JigsawNet: Shredded Image Reassembly using Convolutional Neural Network and Loop-based Composition

Canyu Le and Xin Li

Abstract—This paper proposes a novel algorithm to reassemble an arbitrarily shredded image to its original status. Existing reassembly pipelines commonly consist of a local matching stage and a global compositions stage. In the local stage, a key challenge in fragment reassembly is to reliably compute and identify correct pairwise matching, for which most existing algorithms use handcrafted features, and hence, cannot reliably handle complicated puzzles. We build a deep convolutional neural network to detect the compatibility of a pairwise stitching, and use it to prune computed pairwise matches. To improve the network efficiency and accuracy, we transfer the calculation of CNN to the stitching region and apply a boost training strategy. In the global composition stage, we modify the commonly adopted greedy edge selection strategies to two new loop closure based searching algorithms. Extensive experiments show that our algorithm significantly outperforms existing methods on solving various puzzles, especially those challenging ones with many fragment pieces. Data and code have been made available in https://github.com/Lecanyu/JigsawNet.

Index Terms—Shredded Image Reassembly, General Jigsaw Puzzle Solving, Convolutional Neural Network, Loop Closure Constraints.

I. INTRODUCTION

Reassembling and restoring original information from fragmented visual data is essential in many forensic and archaeological tasks. In the past decades, research progress has been made in reassembling various types of fragments including 2D data such as images [1], [2], frescoes [3], and 3D objects like ancient relics [4], [5] and damaged skeletal remains [6], [7]. These research works could potentially save human being from tedious and time-consuming manual composition in the restoration of valuable documents/objects/evidences in variety of practical cases.

Fractured images can be formulated as solving a arbitrarily-cut jigsaw puzzle. Teaching computers to reliably do this, however, remains challenging, since it was first discussed in [8] in 1964. The difficulty comes from both the local and global aspects of puzzle solving. (1) Locally, we need to identify adjacent pieces and correctly align them. But correlated fragments only share matchable geometry and texture along the fractured boundary. Unlike partial matching studied in classic problems such as image panorama and structure-from-motion, where the overlaps (repeated patterns) are often more significant, here the correlation between adjacent pieces is weak and difficult to identify. (2) Globally, even with a well-designed pairwise alignment algorithm, due to various noise and ambiguity (to be elaborated in Section V), it is usually not always reliable. Effective composition needs to take mutual consistency into account from a more global aspect.

A powerful global composition algorithm, unfortunately, is often complex, computationally expensive, and prone to local optima.

To tackle the above two challenges is non-trivial. For local matching, after a pairwise alignment is computed, reliably identifying whether such an alignment is correct is not easy. Intuitively, smooth transitions in image contexts across the fractured boundary can be a key criterion in formulating or evaluating pairwise alignment compatibility. However, such a smoothness does not simply mean a color or gradient similarity, but is abstract and difficult to model in closed forms. Second, the non-smoothness also often exists in the content of an image near foreground/background contours, silhouettes, or between neighboring objects in the scene. Third, on regions without rich textures (e.g., pure-color backgrounds, or night skies), alignments could have great ambiguity, and incorrect stitching may also produce natural transitions in such cases.

For global composition, when the local pairwise alignments are unreliable (e.g. many incorrect alignments mix with correct ones), finding all the correct ones by maximizing groupwise mutual consistency is essentially an NP-hard problem [9]. An efficient and effective strategy is needed to handle complicated puzzles.

In this work, our main idea and technical contribution in tackling these difficulties are as follows. Locally, we design a Convolutional Neural Network (CNN) to learn implicit image features from fragmented training data, to judge the likelihood of a local alignment being correct. Globally, we generalize and apply the loop-closure constraints, which have been effectively used on SLAM [10], environment reconstruction fields [11], [12], and previous square-shaped jigsaw puzzles [13], [14], to the composition of arbitrarily shredded (geometrically irregular) image fragments.

In summary, the main contributions of this work are

- We design a CNN network to evaluate the pairwise compatibility between fragment pairs. To improve the network performance, two technical components are designed: (1) the transfer of the CNN calculation attention on stitching regions, and (2) an adaptive boosting training procedure for solving the data imbalance problem.

1 Canyu Le is with the Department of Information and Science, Xiamen University, China. E-mail: lecanyu@gmail.com.
2 Xin Li is with School of Electrical Engineering and Computer Science, Louisiana State University, USA. E-mail: xinli@lsu.edu.

Manuscript received September 10, 2018.
We develop a new loop-closure based composition strategy to enforce mutual consistency among poses of multiple pieces. This greatly improves the robustness of global composition, especially in solving complex puzzles.

We have conducted thorough experiments on various benchmarks. Our approach greatly outperforms existing state-of-the-art methods in puzzle solving. Codes and data have been released to facilitate future comparative study on image reassembly and related research.

## II. RELATED WORKS

Originated from Freeman et al. [8], the jigsaw puzzle solving problem has been exploited in many literatures. Generally, we can categorize this problem into solving regular shape puzzles and solving irregular shape puzzles.

### A. Solving Regular-Shaped Jigsaw Puzzles

Square jigsaw puzzles are the most typical cases in regular shape jigsaw puzzle. Recently, multiple literatures have studied this problem. Cho et al. [15] evaluate inter-fragment consistency using the sum-of-squared color difference (SSD) alone the stitching boundary, and used a graphical model to solve the global composition. Pomeranz et al. [16] exploit various measurement strategies to improve the accuracy of pairwise alignment compatibility, and also introduced a consensus metric to the greedy solver in global composition. Gallagher et al. [17] develop a Mahalanobis Gradient Compatibility (MGC) to evaluate the pairwise alignment using changes in intensity gradients, rather than changes in intensity itself near the boundary; in the global composition stage, they greedily generate a minimal spanning tree to connect all the pieces. More recently, state-of-the-art square jigsaw puzzle solving results were reported in [13] and [14]. In [13], Son et al. exploit the loop constraints configuration to filter out false negative alignments; later in [14], the aforementioned MGC measurement is improved by a more accurate intensity gradient calculation, and the overall reassembly is further enhanced by improving the consensus composition.

Those state-of-the-art square puzzle solvers can process even more than a thousand fragments. However, square solvers cannot be used to handle general puzzles that have arbitrary shaped fragments. The key difference is on the assumption of fragmented pieces being square. Such a simplification makes this problem combinatorial: fragments always locate in a 2D array of cells indexed by a pair of grid integers \((i, j)\), and the rotation is just \(k \times \pi/2\). On such square fragments, pairwise compatibility measurement, such as SSD, MGC, and its variants, can simply consider pixel intensity/gradient consistency along horizontal and vertical directions on the straight boundary. From the global aspect, loop closures can be easily formulated and detected on a 2D grid. Algorithms developed based on these simplifications will not work on general puzzles.

A CNN-based method was explored in [18] recently. Pau-mard et al. designed a neural network to predict fragments relative position, and then a greedy strategy is applied for global composition. However, their method can only tackle the simple puzzles and the number of pieces they solved in their experiments is nine.

### B. Solving Irregular-Shaped Jigsaw Puzzles

Irregular-shaped jigsaw puzzles are composed of arbitrarily cut fragmented pieces. Shredded images or documents are typical and practical cases of such puzzles.

Color information from the boundary pixels was used in building image fragment descriptors for their matching. Amigoni et al. [19] extract color content from the fragments’ boundary outlines, and use them to match and align image pieces. Tsamoura et al. [20] apply a color-based image retrieval strategy to identify potential adjacent fragments, and then use boundary pixel’s color to build the contour feature. The pairwise matching is then computed by finding a longest common subsequence [21] between fragments’ contours. In [22], the texture of a band outside the border of pieces is predicted by image inpainting. An FFT-based registration algorithm is then utilized to find the alignment of the fragment pieces.

Fragment's boundary geometry is also commonly used in building features for fragment matching. Zhu et al. [23] approximate contours of ripped pieces by polygons and use the turning angles defined on the polygons as the geometric feature to match fragments. Liu et al. [1] also use polygons to approximate the noisy fragment contours and then extract vertex and line features along the simplified boundary to match partial curves. Each pairwise matching candidate contains a score to indicate how well the matching is. They use those scores to build a weighted graph and apply a spectral clustering technique to filter out irrelevant matching. Zhang et al. [2] build the polygon approximation on both the geometry and color space, and use ICP to compute potential pairwise matches. Multiple pairwise alignments are stored on a multi-graph, weighted by pairwise matching scores. The global composition is solved by finding a simple graph with maximized compatible edge set through a greedy search.

