Lost in Pieces:
An Image Jigsaw Reconstruction Challenge

Dionisios N. Sotiropoulos

December 16, 2025

Abstract

This computational project focuses on addressing the problem of reconstructing an image from a set of segmented pieces, in the spirit of a jigsaw puzzle. The task is to design and implement a complete pipeline that (i) extracts descriptors from image segments using both classical feature extraction methods and deep learning architectures, (ii) defines similarity or compatibility measures between puzzle pieces, and (iii) reconstructs the original image by solving a global arrangement problem. The project emphasizes the practical integration of traditional image descriptors with modern deep feature representations derived from convolutional neural networks (CNNs).

1 Introduction

In many computer vision problems, images are not given as neat, fully observed entities but appear as local patches, segmented regions, or partially occluded objects. Inferring the underlying image or scene from such incomplete evidence is a demanding task, requiring robust feature extraction and careful reasoning about spatial and contextual relationships.

This project adopts a *jigsaw puzzle* perspective: a given image is decomposed into pieces (*tiles* or *regions*) which are then shuffled and may also be rotated. The challenge is to reconstruct the original arrangement of pieces using only visual cues. You are expected not only to apply but also to push beyond the feature extraction concepts covered in the course, combining classical descriptors (region and boundary descriptors, texture features, and local interest point descriptors) with deep representations derived from pre-trained CNNs.

The project has a dual purpose:

- To consolidate and deepen your understanding of feature extraction and image descriptors.
- To introduce a realistic, open-ended problem in which multiple competing design choices and evaluation criteria must be explored and justified.

2 Background

2.1 Feature Extraction in Image Analysis

Feature extraction is central to image analysis and computer vision. The goal is to map raw image data into a compact, informative representation that is suitable for recognition, classification, or matching tasks. The course textbook presents a wide range of feature types, including:

- **Region-based descriptors**.
- **Boundary and shape descriptors**.
- **Texture descriptors**.
- **Local interest point descriptors** which capture distinctive patterns in local neighborhoods.

These descriptors aim to be invariant (or at least robust) to transformations such as translation, rotation, scale, and illumination changes, while remaining discriminative for different image regions.

2.2 Deep Learning-Based Image Representations

Deep convolutional neural networks (CNNs) provide powerful learned image representations. Typical architectures learn hierarchical features that transition from low-level edges and textures to high-level object and scene semantics.

For this project, you are encouraged to use pre-trained CNN models as generic feature extractors. Intermediate activations (feature maps) from such networks can be pooled to form fixed-length descriptors for image patches or border strips. These deep features typically provide strong invariance properties and can complement classical hand-crafted descriptors.

3 Project Objectives

The main objective of the project is to design, implement, and evaluate a system that reconstructs a segmented image from its shuffled pieces. Concretely, the project has the following pedagogical and technical objectives:

(O1) **Feature Extraction:** Implement and experiment with region, boundary, texture, and local feature descriptors for image segments, based on the material covered in the course.
(O2) **Deep Features:** Integrate deep CNN-based features as complementary descriptors for patches or border regions.
(O3) **Adjacency Modeling:** Define similarity or compatibility measures between pairs of pieces (and their sides), using the extracted descriptors.
(O4) **Global Reconstruction:** Formulate the reconstruction problem as a global optimization or assignment problem, and implement an algorithm to find a high-quality solution.
(O5) **Evaluation:** Develop quantitative metrics to evaluate reconstruction quality and to compare different feature configurations.

4 Problem Formulation

4.1 Image Segmentation and Puzzle Pieces

Let the original image be a function

$$I : \Omega \to \mathbb{R}^C, \qquad \Omega = \{1, \dots, H\} \times \{1, \dots, W\},$$

where H and W are the height and width in pixels, and C is the number of channels (e.g. $C = 3$ for RGB).

Fix integers $P, Q \in \mathbb{N}$ such that $P \mid H$ and $Q \mid W$. Define

$$h = \frac{H}{P}, \qquad w = \frac{W}{Q},$$

so that each tile has size $h \times w$ pixels. The set of grid positions is

$$\mathcal{G} = \{(r, c) : 1 \le r \le P,\ 1 \le c \le Q\}.$$

For each $(r, c) \in \mathcal{G}$, the corresponding tile (region) is defined as

$$R_{r,c} = \big\{(x, y) \in \Omega : h(r-1) + 1 \le x \le hr,\ w(c-1) + 1 \le y \le wc\big\}.$$

These tiles form a partition of the image domain:

$$\Omega = \bigcup_{(r,c)\in\mathcal{G}} R_{r,c}, \qquad R_{r,c} \cap R_{r',c'} = \emptyset \text{ for } (r,c) \neq (r',c').$$

Let $N = PQ$ be the total number of tiles. For convenience, we index the tiles as

$$\mathcal{R} = \{R_1, \ldots, R_N\},$$

where the index k is associated with a unique grid position $(r_k, c_k) \in \mathcal{G}$ via a fixed bijection

$$g^\star : \{1, \ldots, N\} \to \mathcal{G}, \qquad g^\star(k) = (r_k, c_k).$$

During puzzle generation, the tiles $\{R_k\}$ are transformed as follows:

- **Shuffling:** A permutation σ of $\{1, \ldots, N\}$ is applied, so that the "logical piece index" k no longer corresponds to its original grid position $g^\star(k)$.
- **Rotation:** For each tile R_k, a rotation angle

  $$\theta_k^\star \in \mathcal{A}, \qquad \mathcal{A} = \{0^\circ, 90^\circ, 180^\circ, 270^\circ\},$$

  is chosen and applied. In the standard project setting, rotations are always allowed and must be estimated by your reconstruction algorithm. (You may optionally consider simpler, non-rotated variants as baselines.)

