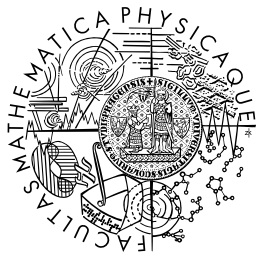


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Marcela Brichtová Piptová

Artwork Reassembly - Neural Networks Approach

Department of Software and Computer Science Education

Supervisor of the master thesis:	doc. RNDr. Barbara Zitová, Ph.D.
Study programme:	Computer Science - Visual Computing and Game Development
Study branch:	IVVP

Prague 2025

I declare that I carried out this master thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor, doc. RNDr. Barbara Zitová, Ph.D., and my consultant, Mgr. Tomáš Karella, for their valuable guidance, support and constructive feedback. I am also grateful to my husband, Ondřej Brichta, for his continuous support throughout the work on this thesis.

Title: Artwork Reassembly - Neural Networks Approach

Author: Marcela Brichtová Piptová

Department: Department of Software and Computer Science Education

Supervisor: doc. RNDr. Barbara Zitová, Ph.D., Department of Image Processing, UTIA CAS

Abstract: This thesis presents a deep learning-based approach to fragment reconstruction, with a focus on artwork restoration. Reconstruction is often necessary in fields like art conservation and archaeology, where reassembling fragmented objects aids in research and preservation. To develop and evaluate our method, we created a synthetic dataset that simulates real-world fragmentation using publicly available artwork. The dataset captures irregular shapes, complex contours, and varying fragment sizes typical of practical reconstruction scenarios.

Our method uses a 1D convolutional neural network with a U-Net-inspired architecture to identify contour-based correspondences between fragments. These local matches are then integrated through a global assembly algorithm, which incrementally merges clusters of pieces based on match confidence while addressing potential inconsistencies. The proposed approach demonstrates strong performance on both synthetic benchmarks and real-world examples, highlighting its potential across a range of reconstruction tasks.

Keywords: artwork restoration, neural network, jigsaw assembly

Contents

Introduction

This thesis addresses the problem of assembling fragmented images, particularly degraded pieces of artwork that have been broken into parts. This task can be viewed as a generalization of the traditional jigsaw puzzle, a well-known game in which an image is divided into many pieces, and the objective is to reconstruct the original image by correctly assembling them.

The problem of image reassembly exists in various forms. In this thesis, we focus exclusively on two-dimensional cases and do not address three-dimensional reconstruction tasks. The image can be divided into rectangular or irregularly shaped pieces. In typical jigsaw puzzles, the pieces are arranged in a regular grid and feature distinctive interlocking tabs and slots along their edges, which help secure the pieces during assembly. In contrast, this thesis focuses on the most general form of the fragmented image assembly problem, where the pieces can vary in size and shape and may be deformed or missing.

Although jigsaw puzzles are primarily associated with recreation, the ability to reconstruct fragmented objects has practical importance in several domains. In archaeology and art restoration, for example, there is a frequent need to reconstruct broken artifacts such as pottery, frescoes, or historical artworks. These tasks are complicated by the fact that fragments are often damaged or incomplete. Computer-aided puzzle reconstruction not only reduces the time and effort required but also makes it possible to restore objects that would be too fragile or complex to handle manually.

Similar challenges also arise in forensic science. For instance, in 2011, DARPA organized a competition [1] focused on reconstructing shredded documents, illustrating the real-world relevance of this problem.

The motivation for this work arose from a collaboration with the National Institute of Optics of the Italian National Research Council (CNR-INO) in Florence, where art restorers sought to reassemble a gelatin dry-plate photographic negative from the archives of the Commissione per l'Edizione Nazionale dei Manoscritti e dei disegni di Leonardo da Vinci (Commissione Vinciana). This negative captures Cesare da Sesto's painting Madonna with Saint John (1477-1523), a work now held by the Museum Bonnat-Helleu in Bayonne (France) after

Figure 1 Gelatin dry-plate photographic negative of Cesare da Sesto's *Madonna with Saint John*, fragmented into 14 irregular pieces.

having been briefly in the Louvre's collection. The plate was fragmented into 14 irregularly shaped pieces—some deformed, others missing altogether—making manual reconstruction both tedious and highly error-prone (Figure 1).

The primary objective of this work is to develop a machine-learning-based algorithm capable of solving fragmented image reassembly problems, such as the reconstruction of the aforementioned negative. Accordingly, we address the general case of jigsaw puzzles without constraints on piece shape. We also account for the possibility of missing or damaged fragments. The only exception is puzzles containing pieces with holes, which are rare in real-world scenarios and not considered in this study. Nevertheless, such pieces can be easily converted into standard fragments by dividing them into smaller parts.

To support the development and evaluation of our method, we also designed an algorithm for synthetic puzzle generation. This generator simulates real-world fragmentation patterns and is used to create a dataset of synthetic puzzle problems. The source images for these puzzles are real artworks, aligning well with our focus on art restoration.

Approach

Computer-aided methods for solving jigsaw puzzles are typically divided into two main stages. The first stage, referred to as *local assembly*, involves identifying

matches between pairs of pieces. The second stage, known as *global assembly*, combines all the pieces to reconstruct the complete image. This thesis emphasizes a machine-learning approach to local assembly while also addressing aspects of global assembly.

Local assembly can be viewed as a specialized form of image registration, except that the pieces do not overlap. Consequently, the only information that can be used to align them is along their borders. The goal is to identify neighboring pairs of pieces and to determine the transformations (rotation and translation) required to align them.

Local assembly methods can be categorized as shape-based (apictorial), content-based, or hybrid approaches. Traditional methods often rely on contour matching to find potential matches, while modern approaches incorporate machine learning. In this thesis, we propose a method using a convolutional neural network to predict the similarity between the contours of two pieces. This predicted similarity is then used to identify matching pairs.

For global assembly, a straightforward approach involves using a greedy algorithm, iteratively combining matched pairs until the full image is reconstructed. However, this approach becomes impractical when the number of false-positive matches is high. Alternative methods employ advanced heuristics or treat the problem as a graph optimization task. Our solution adopts a greedy approach with several enhancements to improve its robustness and performance.

Structure

The thesis is structured as follows:

Chapter 1 clearly defines the goals of the research. In Chapter 2, we review the existing literature on jigsaw puzzle solving, exploring both local and global assembly methods. We examine traditional image-processing techniques and modern deep learning approaches.

Chapter 3 introduces the datasets used for evaluation. We review existing datasets and also describe the creation of our synthetic dataset, designed to simulate real-world fragmented artworks.

The local assembly method is detailed in Chapter 4. This chapter focuses on the neural network that we trained to find pairwise correspondences between image fragments. We explain the architecture, training procedure, and evaluation metrics, and provide the evaluation results of our method.

Chapter 5 focuses on the global assembly phase, describing how the identified pair matches are used to reconstruct the complete image.

Chapter 6 offers a comparative analysis of our approach against the existing PairingNet method [2]. This chapter highlights methodological differences and

compares the evaluation results.

In Chapter 7, we present the results of the evaluation of our method, focusing on both quantitative metrics and qualitative examples. The evaluation includes both synthetic data and real-world examples, such as the dry-plate negative shown in Figure 1.

Finally, Chapter 8 summarizes the key findings of the thesis and suggests future research directions in the field of jigsaw puzzle reconstruction.

Chapter 1

Goals

The main objective of this thesis is to develop a machine-learning-based approach for solving jigsaw puzzle problems, with a particular emphasis on reconstructing fragmented artworks. Our focus is on irregularly shaped fragments of varying sizes, which may also be damaged, as seen in real-world applications such as art restoration and archaeology. A key example is the fragmented dry-plate negative shown in Figure 1. The goal is to design a solution capable of reassembling general fragmented images without making assumptions about the shape of the fragments, with the sole exception being that the fragments do not contain holes meant to be filled by other fragments.

To achieve this, we define the following key objectives:

- **Dataset Creation**: Generate synthetic datasets that closely resemble real-world fragmentation patterns, to be used for method development and evaluation.
- **Local Assembly**: Design a machine-learning-based algorithm to accurately determine whether two pieces belong together and develop a confidence scoring mechanism to rank potential matches.
- **Global Assembly**: Investigate techniques for merging locally matched pairs to reconstruct the complete image.
- **Evaluation**: Define suitable evaluation metrics to assess reconstruction accuracy. Evaluate the algorithm both on synthetic datasets and real-world cases, including the fragmented dry-plate negative.

By addressing these goals, this thesis aims to advance the field of automated jigsaw puzzle reconstruction, contributing to applications in cultural heritage preservation, forensic analysis, and other domains requiring the reassembly of fragmented materials.

Chapter 2

Related Work

In this chapter, we provide an overview of existing methods for solving jigsaw puzzles, highlighting their strengths and limitations.

2.1 Local Assembly

In local assembly, the goal is to find the matching pairs of puzzle pieces. This task can be solved using traditional image processing methods. In recent years, deep learning approaches have also been explored.

2.1.1 Traditional Approach

Traditional methods for solving the local assembly of jigsaw puzzles fall into three primary categories: shape-based, content-based, and hybrid approaches.

Shape-based approaches focus on analyzing the geometric properties of the contours of puzzle pieces. Curve matching techniques, such as the turning function [3] or the Smith-Waterman algorithm [4, 5, 6], have been widely employed. [7] utilized dynamic programming to identify the longest common subsequence of contour features. Most shape-based methods also incorporate polygonal approximations to simplify the representation of piece contours.

Content-based methods leverage image information from puzzle pieces. Typically, they calculate a measure of dissimilarity to identify neighboring pieces by minimizing the difference in color along the shared edges [8]. [9] proposed an automatic reassembly method using color histograms to match pieces of fragmented paintings and images.

Hybrid approaches combine geometric and content-based information to improve the precision of reconstruction [10].

2.1.2 Deep Learning Approach

Despite the extensive exploration of traditional methods, deep learning-based approaches to jigsaw puzzle assembly are comparatively limited. Furthermore, many of these methods are focused on specific puzzle types and thus not applicable to the general jigsaw puzzle problem.

Puzzles with Rectangular Pieces

Research on square puzzle pieces, often organized in 3×3 grids, has been explored in several studies [11, 12, 13, 14, 15, 16, 17, 18, 19]. These problems are inherently simpler due to the limited configurations of neighboring pieces. For example, a neighbor can occupy one of four positions (top, bottom, left, or right), and each position supports only four possible rotations, leading to 16 possible configurations per pair. In contrast, general puzzle problems involve significantly more potential configurations, complicating the matching process.

The authors of [16] and [11] trained convolutional neural networks (CNNs) using triplet loss to classify whether two tiles should be adjacent. Their methods included augmentations such as random shifting and masking of border pixels. Studies such as [12] and [13] focused on predicting positional relationships between tiles in 3×3 grids using CNNs. Augmentations in these approaches included the erosion and substitution of pieces that did not belong to the original image. Generative models have also been explored in this context. For example, [14] employed a generative adversarial network (GAN) to solve 3×3 puzzles. In [18], a Monte Carlo tree search algorithm was used for image reassembly, although this was also limited to the 3×3 grids. Additionally, [19] proposed the use of a Siamese neural network with augmentations like erosion, again focusing on square pieces. Reconstruction of shredded documents was addressed by [20], using strong assumptions about piece shapes and employing self-supervised learning and simulated shredding techniques.

Puzzles with General-Shaped Pieces

Some methods handling general-shape pieces use traditional methods for local assembly, using the neural network only as a classifier of correct matches, which reduces false positives and improves the performance of global assembly. The first approaches used CNN as the backbone architecture for this classification [21], followed by [22] where vision transformers were utilized. The authors of this study also experimented with other backbone architectures, publishing comprehensive comparisons.

A unique approach was presented by [23], who used diffusion models to directly solve the puzzle, bypassing the local assembly step. Their model was

primarily designed for apictorial layout reconstruction and was tested on datasets of room layouts and simple polygonal puzzles. Although effective for puzzles with up to 20 pieces, the method's high computational requirements and reliance on transformers make it unsuitable for larger and more complex cases.

The most relevant work for our study is PairingNet [2], which introduced a method to match contour points in pair of images. For each contour point, they extracted two patches: one capturing image texture and the other that captures the contour details. These patches were processed through a graph convolutional network (GCN) to determine the matching contour points. Texture and contour information were adaptively combined, allowing the model to prioritize the most informative features. Unlike [21] and [22], which relied on manually designed contour features, this method autonomously handles the contour matching task. This is the approach upon which we build, making further improvements to enhance the results.

2.2 Global Assembly

The global assembly task involves selecting the best set of matches from the local assembly step to achieve a fully reconstructed image. Global assembly methods typically employ greedy strategies, often enhanced by various optimizations. In these approaches, a scoring function is used to rank potential matches, and the matches are applied iteratively in the order of highest to lowest score, gradually assembling the complete image [5]. To mitigate errors in the assembly process, backtracking algorithms are frequently employed, allowing the system to explore alternative configurations when facing ambiguities or misplacements. However, this approach often leads to exponential computational complexity as the number of possible piece arrangements grows, significantly impacting scalability for larger puzzles.

Graph-based methods are another common approach to global assembly. In these methods, puzzle pieces are represented as nodes, with potential matches forming weighted edges between them. For example, in [7], a maximum spanning tree algorithm was used to identify the optimal set of matches. Other strategies involve the use of genetic algorithms to optimize the assembly process, as demonstrated by [8] and [24].

Unlike local assembly, the use of deep learning for global assembly is relatively uncommon. One potential approach was already mentioned in the previous section [23], where a diffusion model was used for the entire end-to-end assembly process.

Chapter 3

Data

In this chapter, we present the datasets used for the image reassembly problem. Our primary goal is to solve real-world problems, such as artwork reconstruction. To be able to design and evaluate jigsaw puzzle reconstruction methods, we require datasets that closely mimic real-world fragmentation patterns, capturing the authentic complexity and variability of piece shapes encountered in practical applications. Despite significant prior work in this domain, there remains a scarcity of publicly available datasets that sufficiently meet our requirements. These existing datasets are discussed in Section 3.2. As none of the available datasets fully satisfied our needs, we created and published a synthetic dataset along with the code used for its generation[1].

3.1 Dataset Requirements

To effectively simulate real-world problems such as a fragmented negative shown in Figure 1 and ensure the generalizability of our approach, the dataset must meet the following criteria:

1. **Irregular Structure:** Pieces should not be organized in a regular grid, and the number of their neighbors varies.
2. **Variability in Shape and Size**: Pieces can vary in size and shape, both within a single image and across images.
3. **Complex Borders:** Borders between pieces should range from straight lines to highly intricate curves.
4. **Diverse Image Types:** Images should include various formats such as photographs, illustrations, and other artwork.

[1] `https://github.com/mPiptova/cnn-piece-assemble`

3.2 Existing Datasets

Several datasets have been explored in the context of jigsaw puzzle research, but most do not meet the requirements outlined in Section 3.1. The following section evaluates these datasets against our established criteria.

3.2.1 JigsawNet Dataset

Authors of [21] introduced a synthetic dataset for their JigsawNet model.[2]. Figure 3.1 shows some examples from this dataset. Only the test set is available and it consists of:

- 20 puzzles with 9 pieces each.
- 6 puzzles with 36 pieces each.
- 6 puzzles with 100 pieces each.
- 5 larger puzzles with approximately 400 pieces each.

While the dataset provides puzzles with curved borders, all pieces are of similar size and they are organized in a regular grid. Furthermore, only photographs were used as image sources. Therefore, this dataset fails to meet all of our specified requirements. However, it is still suitable for evaluation of our approach, even though we cannot directly compare our results with theirs, since their objective is to classify piece matches (correct or incorrect) rather than finding them.

3.2.2 PairingNet Dataset

The PairingNet dataset, along with its generation code,[3] was introduced by [2]. It consists of:

- 8196 synthetic fragments from 390 images.
- 320 real fragments from 34 printed images.

This dataset aligns more closely with our use case but is limited to photographs, restricting the diversity of image types. However, it has several limitations. The image quality is relatively low, primarily due to the use of the nearest-neighbor interpolation method during random piece rotations. Second, it does not contain complete images; instead, only a subset of pieces from each image is included,

[2] `https://github.com/Lecanyu/JigsawNet`
[3] `https://github.com/zhourixin/PairingNet`

(a) 9 pieces. **(b)** 36 pieces.

(c) 100 pieces. **(d)** 400 pieces.

Figure 3.1 JigsawNet dataset examples [21].

which makes it impossible to fully reconstruct the original image (see Figure 3.2). This design choice is based on the primary objective of [2] - not to reassemble the original images but to identify neighboring pieces among fragments from multiple images. As a result, this dataset is only suitable for evaluating local assembly methods, which is detailed in Chapter 6.

3.2.3 Shredded Document Dataset

Another dataset, created by [25][4], focuses on shredded document reassembly. It consists of 60 documents that have been synthetically divided into varying numbers of stripes to simulate shredding, along with three real shredded documents. One example is shown in Figure 3.3. Since the fragments form regular rectangular stripes and the source images are limited to documents, this dataset is not aligned with our use case and is not suitable for our experiments.

[4] `https://github.com/xmlyqing00/DocReassembly`

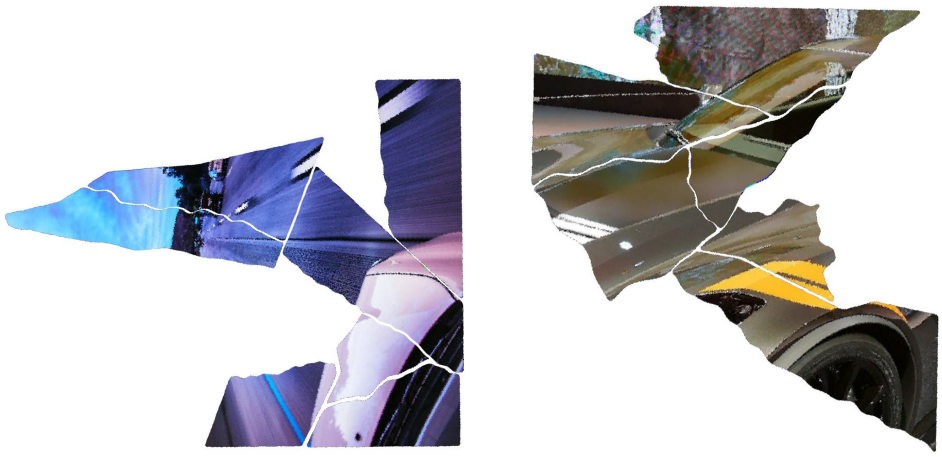


Figure 3.2 PairingNet dataset examples [2].

