Image Reassembly Combining Deep Learning and Shortest Path Problem

Marie-Morgane Paumard[1], David Picard[1,2(✉)], and Hedi Tabia[1]

[1] ETIS, UMR 8051, Université Paris Seine, Université Cergy-Pontoise, ENSEA, CNRS, Cergy-Pontoise, France
{marie-morgane.paumard,picard,hedi.tabia}@ensea.fr
[2] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, 75005 Paris, France

Abstract. This paper addresses the problem of reassembling images from disjointed fragments. More specifically, given an unordered set of fragments, we aim at reassembling one or several possibly incomplete images. The main contributions of this work are: (1) several deep neural architectures to predict the relative position of image fragments that outperform the previous state of the art; (2) casting the reassembly problem into the shortest path in a graph problem for which we provide several construction algorithms depending on available information; (3) a new dataset of images taken from the Metropolitan Museum of Art (MET) dedicated to image reassembly for which we provide a clear setup and a strong baseline.

Keywords: Fragments reassembly · Jigsaw puzzle
Image classification · Cultural heritage · Deep learning

1 Introduction

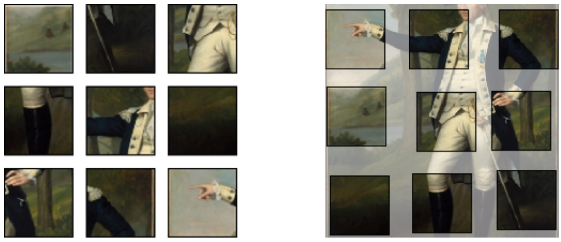
The problem of automatic object reconstruction is very important in computer vision, as it has many potential applications in, e.g. cultural heritage and archaeology. For instance, given numerous fragments of an art masterpiece, archaeologists may spend a long time searching their correct configuration. In recent years, vision-related tasks such as classification [1], captioning [2] or image retrieval [3] have been tremendously improved thanks to deep neural network architectures, and the automatic reassembly of fragments can also be cast as a vision task and improved using the same deep learning methods.

In this paper, we focus on global image reassembly. The fragments are 2D-tiles and the problem consists in finding their approximated position, as shown in Fig. 1. To solve the problem, we build on the method proposed by Doersch et al. [4] that proposes to train a classifier able to predict the relative position of a fragment with respect to another one. We show that solving the reassembly

H. Tabia—This work is supported by the Fondation des sciences du patrimoine, LabEx PATRIMA ANR-10-LABX-0094-01.

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11210, pp. 155–169, 2018.
https://doi.org/10.1007/978-3-030-01231-1_10

problem from an unordered list of fragments can be expressed as a shortest path problem in a carefully designed graph. The structure of the graph heavily depends on the properties of the puzzle such as its geometry (number of positions and their layout), its completeness (a fragment for each available positions) and its homogeneity (all fragments have a correct position in the puzzle).



(a) Fragments (b) Reassembly

Fig. 1. Example of the reassembly task on the MET dataset

Our contributions are the following. First, we propose several deep convolutional neural network architectures for predicting the relative position of a square-cropped fragment with respect to another. The crop allows us to ignore the borders of each piece and to focus on the content in order to achieve a global positioning. Second, we propose several graph construction algorithms that implement the reassembly problem corresponding to the different cases of puzzles depending on the aforementioned properties. Third, we perform extensive experiments of the different neural network and shortest path graph problem combinations on ImageNet [5] and on a new dataset composed of 14,000 images from the Metropolitan Museum of Art (MET). For this new dataset, we provide a clear setup and evaluation procedure that allows future works on the reassembly problem to be compared.

This paper is organized as follows: in Sect. 2, we present related work on puzzle solving and fragment reassembly as well as relevant literature on feature combination as it is an essential step of the relative position prediction. Next, we detail our propositions for the deep neural network building block and the graph construction algorithms that correspond to the different image reassembly problems. In Sect. 4, we present our experimental setups and analyze the results obtained for different combinations of deep neural networks and graphs.

