{-1} N^T \tag{3.2.29}$$

Forgetting about the scaling constant 2β, we have that the matrix P projects the vector $\nabla f(x)$ into a convex region bounded by the constraints.
We assume that the constraints $g_i(x)$ are independent so that the columns of the matrix N will be linearly independent, and hence $N^T N$ will be non-singular and invertible. The vector S can be normalized as

$$s = \frac{P \nabla f}{||P \nabla f||} \tag{3.2.30}$$

If x_i is the starting point for the i-th iteration, from equation (3.2.30) we compute S_i as

$$s_i = \frac{P_i \nabla f(x_i)}{||P_i \nabla f(x_i)||} \tag{3.2.31}$$

where P_i represent the projection matrix P evaluated at the point x_i [23].

3.2.5 Linearly dependent constraints

There might be the case where in the Euclidean space E_m there are linearly dependent hyperplanes, which requires an additional procedure. The steps can be summarized as follow, assume that a point x_0, which can be an initial point or obtained during the optimization, lies in a total of $q+p$ hyperplanes of which only q are into the projection matrix P_q. The vector S is computed according to (3.2.30), and the quantities

$$\theta_i = s^T n_i \qquad (i = q+1, \dots, q+p) \tag{3.2.32}$$

obtained. For any H_i linearly dependent on the original set of q hyperplanes, the corresponding $\theta_i = 0$. Let $\theta = min\ \theta_i,\ i = q+1, \dots, q+p$.
Suppose $\theta \leq 0$ and $\theta = \theta_i$ for $i = q+1$. Then add H_{q+1} to the projection matrix P_q giving P_{q+1}. With P_{q+1} replacing P_q in equation (3.2.30) a new vector s is computed. Replicating this process will retrieve a vector s such that $N_{p+q}s \geq 0$ and $s^T \nabla f(x_0) > 0$ allowing to take a step with an increase in $f(x)$, or a projection matrix P_{q+l}, $1 \leq l \leq p$, will be obtained for which $P_{q+l} \nabla f(x_0) = 0$. In the second case, the discussion at the end of the previous section applies and x_0 is either a global maximum or a suitable ascent direction s can be found. In the case where x_0 is an initial point that lies on the boundary of the closed convex region R, the just discussed procedure applies with $q = 0$ and p representing the number of hyperplanes containing x_0 [24].

3.2.6 Reformulation of the quadratic optimization problem

In order to apply Constraint gradient projection method formulated by Rosen [24], we need to transform the equality constraints in equation (3.2.6) into inequality constraints as in equation (3.2.13).
Let i and k be two tiles of the puzzle, with $i, k \in \{1, \dots, N\}$, then define an

one-hot encoding column vector o_i of dimension N, as follow:

$$o_i(k) = \begin{cases} 0, & \text{if } i \neq k \\ 1, & \text{if } i = k, \qquad \forall k = 1, \dots, N \end{cases}$$

Further, let:

$$\mathbb{1}_i^{(n)} = o_i^t \otimes \overline{1}^T$$

with $\otimes$ being the Kronecker product and $\overline{1}$ being the N-dimensional column vector of one, in other words

$$\mathbb{1}_i^{(n)} = (0, \dots, 0, \overbrace{1, \dots, 1}^{n*i,\dots,n*(i+1)}, 0, \dots, 0)^T$$

where the j-th element is equal to 1 if it is in the i-th row of P.

By inverting the one-hot vector and the vector of ones in the Kronecker product we get the same kind of row-vector but row related.

$$\mathbb{1}'^{(n)}_i = \overline{1}^T \otimes o_i^T$$

Last, we flatten the matrix P in the column-vector $\overline{p}$

$$\begin{aligned} \max \quad & f(p) = \overline{p}^T A \overline{p}, \\ \text{s.t.} \quad & p_{i\cdot}\overline{1} - \overline{1} \geq 0, \\ & -p_{i\cdot}\overline{1} + \overline{1} \geq 0, \\ & p_{\cdot j}\overline{1} - \overline{1} \geq 0, \\ & -p_{\cdot j}\overline{1} + \overline{1} \geq 0, \\ & p_k \geq 0, \qquad \forall i = 1, ..., N \\ & \qquad\qquad \forall j = 1, ..., N \\ & \qquad\qquad \forall k = 1, ..., N^2 \end{aligned} \tag{3.2.33}$$

By plugging the previous formulas of the one-hot encoding vector o_i, $\mathbb{1}_i^{(n)}$ and $\mathbb{1}'^{(n)}_i$ into the problem (3.2.33), we finally obtain

$$\begin{aligned} \max \quad & f(p) = \overline{p}^T A \overline{p}, \\ \text{s.t.} \quad & \mathbb{1}_i^{(N)} \overline{p} - \overline{1} \geq 0, \\ & -\mathbb{1}_i^{(N)} \overline{p} + \overline{1} \geq 0, \\ & \mathbb{1}'^{(N)}_i \overline{p} - \overline{1} \geq 0, \\ & -\mathbb{1}'^{(N)}_i \overline{p} + \overline{1} \geq 0, \qquad \forall i = 1, ..., N \\ & o_{(i)}^{(N^2)} \overline{p} \geq 0, \qquad\qquad\quad \forall i = 1, ..., N^2 \end{aligned} \tag{3.2.34}$$

3.2.7 PSQP tricks

Two new issues arise, how to rearrange constant tiles and how to handle the non-concave property of the global compatibility function. First, the constant tiles are tiles whose borders have identical color information. These types of tiles cause problems because they have total compatibility among them and the same compatibility with all other non-constant tiles. To solve this issue, we simply ignore constant tiles. If the total compatibility is replaced by zero compatibility, the method will not analyze these tiles, and they will be assigned to an empty location by the end of the optimization method [1]. As all the borders of the constant tile are equal, the permutation used to place them does not matter. Note that, we do not distinguish distinct constant tiles, for example, all green tiles and all blue tiles are considered equally and handled in the same way.

The other issue is associated with the non-concavity of the global compatibility function, which means, there is no guarantee that the Gradient Ascent method reaches the global optimum. Andalo et al. [1] stated that in some images with quite a few constant tiles, the final permutation may be a displacement of the correct permutation. To adjust the displacement they proposed the following post-processing step: for all the final permutations π changed by every possible cyclical shift, considering each row and each column, compute the compatibility function $\varepsilon(\pi)$. Then, pick the shift that generates the bigger increase in global compatibility.

The post-processing step can be summarized in the Algorithm 2.

Algorithm 2: Post-Processing

```
**Data:** the final permutation $\pi$
**Result:** permutation $\pi$ of tiles.
$\pi \leftarrow ConstrainedGradientAscent(A, N)$
$st_{H_{max}}, st_{V_{max}} \leftarrow argmax \quad GlobalCompatibility(\pi, st_H, st_V)$
$\pi_{final} \leftarrow Shift(\pi, st_{H_{max}}, st_{V_{max}})$
```

In Algorithm 2, function *ConstrainedGradientAscent* returns the permutation of tiles π, function *GlobalCompatibility* computes the global compatibility $\varepsilon(\pi)$ after applying to π all the possible cyclical shifts horizontally st_H and vertically st_V. $st_{H_{max}}$ $st_{V_{max}}$ are respectively the horizontal shift and vertical shift that generate the highest $\varepsilon(\pi)$. Last, function $Shift$ applies to the permutation π the shifts $st_{H_{max}}$ and $st_{V_{max}}$

4 Learning Compatibility

As of now, most of the puzzle solving works rely on a highly precise hand-craft compatibility measure to predict whether two tiles are neighboring or not.
We aim to learn compatibility by exploiting the power of feature extraction through deep neural networks. The neural network should be capable to find similar embedding for adjacent pieces. These embedding will be given to a similarity function that will tell if the tiles have been classified similarly.
Our methods will be composed of three steps. First, a deep neural network classifies each tile in the image. The embedding of each tile will be given in pair to a correlation similarity function which will return the actual compatibility of the tiles. In the end, one of the previously mentioned solvers will be used to find the best permutation of tiles.
Assuming that given a good compatibility matrix, we can solve any puzzle up to about 400 tiles. From now, we focus on finding a neural network that given all the tiles of the puzzle can extract such compatibility.
For both of our approaches, we use the ResNet50 as the base network and we do transfer learning on it. In each dataset, there are 20 different images, and all tiles have sizes of 28x28 pixels.