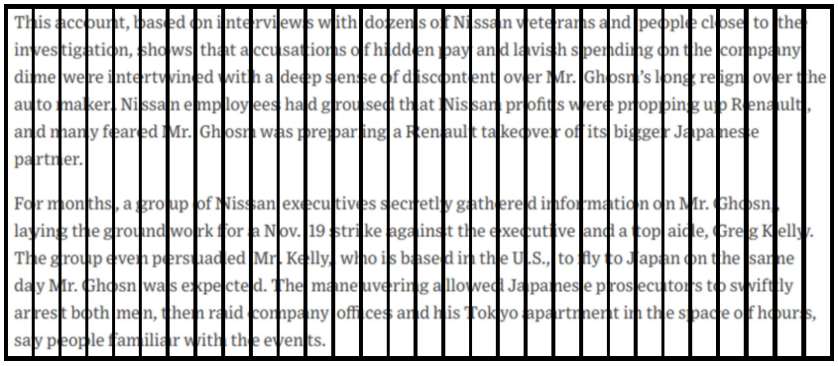


Figure 3.3 Example from the dataset of shredded documents [25].

3.3 Proposed Synthetic Dataset

Since existing datasets do not align adequately with this use case, we generated our own synthetic dataset using publicly available images, for which we designed and implemented a specialized splitting method to segment these images into individual pieces with randomly generated shapes that satisfy our specific requirements. The dataset includes diverse content such as drawings, paintings, and photographs sourced from the National Gallery of Art[5], which aligns well with our objective of artwork reconstruction. Examples are shown in Figure 3.4.

The dataset is divided into training, validation, and test subsets. Images were split into pieces of varying counts, as detailed in Table 3.1. For training and

[5] `https://www.nga.gov/open-access-images.html`

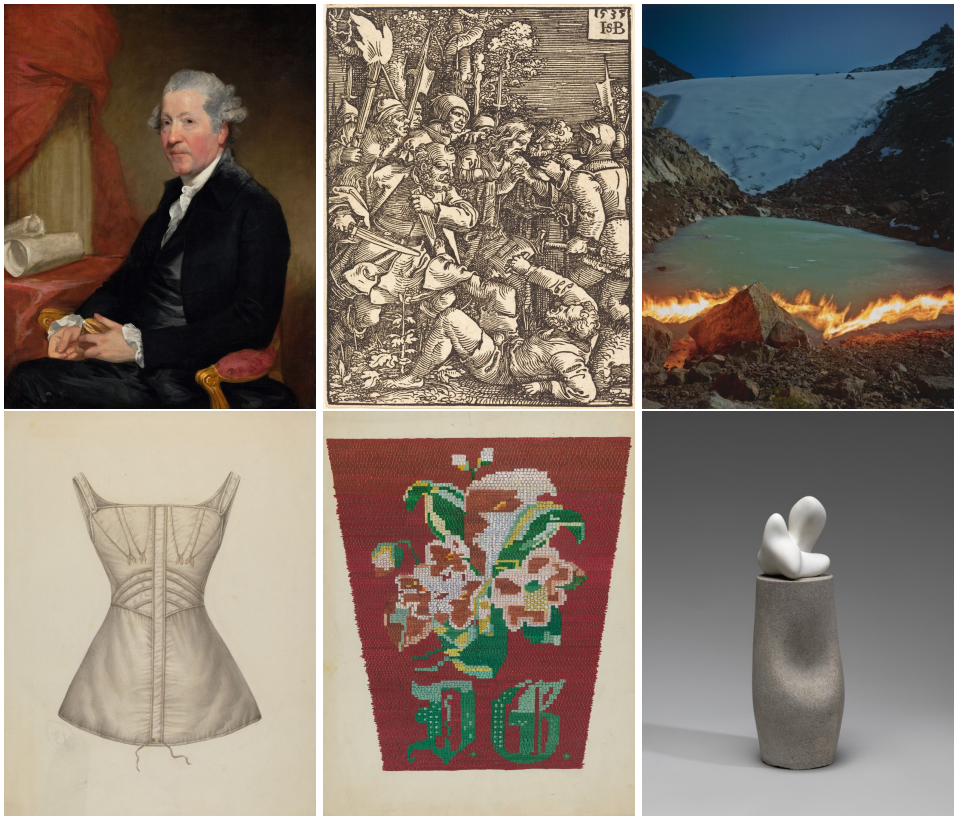


Figure 3.4 Examples of images from our dataset.

validation, piece counts include 50, 100, 200, and 400. For testing, we divided the images into 10, 30, 50, or 100 pieces.

Subset	Train	Validation	Test	Total
10	0	0	30	30
30	0	0	10	10
50	377	67	10	454
100	10	3	5	18
200	5	1	0	6
400	3	1	0	4
Total	395	72	55	522

Table 3.1 Piece counts for each subset of our dataset.

3.3.1 Splitting Algorithm for Synthetic Dataset Generation

Our splitting algorithm ensures that the dataset meets the requirements outlined in Section 3.1 and aims to generate fragmented images as close to real-world examples as possible. The process involves:

1. **Random Curve Generation** (Figure 3.5a)
    - Straight lines are drawn across the image.
    - Points are regularly sampled along these lines, and their position is perturbed using Gaussian noise.
    - The generated points are connected using a cubic spline interpolation, resulting in a smooth curve that passes through all the points. The spline constructs a piecewise cubic function with continuous first and second derivatives, ensuring smooth transitions at the joints.
2. **Region Merging** (Figures 3.5b and 3.5c)
    - The curves generated in step 1 are used to segment the image into regions.
    - The regions are merged iteratively to achieve the desired number of pieces and to eliminate the ones that are too small.
        - In each iteration, the region with the smallest area is selected.
        - The selected region is merged with its neighbor with the largest common border.

(a) Random curves. **(b)** Regions before merging. **(c)** Regions after merging.

Figure 3.5 Image division process used in our splitting algorithm. First, we generate random curves (a), then we use them to divide the image into regions (b), and finally merge regions to obtain the desired number of pieces and eliminate regions that are too small (c).

3. **Piece Extraction**

   - Individual piece images are extracted from the generated regions, and each piece is randomly rotated.
   - The transformation needed to assemble the piece back together is computed.
   - Optionally, each piece is augmented as described in Section 3.3.2.

The configurable parameters of the algorithm include the desired number of pieces, the number of curves, the sampling density of points, and the noise intensity. Examples of generated images are shown in Figure 3.6.

3.3.2 Augmentations

Our research aims to simulate real conditions encountered in actual reconstruction scenarios. When objects fragment in the physical world, the resulting pieces undergo environmental interactions and alterations that can significantly impact their characteristics and spatial relationships. This can lead to eroded boundaries and changes in the color of some fragments. Even in conventional jigsaw puzzles, small gaps often appear between pieces due to imperfections during the cutting process. To mimic these distortions and enhance the robustness of our models, we experimented with two types of augmentations.

- **Erosion:** Simulates physical boundary erosion by removing pixels along piece edges (Figures 3.7 and 3.8b).

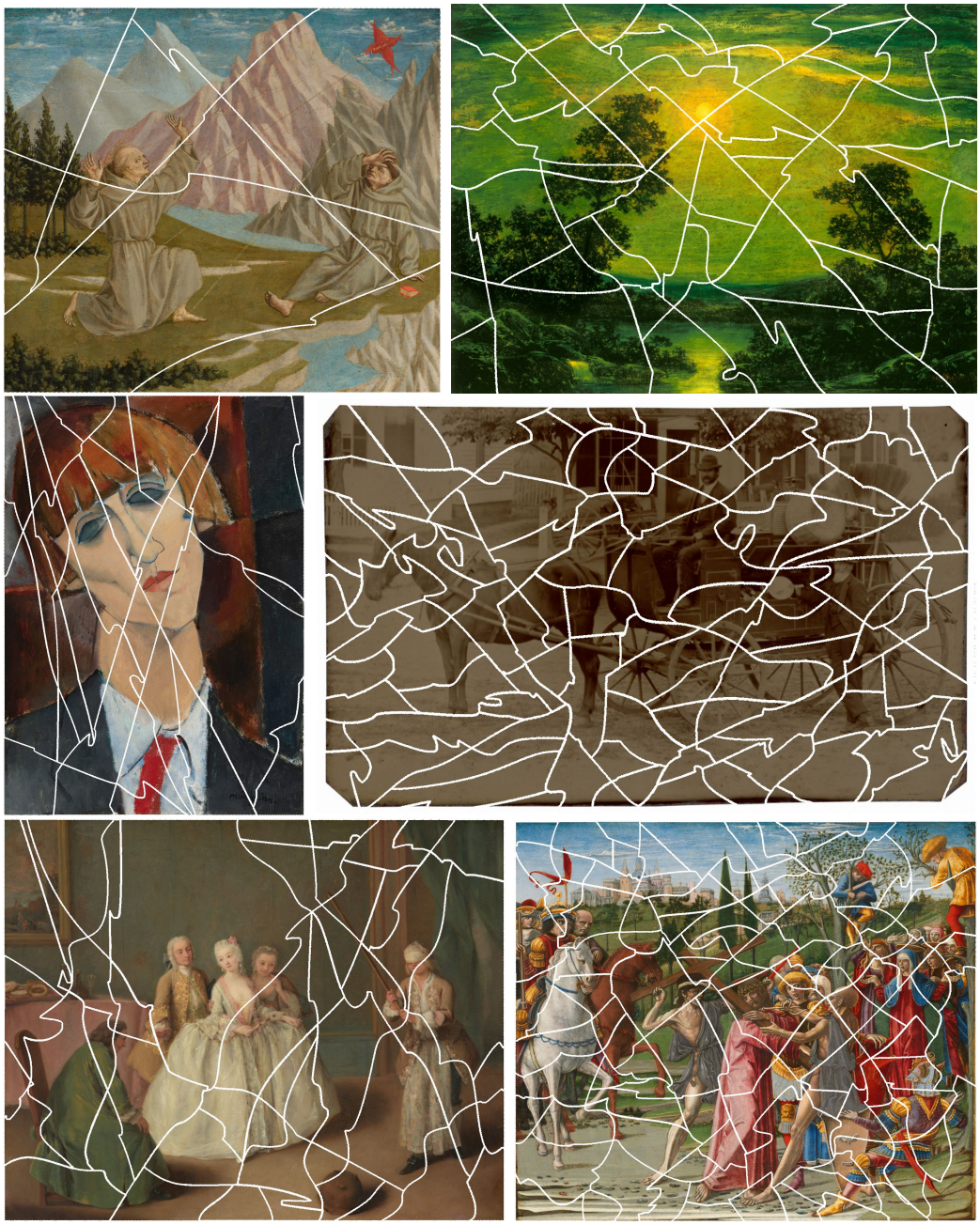


Figure 3.6 Examples of our synthetic fragmented images.

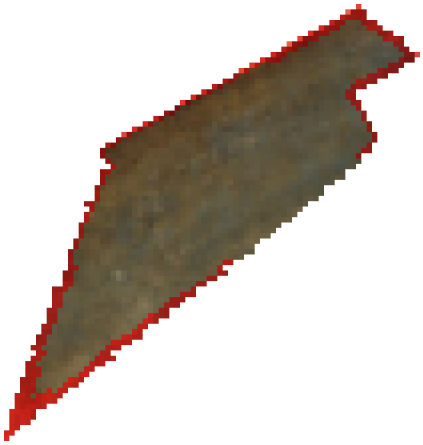


Figure 3.7 Visualization of erosion effect on a piece. Parts highlighted in red were removed.

- **Color Augmentation:** We randomly alter contrast, saturation, hue, and value to simulate diverse environmental lighting conditions and weathering effects on fragment surfaces (Figure 3.8c).

3.4 Data Format

After dividing an image into pieces, we store the data in a structured format to facilitate evaluation. Each puzzle is represented as follows:

1. **Piece Images:** Each puzzle piece is saved as an individual image, containing the visual content of the fragment (Figure 3.9a).
2. **Piece Masks:** For each piece, a corresponding binary mask is provided to facilitate boundary extraction (Figure 3.9b).
3. **Assembly Metadata:**
    - A list of piece transformations (rotation and translation) needed for correct assembly.
    - A list of neighboring piece pairs.

This information is saved in a JSON file, an example of which is shown below:

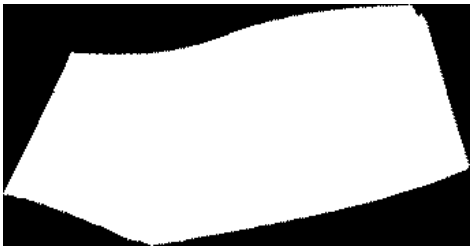
```
{
    "transformed_pieces": [
```

(a) Original image. **(b)** Image with eroded pieces. **(c)** Color augmented image.

Figure 3.8 Example of augmentations used in our synthetic dataset. The first image shows the original non-augmented image, the second image shows the image with eroded pieces, and the third image shows the color-augmented version.


```
        {
            "id": "000",
            "transformation": {
                "rotation_angle": 1.1432936781669358,
                "translation": [
                    -182.21601617933865,
                    400.6434610185635
                ]
            }
        },
        {
            "id": "001",
            "transformation": {
                "rotation_angle": 2.4872918104879265,
                "translation": [
                    31.136386101595804,
                    1177.813497118722
                ]
            }
        },
        ...
    ],
    "neighbors": [
        ["001", "002"], [ "008", "007"], ...
```

(a) Example of piece.



(b) Piece mask.

Figure 3.9 Example of one piece as stored in our synthetic dataset. Figure (a) shows the image, while (b) shows the binary mask.

```
    ]
}
```

Chapter 4

Local Assembly

The objective of local assembly is to identify a viable set of matches between puzzle pieces, determining both which fragments should form the pairs and the exact spatial transformations required for their proper alignment. To achieve this, we train a neural network capable of taking two pieces as input and predicting whether they share a common border and, if so, how to align them. This model is applied to all possible pairs of pieces in the dataset to produce the tentative matches.

This chapter describes the architecture of our model, the training process, the method for extracting matches from the model's output, and the evaluation metrics used to assess its performance.

4.1 Overview

Our model predicts whether pairs of pixels along the boundaries of two input pieces correspond, thereby determining the transformation necessary for their proper alignment. The similarity between pairs of contour pixels is represented as a similarity matrix. The workflow for generating and processing these matrices during training and prediction is illustrated in Figure 4.1 and described in detail in this section.

Consider two pieces, P_k and P_l, with contours represented as sequences of points $(c_0^k, \ldots, c_{n-1}^k)$ and $(c_0^l, \ldots, c_{m-1}^l)$, respectively. The model outputs a similarity matrix $\mathbf{M}^{kl} \in (0,1)^{n \times m}$, where $\mathbf{M}_{ij}^{kl}$ represents the probability that contour points c_i^k of P_k and c_j^l of P_l are matching.

To clarify terminology, this work distinguishes between two levels of correspondences:

1. **Piece-level correspondence**: Indicates whether two pieces share a common border and specifies the transformation required to align them.

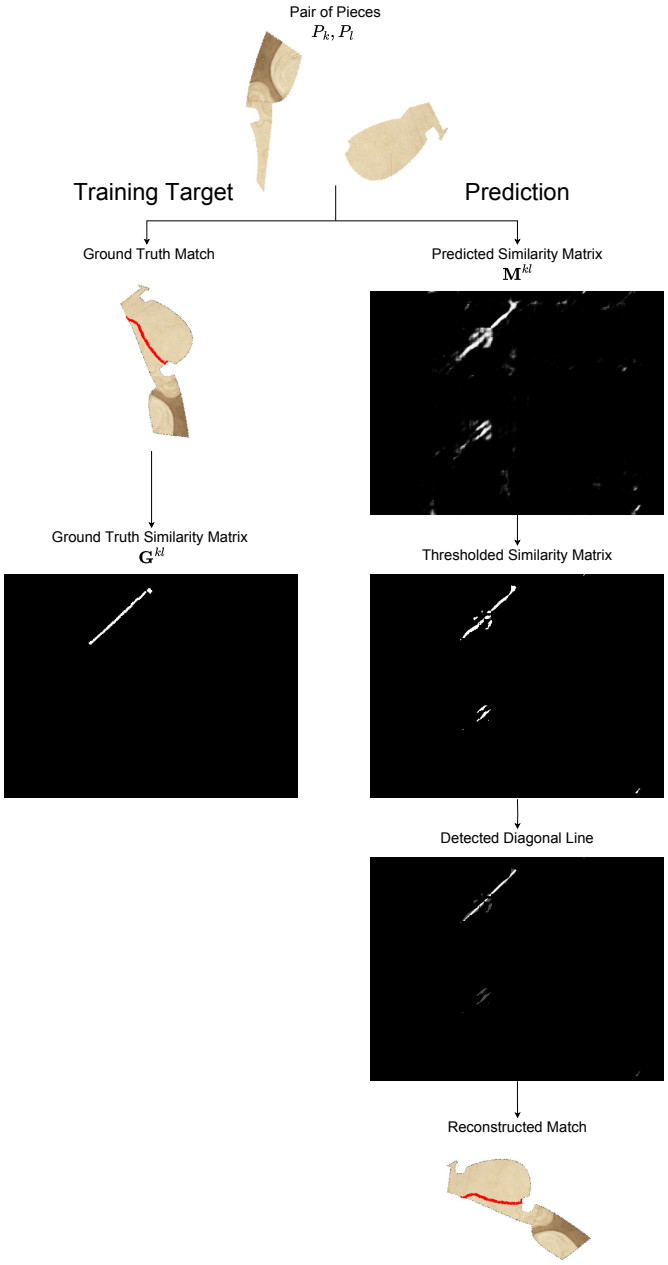


Figure 4.1 Training and prediction workflow for pixel-level contour matching. Given a pair of puzzle pieces P_k and P_l, the network predicts a similarity matrix $\mathbf{M}^{kl}$, indicating matching probabilities between contour points. The predicted matrix is thresholded and processed to extract a diagonal line, which determines the predicted alignment. The final result is a reconstructed match of the two pieces. The ground truth similarity matrix $\mathbf{G}^{kl}$, used as the training target, is computed from known alignments.

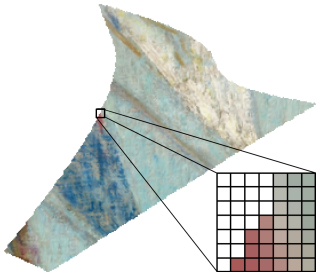


Figure 4.2 Example of patch of size $w \times w$ extracted around a contour point.