You are given only the resulting set of tile images (after shuffling and rotation) and the grid dimensions (P, Q). The ground-truth information

$$g^\star : \{1, \ldots, N\} \to \mathcal{G}, \qquad \{\theta_k^\star\}_{k=1}^N,$$

is stored internally during puzzle generation and will be used at the end to evaluate each reconstructed solution against the original image I.

4.2 Reconstruction Task

The reconstruction task consists of two intertwined subproblems:

(P1) **Piece Placement:** Estimate an assignment

$$\pi : \{1, \ldots, N\} \to \mathcal{G}$$

that places each observed piece at a unique position in the $P \times Q$ grid.

(P2) **Orientation Estimation:** Estimate an orientation

$$\hat{\theta}_k \in \mathcal{A}$$

for each piece k, since rotations are part of the puzzle.

The ideal solution would recover the hidden ground truth, that is,

$$\pi(k) = g^\star(k) \quad \text{and} \quad \hat{\theta}_k = \theta_k^\star \quad \text{for all } k,$$

or at least approximate it as closely as possible. Because $g^\star$ and $\theta_k^\star$ are known to the puzzle-generation code but hidden from you, they provide a precise reference for quantitative evaluation of your reconstruction.

5 Feature Extraction for Puzzle Pieces

In this section we formalize the geometric entities on which features will be computed (tiles and border strips), introduce several families of descriptors defined on generic image regions, and then explain how these descriptors are systematically lifted to the side level in order to support adjacency modeling and global reconstruction.

5.1 Geometry of Tiles and Border Strips

Recall from Section 4 that the original image is partitioned into N non-overlapping rectangular regions (tiles)

$$\{R_k \subset \mathbb{Z}^2\}_{k=1}^{N},$$

each corresponding to one cell of the $P \times Q$ grid.

Each tile R_k has four sides, which we index by

$$\mathcal{S} = \{\text{top}, \text{right}, \text{bottom}, \text{left}\}.$$

For convenience, we may also write $\mathcal{S} = \{N, E, S, W\}$, where N, E, S and W correspond to the top, right, bottom and left sides, respectively.

Fix an integer parameter $w_{\text{b}} \geq 1$ that specifies the width of the *border strip* around each side. For a tile R_k and a side $s \in \mathcal{S}$, we define the corresponding border strip $B_s(R_k)$ as the set of pixels of R_k that lie within w_{b} pixels from side s. For instance, if the bounding box of R_k is

$$R_k = \{(x, y) : x_{\min} \leq x \leq x_{\max},\ y_{\min} \leq y \leq y_{\max}\},$$

then

- $B_{\text{top}}(R_k)$: the w_{b} top-most rows of R_k,
- $B_{\text{bottom}}(R_k)$: the w_{b} bottom-most rows of R_k,
- $B_{\text{left}}(R_k)$: the w_{b} left-most columns of R_k,
- $B_{\text{right}}(R_k)$: the w_{b} right-most columns of R_k.

Border strips provide the natural geometric support for side-level descriptors: they capture the local appearance near a side where two tiles may potentially meet. Every region-based descriptor introduced below can be evaluated either on an entire tile R_k or on a side border strip $B_s(R_k)$.

5.2 Descriptor Families on Regions

Let $\Omega \subset \mathbb{Z}^2$ denote a generic image region (e.g. an entire tile R_k or a border strip $B_s(R_k)$). We now define several descriptor families $\Phi_d(\Omega)$ that map Ω to a feature vector in $\mathbb{R}^{D_d}$.

5.2.1 Color and Texture Descriptors

To capture local appearance, you should extract both color and texture descriptors in a mathematically well-defined way.

Color histograms. Let $I : \Omega \to \mathbb{R}^C$ be the image in some color space (e.g. RGB, HSV, Lab) and let $R \subset \Omega$ denote the set of pixels of a region (a whole piece or a narrow border strip along one side of a piece). If you choose a color space transformation

$$T : \mathbb{R}^C \to \mathbb{R}^C,$$

you may work with transformed colors

$$\tilde{I}(x, y) = T\big(I(x, y)\big).$$

Fix a number of bins $B \in \mathbb{N}$ per channel. For a single channel $c \in \{1, \dots, C\}$, let $\{I_c^{\min}, I_c^{\max}\}$ be the minimum and maximum values (e.g. 0 and 255), and define bin intervals

$$\mathcal{I}_b = \Big[I_c^{\min} + \frac{b-1}{B}\big(I_c^{\max} - I_c^{\min}\big),\ I_c^{\min} + \frac{b}{B}\big(I_c^{\max} - I_c^{\min}\big)\Big), \quad b = 1, \dots, B.$$

The (normalized) histogram of channel c over region R is

$$h_c(b) = \frac{1}{|R|} \sum_{(x,y)\in R} \mathbf{1}\Big(\tilde{I}_c(x, y) \in \mathcal{I}_b\Big), \quad b = 1, \dots, B,$$

where $|R|$ is the number of pixels in R and $\mathbf{1}(\cdot)$ is the indicator function. You can either:

- concatenate the per-channel histograms into a feature vector $(h_1^\top, \dots, h_C^\top)^\top$, or
- define joint histograms over pairs or triples of channels, if desired.

When R is restricted to a border strip (e.g. pixels within a fixed distance from one side of a tile), this same construction yields color descriptors specific to each side.

Texture features from filter banks. Let $\{F_m\}_{m=1}^M$ be a bank of 2D filters (e.g. derivatives of Gaussians, oriented edge/line detectors, or other filters discussed in the course). For each filter F_m, define the filter response on the image as the discrete convolution

$$R_m(x, y) = (I * F_m)(x, y) = \sum_{(u,v)\in\mathbb{Z}^2} I(u, v)\, F_m(x - u, y - v).$$

(If I is multi-channel, you may either convert to grayscale first or apply filters per channel and aggregate the responses.)