4.1 Architecture (ResNet50)

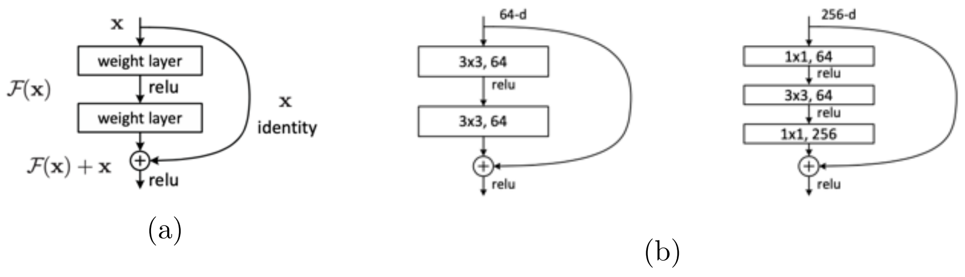


Fig. 4: (a) Residual building block. (b) Left: 2-layer building block for ResNet-34. (b) Right: ”bottleneck” building block for ResNet-50.

With the aim of implementing image classification algorithms for the ImageNet Large Scale Visual Recognition Challenge, a lot of deep networks have been exploited. Deeper neural networks can learn more complex functions and extract more features, but with the increase of depth, the problem of degradation has been exposed, accuracy gets saturated and then degrades quickly. In residual neural networks, the degradation problem is addressed by introducing a deep residual learning framework.

The first introduced residual network was the ResNet-34 [11], where every few stacked layers, instead of fitting a desired underlying mapping, are required to fit a residual mapping. If we define the underlying mapping of a few stacked layers as $\mathcal{H}(x)$ where x is the inputs of the first of these layers, rather than expect stacked layers to approximate $\mathcal{H}(x)$, we explicitly let these layers approximate residual function $\mathcal{F}(x) := \mathcal{H}(x) - x$. Thus, the original function can be reformulated as $\mathcal{F}(x) + x$. It should be easier to optimize the residual mapping that to optimize the original, non-referenced mapping.

The reformulation of $\mathcal{F}(x) + x$ can be achieved by implementing shortcut connections into a feedforward neural network. Shortcut connections allow to skip some layer and add their output to the outputs of the stacked layers, this lets the model learn an identity function that guarantees that higher layers perform at least as good as lower layers, see figure 4 (a) to see an example.

ResNet-34 can be seen as a 34-layer plain network inspired by the VGG-19 where shortcut connections have been added. ResNet-50 is a residual neural network like the ResNet34 but deep 50 layers total, and where the 2-layer building block is replaced with a 3-layer bottleneck block. Shortcuts in ResNet-50 are either identity shortcuts or projection shortcuts used for increasing dimension. In figure 4 (b) we can see the residual learning block for ResNet-34 (left) and ResNet-50 (right). This model should solve the degradation problem and also may mitigate the vanishing gradients problem.

4.2 Baseline approach

In the first attempt, we feed the neural network with a piece of the puzzle and it returns an embedding. We expect to have similar embedding for neighboring

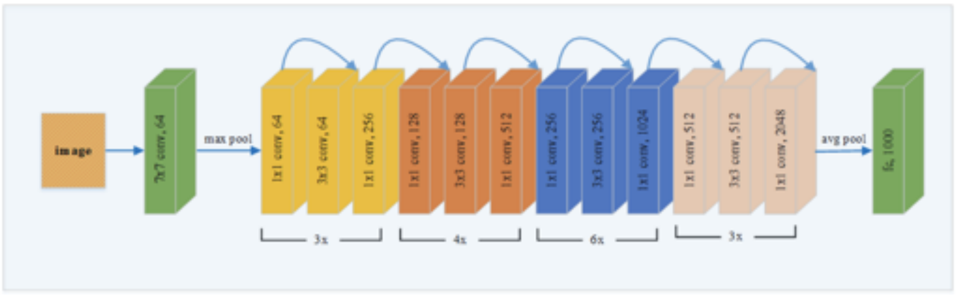


Fig. 5: Architecture of ResNet-50.

pieces. For this approach, the last fully connected layer of the ResNet-50 has been substituted with one that receives in input the features from the last convolutional layer and a position value that represents the position of a tile. Position value can either be 1, or -1, respectively for the left and right position of a horizontal pair, 2 and -2 represent respectively the top and the bottom positions of a vertical pair. The goal is to feed the network with the whole set of tiles of an image without any particular order with each of these position values, such that as output we have an embedding for every combination of tile and position value.

Then, to compute the horizontal and vertical compatibility we measure the similarity of these embedding through the correlation. To do so, we take a pair of embedding where one is from the set forwarded with position value 1 if we are computing horizontal compatibility, 2 otherwise and the other is from the set forwarded with position -1 if we are computing horizontal compatibility, -2 otherwise. Finally, we can compute the compatibility matrix. We want to have high compatibility for adjacent tiles.
To achieve this, in the training phase we use the triplet margin loss,

$$L(a,p,n) = max\left\{d(a_i,p_i) - d(a_i,n_i) + margin, 0\right\}$$

where

$$d(x_i,y_i) = \|x_i - y_i\|_p$$

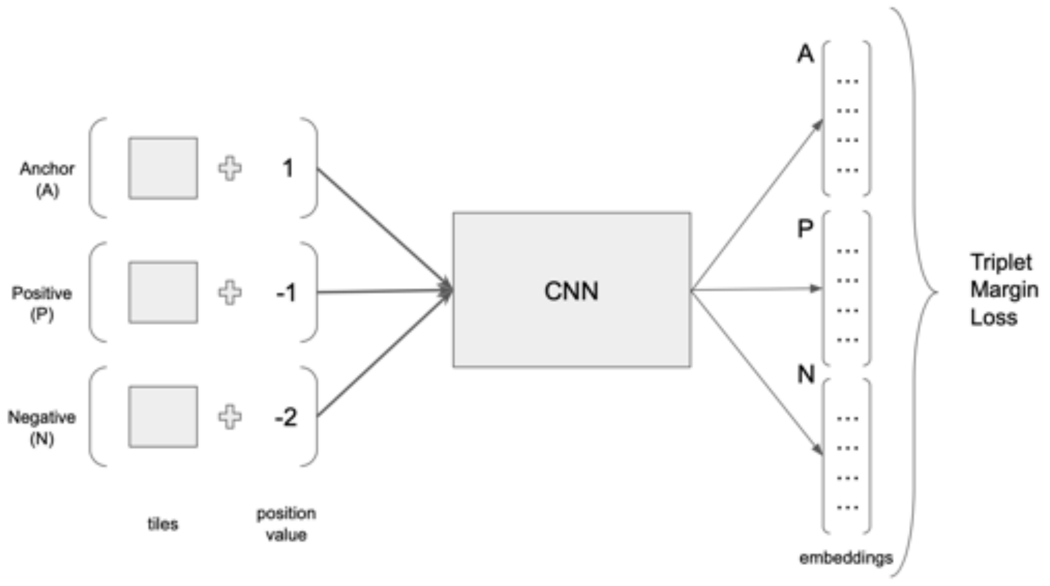


Fig. 6: Example of the whole pipeline for horizontal compatibility.

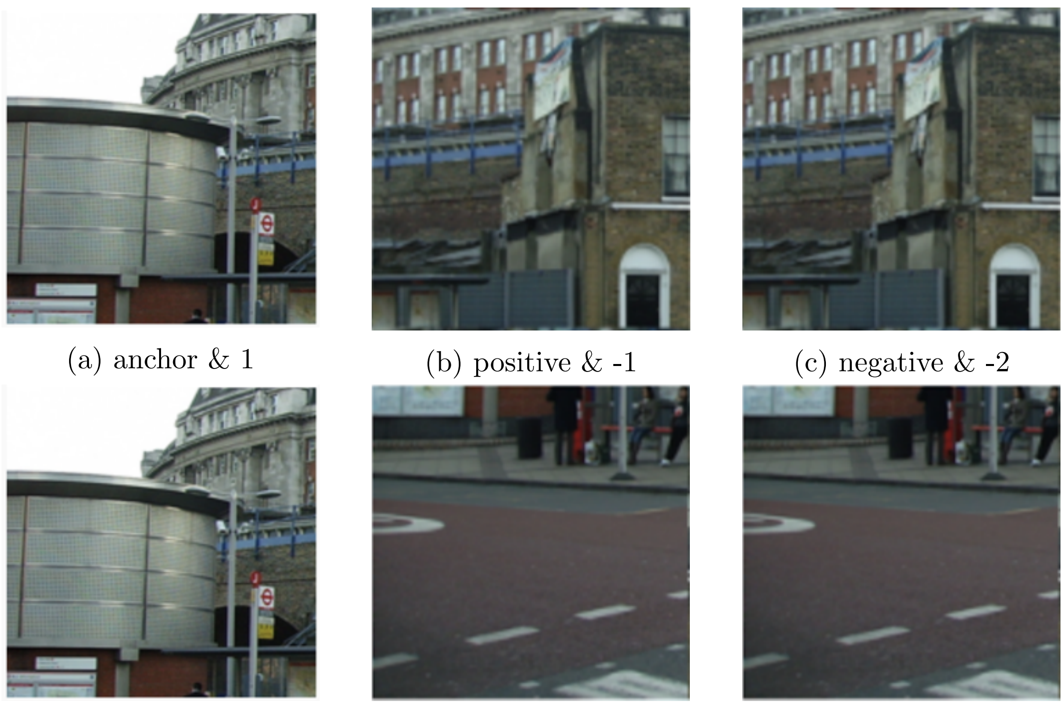
and a, p, n are respectively the anchor, positive and negative examples. The choice of these three examples is critical, so once we have picked an image, our method consists of selecting at random a tile and using it as an anchor example. According if we are looking for horizontal or vertical compatibility, take the tile immediately on the right or at the bottom of the anchor and use it for both positive and negative examples. For the positive and negative examples, we use respectively -1 and -2 for horizontal compatibility or -2 and -1 for vertical compatibility. Each of these examples with a position value run through the network and the resulting embeddings are used to compute the triplet loss.

