Reconstruction of a 1000-Piece Jigsaw Puzzle

Kgetja Bruce Mphekgwane (2593733), Ivy Chepkwony (2431951)

***Abstract*—This report presents the methodology and results for reconstructing a 1000-piece jigsaw puzzle. The pipeline focuses on three main components: segmentation, per-edge feature extraction, and adjacency graph construction. A U-Net model trained on 500 labeled masks achieves a validation IoU of 0.9721, and its predicted masks obtain a mean IoU of 0.9548 (Tutor labels) and 0.9180 (RK labels) on the external evaluation server, with corresponding mDICE scores above 0.95. Using these masks, we compute geometric, color, and texture descriptors for each piece edge and construct a reciprocal-match adjacency graph based on a weighted similarity function. The graph is dense but captures some locally plausible relationships, as confirmed by the visual overlays.**

A final assembly attempt was made for completeness, but the resulting mosaic is not geometrically meaningful. Overall, the pipeline demonstrates strong segmentation performance and a principled approach to edge matching and graph construction, while highlighting the challenges of achieving a full reconstruction under the provided constraints.

I. Introduction

Reconstructing a large jigsaw puzzle from scattered piece images is a challenging computer vision task involving segmentation, feature extraction, edge matching, and global assembly. Classical solvers rely mainly on shape cues [1], while more recent work formulates assembly as a graph or probabilistic optimisation problem [2]. Our approach follows this direction by constructing an adjacency graph from pairwise edge similarities.

In this project, we work with a 1000-piece puzzle dataset consisting of natural images. Only 500 pieces include labeled binary masks, while the remaining pieces must be segmented automatically. Because precise cropped RGB piece images were not provided, full puzzle assembly is not expected to succeed; instead, the focus of the assignment is on designing and evaluating a principled pipeline for segmentation, feature extraction, and adjacency reasoning

The goals of this report are therefore to:

- describe a clear pipeline covering segmentation, feature extraction, PCA embeddings, and graph construction,
- present methods for edge matching supported by qualitative visualisations,
- report quantitative segmentation performance and analyse the resulting adjacency graph,
- reflect on strengths, limitations, and why full reconstruction is challenging under the given constraints.

II. Pipeline Overview

A. High-Level Architecture

Our system is organised into four main stages, implemented as separate Python scripts:

1) **Segmentation** (`segment.py`, `inference.py`): train a U-Net model on 500 labeled images and run inference to predict binary masks for all puzzle images.
2) **Feature extraction** (`extract_features.py`): extract per-piece and per-edge descriptors from the RGB image and corresponding predicted mask.
3) **Edge embeddings & matching** (`embeddings.py`, `matching.py`, `build_graph.py`): compress edge features with PCA, compute similarity scores between edges, and build a pruned adjacency graph using reciprocal matches.
4) **Assembly** (`assemble.py`, `visualize_graph.py`): visualise the adjacency graph and tile connected components of pieces.

III. Methods

A. Data

The dataset contains images of puzzle pieces laid on a uniform background. For segmentation training we are given 500 images with corresponding binary masks. The remaining images lack ground truth masks.

B. Piece Segmentation

1) Model architecture: We use a standard U-Net architecture [3] implemented in `segment.py`. The encoder consists of four convolutional blocks with max-pooling; the decoder uses transpose convolutions (or bilinear upsampling in an alternative configuration) with skip connections from the encoder. The network takes a 3-channel RGB image as input and outputs logits for two classes: background and piece.

2) Training procedure: We trained the U-Net for 30 epochs using Adam (1×10^{-4}), cross-entropy loss, and standard augmentations. Gradient clipping, AMP, and early stopping were applied. The 500 labelled images were split 85%/15% into training and validation sets.

The best checkpoint (according to validation IoU) is saved as `models/unet_best.pth`. This checkpoint is then used at inference to generate predicted masks for all puzzle images that do not already have ground-truth masks.

3) Inference and post-processing: At inference time we:

1) Resize each input image to the chosen training resolution (e.g., 512×512).
2) Forward it through the trained U-Net to obtain a foreground probability map.
3) Resize the probability map back to the original image size.
4) Threshold at $p \geq 0.5$ to obtain a binary mask.