All these existing puzzle solvers generally follow the a three-step composition procedure: (1) design geometry- or color-based features to describe the fragments; (2) compute the inter-fragment correspondences and/or rigid transformations (alignments) between pieces, rank these alignments using a score; (3) globally reassemble the pieces using acceptable pairwise alignments. However, these existing algorithms not only depend on having well designed features, but also often need parameters carefully tuned. This becomes very difficult in general, as puzzles could have different contents and different complexities, and a set of predetermined handcrafted features and hand-tuned parameters may not work for all the various cases. In most experiments reported in all these existing literatures, the puzzles are relatively simple, and the fragment numbers are smaller than 30.

## III. OVERVIEW

As illustrated in Fig [1] our approach contains three components: pairwise alignment candidates extraction, pairwise compatibility measurement, and global composition. Intuitively,
if following a pairwise matching, two fragments can align (under rigid transformation) with natural geometry and texture transition across the boundary, we consider this as a candidate alignment. But before matching, we do not know whether two fragments are adjacent or not. Therefore, we compute matching between every pair of fragments. We adopt the pairwise matching computation strategy used in [2], which formulates the matching as a partial curve matching problem. Specifically, it contains four steps: (1) Through a revised RDP algorithm [24], approximate the noisy fragment boundary contour into a polygon, whose each line segment has similar color; (2) Match each fragment pair by iteratively estimating all the possible segment-to-segment matches; (3) Refine those good segment-to-segment matches using an ICP algorithm; and (4) Evaluate the pairwise matching score by calculating the volume of well aligned pixels. Between each fragment pair, this matching algorithm produces a set of possible alignments, in which both correct and incorrect alignments exist and the number of incorrect alignments is much bigger than correct ones.

Pairwise compatibility measurement. With pairwise alignment candidates, we still need a reliable compatibility evaluator that can examine these candidates: to keep ones that are probably correct and filter out ones that are likely incorrect. Such a detector could reduce the search space in the next global composition step, and benefit both reassembly robustness and efficiency. In most existing reassembly algorithms, heuristic and handcrafted features and evaluation schemes are designed to measure such a compatibility. For example, in [2], the alignment score is defined as the number of matched pixels (i.e., after the ICP transformation on fragments, pixels that have similar color, opposite normal, and small spatial distance). In [20], the matching score is defined as the weighted length of the extracted longest common subsequence. However, these manually designed evaluators do not always work well for different puzzles, and the parameter tuning is often difficult. Hence, in this work, we design a pairwise compatibility detector (classifier) using a CNN network. This network is trained to identify whether the stitching of an image fragment pair under a specific pairwise alignment is correct or not.

Global Composition. Even with a good pairwise compatibility detector, misalignments due to local ambiguity are sometimes inevitable. Such errors need to be handled from a global perspective. We use mutual consensus of many pieces' poses to prune the pairwise alignments and globally compose the fragments. A widely adopted consensus constraint is loop closures: correct pairwise alignments support each other spatially and their relative transformations compose to identity along a closed loop if they are considered in a loop. Such a loop closure constraints have been widely applied on refining or pruning pairwise alignments in many vision-based SLAM and environment reconstruction like [11], [12], [25]. However, enforcing loop closures in jigsaw puzzle solving problem is more challenging than it is in these SLAM and reconstruction tasks. First, in reassembly, outliers dominate inliers, and furthermore, due to the significantly smaller overlap between adjacent pieces and the existence of small loops, incorrect alignments could sometimes form closed loops. Simply applying greedy loop closures, which is a common strategy in the state-of-the-art SLAM systems, will not work reliably. Second, besides loop closure constraints, the global composition should also prevent any inter-fragment intersection, and this extra constraint cannot be formulated in a continuous closed form together with the loop closure constraint. Enforcing it also makes the solving significantly more expensive. Inspired by [12], [13], we develop closed loops searching and merging algorithms on a general multi-graph. These algorithms have promising application on not only fragment reassembly, but also general SLAM and environment reconstruction when datasets are sparsely sampled.

IV. PAIRWISE COMPATIBILITY MEASUREMENT

Pairwise matching computation results in both correct and incorrect alignments. Although we could develop a composition algorithm to prune the incorrect alignments using mutual consensus in the final global step, it is computationally expensive. When the puzzle is complex, fully relying on global pruning is prohibitive. An effective candidates filtering and pre-selection tool is important to both composition efficiency and reliability.

In most existing puzzle solving algorithms, manually designed pairwise matching scores and heuristic thresholds based on experiments or parameter tuning are often used for this filtering. Unfortunately, building handcrafted features and weighting parameters to evaluate the stitching of various puzzles (that have different contents and geometry/size complexity) is in general very difficult. Because evaluating whether the stitched content exhibits a natural transition involves analysis in not only geometry, color, texture, but also higher-level semantics.
Therefore, instead of handcrafted detectors, we formulate the problem of whether an alignment is correct or not as a binary classification problem, and train a CNN to do this pairwise compatibility measurement.

A. A CNN Detector for Compatibility Measurement

1) Overview and Main Idea: Popular convolutional neural networks architectures such as AlexNet [26], VGG16 [27] and ResNet [28] have been developed and applied in many image classification and recognition tasks. But these classic tasks are different from fragment stitching compatibility measurement on two aspects: (1) Instead of dealing with rectangular images that often contain relatively complete contents, in this problem, the shape of image fragment is irregular, and its content is often very local and incomplete. (2) In conventional classification or recognition, features from local to global, and from all over the images may contribute to classification. However, in fragments composition problem, features extracted near the stitching region which could characterize the image content transition smoothness are most important.

Therefore, we design a new CNN network, integrating the desirable properties from the structures of residual block [28] and RoIAlign [29]. The intuition comes from two observations.

First, for pairwise compatibility measurement, calculating complicated and deep feature maps is often unnecessary, because the content in a image fragment is local and incomplete. So unlike many other high-level recognition tasks, it does not need be built upon deep and complicated feature map stacks. Therefore, we build a relatively deep network (29 CONV layers in total, see below for detail), but make the stack of feature map shallow (with the maximum feature map stack being 128). The number of parameters in such net structure are much fewer than the popular deep backbone nets such as ResNet [28], and the training (optimization iteration) and testing (evaluating) is significantly faster.

Fig. 2. Smooth content transition in stitching regions (green boxes). Non-smooth content transition in other regions, such as tree branches versus the background sky, is less important in evaluating the stitching compatibility.

Second, the key clues in differentiating correct and incorrect alignments should locate near the stitching boundary. Correct pairwise reassembly will preserve smooth transition in contents. A simple low-level content smoothness could be the smoothness of color intensity or smoothness of gradients (especially in regions that do not have complicated textures). Fig. 2 illustrates two examples. In the stitching region (the green boxes), natural transitions are important. But in other regions, less smooth transitions in contents may be ignorable.

Therefore, on the one hand, the CNN should be trained to observe the smoothness of contents transition, on the other hand, the network should focus its attention on the critical stitching regions. This will not only speed the learning procedure, but also improve the recognition accuracy.

2) Network Architecture Design: Focusing on Region of Interest (RoI). The region of interest alignment method (RoIAlign) [29] is applied to transfer the attention of calculation to the stitching regions. RoIAlign is a pooling layer in the neural network to extract feature maps from each specific region of interest (RoI). To smoothly calculate the specific output size of this layer, a bilinear interpolation is used. We apply the RoIAlign in the last pooling layer and still use the conventional max pooling in shallow pooling layers. There are two benefits on this design: (1) The max pooling in shallow layers can effectively increase receptive field. (2) In the last RoIAlign, only features located in ROIs are calculated and the final classification (in fully connected layers) is performed mainly based on the stitching regions. Based on this design, the network training is performed on not only local stitching regions but also the whole image context. The ROIs and RoIAlign have been demonstrated in Fig. red boxes.

Network Architecture. The input of the neural network is a series of 160 × 160 × 3 images with corresponding weights and bounding box coordinate which covers the abutted area between two images. The original input image is processed by a convolutional block and 12 residual blocks. The convolutional block (CB) applies the following modules:

(1) Convolution of 8 filters, kernel size 3 × 3 with stride 1.
(2) Batch normalization [30].
(3) A rectified linear unit (ReLU).