Examples of 2D filter banks. You may construct $\{F_m\}_{m=1}^M$ from one or more of the following families:

- *First-order derivatives of Gaussian.* Let $G_\sigma(x, y)$ be a 2D Gaussian with standard deviation σ,

  $$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right).$$

  The horizontal and vertical derivative filters are

  $$F_\sigma^{(x)}(x, y) = \frac{\partial}{\partial x} G_\sigma(x, y), \qquad F_\sigma^{(y)}(x, y) = \frac{\partial}{\partial y} G_\sigma(x, y).$$

  A simple filter bank can be obtained by choosing several scales $\sigma \in \{\sigma_1, \dots, \sigma_S\}$ and including $\{F_{\sigma_s}^{(x)}, F_{\sigma_s}^{(y)}\}_{s=1}^S$ in the bank.
- *Second-order derivatives / Laplacian-of-Gaussian.* The Laplacian-of-Gaussian (LoG) filter

  $$F_\sigma^{(\mathrm{LoG})}(x, y) = \frac{\partial^2}{\partial x^2} G_\sigma(x, y) + \frac{\partial^2}{\partial y^2} G_\sigma(x, y)$$

  emphasizes blob-like structures. Using several σ values yields multi-scale blob detectors for texture analysis.

- *Oriented edge detectors (e.g. Sobel-like filters).* In discrete form, you may use simple 3×3 kernels such as

$$F_{\text{Sobel}}^{(x)} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad F_{\text{Sobel}}^{(y)} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix},$$

  which respond to vertical and horizontal edges respectively. You can obtain additional orientations by rotating these kernels (or by combining $F_{\text{Sobel}}^{(x)}$ and $F_{\text{Sobel}}^{(y)}$ linearly).

In practice, you may build a filter bank by combining several scales (different σ values) and orientations, resulting in a set of filters $\{F_m\}_{m=1}^{M}$ that jointly capture a wide range of local texture patterns.

Given a region R (piece or border strip), you can summarize the texture in terms of statistics of the responses over R. Typical choices include:

- *Mean response:*

$$\mu_m = \frac{1}{|R|} \sum_{(x,y)\in R} R_m(x,y).$$

- *Energy (e.g. L^2 energy):*

$$E_m = \frac{1}{|R|} \sum_{(x,y)\in R} \big|R_m(x,y)\big|^2.$$

- *Standard deviation:*

$$\sigma_m = \sqrt{\frac{1}{|R|} \sum_{(x,y)\in R} \big(R_m(x,y) - \mu_m\big)^2}.$$

The texture descriptor for region R can then be formed by concatenating these statistics over all filters:

$$\Phi_{\text{tex}}(R) = \mathbf{f}_{\text{tex}}(R) = \big(\mu_1, \ldots, \mu_M, \sigma_1, \ldots, \sigma_M, E_1, \ldots, E_M\big)^\top.$$

Gabor filter-based texture. A common choice is to use Gabor filters, which are localized, oriented, band-pass filters. A 2D Gabor kernel with wavelength λ, orientation θ, phase ψ, standard deviation σ, and aspect ratio γ can be written as

$$g(x,y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right),$$

where

$$x' = x\cos\theta + y\sin\theta, \qquad y' = -x\sin\theta + y\cos\theta.$$

Using a bank of Gabor filters with different orientations θ and wavelengths λ, you can compute responses as above and summarize them over region R with energies E_m or related statistics, obtaining a more explicitly orientation- and frequency-sensitive texture descriptor.

5.2.2 Local Interest Point Descriptors

Local interest point descriptors capture the distribution of gradient-based features within a region. We outline a simplified SIFT-style construction for a generic region Ω; later, in Section 5.3, this construction will be restricted to border strips $B_s(R_k)$.

Let (x,y) be a pixel in region Ω, and denote by

$$\nabla I(x,y) = \big(G_x(x,y), G_y(x,y)\big)$$

the image gradient, with magnitude and orientation

$$m(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}, \qquad \theta(x,y) = \operatorname{atan2}\big(G_y(x,y), G_x(x,y)\big) \in [0, 2\pi).$$

Corner detection on border strips (Harris detector). Before computing local descriptors on a side, we must select a set of interest points (corners) within the corresponding border strip.

Let R_k be a tile with border strip $B_s(R_k)$ for side $s \in \mathcal{S}$, as defined in Section 5.1. For each pixel $(x, y) \in B_s(R_k)$, consider the image gradients $G_x(x, y)$ and $G_y(x, y)$ and form the second-moment (structure) matrix over a small window $W_{(x,y)} \subset B_s(R_k)$ centered at (x, y):

$$M(x,y) = \sum_{(u,v)\in W_{(x,y)}} w(u,v) \begin{bmatrix} G_x(u,v)^2 & G_x(u,v)G_y(u,v) \\ G_x(u,v)G_y(u,v) & G_y(u,v)^2 \end{bmatrix},$$

where $w(u, v)$ is a weighting function (typically a Gaussian) that gives higher weight to pixels near the center of the window.

Let $\lambda_1(x, y)$ and $\lambda_2(x, y)$ be the eigenvalues of $M(x, y)$, or equivalently define the Harris response

$$R(x,y) = \det M(x,y) - k\,\big(\operatorname{trace} M(x,y)\big)^2,$$

with a small constant $k \in [0.04, 0.06]$. Intuitively, $R(x, y)$ is large and positive when the gradient changes significantly in *both* directions, which corresponds to a corner-like structure.