2. **Pixel-level correspondence**: Identifies which pixels on the contours of two pieces match when correctly aligned.

Our approach uses pixel-level correspondences to infer piece-level correspondences, which is our primary focus. However, pixel-level metrics are also valuable for model evaluation.

The goal of the model is to predict pixel-level correspondences; therefore, we use a representation of contour points as its input. Following [2], we extract a patch of size $w \times w$ around each contour point (Figure 4.2) and flatten it into a 1D vector, which serves as the input contour point's representation. These feature vectors are stacked to form a 2D tensor of shape (n, D_F), where n is the number of contour points and feature dimension $D_F = w^2 \cdot C$ (with C representing the number of image channels). This process is illustrated in Figure 4.3.

The input tensor is processed through a 1D convolutional neural network (CNN) with a U-Net-like architecture. The goal is to find a suitable representation of each contour point, which can then be used to measure the similarity between the contours of two pieces. The network's output is a tensor of shape (n, D_E), where D_E is the dimensionality of the embedding space. The rows of this tensor, referred to as *contour embeddings*, capture the features of individual contour points. For the i-th piece, we denote the matrix of contour embeddings as $\mathbf{E}^i$, and $\mathbf{e}^i_j$ represents the embedding of the j-th contour point. Details of the network architecture are provided in Section 4.2.

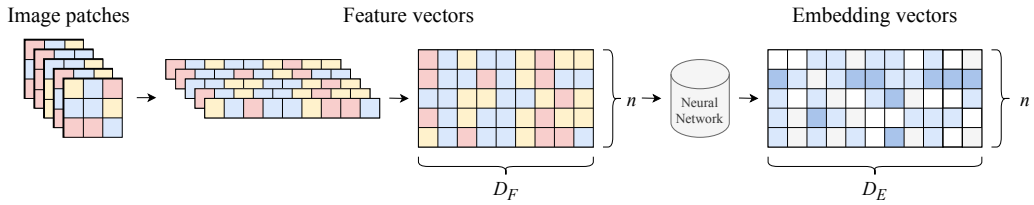


Figure 4.3 The process of obtaining the embedding vectors from image patches.

Given embeddings $\mathbf{E}^k$ and $\mathbf{E}^l$ for pieces P_k and P_l, we compute the similarity

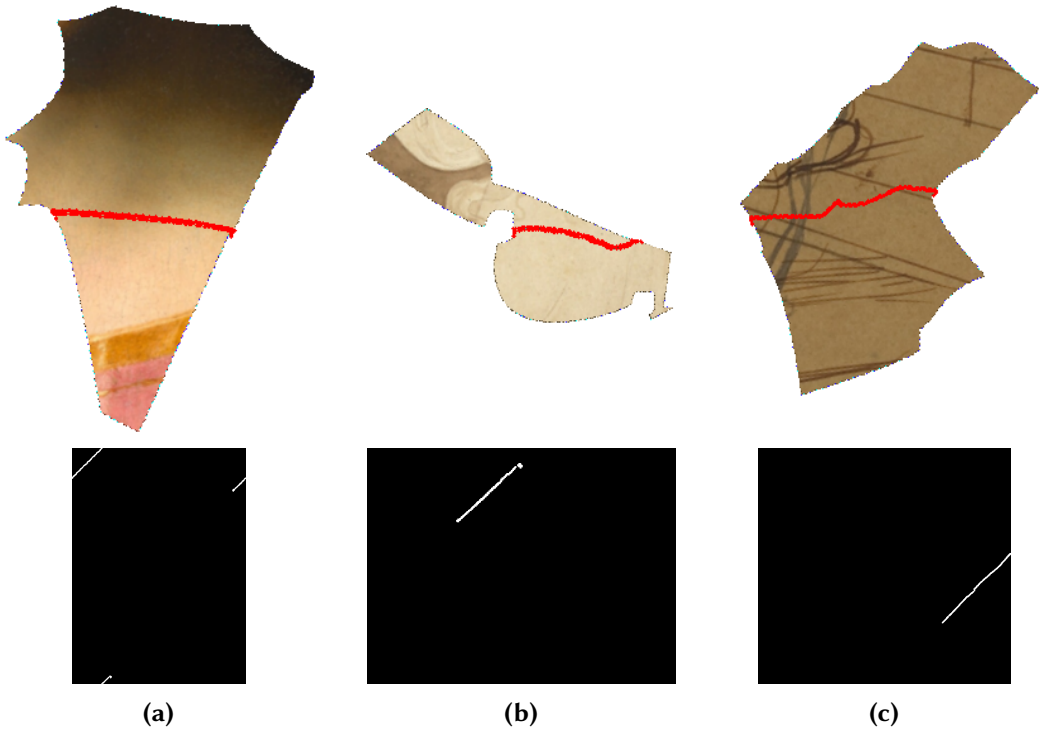


Figure 4.4 Examples of ground truth matches between puzzle pieces (top row) and the corresponding ground truth similarity matrices (bottom row). The red curves indicate contour segments that align between the matched pieces, which are reflected as white lines in the similarity matrices.

matrix $\mathbf{M}^{kl} = \sigma(\mathbf{E}^k \cdot (\mathbf{E}^l)^\top)$, where $\mathbf{M}^{kl}_{ij}$ is the dot product of of corresponding contour embeddings $\mathbf{e}^k_i$ and $\mathbf{e}^l_j$, scaled using the sigmoid function σ to obtain values in the range $(0, 1)$. This serves as a similarity measure for contour points.

The ground truth matrix $\mathbf{G}^{kl} \in \{0, 1\}^{n \times m}$, which represents the known mutual relationships between pieces k and l, is defined as follows:

$$\mathbf{G}^{kl}_{ij} = \begin{cases} 1 & \text{if } \mathrm{dist}(c^k_i, c^l_j) < t \\ & \quad \wedge \ \Big(j = \mathrm{argmin}_o \mathrm{dist}(c^k_i, c^l_o) \ \vee \ i = \mathrm{argmin}_o \mathrm{dist}(c^k_o, c^l_j)\Big) \\ 0 & \text{otherwise} \end{cases}$$

Here, t is a threshold, and $\mathrm{dist}(c^k_i, c^l_j)$ is the Euclidean distance between correctly aligned contour points c^k_i and c^l_j.

Ideally, all 1s in $\mathbf{G}^{kl}$ form a diagonal line, but due to inaccuracies from image manipulation, this is rarely the case. Examples of ground truth matrices are shown in Figure 4.4.

Ground Truth Matrices Properties

There are several important observations about ground truth matrices, which we have to address in the model design:

- **Discontinuities in the diagonal line**: Due to ambiguities in pixel-level correspondences, exact matches are often unrealistic. To address this, dilation is applied to the ground truth matrix to thicken the diagonal line, thereby accommodating inaccuracies and ambiguities.
- **Cyclic boundary conditions**: The contour points of a piece form a closed cycle, and the starting point is arbitrary. This results in diagonal lines that may wrap around the boundaries of the matrix, as shown in Figure 4.4a. The network architecture must account for this cyclic nature, as discussed in the next section.
- **Directionality**: Diagonal lines in the ground truth matrix $\mathbf{G}$ typically slope from bottom-left to top-right, as contours are extracted in a clockwise direction. This introduces asymmetry in the data, as shown in Figure 4.5. If two points c_i^1 and c_j^1 match with c_k^2 and c_l^2, respectively, the order of indices differs: $i < j$ implies $k > l$. To mitigate this, we can either reverse the input contours and ground truth matrices or train separate models for each direction. These solutions are explored in the following section.

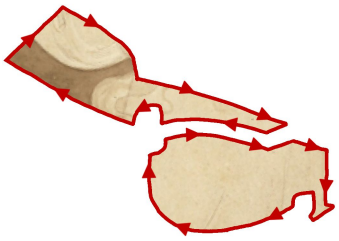


Figure 4.5 Illustration of the impact of directionality on contour matching. When the contours of both pieces are traversed in the same direction, the matching border segments are processed in a clockwise direction for one piece and in a counter-clockwise direction for the other.

4.2 Model Architecture

As described in the previous section, the model takes as an input contour features and for each point of the contour it produces an embedding vector. This process

is illustrated in Figure 4.3. We can then compare these embeddings of contours of different pieces and produce a similarity matrix, which tells us how likely it is for contour points to match.

The neural network used in this work is a 1D convolutional U-Net-like architecture [26], shown in Figure 4.6. The U-Net architecture is a deep learning model designed primarily for image segmentation tasks. Similarly to segmentation tasks, we leverage its ability to combine low-level features with high-level contextual information to extract meaningful representations for each contour point. These representations capture essential details that can be used to measure similarity between the contours of different pieces.

The network follows an encoder-decoder structure with symmetric contracting and expanding paths. The contracting path extracts increasingly abstract feature representations through convolutional and pooling operations. The expanding path progressively reconstructs the output resolution while integrating features from the contracting path via skip connections. The 1D version of this architecture processes one-dimensional data, making it ideal for our problem involving contour embeddings.

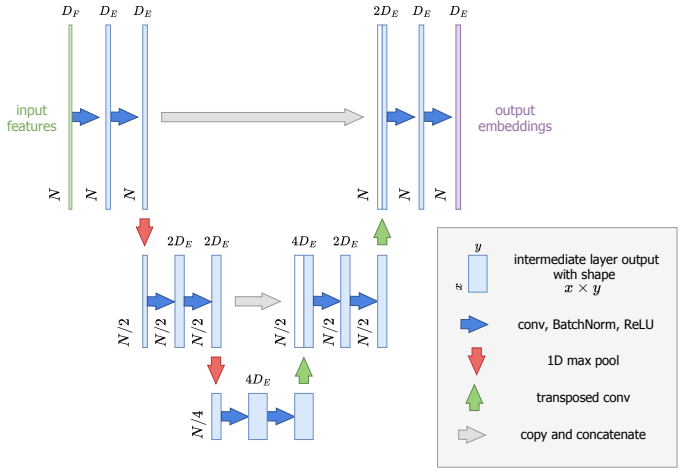


Figure 4.6 Model architecture.

4.2.1 Encoder

The encoder consists of multiple convolutional blocks, each containing the following operations:

- **1D Convolution:** Extracts features from the input tensor using small, learnable filters. Kernel size is configurable, with 3 as a common choice.

- **Batch Normalization:** Normalizes the activations to improve training stability.
- **Activation (ReLU):** Applies a non-linear activation function to introduce non-linearity into the model.
- **1D Max Pooling:** Down-samples the data, reducing its resolution while retaining essential features.

These blocks successively reduce the spatial resolution of the input while increasing the number of feature channels, thereby learning higher-level abstractions at each stage.

4.2.2 Decoder

The decoder mirrors the encoder with the following operations:

- **Transposed Convolution:** Performs up-sampling to increase the resolution of the feature maps.
- **Skip Connections:** Copies feature maps from corresponding layers in the encoder and concatenates them with the up-sampled feature maps in the decoder, providing a direct pathway for low-level spatial features to flow from the encoder to the decoder. This improves the model's ability to reconstruct fine-grained details.
- **Convolution, Batch Normalization and Activation (ReLU):** Refines the up-sampled features to produce the final output. The last layer uses no activation function, as its output is used directly as the output of the model.

4.2.3 Cyclic Boundary Condition

As discussed in the previous section, the contour points of a piece form a closed cycle, and the starting point for extracting these points is arbitrary. This is illustrated in Figure 4.7. The network architecture must account for this cyclic nature to ensure consistent processing of contours. Additionally, different pieces have contours of varying lengths, which must also be addressed.

One potential solution is to leverage cyclic padding, a feature supported by many implementations of convolutional layers. However, this approach has a significant limitation: contours of different lengths cannot be processed in a single batch. Using cyclic padding would restrict the batch size to 1, which is computationally inefficient and undesirable.

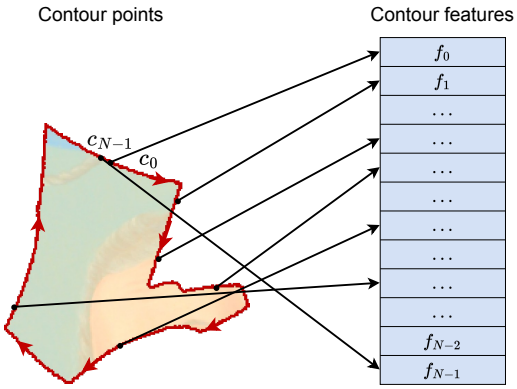


Figure 4.7 Contour point features. Points c_0 and c_{N-1} are spatially close to each other, but the corresponding rows in the feature matrix are not.

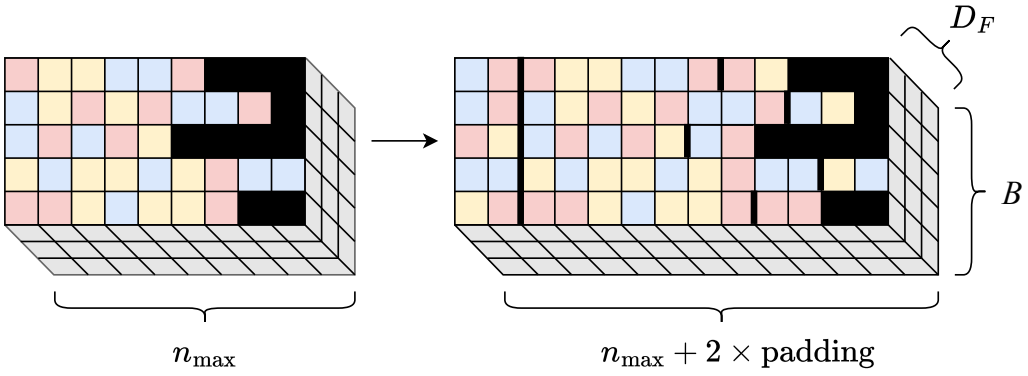


Figure 4.8 Application of cyclic padding to a batch of training data. Black regions mark the zero-padded regions. B denotes the batch size and $n_{\max}$ is the maximum length of a contour in the batch. Here, the padding size is 2.

An alternative solution is to use convolutional layers without padding and apply cyclic padding of an appropriate size (determined by the depth of the network) to the contours beforehand, as shown in Figure 4.8. This method allows for processing multiple contours in a single batch by using zero padding to match their shapes. It is crucial to mask out the zero-padded regions when calculating the loss. This is the approach adopted in our implementation.

4.2.4 Handling Directionality

Another issue, as discussed in the previous section, is the directionality of the contours. Contours can be traversed in either a clockwise or counter-clockwise

direction, which affects the embeddings generated by the network.

One straightforward solution is to process one contour in the clockwise direction and the other in the counter-clockwise direction. In this case, both contours are passed through the same model to generate their respective embeddings.

An alternative approach is to use two separate networks with the same architecture, where one processes contours in a clockwise direction and the other in a counter-clockwise direction. However, our experiments did not show a significant improvement over the single-network approach. Given that the dual-network approach involves a higher number of parameters and, consequently, requires more computational resources, we opted for the single-network approach instead.

In both approaches, two sets of embeddings are generated for each piece:

1. For the single-network method, embeddings are produced for contours traversed in both clockwise and counter-clockwise directions using the same model.

2. For the dual-network method, each network independently generates embeddings for its designated direction.

4.3 Match Reconstruction

Given two pieces P_k and P_l with contours $(c_0^k, c_1^k, \ldots, c_n^k)$ and $(c_0^l, c_1^l, \ldots, c_m^l)$, the embedding model produces embedding matrices $\mathbf{E}^k$ and $\mathbf{E}^l$. Using these embeddings, we compute the similarity matrix $\mathbf{M}^{kl} = \sigma(\mathbf{E}^k \cdot (\mathbf{E}^l)^\top)$, where σ is the element-wise sigmoid function. Each element $\mathbf{M}_{ij}^{kl}$ represents the similarity between contour points c_i^k and c_j^l. This process is illustrated in Figure 4.9.

An example of the similarity matrix is shown in Figure 4.10. The matrix exhibits a strong diagonal line corresponding to matching contour points and weaker lines representing less probable matches. The objective is to identify the most prominent diagonal line and use it to estimate the match between the two pieces. This process is described in the next section.

4.3.1 Reconstruction Steps

This section outlines the steps required to reconstruct the match between two pieces P_k and P_l from the similarity matrix $\mathbf{M}^{kl}$. These steps include thresholding, detecting the strongest diagonal line, and estimating the transformation.

Thresholding In the first step, low-similarity correspondences are removed by applying a threshold to $\mathbf{M}^{kl}$. The result, as shown in Figure 4.10d, retains only high-confidence matches.

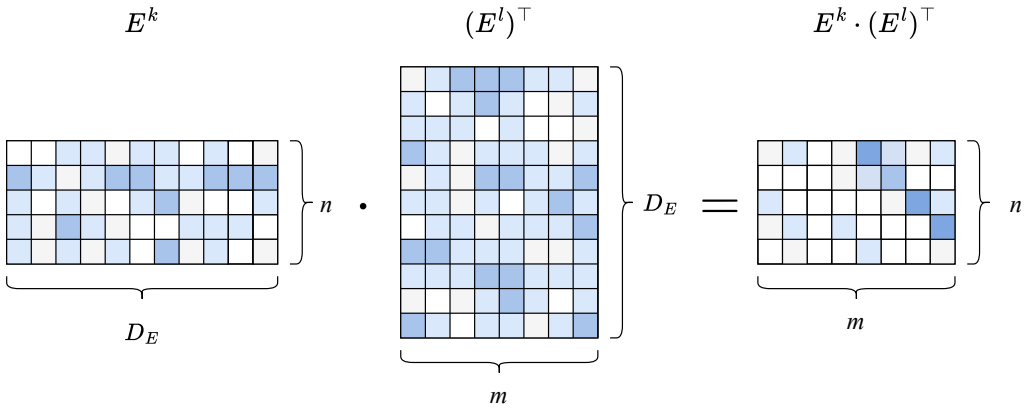


Figure 4.9 Computation of the similarity matrix from contour embeddings. $\mathbf{E}^k$ and $\mathbf{E}^l$ are the embedding matrices for pieces P_k and P_l, respectively. The resulting product yields the similarity matrix $\mathbf{M}^{kl}$, prior to the application of the sigmoid function.

Diagonal Line Detection The next step is to detect the longest diagonal line in the thresholded matrix. We use the Hough transform to achieve this, which allows restricting the search to specific angles (e.g., $(\pi/6, \pi/3)$). Only the most dominant lines of sufficient length are considered valid matches. If no sufficiently long diagonal line is detected, the two pieces are classified as non-neighbors.