The script writes masks as 0/255 PNG files in a dedicated `pred_masks/` directory to keep them separate from any ground-truth masks.

C. Per-Piece and Per-Edge Feature Extraction

Once we have a binary mask for each piece, `extract_features.py` computes per-piece and per-edge descriptors.

1) Contour extraction and edge splitting: For each mask, we:

- Extract the largest external contour using OpenCV.
- Enforce clockwise ordering and smooth the contour using a Savitzky-Golay filter [4].
- Approximate the contour with either a four-corner polygon or a convex hull.
- Split the contour into four sides, corresponding roughly to the four physical edges of the puzzle piece.

Each side is resampled to a fixed number of points (64 points per edge) to enable consistent feature computation across pieces.

2) Geometric descriptors: For each resampled side we compute:

- **Curvature:** discrete curvature along the side via first and second derivatives of the resampled coordinates, followed by smoothing.
- **Fourier shape descriptor [5]:** the magnitude of the first 20 Fourier coefficients of the complex contour representation, normalised by the DC component. This encodes global edge shape while being invariant to translation.
- **Edge type:** a simple classifier labels each edge as “flat”, “tab”, or “blank” based on deviation from a fitted line and the sign of the deviation.

We also compute global per-piece statistics such as contour area, perimeter, aspect ratio, and centroid, which are used later for basic piece-type classification (corner, edge, interior).

3) Appearance and texture descriptors: To capture appearance consistency across neighbouring pieces, we sample RGB colours on either side of each edge. For a subset of edge points, we extract a few pixels inside the piece and a few pixels outside along the local normal direction, accumulate separate 3D colour histograms (4 bins per channel), and compute the mean colour on each side. In addition, we extract a small grayscale patch around the edge and compute an MR8-style [6] texture descriptor using Gaussian, Laplacian-of-Gaussian, and Sobel filters, which captures local contrast and texture differences.

D. Per-Edge Embeddings

The full edge descriptor is high-dimensional, so `embeddings.py` compresses it into a compact vector for efficient matching. For each edge we concatenate curvature statistics, the first ten Fourier magnitudes, colour histograms and means, MR8 texture responses, and edge length. These vectors are then standardised and reduced to 16 PCA [7] components, producing a unified low-dimensional representation that combines shape, colour, and texture cues.

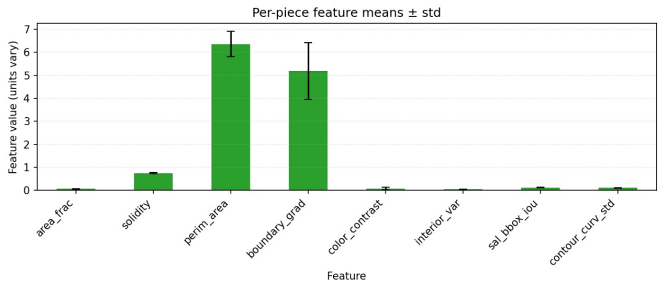

Fig. 1: Per-piece feature means and standard deviations used for edge matching.

To better understand the behaviour of these features before matching, Figure 1 summarises their mean values and variability across all pieces. Features with higher variance (such as colour contrast and boundary gradients) contribute more discriminative power during matching, while low-variance features (such as curvature statistics) tend to be less informative on their own.

E. Edge Matching and Adjacency Graph Construction

1) EdgeMatcher scoring: The core matching logic is implemented in `matching.py`. Given two edges e_i and e_j, the matcher computes:

- A **shape score** based on curvature alignment and Fourier descriptor similarity.
- A **color score** comparing left/right color histograms and mean colors.
- A **Fourier-only score** emphasising global shape.
- An **embedding score [8]** using cosine similarity between PCA embeddings .

These components are combined into a final similarity score:

$$s(e_i, e_j) = 0.35 \cdot s_{\text{shape}} + 0.35 \cdot s_{\text{color}} + 0.15 \cdot s_{\text{fourier}} + 0.15 \cdot s_{\text{embed}}.$$

The weights were chosen to balance geometric and appearance cues. If color dominates, false matches can occur between large regions of uniform texture; if shape dominates, subtle color differences are ignored.