For each side s of tile R_k, we obtain a set of keypoints $\mathcal{K}_s(R_k)$ in three steps:

- *Response thresholding (candidate corners).* Choose a threshold $\tau_H > 0$ for the Harris response. Consider only pixels in the border strip $B_s(R_k)$ and form the candidate set

  $$\mathcal{C}_s(R_k) = \big\{(x,y) \in B_s(R_k) : R(x,y) \geq \tau_H\big\}.$$

  In practice, τ_H can be set either as an absolute value or adaptively (e.g. as a fixed percentile of the responses in $B_s(R_k)$) so that a reasonable number of candidates is retained.
- *Non-maximum suppression (corner localization).* For each candidate $(x, y) \in \mathcal{C}_s(R_k)$, compare $R(x, y)$ to its neighbors in a small window $W_{\mathrm{NMS}}(x, y)$ (e.g. a 3×3 or 5×5 neighborhood restricted to $B_s(R_k)$). Keep (x, y) as a keypoint only if

  $$R(x,y) = \max_{(u,v)\in W_{\mathrm{NMS}}(x,y)} R(u,v),$$

  i.e. it is a local maximum of the Harris response on that side. All other candidates in $\mathcal{C}_s(R_k)$ that are not strict local maxima are discarded.
- *Refinement and keypoint selection.* From the remaining local maxima, keep only those points for which a full $K \times K$ descriptor patch lies entirely inside the border strip $B_s(R_k)$. Concretely, discard any candidate whose distance to the tile corners or to the interior boundary of the strip is smaller than $\lfloor K/2 \rfloor$ pixels.

The remaining points after non-maximum suppression and optional refinement form the side-specific keypoint set

$$\mathcal{K}_s(R_k) = \big\{p_j = (x_j, y_j)\big\},$$

and each $p_j \in \mathcal{K}_s(R_k)$ will serve as the center of a local patch for which a gradient-histogram descriptor is computed.

Patch-based gradient histogram descriptor. Let $\mathcal{K}(\Omega)$ denote a set of keypoints detected within Ω (e.g. via a corner detector). For a keypoint $p_j = (x_j, y_j) \in \mathcal{K}(\Omega)$:

(i) *Local patch extraction.* Extract a square image patch N_j of fixed size $K \times K$ pixels, centered at (x_j, y_j) (for example, $K = 16$ or $K = 24$). Let q index the pixel locations inside this patch.

(ii) *Gradient computation.* For each pixel q in N_j, compute discrete image derivatives $G_x(q)$ and $G_y(q)$ (e.g. using simple finite differences). Define the gradient magnitude and orientation as

$$m(q) = \sqrt{G_x(q)^2 + G_y(q)^2}, \qquad \theta(q) = \operatorname{atan2}\big(G_y(q), G_x(q)\big).$$

Optionally weight each pixel by a Gaussian window centered at p_j,

$$w(q) = \exp\left(-\frac{\|q - p_j\|_2^2}{2\sigma_w^2}\right),$$

with σ_w proportional to K.

(iii) *Orientation histogram construction.* Fix a number of orientation bins B_θ (e.g. $B_\theta = 8$) that partition the interval $[0, 2\pi)$ into equal-width sub-intervals. Let the ℓ-th bin correspond to the interval

$$\mathcal{B}_\ell = \Big[2\pi\frac{\ell - 1}{B_\theta},\ 2\pi\frac{\ell}{B_\theta}\Big), \qquad \ell = 1, \ldots, B_\theta.$$

For each pixel q in the patch N_j, use its gradient orientation $\theta(q)$ to find the unique bin index $b(q) \in \{1, \ldots, B_\theta\}$ such that $\theta(q) \in \mathcal{B}_{b(q)}$. The contribution of pixel q to the descriptor is its weighted gradient magnitude $w(q)\, m(q)$, which is added to the corresponding bin. Formally, the histogram values are defined as

$$h_\ell = \sum_{q \in N_j} w(q)\, m(q)\, \mathbf{1}\big(b(q) = \ell\big), \qquad \ell = 1, \ldots, B_\theta,$$

where $\mathbf{1}(\cdot)$ is the indicator function. Collecting all bins yields the raw descriptor vector

$$\mathbf{v}_j = (h_1, \ldots, h_{B_\theta})^\top \in \mathbb{R}^{B_\theta}.$$

(iv) *Normalization.* Normalize the histogram to reduce the effect of contrast and illumination:

$$\mathbf{f}_j = \frac{\mathbf{v}_j}{\|\mathbf{v}_j\|_2 + \varepsilon},$$

where $\varepsilon > 0$ is a small constant. The resulting vector $\mathbf{f}_j \in \mathbb{R}^{B_\theta}$ is the local descriptor at keypoint p_j.

Descriptor set per region. For a region Ω, this yields a variable-length set of descriptors:

$$\mathcal{F}(\Omega) = \big\{\mathbf{f}_j : p_j \in \mathcal{K}(\Omega)\big\}.$$

To obtain a fixed-dimensional descriptor for Ω, you must aggregate the set $\mathcal{F}(\Omega)$ into a single vector. Assuming that $\mathcal{F}(\Omega) = \{\mathbf{f}_1, \ldots, \mathbf{f}_M\}$, the aggregated vector may be defined as:

$$\bar{\mathbf{f}}(\Omega) = \frac{1}{M}\sum_{j=1}^{M} \mathbf{f}_j \in \mathbb{R}^{B_\theta}.$$

5.2.3 Deep CNN Descriptors

In addition to hand-crafted descriptors, you should obtain deep feature representations using a pre-trained convolutional neural network (CNN). This will allow you to compare the performance of classical features against modern deep representations, and to explore their combination.

CNN model and notation. Let Φ denote a fixed, pre-trained CNN (e.g. a standard architecture trained on ImageNet). Given an input image $X \in \mathbb{R}^{H_0 \times W_0 \times C_0}$, the network computes a sequence of feature maps (layer outputs)

$$F^{(0)}(X), F^{(1)}(X), \ldots, F^{(L)}(X),$$

where $F^{(0)}(X) = X$, and for each layer $\ell = 1, \ldots, L$,

$$F^{(\ell)}(X) \in \mathbb{R}^{H_\ell \times W_\ell \times C_\ell}$$

is a 3D tensor (height $\times$ width $\times$ channels). You will select one or more layers $\ell \in \mathcal{L} \subseteq \{1, \ldots, L\}$ from which to extract features.