Therefore, we assume that from the training set we have knowledge of the placement of pairs of tiles.

We choose to use the MIT dataset to train the network and to do validation on the McGill.

To sum up the whole pipeline, the input of the neural network is a tile and a position value, the CNN extracts some features from the image, which combined with the position value are given as input to the fully connected layer.

The output is an embedding that will be used to compute the compatibility. Two tiles next to each other should have similar embedding.
As we can see from table 1, the accuracy on the training set is more than double the accuracy on the validation set. Hence, as expected, the model is adapting too much to the distribution of the training set, and is not able to fit the validation set, which comes from a different distribution. In synthesis, this methodology is used as a baseline for comparison with the next approaches.



(a) anchor & 1 (b) positive & -1 (c) negative & -2

(d) anchor & 2 (e) positive & -2 (f) negative & -1

Fig. 7: Example of anchor, positive and negative samples as full tiles with position values for horizontal compatibility (top row) and vertical compatibility (bottom row).

In figure 7, we can see an example of anchor, positive and negative samples. For horizontal compatibility, we refer to the row on the top, in fig.7(a) we have the tile for the anchor example and it will be forwarded with position value 1. As positive and negative examples we use the same tile, that is the one on the right of the anchor, but with different position values, as we noted in fig.7(b) and fig.7(c). For vertical compatibility, we do it similarly but using the tile on the bottom of the anchor.

4.3 Self-supervised learning

In order to increase the performances on both datasets, we tried a similar approach to the one used by Noroozi and Favaro [19].
They proposed to build a convolutional neural network to be trained to solve a pretext task, Jigsaw puzzle solving, and then use the same model to solve their main task, object detection. In this way, they aim to have a network that is able to learn as representative and discriminative features as possible. After training the network on the pretext task, they use the just computed weights to initialize a standard AlexNet network, then they re-train the non-convolution layers of the AlexNet for object detection.
As our main task is Jigsaw Puzzle solving, more specifically being able to extract embeddings that are similar for tiles that are next to each other horizontally or vertically, we needed to find a pretext task that is at a lower level. Our choice is to use detecting rotation of tiles as a pretext task.
So, we still start from the ResNet-50 pre-trained on the ImageNet, but we replace the last fully connected layer, which is specialized in classification, with a new one that has four output neurons. Each output neuron corresponds to a different degree of rotation, we limited the cases to 0°, 90°, 180°, and 270°. Since the model achieved pretty accurate results on this task, we decided to move on to the actual task, learning compatibility.
Therefore we initialized a new ResNet with the weights of the one trained on detecting rotation but we changed the fully connected layer with four output neurons with a new one, which takes into account also the position as done for the baseline approach. To select examples, we use the same method proposed for the baseline approach, so we pick at random a tile to use as an anchor example. Next, we pick the tile on its right for horizontal compatibility or on its bottom for vertical compatibility, we call this match. The match tile is used as both positive and negative examples but with different position values (see

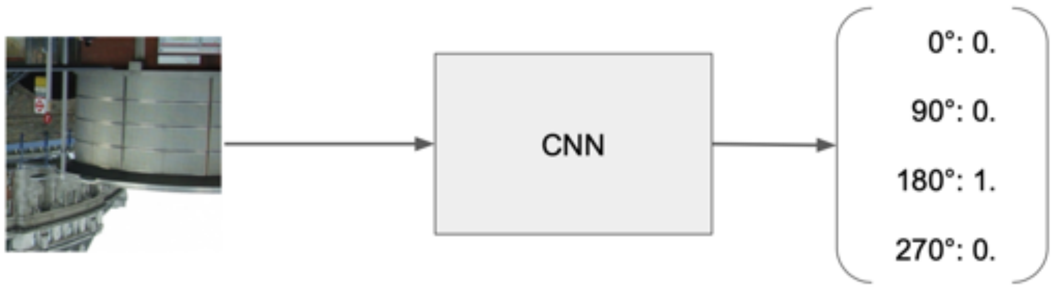


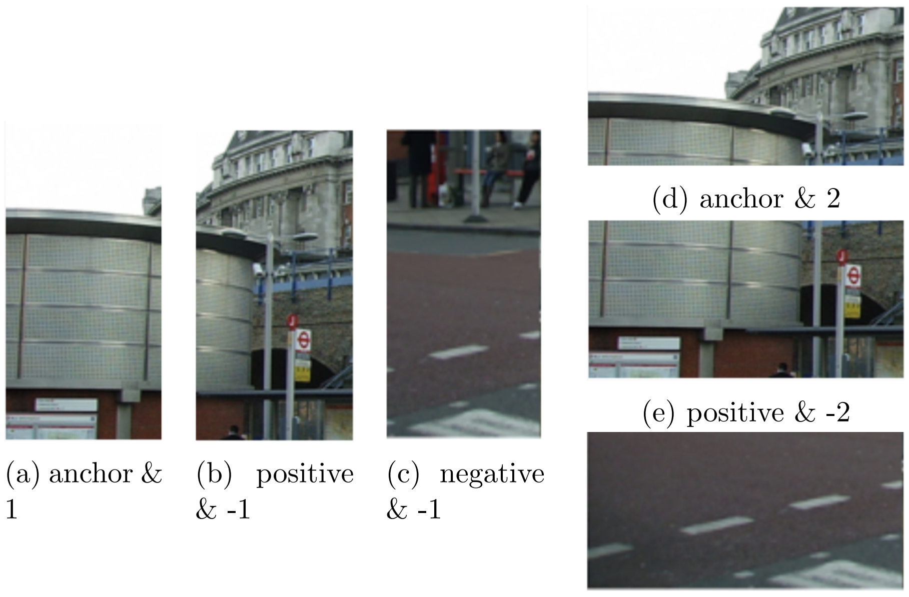
Fig. 8: Example of the pipeline to estimate rotation of tiles. In this case the output indicates that the tile is rotated of 180°.

an example in fig.7). Then, we freeze the convolutional layers and re-train the network to find similar embedding for neighboring tiles.

Even if with this first attempt we reached a bit higher accuracy, it does not seem to perform on McGill as well as on MIT. Therefore, we changed the method to select anchor, positive and negative examples. We tried to use half tiles instead of full tiles, which means, for horizontal compatibility, we randomly choose a full tile, and we split it into two halves, the left one is used as an anchor example and the right one as a positive example. For the negative, we pick a tile that is enough different from the first one, and we keep the half that is not too similar to the anchor. To learn vertical compatibility the steps are similar but all the tiles are rotated 90° counterclockwise. In figure 9 we have an example of how we split tiles for both horizontal and vertical compatibility and also the position values we forwarded together with the half tiles. With this method of selecting examples and frozen convolutional layers, we see an improvement in accuracy on both datasets (see table 4).

Noroozi and Favaro [19], in their paper used an AlexNet which does not have any batch normalization layer, so, one more time, we re-trained our network but freezing both convolutional and batch normalization layers. This brings some slight increase in the accuracy of compatibility, mainly on the test set (see table 5).

Until now, for the choice of the tiles to input into the network during the training phase, we choose randomly the first tile which is split into two halves, one



(a) anchor & 1 (b) positive & -1 (c) negative & -1 (d) anchor & 2 (e) positive & -2 (f) negative & -2

Fig. 9: Example of anchor, positive and negative samples as half tiles with position values for horizontal compatibility (a, b, c) and vertical compatibility (d, e, f).

for the anchor and the other to be used as a positive example. The tile from which we derive the negative example is chosen randomly between the set of tiles with a color difference higher than a threshold according to the Euclidean distance. We observed that the method used to select anchor, positive and negative examples is crucial for our training. If we always select negative samples that are quite different from the anchor we may let the network be ”lazy” and learn only the easy mismatching pairs and may not be able to recognize some that are harder to spot.