2 Related Work

In this section, we first present the related work on puzzle solving. Then we detail the relevant literature on feature combination.

2.1 Puzzle Solving

The reconstruction of archaeological pieces of art leads to better understanding of our history and thus attracts numerous researchers, as Rasheed and Nordin described in their surveys [6,7]. Most publications of this field rely on the border irregularities and aim for precise alignment. They focus on automated reconstruction, such as [8–10] and consider jigsaw puzzle solving with missing fragments or with differently sized tiles [11–13]. These methods perform well on a small dataset with only one source of fragments. On the downside, they stall when fragments come from various sources and they require costly human made annotations. Moreover, they are fragile towards erosion and fragment loss.

Without being interested in jigsaw puzzle solving, Doersch et al. proposed a deep neural network to predict the relative position of two adjacent fragments in [4]. The end goal of the authors is to use this task as a pretraining step of deep convolutional neural network (CNN), harnessing the vast amounts of unlabeled images since the ground truth for such task can be automatically generated. The intuitions for training features able to predict their context are the same as what is found in the text literature with word2vec [14] or skip-thought [15]. In [4], the authors show their proposed task outperforms all other unsupervised pretraining methods. Based on [4], Noroozi and Favaro [16] introduce a network that compares all the nine tiles at the same time. They claim that the complete representation obtained allows discarding the ambiguities that may have been learned with the algorithm proposed by Doersch et al. Gur et al. [17] consider missing fragments, but heavily rely on border to solve the puzzle.

In this paper, we focus on solving the jigsaw puzzle and not on the building of generic images features. In cultural heritage, we have missing pieces, as well as pieces from various images. Therefore, the setup of [16] is impractical, as it requires exactly the nine correct fragments to make a prediction. For this reason, we base our work on the method proposed in [4], but we do not share the same objective and we bring two significant innovations. First, we consider the correlations between localized parts of the fragments when merging the features, something that is difficult to achieve in [4]. We believe these correlations are important, since, e.g., we expect the right part of the baseline fragment to be correlated with the left part of the right fragment. Second, we look for a complete fragment reassembly, which we perform by using the deep neural network predictions to build a shortest path graph problem.

2.2 Feature Combination

Doersch et al. [4] separately processed fragments using a deep CNN with shared weights which output comparable features. These features are then serially concatenated and fed to a multi-layer perceptron (MLP) that performs the classification. The full network has been trained in an end-to-end fashion with standard back-propagation using stochastic gradient descent.

In Doersch et al. [4] formulation, the cross-covariance between the features of both fragments is neglected. Indeed, the output of the CNN can be viewed

as localized pattern activations. The prediction of the relative position depends on the conjunction of specific patterns occurring at specific positions in the first fragment and specific patterns occurring at specific positions in the second fragment. It can be argued that a sufficiently deep MLP can model these cross-covariances, but it also seems easier to model them directly.

In [18], the authors suggest modeling these co-occurrences of patterns using a bilinear model which can be computed using the Kronecker product of the feature vectors. They report improved accuracy on fine-grained classification. However, using the Kronecker product leads to high dimensional features that are intractable in practice. To overcome this burden, the authors of [19] propose to use random projections combined with the Hadamard (element-wise) product to approximate the bilinear model. This strategy is further extended in [20] where the projections are trained in true deep learning fashion. Another factorization based on the Tucker decomposition is also proposed in [21] which allows controlling the rank of the considered co-occurrences.

3 Method

In this section, we detail our proposed method. We start by presenting the deep CNN model upon which we build to solve the image reassembly problem.

3.1 Relative Position Prediction

To solve a puzzle, we need to pick the fragments to use. We compare each selected fragment with the central fragment and compute their relative position. We examine several ways to articulate this problem.