Due to the cyclic nature of the similarity matrix $\mathbf{M}^{kl}$, diagonal lines may extend beyond the matrix boundaries. To address this, the matrix is tiled in both dimensions, ensuring continuity of the diagonal lines for detection (see Figure 4.11).

Extracting Correspondences Once a diagonal line is detected, the corresponding contour points are extracted as a list of tuples:

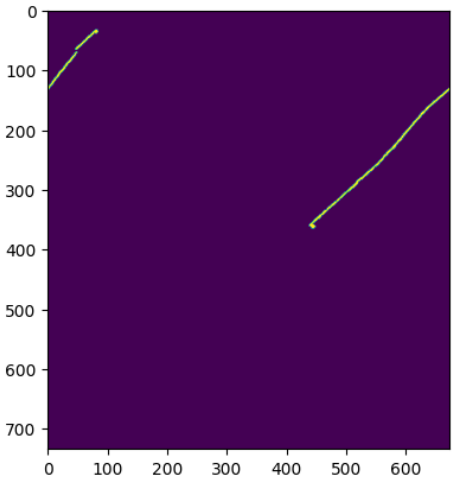
$$((i_1, j_1), (i_2, j_2), \ldots, (i_{n_{\text{pixel}}}, j_{n_{\text{pixel}}})),$$

where n_{pixel} is the number of detected correspondences and (i_k, j_k) are the indices of matching contour points.

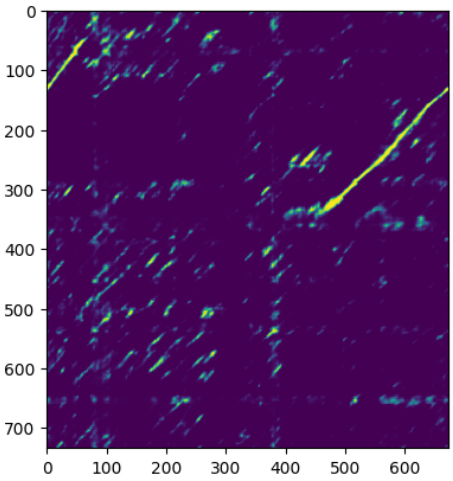
Estimating Transformation To reconstruct the match, we compute the rigid transformation $\mathbf{T}$ (a combination of rotation and translation) that maps the contour points $(c^k_{i_1}, c^k_{i_2}, \ldots, c^k_{i_{n_{\text{pixel}}}})$ to $(c^l_{j_1}, c^l_{j_2}, \ldots, c^l_{j_{n_{\text{pixel}}}})$. This transformation is expressed as:

$$\mathbf{T} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix},$$

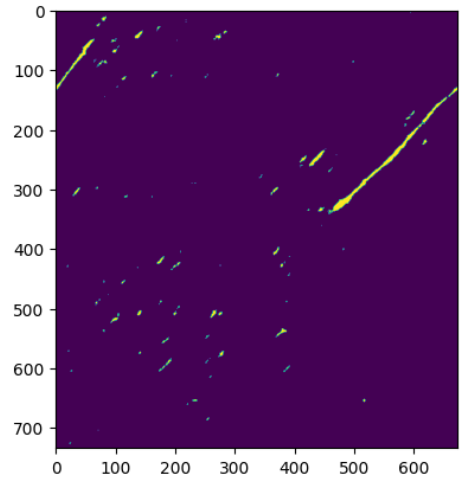
(a) Ground truth match.



(b) Ground truth similarity matrix $\mathbf{G}^{kl}$.



(c) Similarity matrix $\mathbf{M}^{kl}$.



(d) Matrix $\mathbf{M}^{kl}$ after thresholding.

Figure 4.10 Predicted correspondence between pieces P_k and P_l. (a) shows the matching pieces, (b) illustrates the ground truth similarity matrix $\mathbf{G}^{kl}$, (c) displays the predicted similarity matrix $\mathbf{M}^{kl}$, and (d) shows the predicted matrix after thresholding.

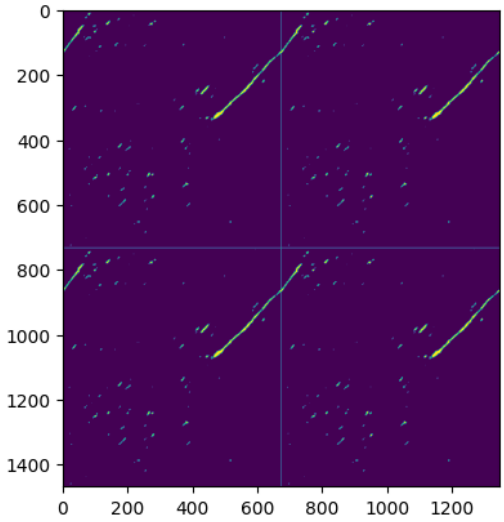


Figure 4.11 Example of tiling for line detection.

where θ is the rotation angle and (t_x, t_y) is the translation vector. The mapping is given by:

$$\begin{bmatrix} c^l_{j_1} \\ c^l_{j_2} \\ \vdots \\ c^l_{j_{n_{\text{pixel}}}} \end{bmatrix}^\top = \mathbf{T} \begin{bmatrix} c^k_{i_1} \\ c^k_{i_2} \\ \vdots \\ c^k_{i_{n_{\text{pixel}}}} \end{bmatrix}^\top .$$

Since an exact solution is unlikely (due to noise and outliers), the RANSAC [27] algorithm is used to robustly estimate $\mathbf{T}$. The transformation from RANSAC serves as the initial guess for fine-tuning using the Iterative Closest Point (ICP)[28, 29] algorithm, which refines the alignment.

4.3.2 Confidence Score

It is important to quantify the quality of a match—that is, the certainty that the given two pieces truly belong together and are correctly aligned. We refer to this measure of match quality as the *confidence score*.

A reliable confidence score is essential for global assembly, as most existing methods depend on it, particularly when ranking matches in a greedy approach.

Given a match (i.e., two pieces and the transformation aligning them), various factors can be considered to compute the confidence score, such as the length of the shared border, the proximity of matching contour points, and the color difference between adjacent regions. These hand-crafted metrics often require extensive hyperparameter tuning and can be challenging to design effectively.

Threshold	Recall	w_l	AUC-ROC
0.5	0.8609	0.3	0.9957
0.6	0.8632	0.3	0.9957
0.7	0.8529	0.3	0.9945
0.8	0.8368	0.3	0.9946
0.9	0.7621	0.3	0.9945

Table 4.1 Best weight w_l and AUC-ROC for different values of threshold for model trained on non-augmented dataset.

To address this, we compute the confidence score directly from the model's output. This approach provides a robust proxy for match quality while eliminating the need for manual tuning.

The confidence score for match between pieces P_k and P_l is computed from the similarity matrix $\mathbf{M}^{kl}$. From the estimated match, we extract contour point pairs using the same method as for the ground truth matrix. The confidence values for individual pairs are taken directly from $\mathbf{M}^{kl}$ and aggregated by computing their mean, yielding a single confidence value per match.

Additionally, our experiments indicate that incorporating match length (i.e., the number of corresponding point pairs) improves the score's reliability. This adjustment assigns higher scores to longer matches while penalizing weak connections where pieces only touch by a few pixels.

The total score for a match is then given by:

$$S(k,l) = \frac{\sum_{i,j \in \text{pairs}_P(k,l)} \mathbf{M}^{kl}_{ij}}{|\text{pairs}_P(k,l)|} \cdot |\text{pairs}_P(k,l)|^{w_l} \tag{4.1}$$

where $\text{pairs}_P(k,l)$ represents the set of matching contour point pairs for pieces P_k and P_l, $\mathbf{M}^{kl}$ is the similarity matrix, and w_l is a hyperparameter weighting the match length.

To assess the impact of w_l, we use the area under the receiver operating characteristic curve (AUC-ROC), which measures how well the score distinguishes between good and bad matches. The effect of different values of w_l on ROC curve and score histograms is shown in Figure 4.12. We evaluated different values of w_l and similarity matrix thresholds using synthetic puzzle problems that were not part of the training or test data.

The tested values for w_l were $\{0, 0.1, 0.2, \ldots, 1\}$, and for the threshold, $\{0.5, 0.6, 0.7, 0.8, 0.9\}$. Evaluations were conducted for both a model trained on a non-augmented dataset (Table 4.1) and a model trained with eroded pieces (Table 4.2).

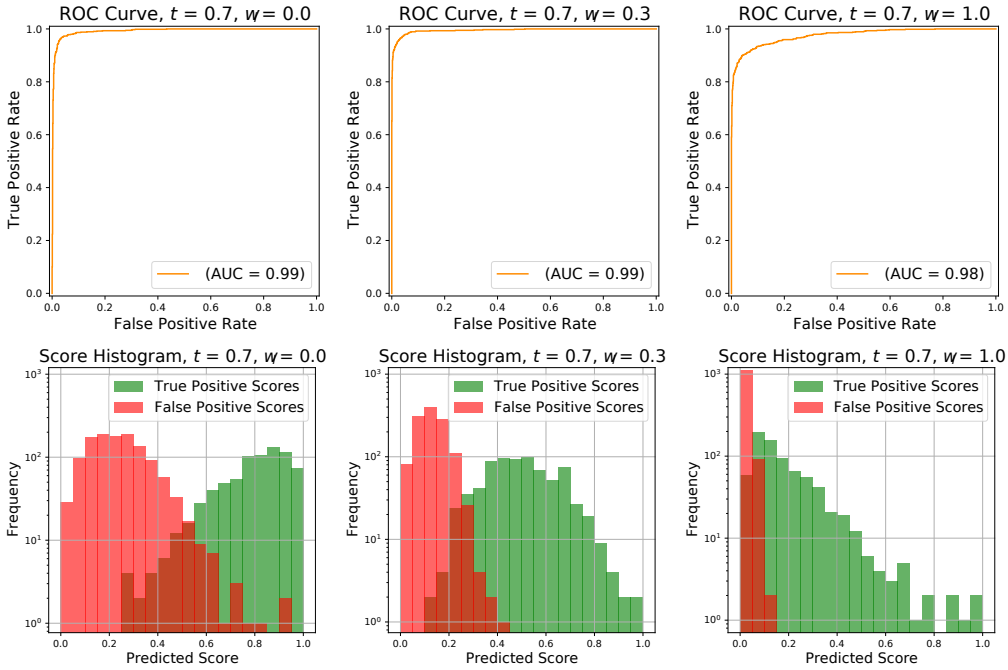


Figure 4.12 Receiver operating characteristic curves and histograms of resulting scores for different values of w_l. Threshold is set to 0.7. Weight $w_l = 0.3$ is the choice with the highest AUC-ROC. The y-axis is logarithmically scaled to maintain a consistent range across all three plots.

Threshold	Recall	w_l	AUC-ROC
0.5	0.7575	0.4	0.9840
0.6	0.7391	0.4	0.9819
0.7	0.6977	0.5	0.9833
0.8	0.6138	0.5	0.9853
0.9	0.4402	0.6	0.9745

Table 4.2 Best weight w_l and AUC-ROC for different values of threshold for model trained on dataset with simulated erosion.

4.4 Training

During the training, we obtain the similarity matrix $\mathbf{M}$ in the same way as described in Section 4.3. It is important to note that when using a batch size larger than 1, the respective embeddings and the matrix contain padding. Therefore, only the unpadded elements of the matrix are valid, and the rest should be masked out.

The task can be viewed as a set of $n \cdot m$ binary classifications, where for each pair of contour points, we predict whether they correspond. For this reason, we use Binary Cross Entropy (BCE) as the loss function:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_{k=1}^{b} \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} \Big[\mathbf{G}_{ij}^{k} \, \log \mathbf{M}_{ij}^{k} + (1 - \mathbf{G}_{ij}^{k}) \, \log(1 - \mathbf{M}_{ij}^{k}) \Big], \tag{4.2}$$

where b represents the batch size, $\mathbf{G}^k$ and $\mathbf{M}^k$ correspond to the ground truth and predicted similarity matrices of the k-th sample, respectively, n_k and m_k are the dimensions of those matrices and $N = \sum_k n_k m_k$ This is the basic version of the loss function, without any masking or weighting.

The simplified pseudocode of the training loop is shown in Algorithm 1. For all experiments, we use the Adam optimizer [30] with a learning rate of 0.0001.

Algorithm 1 Training loop. This pseudocode uses two separate networks for each input, as described in Section 4.2.4. For the case where only one network is used, set $\text{network}_2 = \text{network}_1$. Here, D_F and D_E are the dimensions of the input and output embeddings, $\mathbf{F}^k$ and $\mathbf{F}^l$ are the input features, and $\mathbf{G}^{kl}$ is the ground truth matrix.

$\text{network}_1 \leftarrow \text{EmbeddingNetwork}(D_F, D_E)$
$\text{network}_2 \leftarrow \text{EmbeddingNetwork}(D_F, D_E)$
for epoch $\leftarrow 1$ to epochs **do**
    for each $(\mathbf{F}^k, \mathbf{F}^l, \mathbf{G})$ in batches **do**
        $\mathbf{E}^k \leftarrow \text{network}_1(\mathbf{F}^k)$
        $\mathbf{E}^l \leftarrow \text{network}_2(\textsc{reverse}(\mathbf{F}^l))$
        $\mathbf{M}^{kl} \leftarrow \sigma(\mathbf{E}^k \cdot (\mathbf{E}^l)^\top)$
        $L \leftarrow \text{BCE}(\mathbf{M}^{kl}, \mathbf{G})$
        $L \leftarrow \textsc{Mask}(L, \mathbf{G}^{kl})$
        $\textsc{backward}(L)$
    end for
end for

4.4.1 Dealing with Class Imbalance

In typical jigsaw puzzle problems, there are significantly more pairs of pieces that are not touching than those that do. For example, in a grid-organized jigsaw puzzle with an $n \times n$ grid, there are $2n(n-1)$ touching pairs of pieces, while the total number of piece pairs is $\binom{n^2}{2} = \frac{n^2(n^2-1)}{2}$.

Similarly, for two neighboring pieces, there are many more pairs of contour points that do not match compared to those that do. For pieces with contour lengths n and m, sharing a border of length b, there are $n \times m$ contour point pairs, of which only approximately $c \cdot b$ are labeled as matching. Here, c is a small constant depending on the method used to construct the ground truth matrix.

In machine learning, it is well known that models can overfit to the majority class in imbalanced datasets. In our case, there are two levels of imbalance:

1. **Piece-level imbalance**: Between touching and non-touching pieces.
2. **Pixel-level imbalance**: Between corresponding and non-corresponding contour points.

Both must be addressed to train the model effectively.

Piece-Level Class Imbalance

Our dataset includes only neighboring pairs of pieces, but generating negative pairs during training is straightforward, as the ground truth similarity matrix for non-touching pieces is entirely zeros. In our experiments, we randomly draw pieces from the dataset, without restricting pairs to be from the same image. Including explicitly negative pairs from the same image did not yield significant improvements in our experiments.

The key consideration is the ratio of negative pairs to add. If too few negative pairs are used, the model may overfit to finding correspondences between similarly looking pieces, producing random outputs for dissimilar pairs. Conversely, a high ratio of negative pairs slows down training but does not harm performance if pixel-level imbalance is handled correctly.

Pixel-Level Class Imbalance

Pixel-level imbalance can be addressed directly in the loss function, with two common approaches:

1. Assigning higher weights to positive examples and lower weights to negative examples. This approach uses all available data but requires knowledge of the initial class ratios to fine-tune weights effectively.

2. Using a masking strategy in the loss function. Positive pairs are retained, while a predefined proportion of negative pairs is randomly masked out. This allows precise control of the positive-to-negative ratio but may lead to slower training due to unused data during backpropagation. Handling batches with only negative examples can also be problematic.

We primarily use the first approach but have also experimented with the second. The results of our experiments are presented in Appendix A.

4.5 Evaluation Methodology

When solving the jigsaw puzzle problem, the primary goal is to determine whether the entire image can be accurately assembled or, at the very least, how much of it can be reconstructed. It is also important to evaluate the quality of local assembly independently, so that different methods can be compared without the influence of the global assembly process.

4.5.1 Standard Metrics

The objective of local assembly is to produce a set of matches from the complete set of pieces. To evaluate this, we use standard metrics:

- **Precision**: The ratio of correct matches to all predicted matches.
- **Recall**: The ratio of correct matches to all ground truth matches.
- **F1 Score**: The harmonic mean of precision and recall.

While these metrics provide valuable insights, they should not be the sole basis for comparison. For instance, if the global assembly method is particularly good at handling false positives, precision becomes less important. Similarly, it is not always necessary to find every match to correctly reconstruct the image.

If we represent the jigsaw puzzle as a planar graph, where edges correspond to ground truth matches, reconstructing the image requires finding at least a spanning tree of this graph. For a puzzle with m pieces, each spanning tree contains $m - 1$ edges. According to Euler's formula, the maximum number of edges in a planar graph is $3m - 6$. Thus, the theoretical lower bound on recall for assembling the image is:

$$\frac{m-1}{3m-6} \xrightarrow[m\to\infty]{} \frac{1}{3}.$$

For a grid-organized puzzle with $m = n^2$ pieces and $2n(n-1)$ neighbor pairs, the lower bound becomes:

$$\frac{n^2 - 1}{2n^2 - 2} \xrightarrow[m \to \infty]{} \frac{1}{2}.$$

In cases where there are $m - 1$ neighbor pairs (e.g., images split into parallel strips), every match must be identified to reconstruct the entire image.

This means that puzzles with recall as low as 0.3 can, in theory, be correctly assembled if the identified matches form a spanning tree. However, recall comparisons across puzzles with different structures are not meaningful, as the structural properties significantly affect the recall threshold required for successful assembly.

The specific location of identified matches is often more critical than their quantity. For example, consider an image where one half consists of a textureless sky and the other half has a detailed background:

- A method that perfectly reconstructs the textured part but finds no matches in the sky achieves a recall of 0.5, yet global assembly becomes impossible.
- Another method that identifies 50% of matches in both the sky and textured areas also achieves a recall of 0.5, but it enables partial or complete assembly.

This highlights that recall alone does not fully capture the practical utility of local assembly for global reconstruction.