Thus, we change completely the way we choose anchor, positive and negative samples. As before, we still pick randomly a tile, but now we split it into 3 parts. Then, each part is forwarded two times through the network. The first time we forward them as anchor, positive and negative, respectively, for horizontal compatibility from left to right (see figure 10(b)) and for vertical compatibility from top to bottom (see figure 10(c)). The second time we forward them as negative, anchor, positive for horizontal compatibility from left

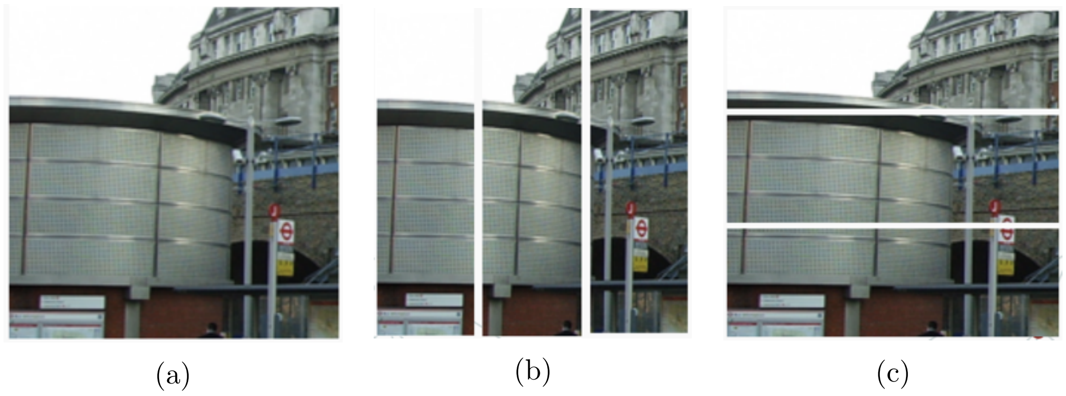


Fig. 10: Example of partitioning in 3 parts of a random tile. (a) the original tile, (b) partitioning for horizontal compatibility, and (c) partitioning for vertical compatibility.

to right and for vertical compatibility from top to bottom. With this technique, we push the network to learn to spot also small differences between different pieces. Moreover, we also try to force the CNN to mainly focus on the right-hand side of the image for horizontal compatibility and on the bottom side of the image for vertical compatibility.
To avoid confusion from learning both horizontal and vertical compatibility at the same time, we preferred to have two different neural networks each one trained only for horizontal compatibility or vertical compatibility.
We define as constant tile a piece whose borders have identical color information, which means monochromatic pieces or pieces with regular patterns (see fig.11). Furthermore, we noticed that more than 50% of each image in the datasets is composed of constant tiles (see fig.12). Hence, we decided to not take into account errors caused by constant tiles. In fact, if we switch two or more constant tiles, visually we do not notice any difference in the image reassembling. In table 6 we can see that without the errors caused by constant tiles we have a large increase in the accuracy.
In addition, if we compare results of the model that use triplets (table 6) with previous models (tables 3, 4, 5) we can immediately notice a huge improvement in the accuracy. With this last implementation, we were able to solve a lot of problems stated before, except how to handle constant tiles.

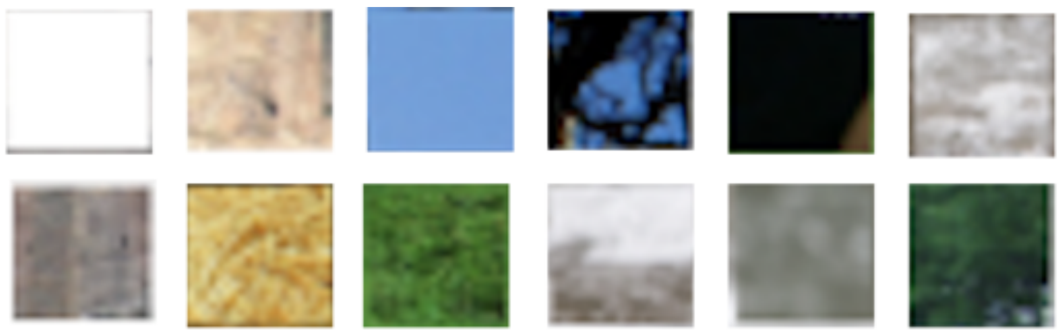


Fig. 11: Some examples of constant tiles, monochromatic or with repetitive patterns.

5 Local compatibility measures

To compare our model, we decided to implement a few hand-crafted compatibility metrics. All of the following methods are based on dissimilarity. In the proposed method by Cho et al [5], the horizontal dissimilarity between two tiles t_i and t_j is calculated as

$$D_{H_{ij}} = \left(\sum_{k=1}^{T} \sum_{l=1}^{3} (|t_i(k,T,l) - t_j(k,1,l)|)^p \right)^{\frac{q}{p}} \tag{5.0.1}$$

where T is the pixel dimension of a tile, 3 is the number of color channels (red, blue, and green), t_i and t_j are $T \times T \times 3$ matrices representing square tiles, and the color difference is computed in the normalized $L^*a^*b^*$ color space.
The local horizontal compatibility between two tiles t_i and t_j is computed from the dissimilarity measure as

$$C_{H_{ij}} \propto exp\left(-\frac{D_{H_{ij}}}{2\sigma_c^2} \right) \tag{5.0.2}$$

where σ_c is defined as the difference between the lowest and the second lowest $D_{H_{ij}}$, for $1 \leq j \leq N$.
The vertical dissimilarity $D_{V_{ij}}$ and the local vertical compatibility $C_{V_{ij}}$ are estimated similarly.
While Cho et al. [5] found their best result with $p = 2$ and $q = 2$, Pomeranz

et al. [22] noticed that the equation 5.0.1 with values $p = 2$ and $q = 2$ is related to the L_2 norm of the vector of differences across tiles borders and that different values of norm $(L_p)^q$ could provide better results. Pomeranz et al. [22] proposed $p = \frac{3}{10}$ and $q = \frac{1}{16}$ which bring to an improved compatibility.
Gallagher [8], proposed a slightly different measure called Mahalanobis Gradient Compatibility which penalizes changes in intensity gradients, learns the covariance between the color channels, and uses the Mahalanobis distance. In other words, it aims to have the boundary of two neighboring tiles with similar gradient distribution to the gradient on both sides of the boundary.
First, define an array of gradients G_{iL} with T rows (T is the pixel dimension of the tile) and with 3 columns (one for each color channel). G_{iL} represents the intensity changes along the right side of the tile t_i, which is on the left-hand side of the pair.

$$G_{iL}(k, c) = t_i(k, T, c) - t_i(k, T - 1, c) \tag{5.0.3}$$

Compute the mean distribution of those gradients on the right side of the tile t_i as

$$\mu_{iL}(c) = \frac{1}{T} \sum_{k=1}^{T} G_{iL}(k, c) \tag{5.0.4}$$

For each color channel, we have that μ_{iR} is the mean difference between the final two columns of t_i. The 3×3 covariance matrix S_{iL} is estimated from G_{iL} and it captures the relationship of the gradients near the edge of the tile between the color channels. Then, the horizontal dissimilarity between tiles t_i and t_j is estimated as

$$D_{LR}(t_i, t_j) = \sum_{k=1}^{T} (G_{ijLR}(k) - \mu_{iL}) S_{iL}^{-1} (G_{ijLR}(k) - \mu_{iL})^T \tag{5.0.5}$$

where $G_{ijLR}(p, c)$ is the gradient from the right side of the piece t_i to the left side of piece t_j, at row position k, and is defined as

$$G_{ijLR}(k, c) = t_j(k, 1, c) - t_i(k, T, c) \tag{5.0.6}$$

Since the junction between pieces t_i and t_j is evaluated based on the distributions estimated from the t_i side of the boundary, the dissimilarity $D_{LR}(t_i, t_j)$ is not symmetric. Thus, similarly, modifying equations from (5.0.3) to (5.0.6), we define $D_{RL}(t_j, t_i)$. In this way, we can compute the horizontal symmetric dissimilarity as

$$D_{H_{ij}} = D_{LR}(x_i, x_j) + D_{RL}(x_j, x_i) \tag{5.0.7}$$

Finally, we convert the just obtained dissimilarities into the horizontal compatibility between tiles t_i and t_j as follows

$$C_{H_{ij}} \propto exp\left(-\frac{D_{H_{ij}}}{K_{min_H}(i)}\right) \tag{5.0.8}$$

where $K_{min_H}(i)$ is the K-min value of the dissimilarity between all other pieces to piece i. As suggested by Gallegher [8] we set $K = 2$.
The equations from (5.0.3) to (5.0.8) are properly adapted to compute the vertical dissimilarity and the local vertical compatibility.