Problem Formulation. The first step towards reassembly consists in discriminating between the fragments that may be of use and the others. On our puzzle, it means that we predict which fragments are allegedly extracted from the same image as a given central fragment, which is a binary classification problem. Once only relevant fragments are selected, we model the position prediction as an 8-classes classification problem, as shown in Fig. 2. Both these classification tasks are performed by a deep CNN described later.

We also propose an alternative model by merging these two networks into a single network. This single network predicts the relative position of the second fragment among the 8 possible positions and a 9th class, activated if the fragment is not part of the same image.

Network Architecture. The global network architecture is described in Fig. 3. Given two input fragments, we first extract fragment representations using a shared feature extraction network (FEN). We tested the most common architectures and empirically found out that a VGG-like [22] network works the best. Therefore, the FEN architecture is inspired by a simplified version of VGG [22]

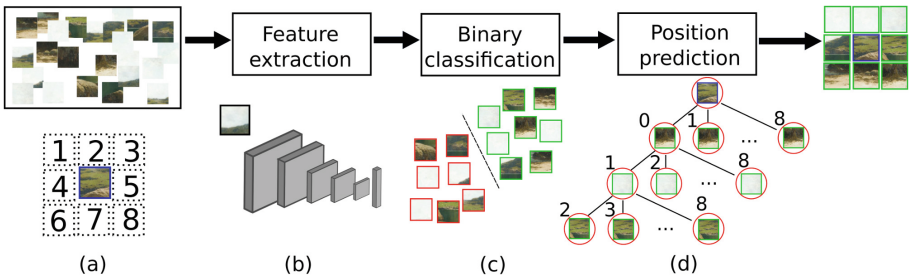


Fig. 2. Overview of our method. Knowing a central fragment, we are looking for the correct arrangement to reassemble the image (a). We extract the feature of all the fragments (b) and we compare them to the features of the central fragment. We predict which fragments are part of the image (c). We retrieve the top eight fragments and we predict their relative position with respect to the central one. We turn the prediction into a graph (d). We then run a shortest path algorithm to reconstruct the image

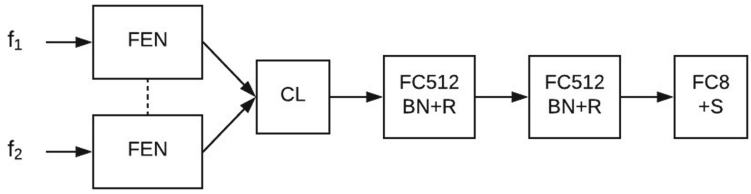


Fig. 3. General network architecture block diagram

and is shown on Table 1. The network is composed of sequences of a 3×3 convolution followed by batch-normalization [23], ReLU activation [24] and max-pooling. We also tried other models based on more recent architectures such as Resnet [25], but we empirically found that they were underperforming compared to the simpler architecture. This can be explained by the fact that contrarily to full images, fragments do not contain as much semantic information and thus require less involved features. Remark also that there is no global pooling [26] in the FEN and thus spatial information is preserved, which we believe is important for the relative position prediction.

The features of each fragment are then combined in a combination layer (CL). Contrarily to the concatenation that is proposed at this stage in [4], we explore variations on the bilinear product in order to model cross-covariances among the features. With $\phi_{\text{FEN}}(f)$ the output of the FEN for fragment f, the full bilinear product is obtained by using the Kronecker product of the features [18]:

$$y_{kron} = \phi_{\text{FEN}}(f_1) \otimes \phi_{\text{FEN}}(f_2). \tag{1}$$

However, this leads to very high dimensional vectors. Similarly to [20], we explore a compressed version using the entry-wise product:

$$y_{had} = (W^\top \phi_{\text{FEN}}(f_1)) \circ (W^\top \phi_{\text{FEN}}(f_2)), \tag{2}$$

where $\circ$ denotes the Hadamard product. This compressed version can be efficiently implemented by changing the output size of the last layer in the FEN.

Finally, the classification stage consists of two sequences of a fully connected layer followed by a batch-normalization and a ReLU activation, and a final prediction layer with softmax activation.