Preprocessing and resizing. Each CNN has a required input size (e.g. $H_0 = W_0 = 224$) and specific preprocessing (normalization) steps. Let R denote a region (a tile or a border strip). Define an operator

$$\mathcal{P} : (\text{image patch}) \to \mathbb{R}^{H_0 \times W_0 \times C_0}$$

that:

- crops the pixels corresponding to the region R,
- resizes the crop to the required input size (H_0, W_0) (e.g. using bilinear interpolation),
- applies any required channel-wise normalization.

The normalized, resized input for region R is then

$$X_R = \mathcal{P}(R).$$

Feature extraction and spatial pooling. For a chosen layer $\ell \in \mathcal{L}$, the corresponding feature map for region R is

$$F_R^{(\ell)} = F^{(\ell)}(X_R) \in \mathbb{R}^{H_\ell \times W_\ell \times C_\ell}.$$

To obtain a fixed-length feature vector from this spatial map, you should apply a pooling operator over the spatial dimensions (H_ℓ, W_ℓ). A common choice is *global average pooling*, defined for channel index $c = 1, \ldots, C_\ell$ by

$$f_c^{(\ell)}(R) = \frac{1}{H_\ell W_\ell} \sum_{i=1}^{H_\ell} \sum_{j=1}^{W_\ell} F_R^{(\ell)}(i, j, c).$$

Collecting these into a vector yields

$$\mathbf{f}^{(\ell)}(R) = \left(f_1^{(\ell)}(R), \ldots, f_{C_\ell}^{(\ell)}(R)\right)^\top \in \mathbb{R}^{C_\ell}.$$

If you use multiple layers $\ell \in \mathcal{L}$, you may concatenate the corresponding vectors to form a single multi-scale deep descriptor:

$$\Phi_{\text{deep}}(R) = \mathbf{f}_{\text{deep}}(R) = \left(\mathbf{f}^{(\ell_1)}(R)^\top, \ \ldots, \ \mathbf{f}^{(\ell_{|\mathcal{L}|})}(R)^\top\right)^\top.$$

By design, these deep descriptors are high-level, learned features that summarize complex patterns (e.g. textures, edges, object parts) without manual engineering.

5.3 From Piece-Level to Side-Level Descriptors

The descriptor families defined above operate on generic regions $\Omega \subset \mathbb{Z}^2$. We now describe a general mechanism to derive *side-level* descriptors for each tile side $s \in \mathcal{S}$, and then specialize it to the most informative descriptor families for adjacency modeling.

General construction. Let $\mathcal{D}$ denote the set of descriptor families you use at the piece level, and for each $d \in \mathcal{D}$ let

$$\Phi_d : (\text{image region}) \to \mathbb{R}^{D_d}, \qquad R \mapsto \Phi_d(R)$$

be the corresponding descriptor mapping. The side-level version of descriptor family d on side s of tile R_k is obtained by restricting Φ_d to the border strip $B_s(R_k)$:

$$\mathbf{f}_{d,s}(R_k) = \Phi_d\big(B_s(R_k)\big) \in \mathbb{R}^{D_d}.$$

Thus, the *definition* of the descriptor does not change; only the spatial support (the region over which it is computed) is restricted to a thin band near the side.

For each side (R_k, s) we may also concatenate multiple descriptor families into a single vector, e.g.

$$\mathbf{f}_s(R_k) = \big(\alpha_{\text{col}}\,\mathbf{f}_{\text{col},s}(R_k)^\top, \alpha_{\text{tex}}\,\mathbf{f}_{\text{tex},s}(R_k)^\top, \alpha_{\text{loc}}\,\mathbf{f}_{\text{loc},s}(R_k)^\top, \alpha_{\text{deep}}\,\mathbf{f}_{\text{deep},s}(R_k)^\top\big)^\top,$$

where the non-negative weights α_{col}, α_{tex}, α_{loc} and α_{deep} control the relative contribution of each descriptor family.

Side-level color descriptors. Let Φ_{col} be the color histogram mapping described in Section 5.2.1, which maps a region R to its normalized (possibly multi-channel) color histogram. The side-level color descriptor for side s of tile R_k is

$$\mathbf{f}_{\text{col},s}(R_k) = \Phi_{\text{col}}\big(B_s(R_k)\big).$$

These descriptors capture how color statistics behave right at the potential interface between neighboring pieces, making them a strong baseline for adjacency estimation.

Side-level texture descriptors. Let Φ_{tex} be the texture descriptor mapping built from filter-bank responses (Section 5.2.1). The corresponding side-level texture descriptor for side s of tile R_k is

$$\mathbf{f}_{\text{tex},s}(R_k) = \Phi_{\text{tex}}\big(B_s(R_k)\big),$$

i.e. you compute filter responses only for $(x, y) \in B_s(R_k)$ and summarize them with the same statistics as in the piece-level definition. This yields side-specific signatures of local textures (e.g. grass, sky, foliage) that are particularly informative for deciding whether two sides are compatible.

Side-level deep CNN descriptors. Let Φ_{deep} denote the deep CNN descriptor mapping introduced in Section 5.2.3. To obtain a side-level deep descriptor, you apply the same procedure to the border strip $B_s(R_k)$:

$$\mathbf{f}_{\text{deep},s}(R_k) = \Phi_{\text{deep}}\big(B_s(R_k)\big).$$

This yields one high-level deep feature vector per side, focused specifically on the visual content near that side.

Local interest point descriptors on borders. We now specialize the generic local descriptor construction of Section 5.2.2 to border strips.

Let R_k be a tile with bounding box

$$R_k = \{(x, y) : x_{\min} \le x \le x_{\max},\ y_{\min} \le y \le y_{\max}\},$$

and let $B_s(R_k)$ be the border strip for side $s \in \mathcal{S}$ defined in Section 5.1. Detect keypoints within $B_s(R_k)$ and denote their set by $\mathcal{K}_s(R_k) \subset B_s(R_k)$.