Whatever methods we use, once for each tile we have the horizontal and vertical compatibility, we set to zero the compatibility values of all non-best-buddy matches and to 1 the compatibility of any two best buddies. This strategy is called best buddies concept and was proposed by Pomeranz et al. [22].

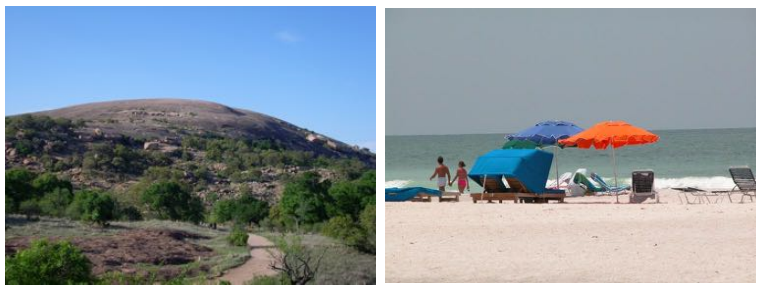

(a) (b)

Fig. 12: Example of images from MIT (a) and McGill (b).

6 Experimental results

For our experiments, we used two publicly available datasets, MIT and McGill. Both of them have 20 images, in MIT all the images have sizes 672x504, while McGill's images have sizes 756x560. In figure 12 are shown two images, one from MIT and one from Mcgill dataset. Both of them have some parts that are monochromatic or show some repetitive pattern. Tiles picked from such areas are considered constant tiles. As we can see, about 50% of these images are constituted by constant tiles. Most of the images in these two datasets are formed similarly to these two. The final goal of the whole model is, given an image split into several square pieces and shuffled with known orientation, being able to reassemble the image to its original structure.

6.1 Relaxation Labeling vs PSQP

As for now, we compare two different methods only on puzzle solving. To do this we test them on ideal compatibility which works as an oracle that always gives the right compatibility between two tiles, and is defined as follows:

$$C_{\mathcal{R}}^{(Oracle)}(i,j) = \begin{cases} 1, & \text{if i,j are the correct neighbors in relation } \mathcal{R} \\ 0, & \text{otherwise} \end{cases}$$

where i and j are two different tiles from the same image placed adjacent to each other. $\mathcal{R}$ is the spatial relationship, $\mathcal{R} \in \{left, up, right, down\}$. For

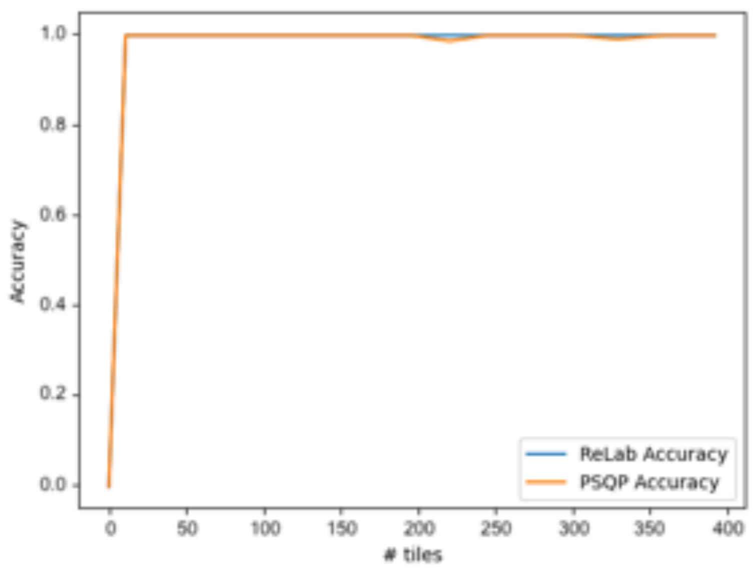


Fig. 13: Accuracy of Relaxation labeling and PSQP according to the number of tiles of the jigsaw puzzle.

example, if j is on the right of i, $C_{\mathcal{R}}^{(Oracle)}(i, j)$ will return 1.
Both algorithms receive as input a compatibility matrix derived from the oracle compatibility and they return a permutation of tiles. We compare such permutations with the correct rearrangement of the tiles to compute the accuracy. We start with a 2x2 puzzle and we increment by 1, one of the two dimensions of the puzzle.

In figure 13, we can see a comparison between the accuracy of the two algorithms on puzzles with a different number of tiles. The relaxation labeling-based solver being a probabilistic method may require more execution before returning its best result. On the other hand, PSQP based solver is a deterministic method and so given an input its result is always the same. However, both algorithms have very high accuracy, but it seems that PSQP always performs pretty well returning almost always a perfect reconstruction. While, with the increase of the size of the puzzles, Relaxation labeling has some oscillations in its performance.

So, we think that with the right measure of compatibility, we can expect that those two algorithms will give us quite good rearrangement results.

6.2 Performance assessment on compatibility learning

The accuracy of all the following models is measured through direct comparison, the horizontal and vertical compatibility are compared to their relative oracle compatibility. This metric computes the ratio between the number of tiles for which the most compatible piece is actually its immediately adjacent neighbor and the total of tiles. For each tile in the images in the dataset we compute the top 1,2,3,4 sets of highest values of compatibility, we checked if in these sets there are the tiles given in the oracle compatibility, then we compute the average among all images in the dataset. Accuracy values stated in all the tables in this section represent the average between horizontal and vertical compatibility.

6.2.1 Baseline approach

In the first attempt, we used the MIT dataset as the training set and the McGill dataset as the validation set. The method to select the tiles in the training phase is what we described in section 4.2, and training took 850 iterations. Hyper-parameters for this model are: learning rate $1e^{-4}$, weight decay $5e^{-4}$ and batch size 256. As we assumed that we already have an algorithm that can solve the puzzles with a good compatibility matrix, in the validation phase we only measure the accuracy of the extracted compatibility and we obtained the results listed in the table 1. Looking at figure 14, as the loss did not flatten, we find out that the model can learn even better the training set. Taking into account that we stopped the training earlier and so, the training loss isn't too close to zero, if we look at the table 1, we can see that the accuracy on the two datasets is pretty different. As expected, the accuracy on the MIT (training set) is more the twice the accuracy on the McGill (test set), hence, the model is not independent of the data of the training set.

ACCURACY	MIT	MCGILL
TOP 1	37.75%	18.75%
TOP 2	48.38%	21.62%
TOP 3	56.25%	25.94%
TOP 4	62.37%	28.70%

Table 1: Top K accuracy of compatibility on MIT (training set) and on McGill (validation set).

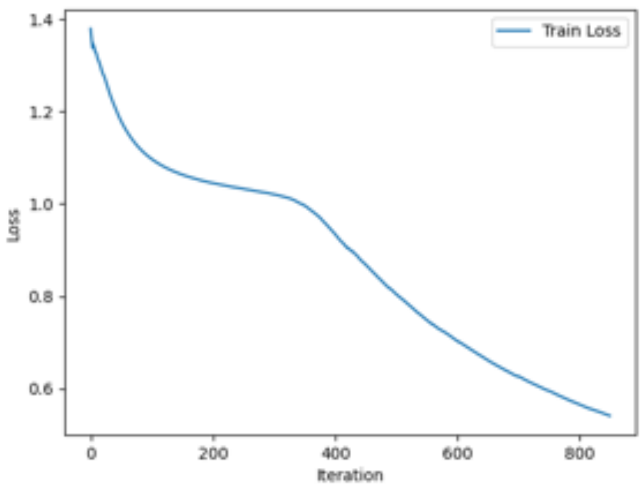


Fig. 14: Training loss over iteration for baseline approach.

6.2.2 Self-supervised learning

From here, we explore models trained with the *self-supervised* approach. For all the following neural networks, training has been done on the MIT dataset and validation on the McGill dataset and full tiles have sizes 28x28.