The residual block (RB(r, h)) has two parameters: the depth of input r and the depth of output h. Each residual block has below architecture:

(1) Convolution of h filters, kernel size 3 × 3 with stride 1.
(2) Batch normalization.
(3) A rectified linear unit (ReLU).
(4) Convolution of h filters, kernel size 3 × 3 with stride 1.
(5) A skip connection.
   If r = h, then directly connect input to the block.
   If r ≠ h, then apply Convolution of h filters of kernel size 3 × 3 with stride 1, and following batch normalization.
(6) A rectified linear unit (ReLU).

The output of the residual towel is passed into either a max pooling or a RoIAlign. The RoIAlign crops and resizes the feature map, which locate in the input bounding box, to 4 × 4 small feature map by using bilinear interpolation. Finally two fully connection layers convert the feature map to the one-hot vector (i.e. a 2 × 1 vector). Fig. 3 illustrates the complete network architecture. Several experiments in Section VI demonstrate the effectiveness of this new network.

3) Training the Detector: Synthesizing shredded images. To train our CNN detector, we build a shredding program to simulate the fragmentation of a given image. The shredding is controlled by three parameters: (1) puzzle complexity (number of cuts to generate), (2) randomized cutting orientation, (3)
B. Solving Data Imbalance

perturbations along the cutting curve. With this generator, we can synthesize big amount of fragmented image data for training and testing.

To train the CNN detector, first, synthesized image fragments are aligned using the aforementioned pairwise matching algorithm; these alignments are used to stitch two image fragments; then, the stiched images are fed into the CNN to train or test. The output of network is normalized by softmax function which represents the probability of true-and-false classification. We call this probability the alignment score $\gamma$. Fig. 4 illustrates some examples of classification results.

![Fig. 4: Some CNN classification results. $\gamma$ is the output probability/score. G.T. stands for the groundtruth. Typically, we use a score threshold 0.5 to distinguish correct and incorrect alignment. Here, (b) is misjudged by the CNN.](image)

(a) $\gamma = 0.00$  
G.T. = False  
(b) $\gamma = 0.52$  
G.T. = False  
(c) $\gamma = 0.78$  
G.T. = True  
(d) $\gamma = 0.94$  
G.T. = True

Fig. 4. Some CNN classification results. $\gamma$ is the output probability/score. G.T. stands for the groundtruth. Typically, we use a score threshold 0.5 to distinguish correct and incorrect alignment. Here, (b) is misjudged by the CNN.

B. Solving Data Imbalance

![Fig. 5: Examples of consistent and inconsistent transition. A correct pairwise alignment (a) often has consistent transition across the stitching region in both texture content and geometry. The incorrect alignments usually exhibit certain inconsistency, either in texture/color content (b), or in geometry (c), along the stitching boundary.](image)

(a)  
(b)  
(c)

Fig. 5. Examples of consistent and inconsistent transition. A correct pairwise alignment (a) often has consistent transition across the stitching region in both texture content and geometry. The incorrect alignments usually exhibit certain inconsistency, either in texture/color content (b), or in geometry (c), along the stitching boundary.

When training the CNN detector, we compute many pairwise alignments from synthesized image fragments. However, these computed alignments are imbalanced. Between each pair of fragments, there is only one correct alignment, but the partial matching could potentially find many non-intersected alignments. These incorrect alignments, which indicate various non-compatible transitions, are valuable for enhancing the detector’s abilities of generalization and recognition. We call such true-alignment versus false-alignment imbalance as between-class imbalance.

Furthermore, among the false alignments, there are errors due to inconsistent boundary geometry and inconsistent image contexts. In the alignment candidate calculation, the original fragment boundary is approximated by a polygon and the alignments (transformations) are computed by matching polygon edge pairs. Much more inconsistent geometry alignments, than inconsistent image content alignments, will be generated. Fig. 5 illustrates examples of these two types of inconsistency. (b) shows a typical context inconsistency in which two color-unrelated fragments were stitched although geometrically the stitching fits very well. (c) shows a geometry inconsistency case where the color context seems to transits well, but the stitching is not very geometrically desirable.

Since the majority of synthesized undesirable alignments are due to geometry inconsistency, they could dominate the learning procedure. Also, the feature that describe image context inconsistency is harder to learn, because the image context could significantly vary from one image to another. Therefore, if we train a CNN directly, the detector tends to be dominated by the majority class of geometry inconsistency, and misses the detection on context inconsistency. The detector will achieve high overall accuracy but low precision. We call this type of imbalance as within-class imbalance.

Although data imbalance problems are related to our synthesis strategy, it is a general and fundamental issue, and is difficult to avoid and overcome by only improving synthesis strategy. In fact, although such geometry inconsistency versus image context inconsistency is the simple imbalance we observed, we don’t know whether within each class of inconsistency, whether there are other minor sub-classes that could be dominated by other false alignments. We need a general strategy to tackle this within-class imbalance.

1) Solving between-class imbalance: Strategies to deal with imbalanced training data for CNN can be categorized into data-level and classifier-level approaches [31], [32]. Data-level approaches modify the original training data by either (1) oversampling, which randomly replicates some minority classes, or (2) undersampling, which randomly remove some majority classes. Classifier-level approaches adjust the objective function or classifier accordingly. For example, probability distribution on imbalanced classes can be computed, then compensated by assigning different weights to different classes [33], [34].

Undersampling data from the majority classes is an easiest approach and it could also desirably improve the efficiency of the training procedure as a smaller training dataset is considered. However, as various incorrect alignments are valuable in training robust detector, we find that through providing more comprehensive alignment data, oversampling the minority class can improve the final classification accuracy. Furthermore, according to Buda et al. [32], oversampling is generally more effective for CNN networks, and will not cause overfitting problem (which was an issue in classic machine learning models). Therefore, considering both the training accuracy and efficiency, we apply both oversampling and undersampling on the synthesized alignment datasets. In our
experiments, we oversample the original positive datasets by 20 times, then randomly downsample the result to a half. Our final training dataset contains approximate 600$k$ alignments (stitched images), in which 70% and 30% are false and true alignments respectively.

2) Solving within-class imbalance: There is also a within-class imbalance in the data. This imbalance problem is a more difficult issue to tackle. As discussed in Section IV-B during pairwise matching, much more false alignments with geometric inconsistency are generated, than false alignments with image context inconsistency. But we do not know which type of inconsistency exists on a specific false alignment, even with the help of groundtruth. Therefore, this imbalance cannot be eliminated through data over/under-sampling or re-weighting the objective function.

The boosting methods, such as Adaboost [35], provides an effective mechanism in solving such imbalance. The boosting method combines multiple weak learners. Each weak learner is trained on the data where the previous weak learners perform badly to complement and fortify overall result. Such a classifier ensemble strategy is suitable for our within-class imbalance problem. Mis-predicted data from the previous learners usually belong to the minority category, and these data will be assigned with a bigger weight in the next learner training.

**Binary classification boosting.** Given training data \(\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}\), \(x_i \in \mathcal{X}, y_i \in \mathcal{Y} = \{-1, 1\}\), to find a mapping \(f : \mathcal{X} \rightarrow \mathcal{Y}\), the Boosting classifier \(f\) uses several standalone learners (i.e., weaker classifiers).

\[
f(x) = \sum_{k=1}^{K} \alpha_k G_k(x). \quad (1)
\]

where \(G_k(x)\) is the \(k\)-th learner, and \(\alpha_k\) is the weight which measures how important this learner is in the final classifier. The classification error can be measured by an exponential loss function [35]:

\[
L(y, f(x)) = \exp(-yf(x)). \quad (2)
\]

If we have a boosting classifier \(f_{k-1}(x)\) from the first \(k-1\) learners, finding the best \(k\)-th learner \(G_k(x)\) to have a minimized loss of Eq. (2) reduces to

\[
(\alpha_k^*, G_k^*(x)) = \arg\min_{\alpha,G} \sum_{i=1}^{n} \exp(-y_if_k(x_i))
\]

\[
= \arg\min_{\alpha,G} \sum_{i=1}^{n} \exp(-y_if_{k-1}(x_i) + \alpha G(x_i))
\]

\[
= \arg\min_{\alpha,G} \sum_{i=1}^{n} w_{k-1,i} \exp(-y_i \alpha G(x_i)) \quad (3)
\]

where \(w_{k-1,i} = \exp(-y_if_{k-1}(x_i))\). If \(w_{k-1,i}\) is larger, it means the previous \(k-1\) ensemble result is undesirable in data \((x_i, y_i)\). This data is assigned a heavier weight for the current learner \(G_k(x)\) training. Therefore, \(w_{k-1,i}\) can be seen as a weight distribution on the training data. The misjudged data will be amplified on the training of next learner.