Table 1. Architecture of the feature extraction network. Conv: convolution, BN: Batch-Normalization, ReLU: ReLU activation. OUT is chosen among 512, 1024, 2048 and 4096, depending on what merging function we use

Layer	Output shape	Parameters shape	Parameters count
Input	$96 \times 96 \times 3$		0
Conv+BN+ReLU	$96 \times 96 \times 32$	$3 \times 3 \times 32$	1k
Maxpooling	$48 \times 48 \times 32$		–
Conv+BN+ReLU	$48 \times 48 \times 64$	$3 \times 3 \times 32$	19k
Maxpooling	$24 \times 24 \times 64$		–
Conv+BN+ReLU	$24 \times 24 \times 128$	$3 \times 3 \times 32$	74k
Maxpooling	$12 \times 12 \times 128$		–
Conv+BN+ReLU	$12 \times 12 \times 256$	$3 \times 3 \times 32$	296k
Maxpooling	$6 \times 6 \times 256$		–
Conv+BN+ReLU	$6 \times 6 \times 512$	$3 \times 3 \times 32$	1.2M
Maxpooling	$3 \times 3 \times 512$		–
Fully connected+BN	OUT		$OUT_{nb\ param}$

3.2 Puzzle Resolution

Once the position is predicted by the neural network for each fragment, we can solve the puzzle, which consists in assigning fragments to a position in the image. We consider several cases depending on whether we already have a well-positioned fragment, and whether we have supernumerary fragments.

Problem Formulation. We first consider the case where we are given the central fragment as well as an unordered list of 8 fragments corresponding to the possible neighbors of the central fragment. Solving the puzzle then consists in solving the assignment problem where each fragment i has to be associated with a position j. Given the relevance $p_{i,j}$ of fragment i at position j, and the assignment variable $x_{i,j} = 1$ if fragment i is at position j, we want to maximize:

$$\max_{x_{i,j}} \sum_{i,j} p_{i,j} \cdot x_{i,j} \tag{3}$$

under the constraints:

$$\forall j, \sum_{i=0}^{8} x_{i,j} = 1 \ , \tag{4}$$

$$\forall i, \sum_{j=0}^{8} x_{i,j} = 1 \ , \tag{5}$$

$$\forall i, j, x_{i,j} \in \{0, 1\}. \tag{6}$$

Remark that only one fragment can occupy a position (Eq. 4) and a fragment can be placed only once (Eq. 5).

Then, if we allow the puzzle to be uncompleted (i.e. some positions are not used), we replace the constraint 4 with:

$$\forall j, \sum_{i=0}^{8} x_{i,j} \leq 1. \tag{7}$$

Similarly, if we have supernumerary fragments (i.e. some fragments are not used), we replace the constraint 5 with:

$$\forall i, \sum_{j=0}^{N} x_{i,j} \leq 1. \tag{8}$$

Finally, if we do not know which fragment is the central fragment, we have to solve the extended assignment problem where one fragment has to be assigned to the central position and the remaining fragment are assigned to the relative positions. This leads to the following problem:

$$\max_{c, x_{i,j}} \sum_{i,j} p_{i,j,c} \cdot x_{i,j,c} \tag{9}$$

under the following constraints:

$$\forall c, j, \sum_{i=0}^{N} x_{i,j,c} \leq 1; \forall c, i \neq c, \sum_{j=0}^{8} x_{i,j,c} \leq 1; \forall c, j, \forall i \neq c, x_{i,j,c} \in \{0, 1\};$$

$$\forall c, j, \forall i = c, x_{i,j,c} = 0.$$

3.3 Graph Formulation

Solving the mentioned problem can be done by finding the shortest path in a corresponding directed graph, which can be done using Dijkstra's algorithm or any of its variants. In this section, we show how to construct such graphs.

Each graph starts with a source S and ends with a sink T. Each subsequent depth level from S corresponds to a fragment. All nodes at a given depth i from S correspond to the position that could be assigned to a fragment i given all previous assignments. Each edge receives the corresponding classification score as the weight.