For each keypoint $p_j = (x_j, y_j) \in \mathcal{K}_s(R_k)$, compute a SIFT-style descriptor $\mathbf{f}_j$ as in Section 5.2.2. In addition, compute a normalized shift coordinate τ_j that encodes its position along side s:

- for the top or bottom side,
$$\tau_j = \frac{y_j - y_{\min}}{y_{\max} - y_{\min}} \in [0, 1],$$
- for the left or right side,
$$\tau_j = \frac{x_j - x_{\min}}{x_{\max} - x_{\min}} \in [0, 1],$$

where $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is the bounding box of tile R_k.

For each side s of tile R_k, you obtain a variable-length set:

$$\mathcal{F}_s(R_k) = \big\{(\mathbf{f}_j, \tau_j) : p_j \in \mathcal{K}_s(R_k)\big\}.$$

To get a fixed-dimensional side-specific descriptor $\mathbf{f}_{\text{loc},s}(R_k)$, aggregate the descriptors in $\mathcal{F}_s(R_k)$ as:

$$\mathbf{f}_{\text{loc},s}(R_k) = \frac{1}{M} \sum_{j=1}^{M} \mathbf{f}_j.$$

These side-specific local descriptors are designed to capture how edges, corners, and small-scale structures run through the border neighborhoods. When two pieces were adjacent in the original image, their corresponding border descriptors should exhibit strong similarities.

Summary. In summary, for each tile R_k and side $s \in \mathcal{S}$, you may construct a collection of side-level descriptors

$$\big\{\mathbf{f}_{\text{col},s}(R_k),\ \mathbf{f}_{\text{tex},s}(R_k),\ \mathbf{f}_{\text{loc},s}(R_k),\ \mathbf{f}_{\text{deep},s}(R_k)\big\},$$

by reusing your strongest region-based descriptor families on the border strips. In practice, you are expected to implement side-level color descriptors, and you are strongly encouraged to explore side-level texture and deep CNN descriptors. Side-level local interest-point descriptors can be treated as an advanced option.

5.4 Adjacency Modeling

Given side-level feature vectors for each piece and each side, together with tile-level (piece-level) descriptors for each piece, you must define *compatibility scores* between candidate neighboring sides. The goal is to quantify how likely it is that side s of piece i was adjacent to side t of piece j in the original image, while also taking into account how globally similar the two pieces are as whole tiles.

Notation. Recall that:

- Pieces are indexed by $i, j \in \{1, \ldots, N\}$.
- Sides are indexed by $s, t \in \mathcal{S} = \{\text{top}, \text{bottom}, \text{left}, \text{right}\}$.
- For each descriptor family d (color, texture, local, deep, etc.) and side s of piece i, you have a side-level feature vector
$$\mathbf{f}_{d,s}(i) \in \mathbb{R}^{D_d},$$
obtained by restricting the descriptor to the border strip $B_s(R_i)$ (Section 5.3).
- For the same family d, you may also compute a *tile-level* (piece-level) descriptor on the whole region R_i:
$$\mathbf{f}_d^{\text{tile}}(i) \in \mathbb{R}^{\tilde{D}_d},$$
using the same mapping Φ_d but on the full tile (e.g. global color histogram, global texture statistics, or piece-level deep CNN features).

We will build, for each side pair $((i, s), (j, t))$, a scalar compatibility score $C((i, s), (j, t)) \in \mathbb{R}$ that combines both side-level and tile-level information.

Side-level and tile-level distances. For each descriptor family d, define:

- a *side-level* distance between side s of piece i and side t of piece j,

$$d_d^{\text{side}}\big((i,s),(j,t)\big) = d_d\big(\mathbf{f}_{d,s}(i),\, \mathbf{f}_{d,t}(j)\big) \geq 0,$$

- and a *tile-level* distance between the whole pieces i and j,

$$d_d^{\text{tile}}(i,j) = d_d\big(\mathbf{f}_d^{\text{tile}}(i),\, \mathbf{f}_d^{\text{tile}}(j)\big) \geq 0.$$

Here $d_d(\cdot,\cdot)$ is any suitable distance for family d (e.g. Euclidean distance on feature vectors, χ^2 distance on histograms, etc.).

From distances to similarities. Each distance is converted to a similarity (compatibility) score via a decreasing function g_d:

$$C_d^{\text{side}}\big((i,s),(j,t)\big) = g_d\Big(d_d^{\text{side}}\big((i,s),(j,t)\big)\Big),$$
$$C_d^{\text{tile}}(i,j) = g_d\big(d_d^{\text{tile}}(i,j)\big).$$

Typical choices include:

- *Gaussian-like similarity* for either side-level or tile-level vectors:

$$g_d(d) = \exp(-\lambda_d d^2), \qquad \lambda_d > 0,$$

- *Cosine similarity* (especially for deep features):

$$C_d^{\text{side}}\big((i,s),(j,t)\big) = \frac{\mathbf{f}_{d,s}(i)^\top \mathbf{f}_{d,t}(j)}{\|\mathbf{f}_{d,s}(i)\|_2\, \|\mathbf{f}_{d,t}(j)\|_2},$$

  and similarly for $C_d^{\text{tile}}(i,j)$ using $\mathbf{f}_d^{\text{tile}}(i)$, $\mathbf{f}_d^{\text{tile}}(j)$, optionally mapped from $[-1,1]$ to $[0,1]$.

Descriptor-specific compatibility terms. For each descriptor family d, you can combine side-level and tile-level contributions into a single compatibility term:

$$C_d\big((i,s),(j,t)\big) = w_d^{\text{side}}\, C_d^{\text{side}}\big((i,s),(j,t)\big) + w_d^{\text{tile}}\, C_d^{\text{tile}}(i,j),$$

where $w_d^{\text{side}}, w_d^{\text{tile}} \geq 0$ control the relative influence of local border evidence vs. global tile similarity for that descriptor family.