To this end, we can generally formulate \(w_{k,i}\) as

\[
w_{k,i} = \exp(-y_i f_k(x_i)) = w_{k-1,i} \exp(-y_i \alpha_k G_k(x_i)). \quad (4)
\]

Eq. (4) means \(w_{k,i}\) is only related with the current learners ensemble. \(w_{k-1,i}\) is a constant in the \(k\)-th calculation. This Eq. (4) explains how to update data weight distribution \(\{w_{k,1}, w_{k,2}, \ldots, w_{k,n}\} \in D_k\) in the \(k\)-th iteration according to the misclassified data.

Since \(w_{k-1,i}\) is a constant for solving \(G_k^*(x)\) and \(y_i, G(x_i) \in \{-1, 1\}\), for \(\forall a > 0\) we can separately formulate the best \(G_k^*(x_i)\) in Eq. (3) as

\[
G_k^*(x) = \arg\min_G \sum_{i=1}^{n} w_{k-1,i} I(y_i \neq G(x_i)) \quad (5)
\]

\[
I(y_i \neq G(x_i)) = \begin{cases} 1 & \text{if } y_i \neq G(x_i) \\ 0 & \text{if } y_i = G(x_i) \end{cases}
\]

Bringing \(G_k^*\) of Eq. (5) into Eq. (3) we have

\[
\alpha_k^* = \arg\min_{\alpha,G} \sum_{i=1}^{n} \exp(-y_i \alpha G_k^*(x_i)) = \arg\min_{\alpha,G} \left[\exp \left(-\alpha \sum_{i=1}^{n} w_{k-1,i} I(y_i \neq G_k^*(x_i)) + \alpha \sum_{i=1}^{n} w_{k-1,i}\right)\right] \quad (6)
\]

Finally, combining Eq. (3) and Eq. (6), we have

\[
\alpha_k^* = \frac{1}{2} \log \frac{1 - E_k}{E_k}. \quad (8)
\]

Equations (5) and (8) tell us how to optimize the current learner in the \(k\)-th iteration. Based on the above derivations, we can design the boosting algorithm for CNN training.

**CNN boost training.** The Eq. (5) in common boosting algorithms is discrete class tag, but the output of our CNN is continuous value. Therefore, to optimize the CNN, we re-formulate Eq. (5) using cross-entropy:

\[
G_k^*(x) = \arg\min_G \sum_{i=1}^{n} w_{k-1,i} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\right] \quad (9)
\]

where \(y_i\) is the groundtruth and \(\hat{y}_i\) is the estimation of CNN after softmax normalization.

During training phase, the class tags in Eq. (9) is \(\{0, 1\}\) instead of \(\{-1, 1\}\). During validation phase, since the output of CNN is a probability of correct alignment, we can use Eq. (10) to convert the probability to a discrete classification result and then apply boosting iteration without any violations.

\[
G_k^*(x) = \begin{cases} -1 & \text{if } \hat{y}_i < p \\ 1 & \text{if } \hat{y}_i \geq p \end{cases} \quad (10)
\]

where \(p\) is a probability threshold (we set \(p = 0.5\) in all of our experiments). The entire training procedure can be summarized in Algorithm 1 and the whole training procedure is illustrated in Fig. 6.
enforce the global loop closure constraints and prune incorrect pairwise alignments.

We call the first strategy as Greedy Loop Closing (GLC). Instead of performing traditional greedy selection on edges (alignments), the greedy selection is conducted in the level of loops. Therefore, high-score edges that violates loop closure will not lead to local minima, and GLC is more robust than existing edge-based searching algorithms. Furthermore, we also develop a second strategy, called Hierarchical Loop Merging (HLM). Instead of greedily selecting closed loops like GLC does, the decision will be made after hierarchical merging operations. Therefore, incorrect local loops that are closed but incompatible with other big loops will not lead to local minima, and HLM is more robust than GLC in solving complicated puzzles.

V. GLOBAL COMPOSITION MAXIMIZING LOOP CONSISTENCY

After pairwise compatibility measurement, a major part of incorrect pairwise alignments have been filtered out. But between many fragment pairs, we still preserve more than one potential alignments. This is because (1) the trained compatibility classifier has not yet reached perfect accuracy, and (2) there is pairwise alignment ambiguity that can not be ruled out locally. Fig. 7 illustrates such an example. Both alignments in Fig. 7(b) and (c) seem to produce natural stitching. Therefore, setting a too high threshold to strictly reject alignments (or even just keep one alignment per pair) may not be a good idea. Instead, we keep several pairwise alignments between a fragment pair, then handle their pruning through this global composition by enforcing groupwise consensus. Most existing global composition algorithms adopt certain types of greedy strategies such as the best-first, spanning-tree growing, or their variants [2, 16, 17], if incorrect/ambiguous alignments have higher matching or compatibility scores and are picked to occupy the positions that belong to other correct pieces, the final composition will fail because of such a local minimum.

Loop closure has been widely adopted as a global consensus constraint in SLAM [10], 3D reconstruction [11, 12], and global structure-from-motion [36, 37], and has demonstrated effective in these tasks. Here, we develop two new strategies to

Algorithm 1 CNN boost training

Input: The training data \( \mathcal{X}, \mathcal{Y} \), the number of learners \( K \)
Output: The compatibility detector/classifier \( f(x) \)

1. Initialize training data weight \( D_0 = \{w_{01}, w_{02}, \ldots, w_{0n}\} \), where \( w_{0i} = \frac{1}{n}, i = 1, 2, \ldots, n \).
2. for \( k = 1 \) to \( K \)
   - Train a network learner \( G_k(x) \) to optimize Eq. (9).
   - Convert to discrete classification result, using Eq. (10).
   - Calculate \( \alpha_k \), using Eq. (3).
   - Update the weight distribution on training data \( D_k = \{w_{k1}, w_{k2}, \ldots, w_{kn}\} \) using Eq. (4).
3. end for

The final classifier is \( f(x) = \sum_{k=1}^{K} \alpha_k G_k^*(x) \)

A. Terminologies and Formulations

We use a directed multi-graph \( G = \{\mathcal{V}, \mathcal{E}\} \) to store all the image fragments and pairwise alignment candidates. Each vertex \( v_i \in \mathcal{V} \) corresponds to an image fragment and a 2D rigid transformation matrix, or pose, \( X_i \in \mathcal{X} \). Between each pair of vertices \( (v_i, v_j) \), there are one or more edges. Each such edge \( e_{i,j,k} \in \mathcal{E} \) corresponds to a pairwise alignment, where \( i, j \) are vertex indices and \( k \) indicates the \( k \)-th potential alignments between them. Every edge \( e_{i,j,k} \) is associated with a 2D rigid transformation matrix \( T_{i,j,k} \), stitching fragment \( i \) to fragment \( j \). For each \( T_{i,j,k} \) we have a compatibility score \( \gamma \), which is the output of the CNN classifier defined in the last section, indicating the probability of its correctness. Many loops \( \{l_1, l_2, \ldots, l_t\} \) can be found in graph \( G \). A loop closure constraint is formulated on a loop \( l \) as

\[
\prod_{(i,j,k) \in l} T_{i,j,k} = I
\]

where \( I \) is the identity matrix. A loop that satisfies this constraint is called a closed loop. Note that, while each edge \( e_{i,j,k} \) is directed, a loop could contain it in its reversed direction \( e_{j,i,k} \). In that case, we shall use its reversed transformation \( T_{j,i,k} = T_{i,j,k}^{-1} \) in evaluating the loop closure constraint. In the following, without causing ambiguity, we may simplify the discussion of loop closure on an undirected graph.

Fig. 8 illustrates an example of simple multi-graph. There are multiple small loops whose lengths are 3 or 4, such as \( (1 \to 2 \to 5 \to 1), (2 \to 3 \to 8 \to 7 \to 2) \),
Fig. 8. Loop closure constraints on a directed multi-graph. (a) A simulative jigsaw puzzle with 9 fragmented pieces. (b) The corresponding graph model is a directed multi-graph.

(2 → 7 → 6 → 5 → 2). If \( T_{121} * T_{251} * T_{151}^{-1} = I \), then those transformations on \((1 → 2 → 5 → 1)\) form a closed loop, and we consider this group of transformations to be mutually consistent. The alignments that satisfy such a loop closure constraint are considered more reliable than those individual pairwise alignments receiving high local alignment scores. This loop closure constraint provides a more global and reliable measure over local pairwise compatibility measures.