Algorithm 1. Graph building from central fragment

procedure CONSTRUCT_EDGES(Y) ▷ Y is the predicted values matrix for i, j
  $empty\_pos \leftarrow [1..9]$
  $used\_pos \leftarrow [S]$
  $next\_frag \leftarrow 1$
  $tree \leftarrow$ ADD CHILDREN($Y, empty\_pos, used\_pos, next\_frag$)
  return $tree$ ▷ The list of the edges: related fragment, position of the previous node, position of the current node, cost of the edge.
end procedure

procedure ADD_CHILDREN($Y, empty\_pos, used\_pos, next\_frag$)
  $edges \leftarrow [\,]$
  if $empty\_pos$ is empty **then**
    $edges \leftarrow [(None, last(used\_pos), T, 0)]$ ▷ Append the $j \rightarrow T$ edge
    return $edges$
  end if
  for pos in $empty\_pos$ **do**
    $edges \leftarrow edges \cup [(next\_frag, last(used\_pos), pos, Y[next\_fragment, pos])]$
    $empty\_pos \leftarrow empty\_pos \setminus pos$
    $used\_pos \leftarrow used\_pos \cup pos$
    $edges \leftarrow edges \cup$ (ADD CHILDREN($Y, empty\_pos, used\_pos, next\_frag + 1$))
  end for
  return $edges$
end procedure

When the central fragment is known and we have the exact number of missing fragments, the construction procedure is given in Algorithm 1. We also give a very simple example with only two relative positions in Fig. 4a.

In the case where the central fragment is known, the size of the resulting graph is $|E| = \frac{n!}{(n-p)!} + \sum_{i=n-p}^{n-1} \frac{n!}{i!}$ for the number of edges and $|N| = 2 + \sum_{i=n-p}^{n-1} \frac{n!}{i!}$ for the number of vertices, with n the number of fragments and p the number of positions. With 8 fragments and positions, this corresponds to $|E| = 150$k and $|N| = 100$k.

In the case where we do not know the central fragment, we simply perform the central fragment selection as a first step. The first expansion from S consists in all the possible cases where each fragment is used as the central fragment. The corresponding subgraphs are the built using Algorithm 1. The size of the resulting graph is unchanged, except we have $n + 1$ fragments, with n the number of the fragment to be assigned to a relative position. With $n = 8$, we obtain $|N| = 1$M and $|E| = 1.3$M. We show in Fig. 4b a simplified example with 3 fragments and 2 relative positions.

Finally, we now consider the case where the puzzle may not be solved with all the fragments we have. This means that we can have more than 8 fragments, coming from various sources. We also may have missing fragments, and consequently, we prefer an algorithm that proposes an incomplete solution than a wrong reassembly. We construct a graph allowing such configurations by enabling the algorithm to pick no fragment. A simplified example of the graph shown in Fig. 5.

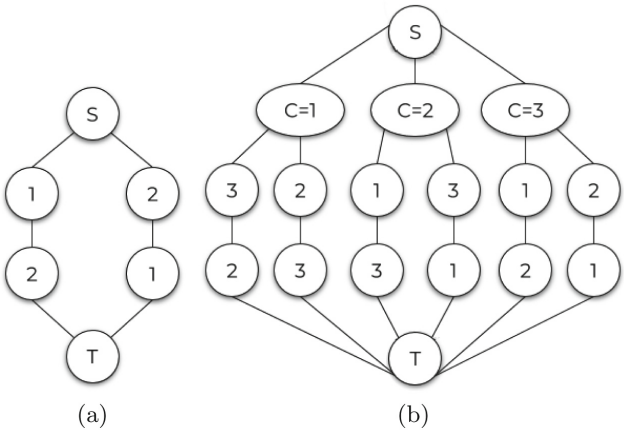


Fig. 4. Examples of graphs for a complete problem with known and unknown central fragment, for empty 2 positions