Combined compatibility score. The overall compatibility score between side s of piece i and side t of piece j is then obtained by summing across descriptor families:

$$C\big((i,s),(j,t)\big) = \sum_{d\in\mathcal{D}} \alpha_d\, C_d\big((i,s),(j,t)\big),$$

where $\alpha_d \geq 0$ are global weights specifying the importance of each descriptor family (color, texture, deep, local, etc.). In expanded form:

$$C\big((i,s),(j,t)\big) = \sum_{d\in\mathcal{D}} \alpha_d\Big(w_d^{\text{side}}\, C_d^{\text{side}}\big((i,s),(j,t)\big) + w_d^{\text{tile}}\, C_d^{\text{tile}}(i,j)\Big).$$

Note that $C((i,s),(j,t))$ is generally *directed* (right side of i vs left side of j is not the same as left side of i vs right side of j); your reconstruction algorithm must respect the geometric orientation of the sides when proposing adjacencies.

Learning or tuning the weights. The weights α_d, w_d^{side}, and w_d^{tile} can be:

- *Manually tuned* (e.g. giving higher weight to color and deep features, moderate weight to texture, etc.).
- *Learned* from data using a simple supervised model.

To learn them, you can:

1. Use the known ground truth adjacencies from the original image to build:
   - a set of *positive* examples: true neighboring side pairs $((i, s), (j, t))$,
   - a set of *negative* examples: randomly sampled non-neighboring side pairs.
2. For each candidate pair, compute a feature vector collecting all descriptor-specific similarities, e.g.

$$\mathbf{z}((i, s), (j, t)) = \big(C^{\text{side}}_{\text{color}}, C^{\text{tile}}_{\text{color}}, C^{\text{side}}_{\text{texture}}, C^{\text{tile}}_{\text{texture}}, C^{\text{side}}_{\text{deep}}, C^{\text{tile}}_{\text{deep}}, \dots\big)^\top.$$

3. Fit a simple linear classifier (e.g. logistic regression, linear SVM) that predicts whether a pair is a true neighbor based on $\mathbf{z}$:

$$S((i, s), (j, t)) = \mathbf{w}^\top \mathbf{z}((i, s), (j, t)) + b,$$

   where $\mathbf{w}$ and b are learned parameters.

The learned score $S((i, s), (j, t))$ can then be used directly as your compatibility score $C((i, s), (j, t))$, with the components of $\mathbf{w}$ playing the role of data-driven weights on the different side-level and tile-level terms.

In all cases, your objective is to design C so that true neighbors receive systematically higher scores than non-neighbors, thereby providing a meaningful signal for the global reconstruction algorithm described next.

5.5 Global Reconstruction Algorithm

Using the side-level compatibility scores defined in Section 5.4, the reconstruction task is formulated as a discrete optimization problem over both *placements* and *orientations* of all pieces.

Configuration space. Recall that $\mathcal{G}$ is the set of grid positions, $N = |\mathcal{G}|$ is the number of pieces, and $\mathcal{A}$ is the finite set of allowed orientations (e.g. $\mathcal{A} = \{0°, 90°, 180°, 270°\}$). A *configuration* of the puzzle consists of:

- an assignment (permutation) $\pi : \{1, \dots, N\} \to \mathcal{G}$ that places each piece at a unique grid position;
- an orientation $\theta_k \in \mathcal{A}$ for each piece $k = 1, \dots, N$.

The space of all configurations $(\pi, \{\theta_k\})$ is extremely large, which motivates the use of approximate optimization methods.

Side-based adjacency score. We now quantify how good a given configuration $(\pi, \{\theta_k\})$ is in terms of how well neighboring pieces fit together along their touching sides.

First, let $\mathcal{E}$ be the set of all pairs of neighboring grid positions in the $P \times Q$ grid, using 4-neighborhood connectivity (only above, below, left, right):

$$\mathcal{E} = \big\{\big((r, c), (r', c')\big) \in \mathcal{G} \times \mathcal{G} : |r - r'| + |c - c'| = 1\big\}.$$

Each element $\big((r, c), (r', c')\big) \in \mathcal{E}$ represents two adjacent cells in the grid.

Given a configuration $(\pi, \{\theta_k\})$:

- let i be the piece placed at position (r, c), i.e. $\pi(i) = (r, c)$;
- let j be the piece placed at position (r', c'), i.e. $\pi(j) = (r', c')$;
- the relative locations of (r, c) and (r', c'), together with the orientations θ_i and θ_j, determine which side s of piece i faces which side t of piece j (for example, the right side of i versus the left side of j, after accounting for rotations).

For that pair of touching sides (i, s) and (j, t), we evaluate the combined compatibility score $C\big((i, s), (j, t)\big)$ defined in Section 5.4. The total adjacency-based score of the configuration is then obtained by summing over all neighboring grid pairs:

$$F_{\text{adj}}(\pi, \{\theta_k\}) = \sum_{\big((r,c),(r',c')\big)\in\mathcal{E}} C\big((i, s), (j, t)\big),$$

where, for each pair $\big((r, c), (r', c')\big)$, the indices i, j, s, t are determined as described above.

Adjacency-based objective function. In the simplest and most natural formulation, the overall quality of a configuration is *exactly* its total adjacency score:

$$F_{\text{tot}}(\pi, \{\theta_k\}) = F_{\text{adj}}(\pi, \{\theta_k\}).$$

The global reconstruction problem is then

$$\max_{\pi,\{\theta_k\}} \; F_{\text{tot}}(\pi, \{\theta_k\})$$

subject to:

$$\pi \text{ is a permutation of } \{1, \ldots, N\} \quad \text{and} \quad \theta_k \in \mathcal{A} \text{ for all } k.$$

Assignment / Integer Linear Programming (ILP) Formulations For smaller puzzles, you may formulate the problem using binary decision variables. One convenient choice is to introduce variables

$$x_{k,g,a} = \begin{cases} 1, & \text{if piece } k \text{ is placed at grid position } g \text{ with orientation } a, \\ 0, & \text{otherwise,} \end{cases}$$

for all $k \in \{1, \ldots, N\}$, $g \in \mathcal{G}$, and $a \in \mathcal{A}$. The constraints

$$\sum_{g\in\mathcal{G}} \sum_{a\in\mathcal{A}} x_{k,g,a} = 1 \quad \forall k, \qquad \sum_{k=1}^{N} \sum_{a\in\mathcal{A}} x_{k,g,a} = 1 \quad \forall g \in \mathcal{G}$$

ensure that each piece is used exactly once and each grid position is occupied by exactly one piece (with a single orientation).