**Induced loops and mergeable loops.** A loop is called a hole or an induced loop, if no two vertices of it are connected by an edge that does not itself belong to this loop. In Fig. 8 (b), \((2 → 5 → 7 → 2)\) and \((7 → 5 → 6 → 7)\) are two induced loops. \((2 → 5 → 6 → 7 → 2)\) is not an induced loop because \((5 → 7)\) is connected through a path (edge) that does not belong to this loop. If one common edge \(e_{i,j,k}\) can be found in closed loops \(l_p\) and \(l_q\), then \(l_p\) and \(l_q\) are adjacent or mergeable. \((2 → 5 → 7 → 2)\) and \((7 → 5 → 6 → 7)\) are mergeable because \((5 → 7)\) is their common edge. (As mentioned above, we consider this on an undirected graph to simplify the notation without causing ambiguity). Merging induced loops results in more complicated loops.

**Composition with Loop Closures.** Based on the above definitions, we can formulate the global composition as an optimization problem in Eq. \((12)\).

\[
E(X, U) = \min \sum_{i,j,k} u_{i,j,k} f(X_i, X_j, T_{i,j,k}) + w_{i,j,k}(1 - u_{i,j,k})
\]

s.t. \(\forall u_{i,j,k} \in \{0, 1\}\), and no fragment intersection

(12)

where \(u_{i,j,k} \in U\) is an indicator variable: \(u_{i,j,k} = 1\) means edge \(e_{i,j,k}\) and associated transformation \(T_{i,j,k}\) are selected, and \(u_{i,j,k} = 0\) otherwise. \(w_{i,j,k}\) is a penalty weight. \(f(X_i, X_j, T_{i,j,k})\) measure the inconsistency between a selected pairwise alignment \(T_{i,j,k}\) and the final poses \(X_i, X_j\) on the nodes. Specifically, it can be formulated using a nonlinear least-square function \([38]\),

\[
f(X_i, X_j, T_{i,j,k}) = e(X_i, X_j, T_{i,j,k})^T \Omega_{ij} e(X_i, X_j, T_{i,j,k})
\]

(13)

where \(e(X_i, X_j, T_{i,j,k}) = \phi \left[T_{i,j,k}^{-1}X_i^{-1}X_j\right]\) and the operator \(\phi\) converts a \(3 \times 3\) transformation matrix to a \(3\)-dimensional vector representing the translation and rotation. If \(T_{i,j,k}^{-1}X_i^{-1}X_j\) is identity, then the output is a zero vector. \(\Omega_{ij}\) is a \(3 \times 3\) weight matrix.

The objective function of Eq. \((12)\) is defined based on the following intuition. Our goal is to solve pairwise alignments selection \(U\) and all of image fragments pose \(X\). In jigsaw puzzle solving, correct pairwise alignments are always compatible with each other, while incorrect ones are prone to produce pose violations. In other words, if an incorrect alignment is selected, it will bring more inconsistencies than a correct alignment. Big pose inconsistency will be reflected by a big error value of \(f(X_i, X_j, T_{i,j,k})\). In this case, to minimize terms \(u_{i,j,k} f(X_i, X_j, T_{i,j,k}) + w_{i,j,k}(1 - u_{i,j,k})\), we tend to discard this edge and get rid of \(f(X_i, X_j, T_{i,j,k})\) by setting \(u_{i,j,k} = 0\) and accept the penalty weight \(w_{i,j,k}\). Therefore, selecting incorrect alignments will bring more penalties than selecting correct alignments. Minimizing Eq. \((12)\) is equivalent to select as many mutually consistent alignments as possible. The loop closure constraint is implicitly included in Eq. \((12)\). Since the edges-alignments are consistent in a closed loop, the optimization of Eq. \((12)\) can be seen as finding edges-alignments to maximize the number of compatible loops.

**B. A Greedy Loop Closing (GLC) Algorithm**

Problem \((12)\) is highly non-linear and has many local minima. Finding its global optimal solution is essentially NP-hard \([7]\). When the directed multi-graph \(G\) is complicated, enumerating all the possible solutions is prohibitive. Hence, developing an algorithm to find an approximate solution is a more effective strategy in practice.

In most SLAM and image reconstruction problems, registration between consecutive frames are mostly reliable, and loop closure is mainly used to refine the poses and suppress accumulative error. Therefore, loop closures are often formulated on a simple graph, and enforced through a best-first greedy strategy, sometimes followed by pose-graph optimization post-processing \([25]\). However, in fragment reassembly, a big portion of computed pairwise alignments are outliers, hence, most existing solvers are prone to local minima and often fail in composing complicated puzzles.

Unlike existing greedy strategies, which iteratively select the best edge that satisfies loop closure and intersection-free constraints, in this algorithm, we iteratively search for loops and fix each found one if it is closed and introduces no inter-fragment intersection. Specifically, the loop searching routine is done through a Depth-First Search (DFS). It starts from a random edge, and randomly grows by merging adjacent edges, as long as no inter-fragment intersection is detected, until a loop \(l\) is found. Then we check whether \(l\) satisfies the loop closure constraint and the intersection-free (between fragments from \(l\) and fragments that are already fixed) constraint. If \(l\) satisfied both constraints, then we fix this loop by selecting all the edges on this loop (by setting indicators to \(1\)) and discarding all their conflicting edges (by setting indicators to \(0\)). If \(l\) violates any of the two constraints, then \(l\) is invalid, and will be ignored.

We keep performing this loop searching and fixing acceptable loops (selecting their loops), until (1) all the nodes in \(G\) have been connected by selected edges, or (2) the DFS
search has no edge to select, or (3) a maximal searching step \( N \) is reached. After loop searching, if there are nodes that are not connected with fixed edges through loops, we greedily select highest-score and intersection-free edge from the left undecided edges to connect them. Finally, we can calculate all the fragments’ poses \( \mathcal{X} \) using the selected edges/alignments in the final graph.

This greedy strategy is usually efficient because any intersection-free closed loop will be fixed and related conflicted edges will be discarded once a valid loop is found. Compared with existing various best-edge first selection strategies adopted in existing reassembly literatures, this algorithm is less sensitive to local minima caused by single pairwise alignments that have high compatibility score but are incorrect.

However, if incorrect pairwise alignments also form a closed loop, then this strategy may get trapped locally again. If an incorrect-alignments-formed closed loop is selected first and its associate fragments occupy the positions where the correct closed loops should locate, then the reassembly will be incorrect in these regions. Fig. 12 (a) illustrates a failed example of this greedy loop closing algorithm. Here the incorrect closed loop \((5 \rightarrow 8 \rightarrow 9 \rightarrow 5)\) was detected before the correct one \((4 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 4)\), and the incorrect loop occupied the positions that belong to the correct loops. The correct loop is then discarded and this leads to an incorrect reassembly.

To further improve the robustness of the global composition, we also design another algorithm through a hierarchical loop merging strategy.

C. A Hierarchical Loop Merging (HLM) Algorithm

In this strategy, instead of directly fixing a found closed loop, we keep all closed loops we found, and perform the selection through an iterative merging. Since the true closed loops are always mutually compatible and false closed loops lead to violations, the correct solutions can be found by merging operation. As Fig. 9 showed, the merging operation will further check the alignments/edges compatibility, and thus the composition reliability will be fortified. The correct probability will significantly improve with more loops merged.

Fig. 9. The difference between merged loop and single long induced loop. Although both of them have same vertex, merged loop has one more compatible edge/alignment \((2 \rightarrow 5)\). When more loops are merged, more interlocking edges/alignments need to be satisfied. Therefore, the composition reliability will be enhanced.

To merge as many closed loops as possible, the algorithm undergoes a bottom-up merging phase then a top-down merging phase.

1) Bottom-up Merging: We start with small induced loops, whose lengths are 3 or 4. The set of loops found in this initial step is denoted as \( \mathcal{L}^0 \). From \( \mathcal{L}^0 \), we try to search loop pairs that are mergeable (see Sec. V-A for definitions of these terminologies).

We say two mergeable loops \( l_p \) and \( l_q \) are incompatible, if after merging, any of the following conditions are violated:

- **Condition 1 (C1) (Pose Consistency):** If any vertex \( v \) is in both \( l_p \) and \( l_q \), then the pose of \( v \)’s associated fragment, derived either from \( l_p \) or \( l_q \), should be consistent.

- **Condition 2 (C2) (Intersection-free):** For any two different vertices \( v_p \in l_p \) and \( v_q \in l_q \), their derived poses should not make the associated fragment overlap with each other.