4.5.2 Balancing Precision and Recall

As in most classification tasks, there is a trade-off between precision and recall. In our case, this trade-off can be controlled by adjusting the weights in the loss function, as described in Section 4.4.1. Determining the optimal balance depends on the global assembly method:

- Greedy methods are sensitive to false positives but do not rely on redundant matches. In theory, matches forming a spanning tree are sufficient for full assembly. Therefore, prioritizing precision over recall may be more effective.
- Some graph-based methods rely on detecting consistent cycles in the graph formed by detected matches, making recall more important.

This dependency complicates direct comparisons of local assembly methods without considering the global assembly process.

4.5.3 Additional Metrics

To address these challenges, we propose evaluating whether it is theoretically possible to assemble the entire image given the set of matches, independent of the global assembly method. Two metrics are defined:

- **Largest Connected Component ($E_{\mathbf{LCC}}$)**: the ratio of the number of pieces in the largest connected component to the total number of pieces, considering only correctly classified matches. This metric reflects the extent to which the image could be reconstructed using an ideal global assembly method without errors.
- **Fully Assembled ($E_{\mathbf{FA}}$)**: A binary metric indicating whether the entire image can be reconstructed—i.e., whether the largest connected component as defined above includes all puzzle pieces.

Both metrics provide a practical perspective on the effectiveness of local assembly, offering insights that complement traditional metrics such as precision and recall.

4.6 Results

In Table 4.3, we present a selection of our evaluation results; the full table is available in Appendix D.

Train Dataset	Test Dataset	P	R	F_1	Acc	E_{FA}	E_{LCC}
Eroded	Eroded	0.2497	0.7584	0.3628	0.6838	0.5542	0.9701
	Non-Augmented	0.3150	0.8498	0.4445	0.6900	0.8292	0.9925
	JigsawNet	0.2569	0.9900	0.3541	0.7232	1.0000	1.0000
	All	0.2739	0.8661	0.3871	0.6990	0.7944	0.9875
Non-Augmented	Eroded	0.2851	0.7181	0.4034	0.7457	0.4806	0.9545
	Non-Augmented	0.3604	0.8725	0.5021	0.7991	0.9590	0.9985
	JigsawNet	0.3215	0.9931	0.4348	0.8240	1.0000	1.0000
	All	0.3223	0.8612	0.4467	0.7896	0.8132	0.9843

Table 4.3 Evaluation results of our model. We compare models trained on our synthetic datasets—with and without erosion augmentation—and evaluate their performance also on the JigsawNet dataset [21].

Chapter 5

Global Assembly

Global assembly represents the second phase of jigsaw puzzle solving, where we use detected piece matches to reconstruct the complete image. In ideal conditions, where all piece matches are correct and no false positives exist, we can solve this using a simple greedy approach that iteratively applies matches until the image is complete.

In real-world scenarios, however, false matches can occur. A common solution to this problem is backtracking, which systematically explores different match combinations to find the correct solution. The main drawback of backtracking is that if an error occurs early in the assembly process, it requires exploring a large number of possible configurations. To address this, we introduce several improvements to the greedy approach, drawing inspiration from evolutionary algorithms. A high-level overview of our approach is shown in Figure 5.1.

During the assembly process, we maintain a dynamic set of clusters of pieces, $\mathcal{C}$, where each cluster represents a partially assembled image.

The process begins by identifying candidate matches and ranking them based on confidence scores. These matches are then represented as a graph, where cycles with consistent transformations are identified and considered "trusted." This provides an initial set of piece clusters. Next, the highest-ranked unused match is iteratively incorporated into existing clusters, followed by merging clusters that share common pieces. Each of these steps is detailed in the following sections.

The algorithm terminates when at least one of the following conditions is met:

- Set of cluster is a single cluster containing all pieces, indicating successful reconstruction.
- All candidate matches have been processed.

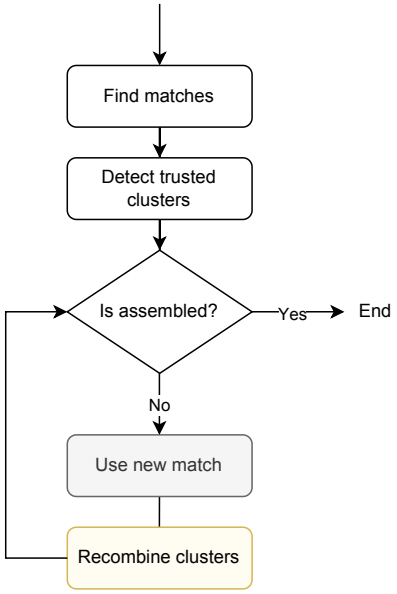


Figure 5.1 High-level overview of our global assembly algorithm.

- A predefined iteration limit is reached.
- No changes occur in the cluster set for a fixed number of iterations (determined by a *patience* parameter).

The algorithm outputs a set of clusters, where each cluster consists of a set of pieces with their respective transformations. In the ideal case, where the entire image is successfully reconstructed, the algorithm yields a single cluster containing all pieces.

5.1 Trusted Clusters

Determining the correctness of a match based solely on its confidence score is not possible. However, during the assembly process, if a match aligns consistently with others, it can often be assumed to be correct.

To take advantage of this, we introduce the concept of *trusted clusters*. A trusted cluster is considered a reliable partial solution, and all subsequent operations must maintain consistency with it. This approach allows us to systematically discard unlikely assembly paths and focus computational efforts on more promising solutions.

Before incorporating a new match, we verify whether it conflicts with any trusted clusters. If a conflict arises, the match is discarded, and the next best

match is considered. Similarly, if merging two clusters results in an inconsistency with an existing trusted cluster, the merge is rejected.

Trusted clusters are identified using two methods: they are either found before the assembly process (cycle-based trusted clusters) or dynamically during assembly.

Cycle-based Trusted Clusters

This method is inspired by graph-based global assembly techniques. We construct a graph of detected matches and search for short cycles, typically of length 3 or 4, as longer cycles would be computationally expensive. To further limit complexity, only the top $10n$ highest-scoring matches are considered, where n is the total number of pieces.

For each detected cycle, we verify whether the transformations between pieces are mutually consistent. If so, the cycle is assumed to be correct, and the corresponding pieces form a trusted cluster, serving as an initial foundation for the assembly process.

Trusted Clusters Identified During Assembly

During the assembly process, if a new match is introduced and it corresponds to a match already present in an existing cluster, it is marked as trusted and added to the set of trusted clusters.

5.2 Cluster Merging

When two clusters share common pieces due to new matches, a merging procedure is triggered. This consists of:

1. Identification of all common pieces between clusters
2. Computation of the alignment transformation
3. Cluster merging

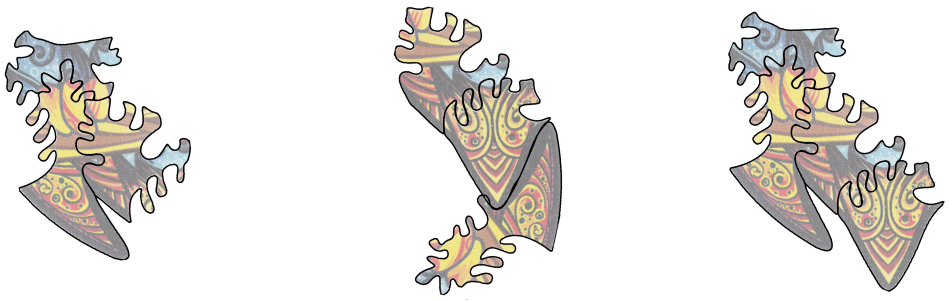
During cluster merging, two types of inconsistencies may arise, indicating incorrect piece placement:

- **Position Inconsistencies**: These occur when clusters share multiple common pieces, but these pieces don't align properly between clusters.

- **Piece Overlap**: This happens when merged pieces occupy the same space. In our implementation, we allow a some level of overlap tolerance to account for small errors that accumulate during assembly.

Traditional backtracking would return to a previous state when encountering errors and try different matches. Our approach instead aims to correct errors as they appear during merging and continues with the next match. This approach provides an effective balance between computational efficiency and accuracy while maintaining robustness against matching errors.

If inconsistencies arise, one cluster is chosen as the reference, typically at random unless a preferred cluster is specified (e.g., a newly introduced match or a trusted cluster). The transformations of the reference cluster are preserved, while conflicting elements from the other cluster are discarded (see Figure 5.2). If the resulting cluster doesn't form a single connected component, only the largest connected component is retained.



(a) First cluster

(b) Second cluster

(c) Merged clusters if transformations from cluster (a) were used

Figure 5.2 Merging of inconsistent clusters. (a) and (b) represent two clusters with inconsistencies, as one piece is placed differently in each. (c) shows the result of merging these clusters, with (a) selected as the reference cluster for the merging process, which determined the position of the conflicting piece.

5.3 Clusters Selection

At each step of the algorithm, we maintain a set of clusters that represent partial reconstructions of the image. These clusters may overlap, meaning a piece can belong to multiple clusters in $\mathcal{C}$. This occurs when two clusters share common pieces but are inconsistent, leading to a lower overall cluster score if merged. Our

approach allows for parallel exploration of multiple solutions, unlike classical backtracking approaches, which explore them sequentially. However, maintaining a large number of clusters increases computational complexity. To mitigate this, we retain only the highest-scoring cluster for each piece.

This selection process is applied whenever new clusters are formed (e.g., through merging) to ensure that only the most promising clusters are retained. Striking the right balance is crucial: retaining too few clusters may hinder successful assembly, while keeping too many increases computational overhead.

5.3.1 Cluster Score

Each cluster is assigned a score. An ideal scoring function should satisfy the following properties:

- Adding a correctly aligned piece increases the score, whereas an incorrectly aligned piece decreases it. In other words, a larger cluster does not always imply a better score.
- The number of neighboring pairs matters. In a graph representation, among clusters with the same number of pieces, those with more edges (neighbor pairs) receive higher scores. A new piece that connects multiple existing pieces contributes more significantly than one that connects only a single piece.

The cluster score function S_C is defined as:

$$S_C(C) = \frac{\sum_{i,j \in \text{neighbors}} S(i,j)}{\sqrt{|C|}} \tag{5.1}$$

where $S(i,j)$ represents the pair score between pieces i and j as defined in Section 4.3.2, and $|C|$ denotes the number of pieces in cluster C. The denominator normalizes the score, reducing the bias toward larger clusters. The square root was chosen empirically for optimal performance.

5.3.2 Selection Algorithm

Using the score defined above, the best clusters are selected for the next iteration. The selection process, detailed in Algorithm 2, ensures that for each piece p, the final cluster set $\mathcal{C}'$ contains the highest-scoring cluster from $\mathcal{C}$ that includes p.

Algorithm 2 Cluster selection

Input: $\mathcal{C}$ sorted by S_C in descending order
Output: $\mathcal{C}'$

$\mathcal{C}' \leftarrow \emptyset$
$P_{\text{used}} \leftarrow \emptyset$ ..... *set of pieces represented in* $\mathcal{C}'$
for all $C \in \mathcal{C}$ **do**
    if $C \not\subseteq P_{\text{used}}$ **then**
        $P_{\text{used}} \leftarrow P_{\text{used}} \cup C$
        $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{C\}$
    end if
end for

5.4 Incorporating a New Match

This section describes how a new match is incorporated into the assembly process. The algorithm is illustrated in Figure 5.3.

First, we select the highest-scoring match that has not yet been used. If this match is inconsistent with trusted clusters, it is discarded and the next best match is considered. In contrast, if any cluster already contains this match, we mark it as trusted and add it to the set of trusted clusters, as described in Section 5.1.

When a new match is selected, it is applied to the current set of clusters, $\mathcal{C}$, attempting to merge it with existing clusters. If it does not share any common pieces with existing clusters, it is treated as a new independent cluster.

If the new match is incorrect, it may corrupt some of the initial clusters. To address this, we apply the cluster selection strategy (see Section 5.3), where the input is the union of the initial cluster set $\mathcal{C}$ and the new clusters created by incorporating the new match.

5.5 Recombining Clusters

The recombination step ensures that clusters capable of being merged are merged. Additionally, in this step, we attempt to reintroduce matches that were previously discarded due to conflicts during the merging process. The recombination process is illustrated in Figure 5.4.

During the merging of inconsistent clusters, some valid matches may have been accidentally discarded. To mitigate this, we reintroduce them during the recombination stage when merging clusters with common pieces.

At this step, all previously used matches are readded to the cluster set, giving them another chance to be incorporated as more context becomes available. To

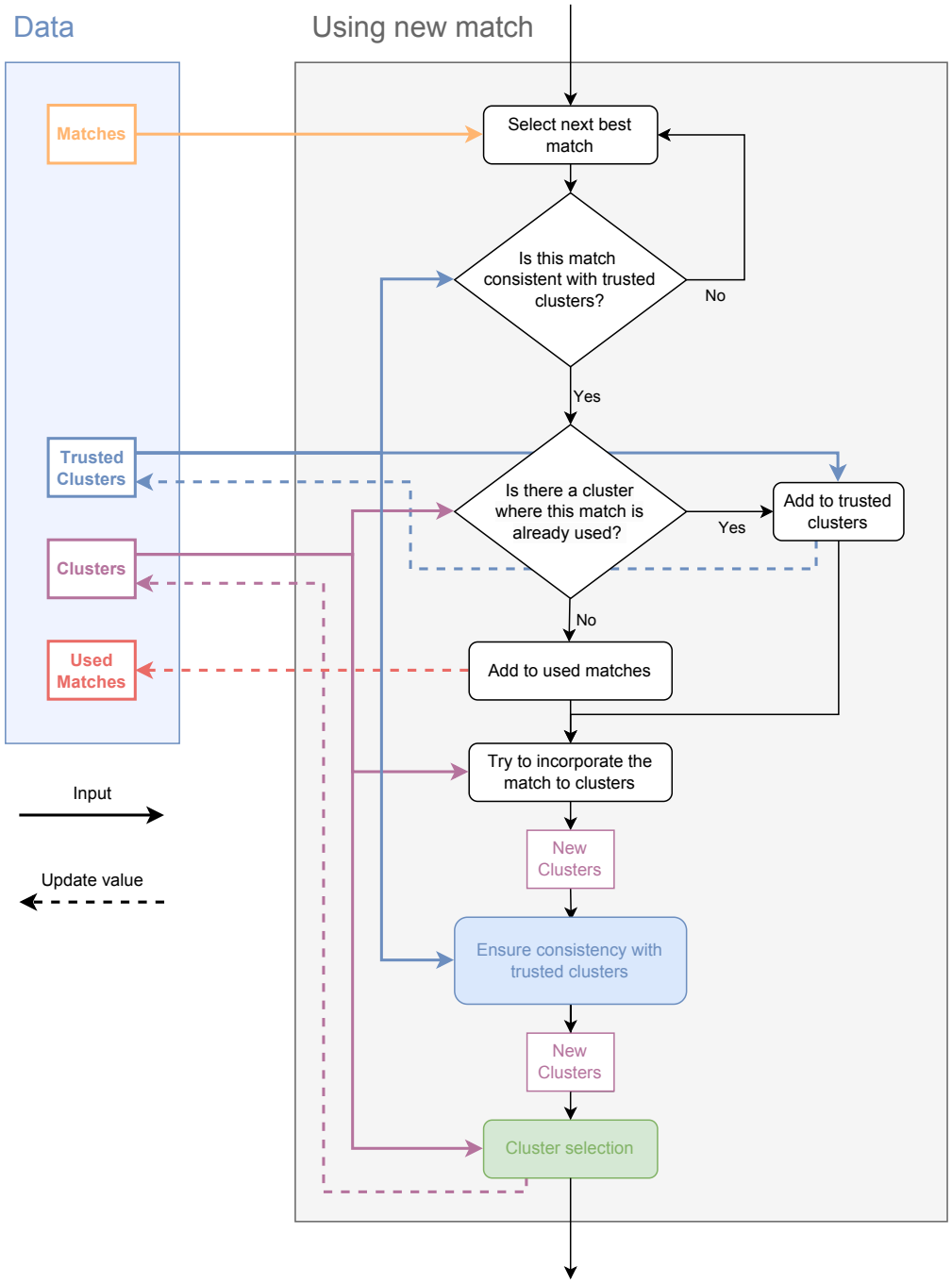


Figure 5.3 Incorporation of a new match.

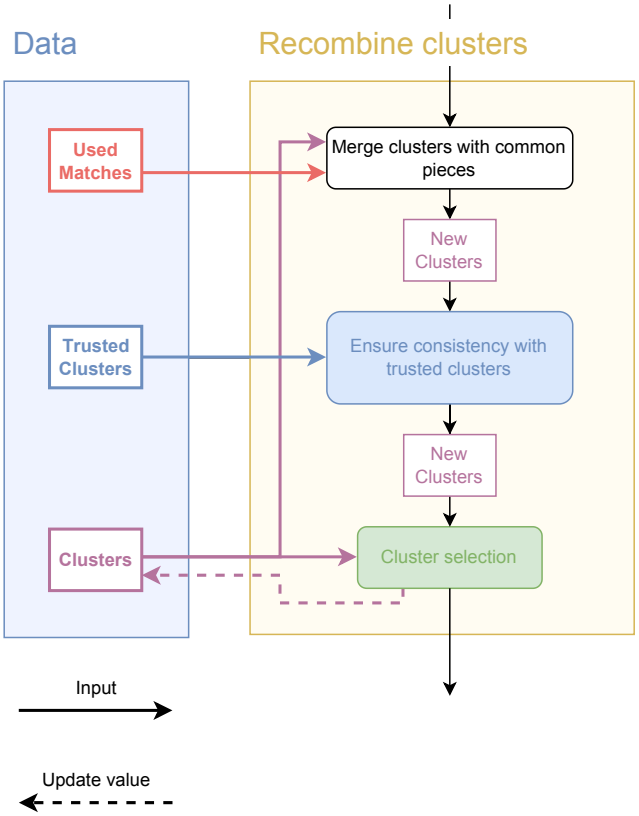
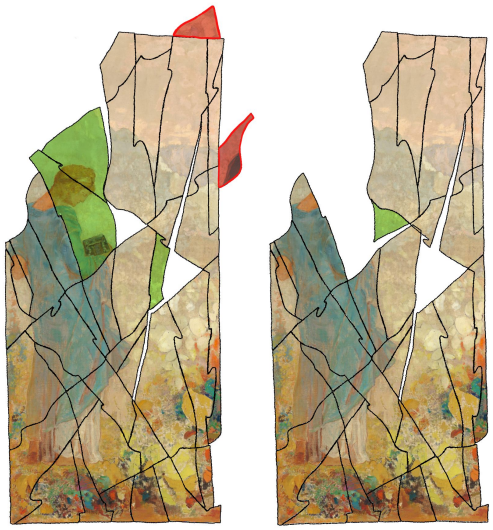


Figure 5.4 Recombination of clusters.