2) Candidate generation and reciprocity: Matching every edge against all others is $O(P^2)$ and expensive at puzzle scale, so `build_graph.py` applies several filters. For each edge we keep only the top-K highest-scoring candidates and discard pairs below a similarity threshold. A match is accepted only if it is *reciprocal* (each piece appears in the other’s top-K list) and the scores are reasonably balanced. These reciprocal pairs form the edges of the adjacency graph, where nodes correspond to pieces (with a coarse type label) and edges store the matched side indices and similarity scores. The full graph is saved as `graph_emb.json`, with a simplified adjacency list also written to `graph.json`.

F. Assembly and Visualisation

The final stage uses the adjacency graph to explore how well the matching captures true neighbour relationships. Full reconstruction was not the primary focus of the project, but an attempt was made, that further proved that the adjacency graph needed more optimization.

We therefore produce three forms of visual output: graph visualisations, matched-edge overlays, and an attempt of the fully reconstrcuted puzzle piece.

1) Graph visualisation: `visualize_graph.py` loads the JSON graph, converts it to a NetworkX graph, and produces overview plots. The matched-edge overlays—using cropped bounding boxes, reprojected edge points, and zoom-in insets—proved particularly valuable for judging whether high-score matches were meaningful or artefacts.

2) Partial assembly (exploratory): We attempted a full reconstruction by cropping pieces from the original images and arranging them according to graph connectivity. However, because the dataset did not provide true per-piece RGB crops, and because the graph itself was highly over-connected, the resulting mosaics are blurry and not geometrically accurate. The `assemble.py` layout therefore functions only as a visual summary of connected components, revealing where the adjacency graph succeeds and where its matches remain ambiguous.

Overall, these visualisations reinforce the main finding of this project: while segmentation and local edge matching are strong, the adjacency graph requires further refinement before a reliable full reconstruction is possible.

IV. Experiments and Results

A. Segmentation Performance

Segmentation performance was monitored during training using an internal train/validation split of the 500 labeled images. The final model checkpoint used for inference corresponds to epoch 29, where the following metrics were reported in Table 1 below:

TABLE I: Best epoch training and validation metrics (Epoch 29).

	Train	Validation
Loss	0.0059	0.0044
IoU	0.9621	0.9721

This indicates that the U-Net not only fits the training data well but also achieves a higher IoU on the held-out validation set, suggesting good generalisation to unseen images.

Using this best-validation checkpoint, we then ran `inference.py` to generate predicted masks for the full dataset. The online server compares our *predicted* masks to two sets of ground-truth labels (Tutor and RK), and reports the following metrics:

TABLE II: Segmentation results on online evaluation server (predicted masks vs. ground-truth).

Label set	mIoU	mDICE	mAccuracy
Tutor	0.9548	0.9710	0.9877
RK	0.9180	0.9572	0.9944

The close agreement between the internal validation IoU (0.9721) and the external mean IoU on the Tutor labels (0.9548) suggests that the model trained generated meaningful and accurate masks for the unlabelled dataset. Visual inspection confirms that most pieces are cleanly segmented, with errors concentrated on very thin tabs, ambiguous boundaries, or challenging lighting conditions.

B. Adjacency Graph Analysis

Using the PCA-enhanced features, we constructed the adjacency graph as shown in Figure 3. The final graph contains 1388 reciprocal edges, with similarity scores ranging from 0.47 to 0.74 (mean ≈ 0.65). Most edges lie between 0.55 and 0.70, with virtually none below the threshold due to reciprocity filtering. This score distribution indicates that the matching function is confident but not very selective: many edges exceed the similarity threshold, resulting in the densely connected graph.

The overview graph confirms this behaviour. A large proportion of pieces fall into a single dominant connected component, showing that the matcher frequently identifies multiple plausible neighbours for a given edge. This density reflects both the strength of shared visual cues among pieces and the limitations of hand-crafted descriptors in visually ambiguous regions.