Estimating rotation. First of all, we have the CNN dedicated to establishing the rotation of tiles. We trained it for 900 iterations, with batch size 256, learning rate $1.4047 * 10^{-4}$, weight decay $5.9747 * 10^{-4}$ and momentum 0.95. For each tile in a batch, we randomly apply a rotation that can be of 0°, 90°, 180°, and 270° and train the model to detect which of these rotations

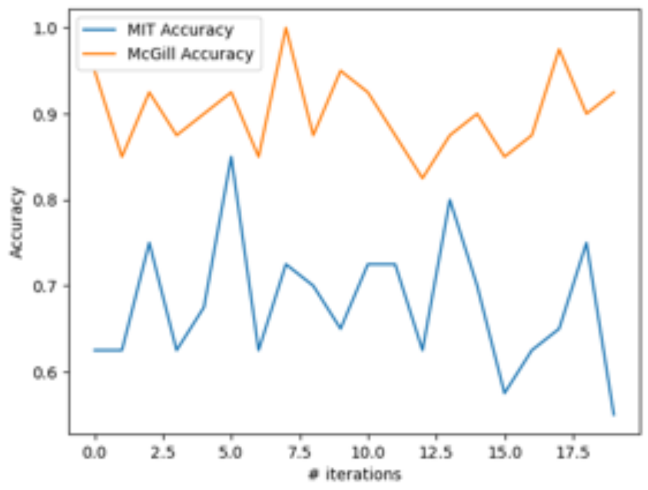
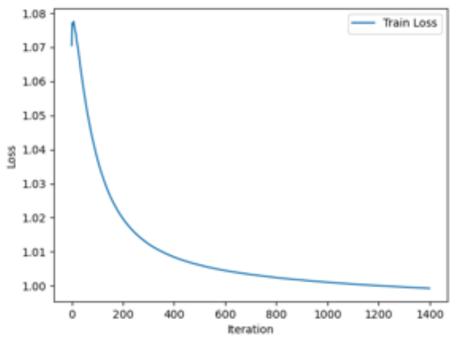


Fig. 15: Comparison of accuracy between MIT and McGill datasets on detecting rotation of tiles.

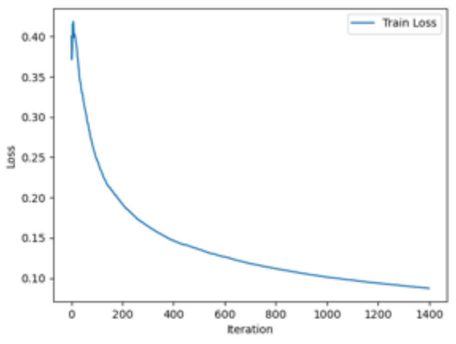
has been applied. We compare accuracy between the training set (MIT) and the validation set (McGill), for each image we compute the rotation of 20 tiles and we can see the results in figure 15. It is easy to see that MIT's accuracy is a bit higher than McGill's. Moreover, we noticed that on both datasets, the network has some drop in accuracy when trying to predict rotation for constant tiles. Since the difference in accuracy is not too pronounced and the behavior is quite similar on both sets, we believe that model can perform quite well on both datasets even in the task of puzzle solving.
Hence, we did a first trial of extracting compatibility. To do so, we removed the last fully connected layer of the CNN, which is specialized in detecting rotation. In this way, we take the output of the last convolutional layer and for each possible pair of tiles, we measure the correlation of these outputs.
Again, we check if the top 1,2,3,4 of horizontal compatibility and vertical compatibility of each tile are present in the actual tiles.
Training has been done on the detection of rotation, so for both datasets, we do not have any knowledge about their compatibility. Thus, we can consider extracting compatibility as an unsupervised task.

ACCURACY	MIT	MCGILL
TOP 1	27.50%	23.33%
TOP 2	36.00%	32.92%
TOP 3	43.87%	40.52%
TOP 4	50.25%	46.73%

Table 2: Top K compatibility accuracy of model trained on detecting rotation of tiles.



(a)



(b)

Fig. 16: Training losses for self-supervised model trained on full tiles (a) and on half tiles (b).

From the results shown in table 2, we can observe that the accuracy on MIT and on McGill are pretty close to each other, however, with pretty low accuracy values. This means that with the help of the pretext task we can achieve similar results for both datasets also on the main task of jigsaw puzzle solving.

Extracting compatibility with full tiles. From previous results, we see that we achieved some good performance on the pretext task, so now we move on to the actual task, learning compatibility. The first attempt at extracting compatibility is trained on the pre-trained net on detecting rotation with frozen convolution layers and on full tiles. Hyper-parameters are batch size 512, learning rate 0.0034271, momentum 0.85, weight decay 2.4587e-08. The new training is done on the MIT and it took 1400 iterations, so we expect a bit

ACCURACY	MIT	MCGILL
TOP 1	38.88 %	25.92 %
TOP 2	48.62 %	37.40 %
TOP 3	56.62 %	45.69 %
TOP 4	62.13 %	51.35 %

Table 3: Top K compatibility accuracy of re-trained model with frozen convolutional layers and on full tiles.

better performance on that dataset. Accuracy values shown in table 3 are a bit higher for the MIT dataset than what table 2 shows, but the accuracy on McGill of the two models is not too different. This could mean that the network is finally learning, however, it is not able to generalize on the McGill.

Extracting compatibility with half tiles. As we want our model to have good results on any datasets, we change a bit the training phase of extracting embedding. Instead of using two close full tiles, one for anchor and positive examples and the other for negative example, and giving different position values to the positive and negative, now we pick two tiles that are no more strictly close to each other, and we split the first one into two halves and one half becomes the anchor and the other the positive example. While the negative example is a random half of the other tile. We used the same hyper-parameters as for the model trained on full tiles. In figure 16 we see that both models trained on full tiles and half tiles, during training have the same behavior. However, if we compare table 3 and table 4 we can see that using half tiles for training to extract compatibility, accuracy on McGill is considerably increased and the model achieved similar accuracy on both datasets, meaning that the model is finally learning features agnostic from the training set.

ACCURACY	MIT	MCGILL
TOP 1	37.38 %	32.40 %
TOP 2	45.88 %	39.48 %
TOP 3	54.00 %	47.19 %
TOP 4	59.87 %	53.33 %

Table 4: Top K compatibility accuracy of re-trained model with frozen convolutional layers and on half tiles.

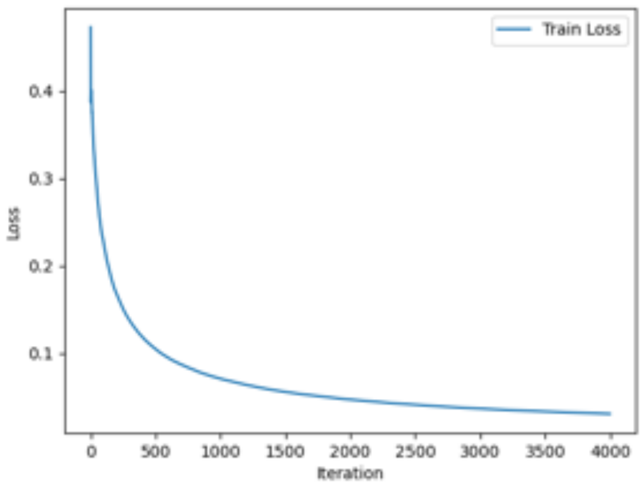


Fig. 17: Training loss over iteration for self-supervised approach trained with both convolutional and batch normalization layers frozen .

In table 5, we have the accuracy results of the model trained with both convolutional and batch normalization layers frozen. and with the number of iterations increased to 4000 (see figure 17). And so, we have another small increase in the accuracy of the network on the McGill (test set).

ACCURACY	MIT	MCGILL
TOP 1	37.98 %	33.85 %
TOP 2	47.25 %	42.92 %
TOP 3	54.75 %	49.92 %
TOP 4	60.25 %	55.52 %

Table 5: Top K compatibility accuracy of re-trained model on half tiles with frozen both convolutional and batch normalization layers.

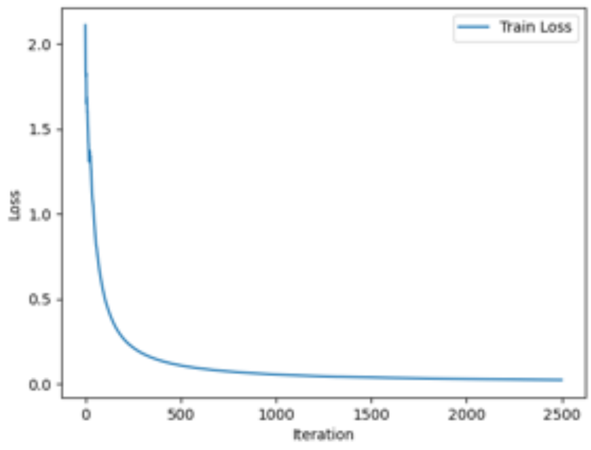


Fig. 18: Training loss of self-supervised model trained on MIT, with triplets derived from single tiles.