(a) (b)

Figure 5.5 Example of two output clusters. The clusters share many common pieces but could not be merged because cumulative inaccuracies would cause some pieces to overlap. The differing pieces are highlighted in green (correct) and red (incorrect).

avoid redundant computation, these matches are merged only with larger clusters from the previous stage, rather than with each other.

After trying to merge all clusters with common pieces, we once again ensure consistency with trusted clusters and select the best clusters as described in Section 5.3.

5.6 Output

The algorithm produces a set of clusters $\mathcal{C}$, where each cluster represents a partial assembly of the image, consisting of a set of pieces and their respective transformations. In the ideal case, where the entire image has been reconstructed, the algorithm outputs a single cluster that contains all pieces.

Otherwise, multiple clusters are produced. As described in Section 5.3, multiple clusters may share common pieces. An example of such a case is shown in Figure 5.5, where the two clusters are nearly identical, but could not be merged due to cumulative inaccuracies in the assembly process. However, since both clusters contain valuable information about the assembly, we choose to output all of them, even if it results in some redundancy.

Chapter 6

Comparison with PairingNet

Our work builds on the approach introduced in [2], which is denoted as PairingNet. Although we adopt its core idea, we introduce several significant modifications. In this chapter, we outline the key differences between our method and PairingNet and present a comparative evaluation.

6.1 Key Differences

The main concept of PairingNet involves extracting the contour of each piece, then obtaining a 7×7 patch around each contour point, which is processed by a neural network to generate embedding vectors. These embeddings are then used to establish correspondences between contour points of two pieces. While PairingNet addresses both pair-matching (where two pieces are given as input and the output is either a transformation or no match) and pair-searching (where a set of images is provided and the model predicts the probability of a match for each pair), we focus primarily on the pair-matching problem, referred to in this work as local assembly.

The most significant difference in our approach is the use of a 1D convolutional network to generate embeddings instead of a graph convolutional network. Additionally, PairingNet utilizes two types of input features, contour information and texture information, training separate networks for each and later fusing the embeddings. In contrast, we use only texture information, as it implicitly contains contour features, significantly simplifying the architecture.

Both approaches construct a similarity matrix by multiplying the embeddings of two pieces. However, the post-processing and loss function differ. We formulate the task as a set of binary classification problems and use Binary Cross Entropy (BCE) as the loss function. In contrast, PairingNet applies a softmax function in both directions, multiplies the results, and employs the following loss function:

$$\mathcal{L}^{ij} = \sum_{k,l} \left(\beta_1 (1 - \mathbf{M}_{kl}^{ij})^{\gamma} \log \mathbf{M}_{kl}^{ij} \mathbf{G}_{kl}^{ij} + \beta_2 \left(\mathbf{M}_{kl}^{ij} \right)^{\gamma} \log \left(1 - \mathbf{M}_{kl}^{ij} \right) \left(1 - \mathbf{G}_{kl}^{ij} \right) \right) \tag{6.1}$$

where β_1, β_2, and γ are hyperparameters such that $\beta_1 + \beta_2 = 1$, $\mathbf{M}_{kl}^{ij}$ is the predicted similarity matrix and $\mathbf{G}_{kl}^{ij}$ is the ground truth similarity matrix. We opted for a binary classification approach to avoid the need for hyperparameter tuning while providing a more intuitive and simplified framework.

Another key difference lies in transformation reconstruction. PairingNet enhances the similarity matrix by applying dilation with a diagonal kernel to reinforce diagonal patterns before directly using RANSAC for transformation estimation. In contrast, we apply a Hough transform to detect diagonal lines, extracting only the points along the detected line for RANSAC. Additionally, we refine the estimated transformation using the Iterative Closest Point (ICP) algorithm to further improve accuracy.

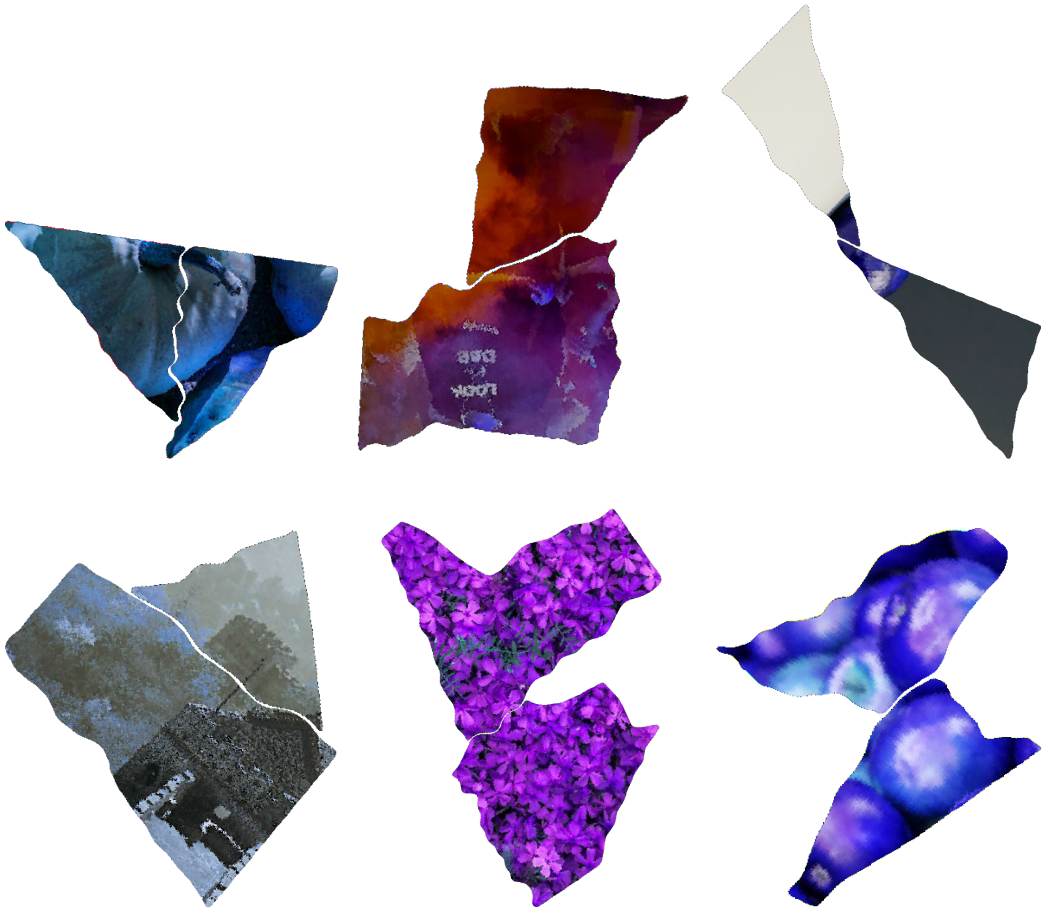
6.2 Dataset Comparison

Similarly to us, [2] designed their own algorithm for fragmenting images into pieces and used it to generate the dataset. Their approach to the fragmentation process is top-down, with the image being recursively divided into smaller and smaller pieces. Fragments are split either by straight lines or by a curve simulating hand-torn paper. In contrast, our approach is more bottom-up, with the image being initially fragmented into many small pieces, which are then iteratively merged together until the desired number of pieces is reached.

Both of these approaches are comparable in terms of the diversity of the shape of the generated pieces. For the purpose of local assembly, these two approaches are probably equivalent. However, for the global assembly, we believe that our approach generates more diverse patterns, as some patterns, like crossings of two lines, are impossible or unlikely to achieve by iteratively dividing the image.

Their dataset consists of 8,196 pieces and 14,951 matching pairs. The number of pieces in the training, validation, and test sets is detailed in Table 6.2a. In addition, in the test set, they categorized the matches into three difficulty levels based on the length of the shared border. In contrast, our dataset contains 21,877 pieces and 41,861 matches for training. The number of pieces in the training, validation, and test sets is detailed in Table 6.2b.

It is important to highlight that, in PairingNet's dataset, pieces originating from the same image can appear in different splits. This results in visually similar pieces being present in both the training and test sets. In contrast, we split



(a) Low difficulty. **(b)** Medium difficulty. **(c)** High difficulty.

Figure 6.1 Example of matching pairs of pieces from the PairingNet dataset [2] and their difficulty.

the original images into distinct training, validation, and test sets before fragmenting them into pieces, ensuring that pieces in the training and test sets are as different as possible.

6.3 Comparative Evaluation

In this section, we compare the performance of our model with that of PairingNet. First, we evaluate our model, trained on our own dataset, using the test split from the PairingNet dataset. We then compare the recall of the correctly matched pairs with the results reported by the authors in [2]. Additionally, we train our model on the PairingNet training set and evaluate it on both our test set and

	Total	Train	Validation	Test			
				Full	High	Medium	Low
Fragments	8,196	4,098	819	3,279	1,071	1,671	1,090
Pairs	14,951	3,654	137	2,370	663	1,103	604

(a) PairingNet dataset

	Total	Train	Validation	Test
Fragments	27,866	21,877	4,190	1,799
Pairs	53,577	41,861	8,130	3,586

(b) Our synthetic dataset

Figure 6.2 Comparison of dataset sizes in terms of number of fragments and matching pairs: (a) the PairingNet dataset, and (b) our synthetic dataset.

the PairingNet test set to assess its generalization across datasets.

6.3.1 Evaluating Our Model on the PairingNet Dataset

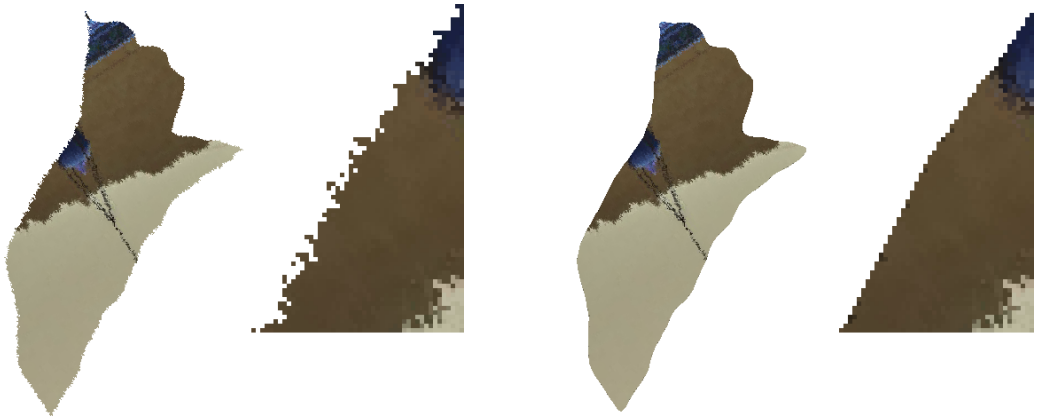
As an initial experiment, we tested our best-performing model trained on our synthetic data on the PairingNet dataset. Our first results were suboptimal, with a total recall of only 0.684, significantly lower than their reported recall of 0.835.

Upon investigating their data and source code, we discovered that their pieces were rotated using nearest-neighbor interpolation. This method resulted in low image quality and introduced artifacts (as shown in Figure 6.3a), which negatively impacted the performance of our model. Our model was not trained on such degraded images, and these artifacts are also uncommon in real-world scenarios, where digitalized fragments are typically preprocessed to obtain smooth contours.

To address this issue, we applied preprocessing to improve the quality of PairingNet's data, making it more comparable to both our dataset and real-world examples. Specifically, we extracted the binary mask of each piece, applied Gaussian blurring, and then re-thresholded it to obtain a refined binary mask. The image content remained largely unchanged, with newly introduced gaps filled by averaging values from neighboring pixels. However, this process resulted in some loss of sharp edge details, as can be seen in Figure 6.3b.

After preprocessing, our model achieved results comparable to those reported in the original PairingNet paper. In particular, our performance was nearly identical on low- and medium-difficulty samples but remained lower for high-difficulty cases. The results are shown in Table 6.1.

For this evaluation, we used the model trained on the variant of eroded data.



(a) Original image, full piece and detail.

(b) Preprocessed image, full piece and detail.

Figure 6.3 Preprocessing effects on the PairingNet dataset.

6.3.2 Training Our Model on the PairingNet Dataset

To further compare the approaches, we trained our model using the PairingNet dataset without preprocessing. However, a difference emerged in data processing.

In our pipeline, contour points are extracted directly from the piece masks. In contrast, PairingNet provides precomputed contour points, which did not match those we obtained from their masks. Analyzing their source code revealed that their contour points were extracted from the original, unrotated images and later rotated using the same transformation applied to the piece. Due to the use of nearest-neighbor interpolation, this process significantly altered the contours.

In their paper, [2] reported using an embedding dimension of $D_E = 64$. Our experiments with our dataset showed the best results with $D_E = 128$. For a fair comparison, we trained the models using both configurations: $D_E = 64$ and $D_E = 128$.

As in the previous experiment, we achieved similar results on low- and medium-difficulty samples but performed worse on high-difficulty cases. Detailed results are shown in Table 6.1.

The evaluation results indicate that the model trained on the PairingNet dataset is likely overfitted to its training data, as it performs significantly worse on our test set. Specifically, it achieves an overall recall of only 0.389 on eroded data and 0.463 on non-augmented data, compared to 0.758 and 0.850 respectively for the model trained on our dataset. When evaluated on the PairingNet test set, both models perform comparably, suggesting that our training data offer better generalization across datasets. The complete results are provided in the appendix C.

Model	Training Dataset	Test Pre-processing	D_E	Recall			
				All	Low	Medium	High
PairingNet	PairingNet	-	64	0.835	0.961	0.879	0.606
Ours	Ours	No	128	0.684	0.894	0.736	0.407
		Yes		0.779	0.964	0.878	0.445
Ours	PairingNet	No	64	0.809	0.967	0.883	0.541
			128	0.796	0.952	0.864	0.539

Table 6.1 Comparison of results on the PairingNet dataset for the pair-matching task.

Chapter 7

Results

In this chapter, we present the evaluation results of our method. We provide both quantitative results on our test set and qualitative results on out-of-distribution images, including real-world examples.

7.1 Quantitative Results

The algorithm produces a set of clusters, $\mathcal{C}$, which may share common pieces. To ensure that each piece and match is counted only once when computing the metrics, we extract a set of disjoint clusters, $\mathcal{C}_{\text{corr}}$. Each cluster in $\mathcal{C}_{\text{corr}}$ is a subset of some clusters in $\mathcal{C}$ and is correctly assembled, meaning the relative transformations between the pieces are within an acceptable tolerance. In our evaluations, we use a rotation tolerance of $\theta_{\text{tol}} = 0.17\text{rad} \approx 10^\circ$ and a translation tolerance of $t_{\text{tol}} = 25$ px. For reference, the original images in our experiments have a maximum dimension of 1600 px, and the average fragment size scales inversely with the number of pieces—for 100-piece puzzles, the smallest fragments are typically around 200×200 px.

7.1.1 Metrics

We evaluate the performance of our method using the following metrics:

- **Largest Connected Component (E_{LCC})**: The size of the largest correctly assembled cluster relative to the total number of pieces:

$$E_{\text{LCC}} = \frac{\max_{C \in \mathcal{C}_{\text{corr}}} |C|}{N}. \tag{7.1}$$

- **Number of Connected Components (E_{NCC})**: The total count of correctly assembled clusters:

$$E_{\text{NCC}} = |\mathcal{C}_{\text{corr}}|. \tag{7.2}$$

- **Rotation Error (E_θ)**: The average rotation error of correctly assembled pieces, measured in radians. Using the ground truth set of neighboring piece pairs, $\mathcal{N}_{\text{gt}}$, we define the set of neighbors for each cluster C:

$$\mathcal{N}_C = \{(i,j) \mid (i,j) \in \mathcal{N}_{\text{gt}},\ i \in C,\ j \in C\}. \tag{7.3}$$

  The rotation error is then computed as:

$$E_\theta = \frac{\sum_{C \in \mathcal{C}_{\text{corr}}} \sum_{(i,j) \in \mathcal{N}_C} |\theta_{ij}^{\text{gt}} - \theta_{ij}^{C}|}{\sum_{C \in \mathcal{C}_{\text{corr}}} |\mathcal{N}_C|}, \tag{7.4}$$

  where θ_{ij}^{gt} is the true angle between pieces i and j, and θ_{ij}^{C} is the estimated angle in cluster C.

- **Translation Error (E_t)**: The average translation error of correctly assembled pieces, measured in pixels. Defined analogously to the rotation error:

$$E_t = \frac{\sum_{C \in \mathcal{C}_{\text{corr}}} \sum_{(i,j) \in \mathcal{N}_C} \|t_{ij}^{\text{gt}} - t_{ij}^{C}\|}{\sum_{C \in \mathcal{C}_{\text{corr}}} |\mathcal{N}_C|}, \tag{7.5}$$

  where t_{ij}^{gt} and t_{ij}^{C} are the ground truth and estimated translation vectors between pieces i and j.

- **Registration Error (E_{reg})**: A combined metric that normalizes rotation and translation errors by their respective tolerances:

$$E_{\text{reg}} = \frac{1}{2}\left(\frac{E_\theta}{\theta_{\text{tol}}} + \frac{E_t}{t_{\text{tol}}}\right). \tag{7.6}$$

7.1.2 Evaluation Data

For evaluation, we use our synthetic data. We have two test datasets consisting of the same images divided into the same pieces, and the only difference is that to one of them was applied simulated erosion to simulate damaged pieces. The original images have different shapes and sizes, but none of them has larger size above 1600px. Each image is fragmented into 10, 30, 50 or 100 pieces, the counts are shown in Table 7.1.

Pieces per image	Number of images
10	30
30	10
50	10
100	5
Total	55

Table 7.1 Composition of synthetic test datasets.

The results for a model trained on eroded data are shown in Table 7.3 and the results for a model trained on non-eroded data are shown in Table 7.4.

We also evaluate our method on the JigsawNet dataset (Section 3.2.1). The composition of this dataset is shown in Table 7.2 and we report the results in Table 7.5.

We did not evaluate our method on the 400-piece JigsawNet dataset due to performance limitation. The large number of pieces led to excessive processing time, making it impractical for our current evaluation. This limitation is tied to the global assembly process and could be addressed in future research through optimization techniques to improve efficiency for larger datasets.

For both of our models, only one of the images from the JigsawNet dataset was not completely assembled, and it was the example with 9 pieces (Figure 7.1).