To better understand which edges correspond to meaningful neighbours, we inspected the highest-scoring reciprocal matches using the overlay visualisation tool. These overlays reveal a mixture of outcomes: some matched edges exhibit consistent curvature and colour cues, while others are clearly spurious despite receiving high similarity scores.

A representative failure case is shown in Figure 2. Although the matcher assigns a score of 0.75, the two edges belong to completely different visual regions. Such false positives commonly arise when unrelated edges share similar curvature patterns or background colours, and they contribute directly to the graph's over-connectivity.

These observations highlight the central challenge in the adjacency stage: while the descriptors capture useful information, they are not sufficiently discriminative to separate true neighbours from visually similar distractors at puzzle scale. This explains why, despite reasonable local matches, the global graph structure remains too dense for reliable downstream assembly.

C. Assembly Behaviour

Because the adjacency graph is dense and over-connected, and because precise cropped RGB piece images were not available, the assembly attempt does not yield a meaningful reconstruction. Furthermore, the resulting assembled image is extremely large (several thousand pixels in each dimension) and therefore omitted from the report.

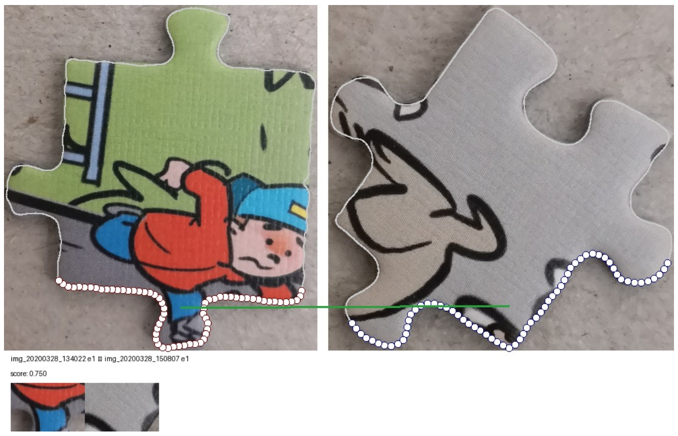

Fig. 2: Example of a high-scoring but incorrect match produced by the edge similarity function. The shape and colour cues lead to a similarity score of 0.75 despite the edges belonging to unrelated puzzle regions.

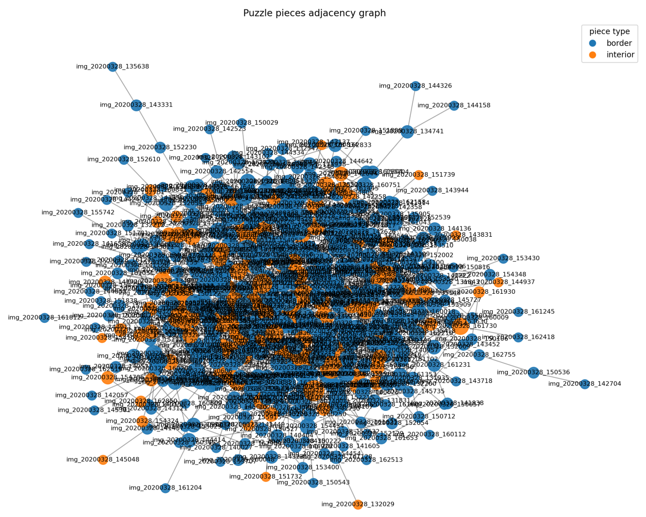

Fig. 3: Global adjacency graph generated from features. Each node corresponds to a puzzle piece (coloured by corner/edge/interior type), and edges represent reciprocal top-K matches. The prominence of a large connected component illustrates the over-connectivity of the matching stage and the difficulty of distinguishing true neighbours in visually homogeneous regions.

V. Reflection

A. What Worked Well

Several components of the pipeline performed strongly:

- **Segmentation robustness:** the U-Net achieved high validation IoU (0.9721) and strong external performance against both Tutor and RK label sets. Mask quality directly enabled reliable contour extraction.
- **Feature design:** combining curvature, Fourier descriptors, colour histograms, and MR8 responses provided diverse cues for edge comparison. This avoided reliance on any single ambiguous feature type.