Extracting compatibility with triplets. Next, we have the results of our final models, which are trained on full tiles split in triplets and forwarded two times for different combinations of anchor, positive and negative examples. Training took 2500 iterations for each model with batch size 256, learning rate $3.4271 * 10^{-3}$, momentum 0.85, weight decay 2.45872e-08. Figure 18 show both models trained for horizontal and vertical compatibility behaved during training. In table 6 and 7 are reported only the accuracy on the McGill, we can easily notice that with this method of selecting examples we bring a huge improvement on our outcomes. We also noticed that with a strong presence of constant tiles, a tile may be misplaced but visually this does not bring any

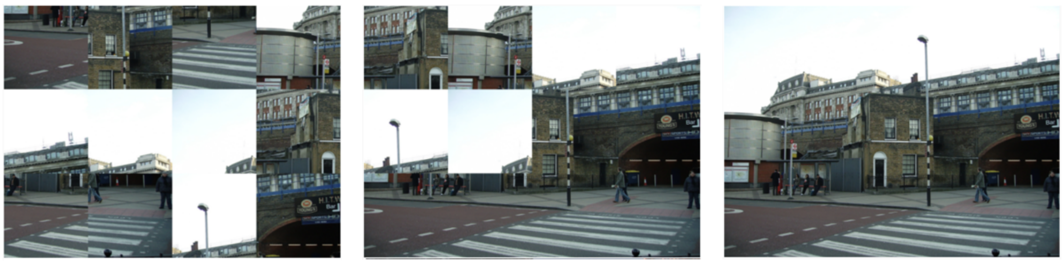
TOP K	With	Without
K=1	55.42 %	74.17 %
K=2	67.92 %	80.00 %
K=3	75.83 %	85.42 %

Table 6: Accuracy on compatibility with and without errors caused by constant tiles using the model trained with triplets from one tile.

Method	With	Without
ReLab	45.00 %	60.00 %
PSQP	49.17 %	67.92 %

Table 7: Accuracy on puzzle solving with and without errors caused by constant tiles on McGill dataset.

difference in the reassembling of the image. Hence why, in table 6 and 7, we have two columns, "With" are computed top 1,2,3 accuracy considering only the correct placement of the tiles. Instead, in "Without" column we do not consider as error if a constant tile is placed in the position of another with the same color information. Basically, even if a tile is in the wrong position, but visually looks the same as the corrected one, we actually do not consider it misplaced.
From table 6 for compatibility and table 7 for puzzle solving, it is easy to observe that without considering errors caused by constant tiles we have pretty good performances. The two puzzle solving algorithms perform almost identically, PSQP has slightly higher outcomes. Looking at figure 19, we can see an example of image reassembling. Figure 19 (a) shows an example of input for our neural network, all the tiles of the image have been shuffled. Figure 19(b) displays the final output, after CNNs and puzzle solver. While figure 19(c) represents the original image. In this example, we have a correct reconstruction of about 66%, which reflects the results of the previous tables.



(a) (b) (c)

Fig. 19: Example of image reassembling with tile of size 168x168, (a) starting point with all tiles shuffled, (b) the result after extracting compatibility and using Relaxation labeling. (c) original image, the goal.

Method	**With**	**Without**
Ours	55.42 %	74.17 %
Sholomon et al. [29]	80.56 %	82.94 %
Pomeranz et al. [22]	82.50 %	83.26 %
Gallagher [8]	90.21 %	91.38 %

Table 8: Accuracy on compatibility for different methods, with and without errors caused by constant tiles.

Table 8 shows a comparison between our proposed method and some of the best hand-crafted compatibility metrics. We can clearly see that considering errors derived from constant tiles, our method is quite far from the others. But in the case where we do not consider such errors, our solution reaches results pretty close to the other compatibility measures.

Therefore, in table 9, we can observe the comparison of the same methods, but of accuracy on puzzle solving with PSQP by direct comparison and the number of visually perfectly reconstructed images without considering errors derived from constant tiles. Our method is comparable to the solution proposed by Sholomon et al. [29] when talking about the number of visually perfect reconstructed images.
These last results show how much our residual neural network is affected by constant tiles.

Method	**D (%)**	**# of perfect**
Ours	67.92 %	7
Sholomon et al. [29]	86.19 %	7
Pomeranz et al. [22]	88.75 %	12
Gallagher [8]	92.42 %	13

Table 9: Accuracy on puzzle for compatibility metrics with PSQP by direct comparison (D) and visual perfect reconstruction, without errors caused by constant tiles.

In figure 20, we have a comparison between results obtained with our technique (left column) and the hand-crafted compatibility metrics proposed by Pomeranz et al. [22] (middle column) and by Gallagher [8] (right column). We note that our approach is still not close to the others but is neither too far.

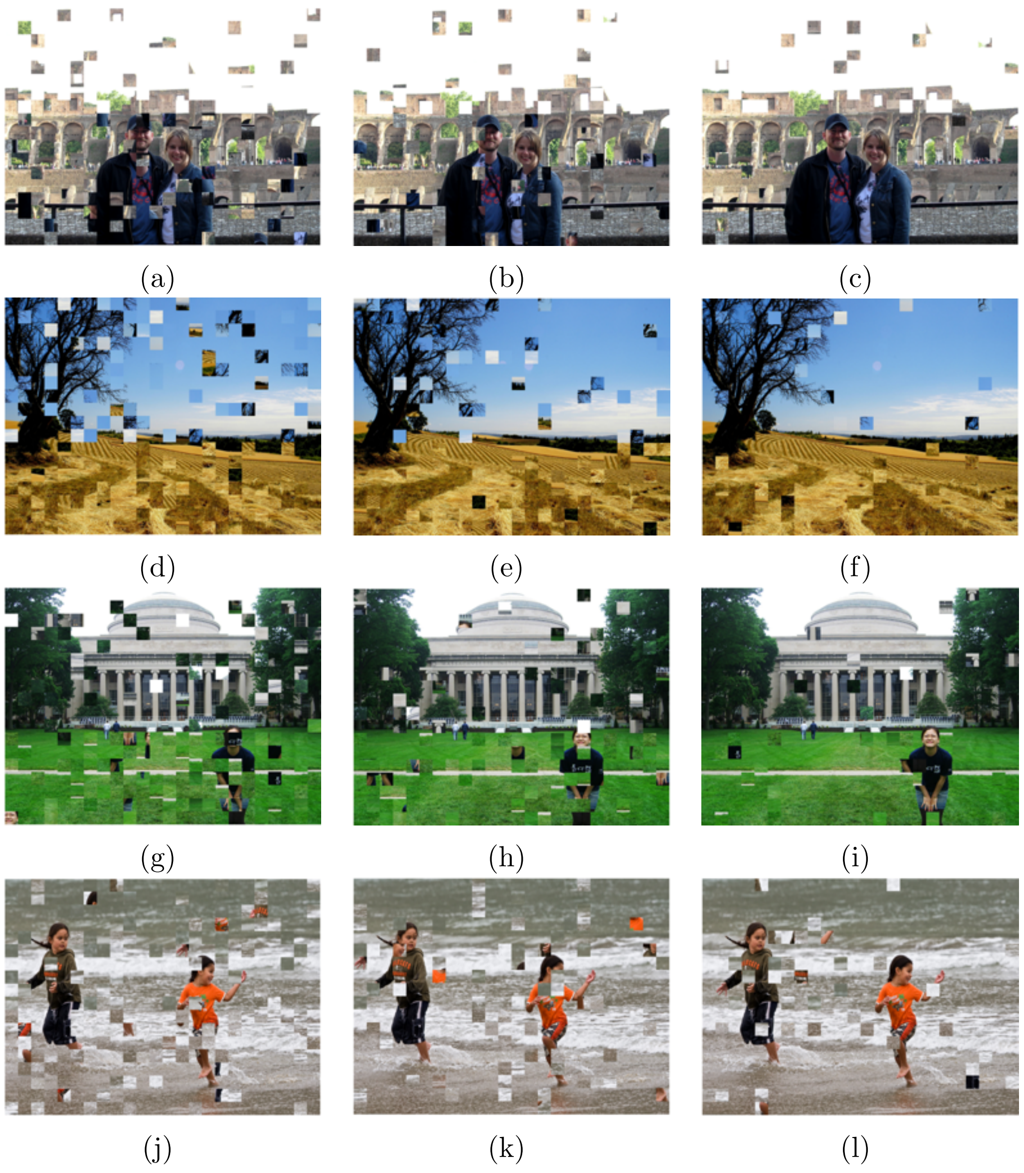


Fig. 20: Jigsaw puzzles with 432 tiles of size 28x28 pixels. In each row, there are our final permutation, the result obtained with Pomeranz et al.[22], the result obtained with Gallagher[8]. All three compatibility metrics are combined with PSQP.