Figure 7.1 The only incorrectly assembled image from the JigsawNet dataset using our method.

Pieces per image	Number of images
9	20
36	6
100	6
400	5
Total	37

Table 7.2 Composition of JigsawNet test dataset.

Train Dataset	# Pieces	E_θ	E_t	E_{reg}	E_{LCC}	E_{NCC}
Eroded	10	0.0018	1.7217	0.0399	0.9522	1.47
	30	0.0044	2.4723	0.0623	0.9500	2.00
	50	0.0108	4.3382	0.1186	0.9800	1.90
	100	0.0144	4.4985	0.1322	0.9660	4.20
	All	**0.0051**	**2.5863**	**0.0667**	**0.9581**	**1.89**
Non-Augmented	10	0.0009	1.5257	0.0280	0.9622	1.37
	30	0.0025	2.0379	0.0352	0.9900	1.30
	50	0.0023	1.6697	0.0335	0.9900	1.50
	100	0.0043	2.0358	0.0410	0.9860	2.00
	All	**0.0017**	**1.6914**	**0.0315**	**0.9745**	**1.44**
	All	**0.0034**	**2.1389**	**0.0421**	**0.9663**	**1.66**

Table 7.3 Results for our synthetic test datasets and model trained on dataset with simulated erosion.

Train Dataset	# Pieces	E_θ	E_t	E_{reg}	E_{LCC}	E_{NCC}
Eroded	10	0.0018	1.7404	0.0344	0.9148	1.80
	30	0.0048	2.5816	0.0647	0.8833	3.20
	50	0.0100	3.8897	0.0920	0.8920	5.30
	100	0.0127	3.5292	0.0981	0.7100	12.00
	All	**0.0048**	**2.4467**	**0.0562**	**0.8863**	**3.62**
Non-Augmented	10	0.0009	1.5489	0.0284	0.9896	1.10
	30	0.0016	1.6343	0.0319	0.9967	1.10
	50	0.0022	1.5146	0.0316	0.9960	1.20
	100	0.0030	1.5218	0.0393	0.9960	1.40
	All	**0.0014**	**1.5557**	**0.0306**	**0.9926**	**1.15**
	All	**0.0031**	**2.0012**	**0.0434**	**0.9395**	**2.38**

Table 7.4 Results for our synthetic test datasets and model trained on non-augmented data.

Train Dataset	# Pieces	E_θ	E_t	E_{reg}	E_{LCC}	E_{NCC}
Eroded	9	0.0010	1.9167	0.0349	0.9944	1.05
	36	0.0007	1.9066	0.0339	1.0000	1.00
	100	0.0016	1.8682	0.0358	1.0000	1.00
	All	**0.0011**	**1.9057**	**0.0349**	**0.9965**	**1.03**
Non-Augmented	9	0.0010	1.8957	0.0410	0.9944	1.05
	36	0.0007	1.8980	0.0402	1.0000	1.00
	100	0.0016	1.9166	0.0430	1.0000	1.00
	All	**0.0011**	**1.9000**	**0.0412**	**0.9965**	**1.03**

Table 7.5 Results for JigsawNet test dataset.

7.2 Qualitative Results

To gain deeper insights, we analyze results on out-of-distribution images. We consider two real-world examples: a digitized jigsaw puzzle with irregular pieces and a fragmented dry-plate negative capturing Cesare da Sesto's *Madonna with Saint John* (Figure 1).Additionally, we include synthetic examples not present in the training set to evaluate the model's reliance on color and shape information.

An overview of the most common errors made by our method is provided in Appendix B.

7.2.1 Digitalized Jigsaw Puzzle

We evaluated our method on jigsaw puzzle with 101 irregularly shaped pieces. Unlike standard puzzles, its overall shape is non-rectangular. One piece contained a hole designed to fit another smaller piece— a case our method does not handle— so we excluded this piece, using only 100 pieces.

The algorithm successfully assembled the puzzle with all 100 pieces correctly placed (Figure 7.2). The total assembly time was approximately 25 minutes.

7.2.2 Fragmented Negative

We applied our method to a fragmented negative, which was introduced as our main motivation in the Introduction (see Figure 1). The algorithm successfully assembled all but three pieces correctly (see Figure 7.3). One piece was intentionally omitted because it fits into another piece with a hole, a scenario that our method does not currently handle. The other two pieces were likely placed incorrectly because they are too small compared to the rest and lack distinctive features that would help the algorithm determine their correct placement.

7.2.3 Puzzles Without Image Information

To assess the model's reliance on shape versus color, we created synthetic puzzles without any image information.

For a 30-piece monochromatic puzzle, the algorithm correctly assembled 28 pieces, with one misaligned (top right corner) and one missing. The result is shown in Figure 7.4. For 100 pieces, the algorithm was able to find some correct clusters, but overall assembly was unsuccessful.

Figure 7.2 Assembly of digitized jigsaw puzzle with 100 pieces. One piece was manually omitted because it fits into another piece with a hole, a scenario that our method does not currently handle.

(a) Negative assembled by our method. **(b)** Missing pieces

Figure 7.3 Reassembly of the fragmented dry-plate negative of Cesare da Sesto's *Madonna with Saint John*. Eleven of fourteen fragments were correctly aligned; three remain unresolved.

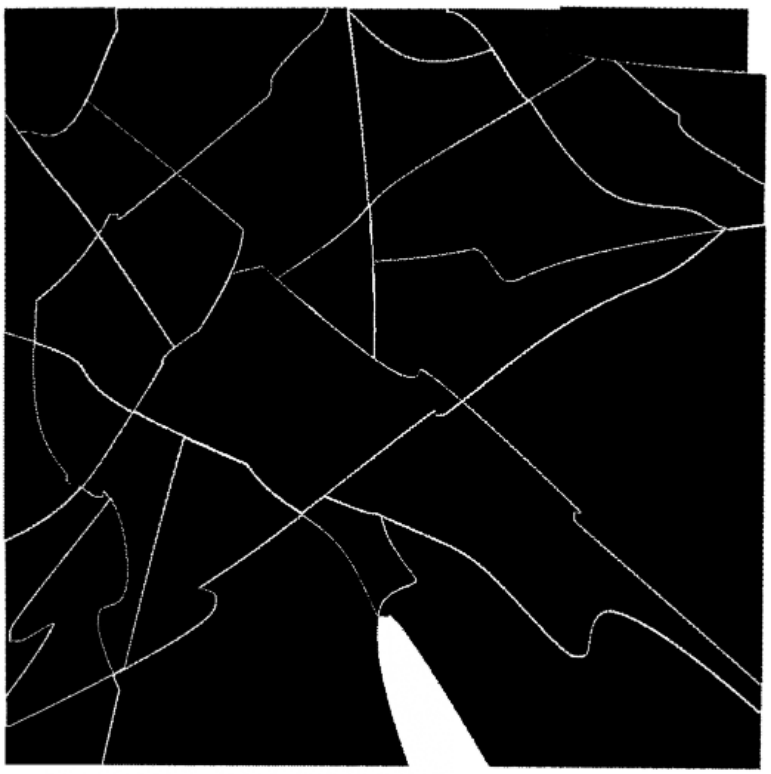


Figure 7.4 Assembly of monochromatic puzzle with 30 pieces. The piece at the top right corner is slightly shifted to the left and one piece at the bottom is missing, otherwise the assembly was successful.

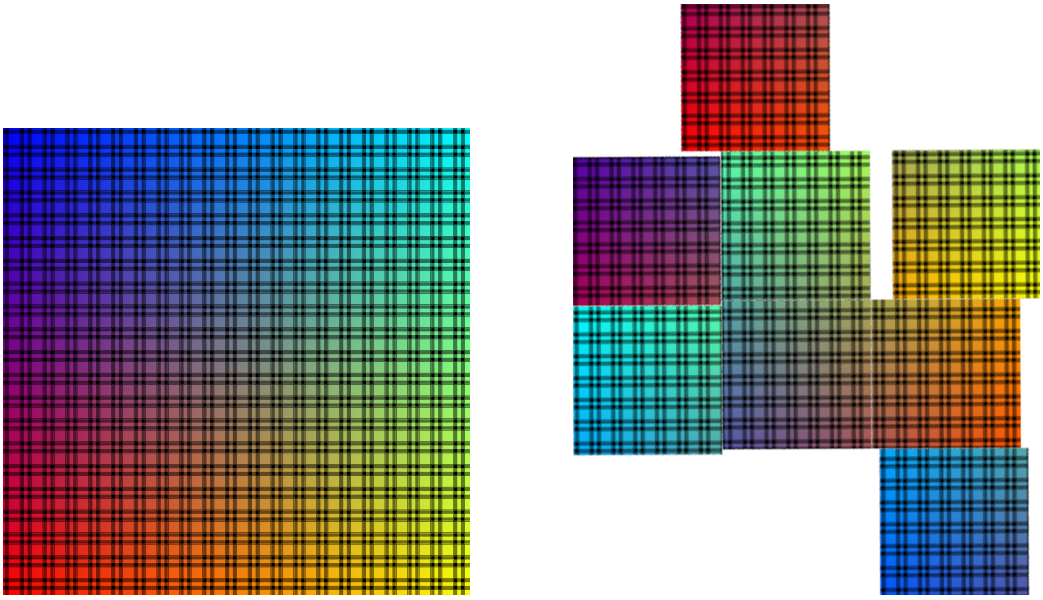


Figure 7.5 An image with a color gradient overlaid with a regular texture, fragmented into 9 square pieces. The left image shows the original, while the right image displays the assembly produced by our method, which is almost entirely incorrect. This highlights that our model does not rely heavily on color information for reconstruction.

7.2.4 Puzzles with Rectangular Pieces

Many studies focus on puzzles with regular, rectangular pieces, making it relevant to evaluate our method in this context. With all pieces having the same shape and size, the method can use only the color information to assemble the puzzle.

We tested a color gradient puzzle, where the model failed to find any matches, resulting in an unsuccessful assembly. When a simple repeating texture was overlaid, the model detected matches, but most were incorrect. The result is shown in Figure 7.5.

For puzzles composed of irregular textures:

- A 3×3 grid (9 pieces) was fully assembled (Figure 7.6).
- A 4×4 grid (16 pieces) had 12 correctly placed pieces.
- A 5×5 grid (25 pieces) had 19 correctly placed pieces.

We also tested artwork images divided into rectangular pieces. In most cases, the model failed to assemble even small puzzles (e.g., 9 pieces). One such example is shown in Figure 7.7, where the algorithm correctly identified two clusters but failed to merge them.

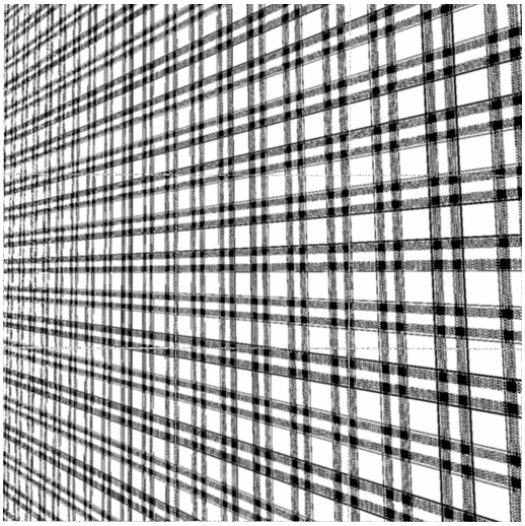


Figure 7.6 Texture image, fragmented into 9 square pieces and correctly assembled by our method.

Figure 7.7 An artwork fragmented into 9 rectangular pieces and assembled using our method. The algorithm successfully formed two clusters—one containing 7 pieces and the other 2—but was unable to merge them into a complete assembly.

7.2.5 General Findings

The experiments with apictorial and rectangular puzzle pieces suggest that the model prioritizes shape and change in intensity over absolute color. One possible reason is that it was primarily trained on fragmented rectangular artworks, reinforcing the assumption that straight edges (image borders) do not match, even if they share the same color.

These findings also suggests that a simplified input representation for the model may be sufficient. Instead of three color channels, a single-channel grayscale or edge-based representation could be viable alternatives. This could be subject to further research.

Chapter 8

Conclusion

In this thesis, we successfully addressed the challenging problem of artwork reconstruction, particularly focusing on real-world fragmentation patterns that are often encountered in historical and artwork-based puzzles. One of the key milestones was the reconstruction of a fragmented dry-plate negative, a complex example that was solved effectively using our methodology. This achievement was made possible through the creation of a synthetic dataset tailored to replicate realistic fragmentation scenarios, which was instrumental in training and evaluating the model. The diversity of our dataset in image content, including various types of artwork such as drawings, paintings, and photographs, ensured the robustness and generalizability of our approach. The model's performance on this custom dataset was critical in testing its potential for practical applications in artwork restoration and puzzle reconstruction.

Our results on one of the publicly available datasets, the JigsawNet dataset [21], were highly promising, with near-perfect performance achieved on puzzles containing up to 100 pieces. The model demonstrated exceptional accuracy, with only a single error in all examples. However, a limitation of the global assembly method became evident during this evaluation, as the model struggled to efficiently handle the 400-piece puzzles. Although it performed well on smaller puzzles, the size of larger puzzles posed challenges, revealing areas where the current approach can be improved for scalability.

Furthermore, we performed a comparative analysis with PairingNet, a fragment assembly method introduced in [2]. Our approach achieved comparable results, particularly on low- and medium-complexity puzzles, even when trained solely on our synthetic dataset. We also trained our model using the PairingNet training set and observed similar performance on its corresponding test set. However, when evaluated with our own test data, the model trained on the PairingNet dataset performed significantly worse, suggesting that our dataset offers better generalization to various fragmentation scenarios.

These outcomes highlight the effectiveness of our global and local assembly strategies in solving complex artwork reconstruction problems. Our approach demonstrated progress in both the accuracy and efficiency of reconstructing fragmented artwork, providing a reliable solution to a long-standing problem in the field.

8.1 Contribution

The contributions of this thesis can be summarized as follows:

- **Synthetic Dataset Creation:** The development of a synthetic dataset tailored for artwork reconstruction, containing diverse content and designed to mimic real-world fragmentation patterns.
- **Local Assembly Method:** The proposal of a novel local assembly method that uses a 1D convolutional neural network (CNN) with a U-Net-like architecture to identify pixel-level correspondences along the contours of the puzzle piece. This method enhances piece-level matching by computing similarity and aligning transformations using RANSAC and ICP.
- **Confidence Scoring Mechanism:** Introduction of a confidence scoring approach that quantifies the reliability of piece matches based on the similarity matrix and match length, optimizing the model's ability to prioritize high-confidence matches during the assembly process.
- **Global Assembly Algorithm:** The proposal of a global assembly method that reconstructs images by merging puzzle piece matches into clusters, resolving inconsistencies during the merging process without backtracking. The algorithm ranks candidate matches by confidence score, detects cycles in the match graph to identify trusted clusters, and selects the highest-scoring clusters to efficiently assemble larger puzzles.
- **Comprehensive Evaluation:** Evaluation of the proposed method on both synthetic and real-world data, including two publicly available benchmarks: JigsawNet [21] and PairingNet [2] datasets. Additional experiments were conducted to assess the model's performance using only shape (monochromatic puzzles) or only color (grid-like rectangular puzzles) information.

These contributions aim to advance the state of artwork reconstruction, particularly in handling real-world fragmentation.

8.2 Future Work

Although the approach presented in this thesis shows promising results, there are several avenues for future work that could further enhance the performance and applicability of the method.

8.2.1 Feature Extraction

Currently, the model takes as input the features extracted from image patches along the piece contours, using all three RGB channels. However, experiments with puzzles consisting solely of rectangular pieces suggest that the model does not heavily rely on color information but instead focuses on intensity changes, such as edges. This observation raises the possibility that providing all three color channels might be excessive. Further experiments could explore alternative image representations, such as grayscale images or just edge representations.

Another possibility for future work is to explore the use of a neural network to extract features for each contour point from the image, rather than relying on explicit image patches. The advantage of this approach is that the network would have access to the entire image, allowing it to capture global contextual information that may enhance feature representation and improve matching accuracy.

8.2.2 Global Assembly for Large Puzzles

Our experiments revealed that the global assembly solution does not scale efficiently. Although it can successfully assemble puzzles with up to 100 pieces in a few minutes or up to an hour in the worst-case scenarios, attempting to assemble 400-piece puzzles from the JigsawNet dataset can take several days. This remains the primary limitation of our method. Future work should focus on developing more efficient global assembly strategies or optimizing the implementation to handle larger puzzles more quickly.

8.2.3 Match Reconstruction

Currently, the match reconstruction process is based on traditional computer vision techniques such as the Hough transform, RANSAC, and ICP. Future work could explore the use of deep learning approaches to directly estimate the transformation between pieces.

Bibliography

[1] Tom Geller. "DARPA Shredder challenge solved". In: *Communications of The ACM - CACM* 55 (Aug. 2012). DOI: 10.1145/2240236.2240242.

[2] Rixin Zhou et al. *PairingNet: A Learning-based Pair-searching and -matching Network for Image Fragments*. 2023. arXiv: 2312.08704 [cs.CV]. URL: https://arxiv.org/abs/2312.08704.

[3] Liangjia Zhu, Zongtan Zhou, and Dewen Hu. "Globally Consistent Reconstruction of Ripped-Up Documents". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.1 (2008), pp. 1–13. DOI: 10.1109/TPAMI.2007.1163.

[4] H.C. da Gama Leitao and J. Stolfi. "A multiscale method for the reassembly of two-dimensional fragmented objects". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.9 (2002), pp. 1239–1251. DOI: 10.1109/TPAMI.2002.1033215.

[5] Lalitha K.S. et al. "Graph-Based Clustering for Apictorial Jigsaw Puzzles of Hand Shredded Content-less Pages". In: *Intelligent Human Computer Interaction*. Ed. by Anupam Basu et al. Cham: Springer International Publishing, 2017, pp. 135–147. ISBN: 978-3-319-52503-7.

[6] Anke Stieber et al. "A Contour Matching Algorithm to Reconstruct Ruptured Documents". In: *Pattern Recognition*. Ed. by Michael Goesele et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 121–130. ISBN: 978-3-642-15986-2.

[7] Andre Pimenta et al. "Document reconstruction using dynamic programming". In: *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2009, pp. 1393–1396. DOI: 10.1109/ICASSP.2009.4959853.

[8] Dror Sholomon, Eli David, and Nathan S. Netanyahu. *A Generalized Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles of Complex Types*. 2017. arXiv: 1711.06768 [cs.CV]. URL: https://arxiv.org/abs/1711.06768.