When two mergeable loops satisfy both C1 and C2, we can merge them into a bigger loop. This will result in a valid composition locally. If two mergeable loops violates one of these conditions and are incompatible, then merging them leads to an invalid composition. This means at least one of these two loops are incorrect.

We merge loops from \( \mathcal{L}^0 \), and add the new merged loops into a new set \( \mathcal{L}^1 \). Then, iteratively we repeat this procedure to get \( \mathcal{L}^2, \mathcal{L}^3, \ldots, \mathcal{L}^n \) until no more loops can be merged. With the growth of the compatible loops, the probability of these big loops being correct significantly increases. In the last loop set, \( \mathcal{L}^n \), we select a loop that has the highest sum of score and denote it as \( l^* \). \( l^* \) corresponds to the biggest reassembled patch that we have got so far through this bottom-up merging procedure. Fig. 10 gives an illustrative example of this merging procedure.

![Fig. 10. Bottom-up merging. For convenience, we use small squares to represent image fragments whose actual shapes are irregular. In each iteration, we try to merge all mergeable loop pairs in \( \mathcal{L}^k \) and add the merged bigger loop into \( \mathcal{L}^{k+1} \). Merged bigger loops have higher probability to be correct than those smaller loops.](image)

Controlling the Complexity. When merging loops in \( \mathcal{L}^i \), if we enumerate all the possible merges between every pair of loops in each \( \mathcal{L}^i \), then the algorithm’s time and space complexity will both grow exponentially. In a worst case, the \( \beta \) closed loops in set \( \mathcal{L}^i \) could become \( \beta^2 \) loops in \( \mathcal{L}^{i+1} \), and then grow to \( \beta^4 \) in the next level. Therefore, to restrict this exponential growing, in each level \( \mathcal{L}^i \), we restrict the maximal number of merges we try to be a constant number \( \theta_M \) (\( \theta_M \) is set to 500 in all our experiments). Then, in \( \mathcal{L}^3 \), we will at most get \( \theta_M \) loops (most likely, fewer than that as some merge will
be unacceptable and discarded). From all the $\frac{\theta_M(\theta_M-1)}{2}$ loop pairs, we randomly consider $\theta_M$ merges.

2) Top-down Merging: If all the fragments (nodes) are merged into a big loop, then $l^*$ gives us the final composition. But $l^*$ may not contain all the correct pairwise alignments: some correct alignments may not be detected (through pairwise matching) or have relatively weak compatibility, these fragments may need to be stitched onto the main components through individual edge connections. Therefore, we further perform a top-down merging to stitch these left-out fragments (isolated vertices) or sub-patches (sub-loops).

The top-down merging starts with $l^*$ and first check each loop in $L^{n-1}$. If a loop is found to be compatible with $l^*$, we will merge it to $l^*$. Specifically, we define loops $l_1$ and $l_2$ to be valuable to each other, if they are compatible to each other and $l_1$ contains some vertices that are not in $l_2$.

We grow $l^*$ by iteratively merging it with new valuable loops from $L^{n-1}$, then $L^{n-2}$, to finally, loops from $L^0$. Fig. 11 illustrates a procedure of this top-down merging.

![Fig. 11. Top-down merging. Iteratively, we try to merge the current maximal loop $l^*$ with its valuable loops from $L^{n-1}, L^{n-2}, ..., L^0$.](image)

Adding Left Edges after Top-down Merging. Finally, if there are still isolated vertices, which share edges with vertices in the merged $l^*$ but are not merged. We then just perform a greedy growing algorithm, to iteratively pick a highest-scored edge that does not introduce fragment intersection, until no more edge can be further added.

The entire Hierarchical Loop Merging algorithm is summarized in Algorithm 2.

Fig. 12 shows an example in which the GLC algorithm (Section V-B) fails but the HLM algorithm succeeds. With HLM, the incorrect closed loop $(5 \rightarrow 8 \rightarrow 9 \rightarrow 5)$ will be discarded, since it has conflict with the correct loop $(4 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 4)$ during merging. In contrast, in GLC, greedily selecting this incorrect loop leads to an undesirable local minimum and a failure in the reassembly.

3) Accuracy and Complexity Analysis on HLM: The HLM algorithm aims to extract as many compatible loops as possible from the given multi-graph $G$. This is consistent with minimizing Eq. (12). Maximizing the number of compatible loops selected will minimize the second (indicator variable penalty) term, and since these loops are compatible to each other, it does not increase the first term. Such a merging based procedure offers a mechanism to prune false loops, and thus, can better avoids local minima in global composition.

Complexity analysis. In Algorithm 2 we use heap arrays to store vertex and edge indices, and use index sets to represent closed loops. To find a mergeable or valuable loop pair $(l_p, l_q)$, whose lengths are $k_1, k_2$ respectively, we need $O(k_1 + k_2)$ to search common elements within two heaps. For the same reason, the evaluation of condition C1 can be finished in $O(k_1 + k_2)$. The complexity of checking condition C2 is related with image resolution, because we check the fragment intersection on a canvas. If the image fragment composed from a loop has $t$ pixels, then the time complexity is $O(t)$. Next, the loop merging operation will insert one loop’s heap data structure into another, and this can be finished in $O(k_1 + k_2)$. Since the image resolution $t > > k_1, k_2$, the complexity of a loop merging can be estimated as $O(t)$, where $t$ is the pixel number of the puzzle image. In our implementation, we speed up this intersection detection by implementing it using CUDA.

In bottom-up merging, with the complexity control, the double while-loops will be run $n\theta_M$ times, where $n$ is total iteration number. Therefore, the total time complexity of this stage is $O(n\theta_M t)$ (usually, $n \approx 20$ in hundreds of pieces of puzzle). In top-down merging, we will try merging every loop in each level $L^i$ with $l^*$ once. Therefore, with totally $O(n\theta_M)$ loops, the complexity is also $O(n\theta_M t)$. In the final
(a) Composition by Greedy Loop Closing (GLC)

(b) Composition by Hierarchical Loop Merging (HLM)

Fig. 12. GLC and HLM in Global Composition. (a) With the GLC algorithm: once a closed induced loop is found, it is fixed. If such a loop is incorrect, the composition will be wrong. (b) With the HLM algorithm: with the same initial closed loops as (a), correct loops get merged and incorrect ones are eventually discarded as they cannot be merged.

Fig. 13. Convergence Comparisons of Optimization on Learners with versus without RoIs. Learners with RoIs converges faster: at 5000-th iteration (blue boxes), the loss errors of learners with RoIs are significantly smaller. Learners with RoIs also converge to smaller loss errors.

Fig. 14. Precision and recall curves under different net configurations.

VI. EXPERIMENTS

We conducted experiments on two public datasets: MIT datasets [15] and BGU datasets [16]. However, these two datasets only contain a limited set of images. Popular general-purpose image databases, such as ImageNet, are not suitable for testing jigsaw puzzle solving, because most images in these database have relatively low resolution and each often only contains a single/simple object. Therefore, we also create a new benchmark dataset. We use a website spider to automatically download images from the copyright-free website Pexels [39]. 125 downloaded images, under different categories (e.g. street, mountain, botanical, etc), were randomly selected as training (100) and testing (25) data. These images are randomly cut to generate puzzles of 36 pieces, 100 pieces. We denote this set of data as TestingSet1. We also use 5 additional high-resolution images to create challenging puzzles, each of which have around 400 pieces. We denote this set of data as TestingSet2. We have released our training and testing datasets in https://github.com/Lecanyu/JigsawNet.

A. Evaluating the CNN Performance

We implemented the CNN using TensorFlow [40] and trained it on the aforementioned 100 training images in our own dataset. After randomly partitioning all these images into pieces, we calculated pairwise alignments between every pair of fragments, and got around $600k$ alignments. These alignments, together with (1) the RoI bounding box information (which can be calculated from the alignments), and (2) a label indicating whether the alignment is correct or not (such information is available as we have the groundtruth on each image). We use Adam [41] as the optimization solver and set the batch size to 64 and learning rate to $1e^{-4}$. The loss function is built by combining a cross-entropy term and an $l_2$-regularization term whose weight decay is $1e^{-4}$. The number of training iteration is $30k$ for every learner. The final evaluation is built up using 5 learners.