7 Conclusions and Future works

This thesis deals with the problem of reassembling images from non-overlapping square tiles, to be placed in a rectangular grid of the same shape and size as the original image. We work with pieces that have linear boundaries and specific orientations. In literature, there are already several works about puzzle solving. But almost all of them propose either direct puzzle solving through the search for the best permutation or hand-crafted compatibility measure to be paired with some advanced solver. Just a few of them address the problem of puzzle solving as learning compatibility in a supervised manner. At the end of our work, we were able to propose a neural network, that working in a self-supervised environment, is capable of extracting compatibility from an image with good results though they are still not comparable with the best hand-crafted compatibility measures.

First, we compared two solvers for image reassembling, relaxation labeling, and PSQP, finding out that with a good compatibility matrix we can perfectly reconstruct any puzzle with 400 pieces and above. In search of a neural network capable of automatically devising a compatibility metric, we started implementing a baseline approach which led to poor generalization. Thus, we needed to search for a method to either extract agnostic features from the training set and not adapt to its distribution. The method proposed by Noroozi and Favaro [19] seems to have some good performance but the choice of the pretext task is probably crucial in the knowledge transfer step. Exploring the self-supervised domain, we elaborated different solutions to cope with the issue of making the neural network understand that some portion of a tile is more relevant than others, for example, the right part of the tile is more relevant when dealing with horizontal compatibility, similarly for vertical compatibility. After all experiments, we devised a model, which is trained with triplets derived from a single tile, and that emphasizes all the important aspects. Currently, our solution is not completely at the same level as the best hand-crafted compatibility metrics but, anyway, it achieves fairly good results. The biggest problem is how to deal with constant tiles, we have seen that if we do not consider errors caused by constant tiles we have pretty high outcomes.

In the literature, all the already existing solutions based on neural networks either aim to directly find the best permutation on small puzzles or use supervised learning. Furthermore, to the best of our knowledge, the presented model is the first unsupervised approach to automatically extract the compatibility of tiles for jigsaw puzzle solving.

Future works. As we have shown, by being able to fully address the handling of constant tile, it will be possible to have an unsupervised neural network-based compatibility metric for the jigsaw puzzle solving that can overcome all the hand-crafted metrics already present in literature. Thus, some improvements can be

- training the deep neural network to be agnostic from constant tiles;
- handle tiles with any rotation;
- upgrade the solvers used for tiles reordering to some that do not require post-processing.

References

[1] FA. Andalo, G. Taubin, S. Goldenstein, PSQP: Puzzle Solving by Quadratic Programming. IEEE Trans Pattern Anal Mach Intell. (2016)

[2] B. Brown, C. Toler-Franklin, D. Nehab, M. Burns, D. Dobkin, A. Vlachopoulos, C. Doumas, S. Rusinkiewicz, and T. Weyrich, A system for high-volume acquisition and matching of fresco fragments: Reassembling Theran wall paintings, in ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008, vol. 27, no. 3, pp. 84:1– 84:9. (2008)

[3] G. Burdea and H. Wolfson, Solving jigsaw puzzles by a robot, IEEE Transactions on Robotics and Automation, vol. 5, no. 6, pp. 752–764. (1989)

[4] T. Cho, S. Avidan, and W. Freeman, The patch transform, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 8, pp. 1489–1501. (2010)

[5] T. Cho, S. Avidan, and W. Freeman, A probabilistic image jigsaw puzzle solver, in Conference on Computer Vision and Pattern Recognition (CVPR), pp. 183–190. (2010)

[6] E. Demaine and M. Demaine, Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity, Graphs and Combinatorics, vol. 23, pp. 195–208, (2007)

[7] H. Freeman and L. Gardner., Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. IEEE. Trans. on Electronic Computers, (1964)

[8] A. Gallagher, Jigsaw Puzzles with Pieces of Unknown Orientation, Proc. CVPR. (2012)

[9] D. Goldberg, C. Malon, and M. Bern, A global approach to automatic solution of jigsaw puzzles, in 18th Annual ACM Symposium on Com- putational Geometry (SoCG), pp. 82–87. (2002)

[10] Marie-Morgane Paumard, David Picard, Hedi Tabia. Image Reassembly Combining Deep Learning and Shortest Path Problem. European Conference on Computer Vision (ECCV 2018), Sep 2018, Munich, Germany. hal-01869765v2

[11] He Kaiming, Zhang Xiangyu, Ren Shaoqing and Sun Jian, Deep Residual Learning for Image Recognition. (2015)

[12] R.A.Hummel, S.W. Zucker, On the foundations of relaxation labeling processes.IEEE TPAMI 5(3), 267–287 (1983)

[13] E. Justino, L. Oliveira, and C. Freitas, Reconstructing shredded documents through feature matching, Forensic Science International, vol. 160, no. 2, pp. 140–147. (2006)

[14] M. Khoroshiltseva, B. Vardi, A. Torcinovich, A. Traviglia, O. Ben-Shahar, and M. Pelillo, Jigsaw Puzzle Solving as a Consistent Labeling Problem. (2021)

[15] D. A. Kosiba, P. M. Devaux, S. Balasubramanian, T. L. Gandhi, and K. Kasturi. An automatic jigsaw puzzle solver. In Proc. ICPR. (1994)

[16] W. Marande and G. Burger, Mitochondrial dna as a genomic jigsaw puzzle, Science, vol. 318, no. 5849, p. 415. (2007)

[17] J. McBride and B. Kimia, Archaeological fragment reconstruction using curve-matching, in Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). (2003)

[18] T. R. Nielsen, P. Drewsen, and K. Hansen. Solving jigsaw puzzles using image features. Pattern Recogn. Lett., 29. (2008)

[19] M. Noroozi and P. Favaro, Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles. (2017)

[20] Paumard, M., Picard, D., Tabia, H., Deepzzle: Solving visual jigsaw puzzles with deep learning and shortest path optimization. CoRR abs/2005.12548 (2020)

[21] M. Pelillo, The dynamics of nonlinear relaxation labeling processes. J.Math.Imag.Vis. 7(4), 309–323 (1997)

[22] D. Pomeranz, M. Shemesh, and O. Ben-Shahar, A fully automated greedy square jigsaw puzzle solver, in Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9–16. (2011)

[23] Rao, S. Singiresu, Engineering optimization: theory and practice, John Wiley & Sons. (2019)

[24] J.B. Rosen, The Gradient Projection Method for Nonlinear Programming, Journal of the Society for Industrial and Applied Mathematics , Mar., 1960, Vol. 8, No. 1 , pp. 181-217, (Mar., 1960)

[25] A. Rosenfeld, R.A.Hummel, S.W. Zucker, Scene labeling by relaxation operations. IEEE Trans. Syst. Man & Cybern. 6, 420–433 (1976)

[26] M. Sagiroglu and A. Ercil. A texture based matching approach for automated assembly of puzzles. In Proc. ICPR. (2006)

[27] J. Schwartz and M. Sharir, Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves, The International Journal of Robotics Research, vol. 6, no. 2, pp. 29–44. (1987)

[28] E. Seneta, Non-negative matrices and Markov chains. Springer Verlag, (2006)

[29] D. Sholomon, O. David, and N. Netanyahu, A genetic algorithm-based solver for very large jigsaw puzzles, in Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1767–1774. (2013)

[30] D. Sholomon, Dror and David, E. Omid and Netanyahu,S. Nathan, DNN-Buddies: A Deep Neural Network-Based Estimation Metric for the Jigsaw Puzzle Problem, Artificial Neural Networks and Machine Learning – ICANN 2016, p.170-178 (2016)

[31] K. Son, J. Hays, D.B. Cooper, Solving square jigsaw puzzle by hierarchical loop constraints. IEEE TPAMI 41(9), 2222–2235 (2018)

[32] F.-H. Yao and G.-F. Shao, A shape and image merging technique to solve jigsaw puzzles. Pattern Recognition Letters. (2003)

[33] Y. Zhao, M. Su, Z. Chou, and J. Lee, A puzzle solver and its application in speech descrambling, in International Conference on Computer Engineering and Applications (CEA), pp. 171–176. (2007)

[34] L. Zhu, Z. Zhou, and D. Hu, Globally consistent reconstruction of ripped-up documents, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 1, pp. 1–13. (2008)

[35] Line search by SciPy: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.line_search.html`

[36] John Spilsbury: `https://en.wikipedia.org/wiki/John_Spilsbury_(cartographer)`

Acknowledgements

First of all, i have to thank to my thesis supervisor Marcello Pelillo and thank you for being a source of inspiration.
I also want to thank Alessandro Torcinovich very much for having followed me for the entire duration of this project and I am grateful for his patience not only in correcting but also in explaining in depth why of my mistakes.
Last, a big thanks to my course mates, my friends and my family which supported me during the academic journey.