B. What Failed and Why

Several challenges limited the ability to produce a meaningful global reconstruction:

- **Graph over-connectivity:** similarity thresholds and top-K selection produced a dense adjacency graph. Many plausible but incorrect matches survived reciprocity filtering.
- **Ambiguous textures:** uniform regions caused many false positives because colour and texture statistics were insufficiently discriminative.
- **No geometric assembly constraints:** without piece rotations, spacing constraints, or cycle-consistency checks, the assembly stage could not reason about spatial layout.

C. Limitations

- **Quadratic matching cost:** comparing every edge against all others scales poorly with puzzle size.
- **Error propagation:** any inaccuracy in the segmentation masks carries through to contour extraction and feature computation.
- **Hand-crafted matching:** the similarity function is manually designed and not learned from data, limiting its ability to capture true adjacency.

D. Future Directions

Potential improvements include:

- **Learned edge embeddings** via CNN or transformer encoders [9].
- **Approximate nearest neighbour search** for scalable candidate generation.
- **Graph pruning using piece-type and degree constraints.**
- **Global optimisation** over piece poses with geometric consistency constraints.
- **Explicit handling of rotations** during matching and assembly.

VI. Conclusion

We developed a pipeline for automated reconstruction of a 1000-piece jigsaw puzzle from natural images. The U-Net segmentation stage performs strongly on both internal validation and external label sets, enabling reliable contour extraction. Using these masks, we computed geometric and appearance-based edge descriptors, compressed them with PCA, and constructed an adjacency graph through reciprocal matching.

Although the graph captures some locally plausible neighbours, it remains highly over-connected, highlighting the difficulty of distinguishing edges in visually homogeneous regions. As expected for this assignment and given the lack of true per-piece RGB crops, full puzzle assembly was not feasible.

Overall, the pipeline demonstrates robust segmentation and a principled approach to feature design and adjacency estimation, while revealing the key challenges that must be addressed for reliable large-scale puzzle reconstruction.

VII. Reproducibility and Artifacts

All code is written in Python and relies on standard libraries (PyTorch, OpenCV, NumPy, SciPy, scikit-learn, NetworkX, and Matplotlib). The following artifacts are produced by the pipeline:

- **Segmentation:** a U-Net model is trained using `segment.py`, and the best checkpoint is saved; `inference.py` then uses this checkpoint to generate predicted masks.
- **Features:** per-piece and per-edge features in `outputs/test_features.pkl`, produced by `extract_features.py`.
- **Embeddings:** PCA-enhanced features in `outputs/features_emb.pkl`, produced by `embeddings.py`.
- **Graph:** adjacency graph files `outputs/graph_emb.json` and `outputs/graph.json`, produced by `build_graph.py`.
- **Visuals:** graph overview PNGs, edge overlays, and exported GraphML/GEXF files in `outputs/visuals/`, produced by `visualize_graph.py`.
- **Partial assemblies:** component mosaics and a final stacked image produced by `assemble.py`.

A `requirements.txt` file enumerates the Python dependencies. Running the sequence:

1) `python inference.py`
2) `python extract_features.py`
3) `python embeddings.py`
4) `python build_graph.py`
5) `python visualize_graph.py`

reproduces the core results described in this report.

References

[1] H. Freeman, "The determination of pattern by shape recognition," *Proceedings of the National Academy of Sciences*, vol. 51, pp. 47–53, 1964.

[2] T. S. Cho, S. Avidan, and W. T. Freeman, "A probabilistic framework for solving jigsaw puzzles," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 183–190.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, pp. 234–241.

[4] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[5] E. Persoon and K.-S. Fu, "Shape discrimination using fourier descriptors," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 3, pp. 170–179, 1977.

[6] M. Varma and A. Zisserman, "A statistical approach to texture classification from single images," *International Journal of Computer Vision*, vol. 62, no. 1, pp. 61–81, 2005.

[7] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 11, pp. 559–572, 1901.

[8] G. Salton and M. J. McGill, "Introduction to modern information retrieval," 1983.

[9] S.-H. e. a. Son, "Solving mixed-type jigsaw puzzles using genetic algorithms and deep learning," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2014.