Using RoI not only speeds up the training convergence but also enhances the detector’s accuracy. As Fig. 13 shows the convergence rates of the two learners (with versus without RoI component). With RoI, the training converges much faster and reaches smaller loss. For example, at the 5k-th iteration (purple boxes), the losses in (a)(c) are 0.04 and 0.23, and the losses are 0.1 and 0.5 in (b)(d), respectively.

greedy selection, the time complexity is linear to the remaining edges. So if the multi-graph $G$ has $e_g$ edges, the complexity is bounded by $O(e_g t)$. In summary, the overall complexity of HLM is $O(nθM + e_g t)$.

Training. We implemented the CNN using TensorFlow [40] and trained it on the aforementioned 100 training images in our own dataset. After randomly partitioning all these images into pieces, we calculated pairwise alignments between every pair of fragments, and got around $600k$ alignments. These alignments, together with (1) the RoI bounding box information (which can be calculated from the alignments), and (2) a label indicating whether the alignment is correct or not (such information is available as we have the groundtruth on each image). We use Adam [41] as the optimization solver and set the batch size to 64 and learning rate to $1e^{-4}$. The loss function is built by combining a cross-entropy term and an $l_2$-regularization term whose weight decay is $1e^{-4}$. The number of training iteration is $30k$ for every learner. The final evaluation is built up using 5 learners.

Using RoI not only speeds up the training convergence but also enhances the detector’s accuracy. As Fig. 13 shows the convergence rates of the two learners (with versus without RoI component). With RoI, the training converges much faster and reaches smaller loss. For example, at the 5k-th iteration (purple boxes), the losses in (a)(c) are 0.04 and 0.23, and the losses are 0.1 and 0.5 in (b)(d), respectively.
TABLE I
Classification results on different network configurations. TP, TN, FN, and FP represents the number of true positive, true negative, false negative, and false positive, respectively. The baseline algorithm is the original network without using boosting or RoI. Boosting indicates that the final classification comes from the ensemble strategy. RoI means that the RoIAlign layer is applied to replace the standard pooling layer. All the networks in this comparisons use the same hyperparameters.

| Baseline | Boost | Boost+RoI |
|----------|-------|----------|
| TP TN FN FP Proc. Recall |
| 1084 48149 637 24 63.2% 97.8% |
| 1093 48404 382 15 74.1% 98.6% |
| 1075 48565 221 33 82.9% 97.0% |
| 1087 48589 197 21 84.7% 98.1% |
| TestingSet1 |
| Baseline | Boost | Boost+RoI |
| 3848 202433 2386 102 61.7% 97.4% |
| 3879 203337 1482 71 72.4% 98.2% |
| 3880 204031 788 150 82.8% 96.2% |
| 3855 204156 663 95 85.3% 97.6% |
| TestingSet2 |
| Baseline | Boost | Boost+RoI |
| 2421 711532 15196 32 13.7% 98.7% |
| 2396 721564 5155 57 31.7% 97.7% |
| 2349 725185 1534 104 60.5% 95.8% |
| 2362 726049 670 91 77.9% 96.3% |

Testing. The testing is performed on the public datasets and our testing benchmarks. We compared the detector’s performance on using different network configurations, i.e., with or without using RoI and boosting. We calculated the recall and precision to evaluate the classification (compatibility detection) results. Fig. 14 shows the recall and precision curves with the four different configurations. Table I illustrates the classification result statistics on the three testing datasets. Our proposed strategy that integrates both RoI and adaptive boosting overall performs the best. It results in the best precision than the other three, and the second best recall (only slightly worse than the boosting-only strategy).

B. Comparison of Pairwise Compatibility Detection with Existing Strategies

We compared our approach with two representative methods, Tsamoura et al. [20], and Zhang et al. [2]. Pairwise alignments in [20] are computed using boundary pixel color information and a longest common subsequence (LCS) algorithm. Pairwise alignments in [2] are computed using contour geometry and an ICP registration. Other matching algorithms in literatures can be considered as variants of these two approaches. In these algorithms, a matching/alignment score is usually produced as the output of the partial matching, and it is used to prune locally good/bad alignments. We normalize these algorithms’ scores to [0, 1] so that they can be compared with our network’s classification output. Then use different thresholds to draw precision and recall curves. The results are illustrated in Fig. 15 (a).

We also calculate the precision and recall values in Fig. 15 (b) by greedily selecting the highest score alignment to reassemble until all of fragments are connected. This greedy strategy can directly reflect whether the scoring mechanism is desirable. From these experiments, we can tell that the proposed compatibility detector significantly improves the accuracy and outperforms existing scoring mechanisms in evaluating pairwise alignments.

C. Comparison of Global Reassembly Results

Finally, but most importantly, we evaluated and compared the overall composition performance on all our testing datasets, where the puzzles have various complexity, from simple ones with 9 pieces to complex ones that have around 400 pieces.

In addition to the comparison with the aforementioned solvers [2] and [20], we also compared the puzzle solving results obtained from using different global composition strategies. Specifically, first, we implement the commonly adopted best-first (BF) strategy (which iteratively picks and stitches the best non-intersecting compatible pairwise alignment). This strategy and its variants are commonly adopted in many existing jigsaw puzzle solvers [1][16]. We compare it with our greedy loop closing (GLC) and hierarchical loop merging (HLM) algorithms. All these three strategies, BF, GLC, and HLM, use the same CNN compatibility detector to prune raw pairwise alignments, and only differ in the global composition strategies. Hence, this comparison can also be viewed as an evaluation of our proposed global composition algorithms.

To quantitatively measure the reassembly result, we consider the following metrics, some of which were also used in evaluating square puzzle solvers [15]. [17].

- **Pose Correctness Ratio (PCR):** the ratio of fragments whose final poses match the ground truth (i.e., the deviation is smaller than a threshold: here the rotation and shift errors are within 5° and 100 pixels).
- **Alignment Correctness Ratio (ACR):** the ratio of correctly selected pairwise alignments in the final composition.
- **Largest Component Ratio (LCR):** the size (ratio) of the biggest recomposed component.

All experiments were performed on an Intel i7-4790 CPU with GTX 1070 GPU and 16 GB RAM. The evaluation metrics are reported in Tables II - V. The Avg. time indicates the average time needed in solving each puzzle in that testing dataset. Some composition results are illustrated for a side-by-side comparison in Figs. 17 - 20.

We can see from these results that [20] and [2] worked for simpler puzzles (e.g., the number of pieces is small). With the increase of the puzzle complexity, their performance decrease dramatically. In contrast, our algorithm is stable in solving these big puzzles. Also, with our CNN compatibility detector,
even adopting simple greedy composition often produces good reassemblies, especially for small puzzles (fragments are bigger and less ambiguous). Compared with the greedy strategy, the GLC and HLM strategies produce more reliable results because of the usage of loop closure constraints.

**GLC versus HLM.** In previous experiments, the performance difference between GLC and HLM seems small. This is because CNN has filtered massive incorrect alignments, the number of remaining incorrect closed loops becomes small. To verify this observation, we use an experiment that discards our CNN detector and only uses the pairwise matching score of [7] to prune alignments in the local phase. The experiments were performed on the above MIT dataset where images were partitioned into just 9 pieces. The results are reported in Table VII. When there are many incorrect alignments, incorrect closed loops would also appear. In such scenarios, the HLM strategy outperforms the GLC algorithm. Therefore, when dealing with easier puzzles in which there are fewer incorrect/ambiguous alignments, GLC is suitable and it is more efficient. But when dealing with big and difficult puzzles, which have many small pieces and their alignments become highly unreliable, HLM offers a more robust global composition.

**Comparisons with Other Puzzle Solvers.** Without access to source/executable codes, we were not able to perform experiments using other notable solvers, such as [1] and [23]. However, we expect these solver would perform similarly to [20] and [2] in complex puzzles. Locally, these algorithms are also built upon handcrafted geometry or color based fragment descriptors and pairwise matching schemes. In the global composition phase, they still use variants of greedy edge growing strategies, which heavily rely on the pairwise matching scores which are often ambiguous and unreliable when puzzles become complicated.

### VII. Failure Cases and Limitations

Our global composition algorithm degenerates to greedy edge growing strategies if there are no sufficient loops. In such cases, when reassembling ambiguous pieces or sub-patches, the loop closure based composition could also be sensitive to the accuracy of local alignments during the greedy selection phase. Fig. 16 demonstrates such a failure case. The correct sub-patch does not have many alignments computed with its neighboring pieces, and no loop was found to link it with the main patch. Without loop closure, the greedily selected alignment happens to be incorrect and a wrong piece is stitched.