Precompute adjacency scores $C_{k,\ell}^{g,g',a,b}$ for each neighboring grid pair $(g, g') \in \mathcal{E}$, piece pair (k, ℓ), and orientation pair (a, b), where $C_{k,\ell}^{g,g',a,b}$ encodes the compatibility of the sides that meet when k is placed at g with orientation a and ℓ at g' with orientation b. The objective becomes

$$\max_{x} \sum_{(g,g')\in\mathcal{E}} \sum_{k,\ell=1}^{N} \sum_{a,b\in\mathcal{A}} C_{k,\ell}^{g,g',a,b}\, x_{k,g,a}\, x_{\ell,g',b},$$

subject to the assignment constraints above. This yields a quadratic assignment or integer quadratic programming formulation. It is generally expensive to solve exactly, but it provides a clean mathematical reference for small-scale instances.

Heuristic search and metaheuristics. For realistic puzzle sizes, you will typically resort to heuristic or metaheuristic algorithms that search over configurations $(\pi, \{\theta_k\})$:

- **Greedy construction and local search:** Start from an initial configuration (random or heuristic). At each step, propose local moves such as:
  - swapping the positions of two pieces (and possibly adjusting their orientations),
  - rotating a single piece in place (changing θ_k),
  - moving a piece to a new grid position.

  Accept moves that increase F_{tot} and iterate until no simple move improves the objective.

Practical expectations. You are expected to:

- **Explicitly define** your objective F_{tot} (here equal to F_{adj}), clearly indicating:
  - which side-level and tile-level compatibility terms are used in the definition of $C((i, s), (j, t))$,
  - how different descriptor families (color, texture, deep, local) contribute via their weights.
- **State the constraints** (permutation of pieces, one orientation per piece, one piece per grid location).
- **Justify your choice of optimization method** in terms of puzzle size, complexity of the objective, and available computational resources.
- **Compare** at least one simple baseline (e.g. using only color-based compatibilities, or only deep features) with your full combined adjacency objective, both in terms of F_{tot} and reconstruction accuracy with respect to the known ground truth.

This explicit formulation of an adjacency-based objective and constraints will help you analyze which aspects of your feature design and optimization strategy contribute most to successful reconstruction.

5.6 Software Environment

You are free to choose the programming environment, but a typical choice is:

- Python, using libraries such as NumPy, SciPy, scikit-image, OpenCV, and PyTorch or TensorFlow for deep features.
- Alternatively, MATLAB with the Image Processing Toolbox and Deep Learning Toolbox.

The code base should be modular, with clear separation between:

- Data loading and puzzle generation.
- Feature extraction modules.
- Adjacency computation.
- Global reconstruction algorithm.
- Evaluation and visualization.

5.7 Suggested Milestones

A possible sequence of milestones is:

(M1) Implement basic puzzle generation and visualization (segmentation, shuffling of pieces).
(M2) Implement and test classical feature extraction for pieces and their borders.
(M3) Implement deep feature extraction using a pre-trained CNN.
(M4) Define and validate adjacency measures on small puzzle instances.
(M5) Implement and test the global reconstruction algorithm on regular grid puzzles.
(M6) Perform systematic experiments and evaluation on a set of images.

6 Evaluation and Deliverables

6.1 Quantitative Evaluation Metrics

To assess the reconstruction quality, you should define and compute at least the following metrics:

- **Piece placement accuracy:** fraction of pieces that are placed in their correct absolute position.
- **Neighbor accuracy:** fraction of true neighboring pairs in the original image that are recovered as neighbors in the reconstruction.
- **Rotation accuracy:** fraction of pieces whose orientation is correctly estimated.

Comparisons should be made between different feature configurations, e.g.:

- Classical features only.
- Deep features only.
- Combined classical and deep features.

6.2 Qualitative Evaluation

In addition to numerical metrics, the visual quality of the reconstructed images should be inspected:

- Present original and reconstructed images side by side.
- Highlight incorrectly placed pieces or mismatched boundaries.
- Discuss typical failure cases and their relation to feature or model limitations.

6.3 Expected Deliverables

The project deliverables are expected to include:

(D1) **Code:** A well-documented code repository containing all modules needed for reproduction of the experiments.

(D2) **Technical report:** A written report describing:

- The problem setting and motivation.
- The feature extraction methods used, with clear explanation of the underlying concepts.
- The design of adjacency measures and the global reconstruction algorithm.
- Experimental setup, datasets, and evaluation metrics.
- Quantitative and qualitative results, with discussion.

(D3) **Demonstration:** A small interactive demo that illustrates the reconstruction process.

6.4 Team Work and Collaboration

Students are encouraged to work on this project in small teams in order to share ideas, divide implementation effort, and practice collaborative development. The following rules apply:

- Teams may consist of *at most three* students.
- Each team submits a *single* code repository and a *single* technical report, clearly stating the team members on the cover page and in the report.
- All team members are jointly responsible for the contents of the submission (code, experiments, and report). Every member should be able to explain the main design choices, algorithms, and results.

- Work should be reasonably distributed among team members (e.g. feature extraction, adjacency modeling, reconstruction algorithm, experiments, writing), and this distribution should be briefly described in the report (e.g. in a short "Contributions" paragraph).

Teams are free to discuss high-level ideas with other groups, but sharing of code or written text across teams is not allowed. Each team is expected to produce an original implementation and report.