[9] Efthymia Tsamoura and Ioannis Pitas. "Automatic Color Based Reassembly of Fragmented Images and Paintings". In: *IEEE Transactions on Image Processing* 19 (2010), pp. 680–690. URL: https://api.semanticscholar.org/CorpusID:10272386.

[10] Davit Gigilashvili et al. "Computational techniques for virtual reconstruction of fragmented archaeological textiles". In: *Heritage Science* 11.1 (2023), p. 259. DOI: 10.1186/s40494-023-01102-3. URL: https://doi.org/10.1186/s40494-023-01102-3.

[11] Daniel Rika et al. "A novel hybrid scheme using genetic algorithms and deep learning for the reconstruction of portuguese tile panels". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. ACM, July 2019. DOI: 10.1145/3321707.3321821. URL: http://dx.doi.org/10.1145/3321707.3321821.

[12] Marie-Morgane Paumard, David Picard, and Hedi Tabia. *Jigsaw Puzzle Solving Using Local Feature Co-Occurrences in Deep Neural Networks*. 2018. arXiv: 1807.03155 [cs.CV]. URL: https://arxiv.org/abs/1807.03155.

[13] Marie-Morgane Paumard, David Picard, and Hedi Tabia. "Deepzzle: Solving Visual Jigsaw Puzzles With Deep Learning and Shortest Path Optimization". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 3569–3581. DOI: 10.1109/TIP.2019.2963378.

[14] Ru Li et al. "JigsawGAN: Self-supervised Learning for Solving Jigsaw Puzzles with Generative Adversarial Networks". In: *CoRR* abs/2101.07555 (2021). arXiv: 2101.07555. URL: https://arxiv.org/abs/2101.07555.

[15] Dov Bridger, Dov Danon, and Ayellet Tal. *Solving Jigsaw Puzzles with Eroded Boundaries*. 2019. arXiv: 1912.00755 [eess.IV]. URL: https://arxiv.org/abs/1912.00755.

[16] Dror Sholomon, Eli David, and Nathan Netanyahu. "DNN-Buddies: A Deep Neural Network-Based Estimation Metric for the Jigsaw Puzzle Problem". In: Sept. 2016, pp. 170–178. ISBN: 978-3-319-44780-3. DOI: 10.1007/978-3-319-44781-0_21.

[17] Marie-Morgane Paumard, David Picard, and Hedi Tabia. "Image Reassembly Combining Deep Learning and Shortest Path Problem". In: *CoRR* abs/1809.00898 (2018). arXiv: 1809.00898. URL: http://arxiv.org/abs/1809.00898.

[18] Marie-Morgane Paumard, Hedi Tabia, and David Picard. *Alphazzle: Jigsaw Puzzle Solver with Deep Monte-Carlo Tree Search*. 2023. arXiv: 2302.00384 [cs.CV]. URL: https://arxiv.org/abs/2302.00384.

[19] Xingke Song et al. “Siamese-Discriminant Deep Reinforcement Learning for Solving Jigsaw Puzzles with Large Eroded Gaps”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.2 (2023), pp. 2303–2311. DOI: 10.1609/aaai.v37i2.25325. URL: https://ojs.aaai.org/index.php/AAAI/article/view/25325.

[20] Thiago M. Paixão et al. *Self-supervised Deep Reconstruction of Mixed Strip-shredded Text Documents*. 2020. arXiv: 2007.00779 [cs.CV]. URL: https://arxiv.org/abs/2007.00779.

[21] Canyu Le and Xin Li. “JigsawNet: Shredded Image Reassembly using Convolutional Neural Network and Loop-based Composition”. In: *arXiv preprint arXiv:1809.04137* (2018).

[22] Yangjie Cao et al. “2D Irregular Fragment Reassembly With Deep Learning Assistance”. In: *IEEE Access* 12 (2024), pp. 28554–28563. DOI: 10.1109/ACCESS.2024.3368004.

[23] Sepidehsadat (Sepid) Hossieni et al. “Puzzlefusion: Unleashing the Power of Diffusion Models for Spatial Puzzle Solving”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 9574–9597. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/1e70ac91ad26ba5b24cf11b12a1f90fe-Paper-Conference.pdf.

[24] Elena Sizikova and Thomas Funkhouser. “Wall Painting Reconstruction Using a Genetic Algorithm”. In: *J. Comput. Cult. Herit.* 11.1 (Dec. 2017). ISSN: 1556-4673. DOI: 10.1145/3084547. URL: https://doi.org/10.1145/3084547.

[25] Yongqing Liang and Xin Li. “Reassembling Shredded Document Stripes Using Word-path Metric and Greedy Composition Optimal Matching Solver”. In: *IEEE Transactions on Multimedia* (2019).

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[27] Martin A. Fischler and Robert C. Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[28] P.J. Besl and Neil D. McKay. “A method for registration of 3-D shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791.

[29] Zhengyou Zhang. “Iterative point matching for registration of free-form curves and surfaces”. In: *International journal of computer vision* 13.2 (1994), pp. 119–152.

[30] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

Appendix A

Class Imbalance - Evaluation Results

For the following experiments, we used a network with $D_E = 128$, non-augmented training data, a batch size of 16, and a training loss of 0.0001.

A.1 Weights

Weight of Pos. Samples	# Pieces	P	R	F_1	Acc
10	10	0.536	0.927	0.679	0.719
	30	0.364	0.857	0.511	0.784
	50	0.312	0.865	0.459	0.831
	100	0.229	0.841	0.359	0.863
	All	0.360	0.872	0.502	0.799
15	10	0.488	0.922	0.639	0.664
	30	0.289	0.869	0.433	0.700
	50	0.261	0.863	0.400	0.785
	100	0.195	0.849	0.316	0.832
	All	0.308	0.876	0.447	0.745
25	10	0.447	0.920	0.601	0.608
	30	0.231	0.852	0.363	0.607
	50	0.188	0.874	0.310	0.678
	100	0.121	0.861	0.212	0.712
	All	0.247	0.877	0.372	0.651

Table A.1 Results for different weights of positive samples. Weight w means that for the loss computation, positive samples are weighted by w, while the weight of negative samples is 1.

A.2 Masking

Ratio of Neg. Samples	# Pieces	P	R	F_1	Acc
1	10	0.353	0.869	0.502	0.446
	30	0.150	0.787	0.251	0.383
	50	0.096	0.799	0.172	0.364
	100	0.053	0.771	0.099	0.372
	All	0.163	0.806	0.256	0.391
2	10	0.364	0.891	0.517	0.465
	30	0.159	0.818	0.266	0.406
	50	0.106	0.827	0.188	0.407
	100	0.064	0.836	0.120	0.446
	All	0.173	0.843	0.273	0.431
4	10	0.394	0.911	0.550	0.521
	30	0.184	0.831	0.301	0.493
	50	0.122	0.840	0.214	0.488
	100	0.075	0.817	0.138	0.539
	All	0.194	0.850	0.301	0.510
8	10	0.394	0.907	0.549	0.521
	30	0.193	0.855	0.315	0.510
	50	0.136	0.844	0.235	0.544
	100	0.095	0.844	0.170	0.630
	All	0.204	0.863	0.317	0.551
16	10	0.408	0.912	0.564	0.547
	30	0.218	0.867	0.348	0.571
	50	0.159	0.872	0.269	0.609
	100	0.116	0.865	0.204	0.696
	All	0.225	0.879	0.346	0.606

Table A.2 Results for different ratios of negative samples. A ratio of n means that for the loss computation, n times more negative samples are used.

Appendix B

Common Errors

This appendix discusses common types of errors that can occur during the assembly process of fragmented objects.

B.1 Error Accumulation

In cases involving eroded fragments, pieces are sometimes not placed with perfect accuracy. As the assembly progresses, these small inaccuracies can accumulate, eventually preventing new pieces from being placed without causing significant overlaps—which the algorithm disallows. An example of such a situation is shown in Figure B.1.

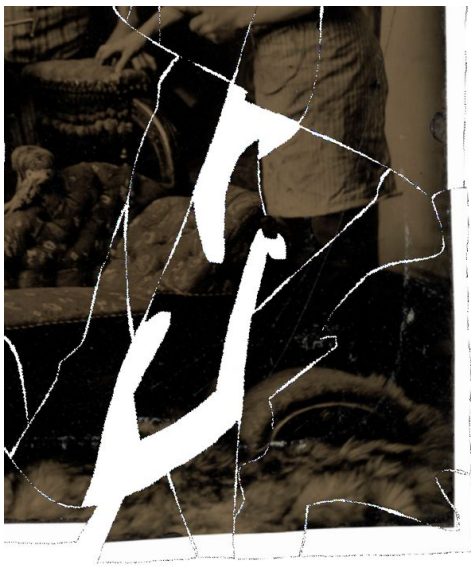


Figure B.1 Assembly failure due to accumulated placement errors (cropped).

In other instances, these cumulative errors result in noticeable gaps between pieces rather than overlaps, as illustrated in Figure B.2.

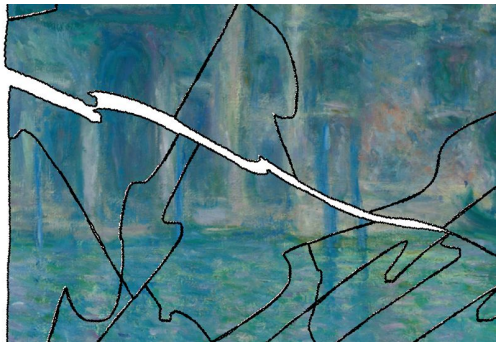


Figure B.2 Large gaps caused by accumulated placement inaccuracies (cropped).

B.2 Incorrectly Placed Borders

When there is insufficient visual information near the borders of an image, edge pieces may be incorrectly matched. Since such misplacements typically do not cause overlaps or inconsistencies, they often go uncorrected in later assembly stages. Examples of this type of error are shown in Figure B.3.

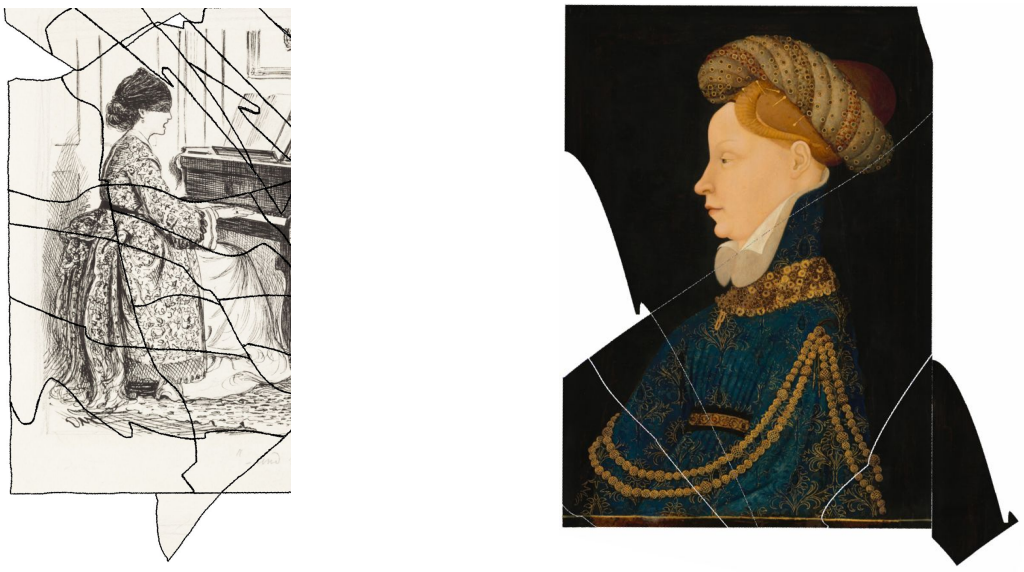


Figure B.3 Examples of incorrect matches along the image borders leading to unsuccessful assembly.

Appendix C

Comparison of Models Trained Using Our and PairingNet Datasets

Table C.1 presents a comparison of models trained on our dataset and the PairingNet dataset, evaluated on our test set.

Train Dataset	D_E	Test Dataset	# Pieces	P	R	F_1	Acc	E_{LCC}
Ours	128	Eroded	10	0.458	0.840	0.592	0.628	0.939
			30	0.239	0.737	0.361	0.656	0.967
			50	0.183	0.752	0.294	0.699	0.992
			100	0.120	0.704	0.204	0.752	0.983
			All	0.250	0.758	0.363	0.684	0.970
		Non-Augmented	10	0.465	0.890	0.610	0.635	0.978
			30	0.246	0.813	0.378	0.648	0.995
			50	0.190	0.821	0.308	0.692	0.997
			100	0.359	0.875	0.481	0.785	1.000
			All	0.315	0.850	0.445	0.690	0.992
PairingNet	128	Eroded	10	0.223	0.501	0.309	0.278	0.668
			30	0.069	0.384	0.118	0.241	0.437
			50	0.040	0.356	0.072	0.240	0.428
			100	0.017	0.279	0.031	0.227	0.282
			All	0.087	0.380	0.132	0.247	0.454
		Non-Augmented	10	0.231	0.534	0.323	0.280	0.678
			30	0.076	0.422	0.128	0.245	0.620
			50	0.047	0.419	0.084	0.243	0.596
			100	0.019	0.328	0.036	0.220	0.334
			All	0.093	0.426	0.143	0.247	0.557
PairingNet	64	Eroded	10	0.216	0.494	0.301	0.261	0.682
			30	0.073	0.417	0.125	0.230	0.663
			50	0.037	0.337	0.067	0.227	0.404
			100	0.018	0.306	0.034	0.216	0.298
			All	0.086	0.389	0.132	0.234	0.512
		Non-Augmented	10	0.245	0.582	0.345	0.290	0.729
			30	0.081	0.466	0.138	0.232	0.650
			50	0.048	0.445	0.087	0.228	0.662
			100	0.021	0.358	0.039	0.206	0.474
			All	0.099	0.463	0.152	0.239	0.629

Table C.1 Comparison of results on our test set for models trained on our and PairingNet [2] datasets.

Appendix D

Local Assembly Results

For these experiments, we used a network with $D_E = 128$, batch size 16 and training loss 0.0001. Ratio of negative samples in the training set was 0.1 and the weight of positive samples in loss computation was set on 10. The kernel size for convolutional layers was 3. The results are presented in Table D.1.

Train Dataset	Test Dataset	# Pieces	P	R	F_1	Acc	E_{FA}	E_{LCC}	E_θ	E_t	E_{reg}
Eroded	Eroded	10	0.4576	0.8399	0.5923	0.6283	0.6667	0.9387	0.0039	2.4405	0.0522
		30	0.2388	0.7373	0.3608	0.6559	0.5000	0.9667	0.0077	2.8623	0.0704
		50	0.1830	0.7520	0.2939	0.6987	0.6500	0.9920	0.0118	3.1975	0.0879
		100	0.1196	0.7044	0.2043	0.7524	0.4000	0.9830	0.0177	3.3993	0.1088
		All	0.2497	0.7584	0.3628	0.6838	0.5542	0.9701	0.0103	2.9749	0.0798
	Non-Augmented	10	0.4647	0.8898	0.6105	0.6350	0.7667	0.9780	0.0012	1.6818	0.0315
		30	0.2465	0.8133	0.3783	0.6478	0.7000	0.9950	0.0029	1.7974	0.0384
		50	0.1897	0.8210	0.3078	0.6922	0.8500	0.9970	0.0050	1.9927	0.0479
		100	0.3593	0.8750	0.4814	0.7852	1.0000	1.0000	0.0073	2.0079	0.0548
		All	0.3150	0.8498	0.4445	0.6900	0.8292	0.9925	0.0041	1.8699	0.0431
	JigsawNet	9	0.6761	1.0000	0.8067	0.8403	1.0000	1.0000	0.0019	2.1091	0.0406
		36	0.1912	0.9944	0.3208	0.5963	1.0000	1.0000	0.0014	2.0873	0.0390
		100	0.1300	0.9926	0.2298	0.7583	1.0000	1.0000	0.0026	2.1147	0.0429
		400	0.0304	0.9730	0.0590	0.6980	1.0000	1.0000	0.0027	2.1017	0.0429
		All	0.2569	0.9900	0.3541	0.7232	1.0000	1.0000	0.0021	2.1032	0.0414
	All		0.2739	0.8661	0.3871	0.6990	0.7944	0.9875	0.0055	2.3160	0.0548
Non-Augmented	Eroded	10	0.4103	0.7881	0.5395	0.5680	0.6889	0.9341	0.0035	2.4247	0.0506
		30	0.3051	0.7638	0.4351	0.7664	0.5000	0.9378	0.0070	2.6426	0.0646
		50	0.2495	0.6732	0.3641	0.8054	0.3333	0.9673	0.0109	3.0342	0.0826
		100	0.1752	0.6472	0.2747	0.8430	0.4000	0.9787	0.0168	3.2602	0.1039
		All	0.2851	0.7181	0.4034	0.7457	0.4806	0.9545	0.0096	2.8404	0.0754
	Non-Augmented	10	0.5361	0.9269	0.6793	0.7187	0.9111	0.9947	0.0013	1.5699	0.0299
		30	0.3644	0.8573	0.5114	0.7840	0.9250	0.9992	0.0028	1.6116	0.0350
		50	0.3124	0.8649	0.4588	0.8309	1.0000	1.0000	0.0049	1.8796	0.0458
		100	0.2289	0.8407	0.3589	0.8627	1.0000	1.0000	0.0067	1.8473	0.0505
		All	0.3604	0.8725	0.5021	0.7991	0.9590	0.9985	0.0039	1.7271	0.0403
	JigsawNet	9	0.7362	1.0000	0.8481	0.8806	1.0000	1.0000	0.0018	2.0663	0.0397
		36	0.2648	0.9944	0.4183	0.7348	1.0000	1.0000	0.0013	2.0000	0.0372
		100	0.2405	0.9981	0.3875	0.8854	1.0000	1.0000	0.0025	2.0071	0.0408
		400	0.0446	0.9798	0.0852	0.7953	1.0000	1.0000	0.0026	2.0284	0.0413
		All	0.3215	0.9931	0.4348	0.8240	1.0000	1.0000	0.0020	2.0254	0.0398
	All		0.3223	0.8612	0.4467	0.7896	0.8132	0.9843	0.0052	2.1977	0.0518

Table D.1 Local assembly performance across different training and testing datasets. Metrics are reported for varying numbers of puzzle pieces.