Two reasons can lead to the generation of insufficient closed loops: (1) pairwise alignment proposal algorithm misses some correct alignments. (2) The CNN misjudges some good alignments. To analyze them, we checked the pairwise calculation and CNN classification results. In this puzzle, among the total 760 correct pairwise alignments, only 501 (66%) were extracted. And within these 501 correct alignments, 32 were misjudged by the CNN (i.e. Recall = 93.6%). Also, among the total 153 incorrect pairwise alignments, 152 incorrect alignments were misjudged by the CNN (about 0.1% false positive). Therefore, the big amount of false alignments and missed correct alignments from the pairwise alignment extraction step seems to be the major reason. From this analysis, we can conclude that a better pairwise alignment computation algorithm would further improve the reassembly performance.
VIII. CONCLUSIONS

We developed a new fragment reassembly algorithm to restore arbitrarily shredded images. First, we design a first CNN based compatibility detector to judge whether an alignment is correct, by evaluating whether the stitched fragment looks natural. Second, we developed two new global composition algorithms, GLC and HLM, to improve the composition using mutual consistency from loop closure. With these two technical components, our algorithm has greatly outperformed the existing reassembly algorithms in handling various jigsaw puzzles. Besides jigsaw puzzle solving, these strategies are general and could potentially be extended to other sparse reconstruction tasks.

ACKNOWLEDGMENTS

This work was partly supported by the National Science Foundation IIS-1320959. Canyu Le was supported by the National Natural Science Foundation of China 61728206, and his work was done while he was a visiting student at Louisiana State University.

REFERENCES

[1] H. Liu, S. Cao, and S. Yan, “Automated assembly of shredded pieces from multiple photos,” IEEE Transactions on Multimedia, vol. 13, no. 5, pp. 1154–1162, 2011.
[2] K. Zhang and X. Li, “A graph-based optimization algorithm for fragmented image reassembly,” Graphical Models, vol. 76, no. 5, pp. 484–495, 2014.
[3] B. J. Brown, C. Toler-Franklin, D. Nehab, M. Burns, D. Dobkin, A. Vlachopoulos, C. Doumas, S. Rusinkiewicz, and T. Weyrich, “A system for high-volume acquisition and matching of fresco fragments: Reassembling theran wall paintings,” in ACM transactions on graphics (TOG), vol. 27, no. 3. ACM, 2008, p. 84.
[4] K. Son, E. B. Almeida, and D. B. Cooper, “Axially symmetric 3d pots configuration system using axis of symmetry and break curve,” in Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013, pp. 257–264.
[5] M. Dellepiane, F. Niccolucci, S. P. Serna, H. Rushmeier, L. Van Gool et al., “Reassembling thin artifacts of unknown geometry,” Proc of the 12th International Conference on Virtual Reality, Archaeology and Cultural Heritage, Switzerland: Euro Graphics Association Airelavelle, pp. 51–61, 2011.
[6] Z. Yin, L. Wei, M. Manhein, and X. Li, “An automatic assembly and completion framework for fragmented skulls,” in International Conference on Computer Vision (ICCV), 2011, pp. 2532–2539.
[7] K. Zhang, W. Yu, M. Manhein, W. Waggener, and X. Li, “3d fragment reassembly using integrated template guidance and fracture-region matching,” in Proc. IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2138–2146.
[8] H. Freeman and L. Garder, “A pictorial jigsaw puzzles: The computer solution of a problem in pattern recognition,” IEEE Transactions on Electronic Computers, no. 2, pp. 118–127, 1964.
[9] E. D. Demaine and M. L. Demaine, “Jigsaw puzzles, edge matching, and polynomino packing: Connections and complexity,” Graphs and Combinatorics, vol. 23, no. 1, pp. 195–208, 2007.
[10] R. Murr-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255–1262, 2017.
[11] S. Choi, Q.-Y. Zhou, and V. Kolotou, “Robust reconstruction of indoor scenes,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5556–5565.
[12] C. Le and X. Li, “Sparse3d: A new global model for matching sparse rgb-d dataset with small inter-frame overlap,” Computer-Aided Design, vol. 102, pp. 33–43, 2018.
[13] K. Son, J. Hays, and D. B. Cooper, “Solving square jigsaw puzzles with loop constraints,” in European Conference on Computer Vision. Springer, 2014, pp. 32–46.
[14] K. Son, J. Hays, D. B. Cooper et al., “Solving small-piece jigsaw puzzles by growing consensus,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1193–1201.
[15] T. S. Cho, S. Avidan, and W. T. Freeman, “A probabilistic image jigsaw puzzle solver,” in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010, pp. 183–190.
[16] D. Pomeranz, M. Shemes, and O. Ben-Shahar, “A fully automated greedy square jigsaw puzzle solver,” in Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. IEEE, 2011, pp. 9–16.
[17] A. C. Gallagher, “Jigsaw puzzles with pieces of unknown orientation,” in Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012, pp. 382–389.
[18] M.-M. Paumard, D. Picard, and H. Tabia, “Jigsaw puzzle solving using local feature co-occurrences in deep neural networks,” arXiv preprint arXiv:1807.03155, 2018.
[19] F. Amigoni, S. Gazzani, and S. Podico, “A method for reassembling fragments in image reconstruction,” in Proceedings 2003 International Conference on Image Processing, vol. 3, 2003, pp. III–581.
[20] E. Tsamoura and I. Pitas, “Automatic color based reassembly of ripped-up documents,” IEEE Transactions on Image Processing, vol. 19, no. 3, pp. 680–690, 2010.
[21] H. J. Wolfson, “On curve matching,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 12, no. 5, pp. 483–489, May 1990.
[22] M. S. Sagiroglu and A. Ercil, “A texture based matching approach for automated assembly of puzzles,” in 18th International Conference on Pattern Recognition (ICPR’06), vol. 3, 2006, pp. 1036–1041.
[23] L. Zhu, Z. Zhou, and D. Hu, “Globally consistent reconstruction of ripped-up documents,” IEEE Transactions on pattern analysis and machine intelligence, vol. 30, no. 1, pp. 1–13, 2008.
[24] D. H. Douglas and T. K. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” Cartographica: The International Journal for Geographic Information and Geovisualization, vol. 10, no. 2, pp. 112–122, 1973.
[25] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g 2 o: A general framework for graph optimization,” in Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011, pp. 3607–3613.
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.
[27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
Fig. 17. Some reassembly results on the MIT data. Each puzzle contains 9 pieces.

Fig. 18. Some reassembly results on the BGU data. The first row shows a reassembly of a 36-piece puzzle. The second and third rows show reassemblies of a 100-piece puzzles.

[28] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[29] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in Computer Vision (ICCV), 2017 IEEE International Conference on. IEEE, 2017, pp. 2980–2988.

[30] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” arXiv preprint arXiv:1502.03167, 2015.

[31] H. He and E. A. Garcia, “Learning from imbalanced data,” IEEE Transactions on knowledge and data engineering, vol. 21, no. 9, pp. 1263–1284, 2009.

[32] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” arXiv
Fig. 19. Some reassembly results on the TestingSet1 data. The first row shows a reassembly of a 36-piece puzzle. The second and third rows show the reassemblies of two 100-piece puzzles.

Fig. 20. Some reassembly results on the TestingSet2 data. Each puzzle has around 400 pieces.

[33] C. Elkan, “The foundations of cost-sensitive learning,” in International joint conference on artificial intelligence, vol. 17, no. 1. Lawrence Erlbaum Associates Ltd, 2001, pp. 973–978.

[34] M. D. Richard and R. P. Lippmann, “Neural network classifiers estimate bayesian a posteriori probabilities,” Neural computation, vol. 3, no. 4, pp. 461–483, 1991.

[35] Y. Freund, R. Schapire, and N. Abe, “A short introduction to boosting,” Journal-Japanese Society For Artificial Intelligence, vol. 14, no. 771, pp. 1612, 1999.

[36] Z. Cui and P. Tan, “Global structure-from-motion by similarity averaging,” in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 864–872.

[37] S. Zhu, T. Shen, L. Zhou, R. Zhang, J. Wang, T. Fang, and L. Quan,
“Parallel structure from motion from local increment to global averaging,” \textit{arXiv preprint arXiv:1702.08601}, 2017.

[38] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based slam,” \textit{IEEE Intelligent Transportation Systems Magazine}, vol. 2, no. 4, pp. 31–43, 2010.

[39] “Pexels,” \url{https://www.pexels.com/}.

[40] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard \textit{et al.}, “Tensorflow: a system for large-scale machine learning,” in \textit{OSDI}, vol. 16, 2016, pp. 265–283.

[41] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” \textit{arXiv preprint arXiv:1412.6980}, 2014.