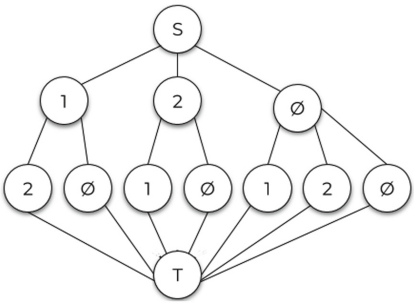


Fig. 5. Example of a graph allowing empty positions

The graph building algorithm is similar to the Algorithm 1; if we add a position Ø at the *antecedents* list, we do not exclude it from the further available choices, as detailed in Algorithm 2. This graph has:

$$|N| = 2 + \sum_{l=0}^{n} \sum_{k=p-l}^{p} \binom{l}{p-k} \frac{(k+1)p!}{k!} \tag{10}$$

vertices and

$$|E| = \sum_{k=p-n}^{p} \binom{l}{p-k} \frac{(k+1)p!}{k!} + \sum_{l=0}^{n} \sum_{k=p-l}^{p} \binom{l}{p-k} \frac{(k+1)p!}{k!} \tag{11}$$

edges, with n fragments and p positions. If the breadth of the graph is limited by the number of position, the depth depends on the number of fragments. In the case of 10 fragments and 8 relative positions, the size of the graph is $|E| = 5 \cdot 10^9$ and $|N| = 4 \cdot 10^8$.

Once the graph has been set up, the shortest path from S to T can be found with Dijkstra's algorithm [27] for which the complexity is $\mathcal{O}(|E| + |N| \times \log(N))$.

Algorithm 2. Graph building with empty positions

```
1: procedure ADD_CHILDREN(Y, empty_pos, used_pos, next_frag)
2:     edges ← [ ]
3:     if empty_pos is empty or next_frag > n then
4:         edges ← [(None, last(used_pos), T, 0)]          ▷ Append the j → T edge
5:         return edges
6:     end if
7:     for pos in empty_pos∪Ø do
8:         edges ← edges ∪ [(next_frag, last(used_pos), pos, Y[next_fragment, pos])]
9:         if pos in empty_pos then
10:            empty_pos ← empty_pos \ pos
11:        end if
12:        used_pos ← used_pos ∪ pos
13:        edges ← edges ∪ (ADD CHILDREN(Y, empty_pos, used_pos, next_frag + 1))
14:    end for
15:    return edges
16: end procedure
```

Greedy Method. We implement a greedy method to enable us to benchmark the Dijkstra algorithm. We solve iteratively the puzzle, picking at each step the top value from the neural network predictions. We expect this method will make worst choices than Dijkstra's considering the dependencies between the steps.

4 Experiments

In this section, we first describe our experimental setup as well as our new dataset related to cultural heritage. Then, we give experimental results on the classification task and on full image reassembly.

4.1 Experimental Setup

The neural networks are trained using fragments from 1.2M images of ImageNet. We use 50k images to evaluate the classification accuracy. Each image is resized and square-cropped to 398×398 pixels, and divided into 9 parts separated by a 48-pixel margin, corresponding to the erosion of the fragments. Each fragment

has a size of 96×96 pixels and has to be contained in one of the 9 parts, which means that it can be chosen within a ± 7-pixels range in each direction.

For the reassembly, the neural networks are then fine-tuned on a cultural heritage dataset, consisting of 14,000 open-source images from the Metropolitan Museum of Art. Such dataset is close to our aimed application, puzzle solving for cultural heritage

4.2 Classification

To evaluate our proposed architectures for classification, we reproduce the architecture Doersch et al. detailed in [4]. The authors reported a 40% accuracy on ImageNet for the 8-classes classification task. Replicating the architecture of their neural network, we obtain an accuracy of 57%. This may be explained by the tuning of the hyperparameters.

In Table 2, we report the accuracy for the different combination layers on the 8-classes problem on ImageNet validation images. As we can see, the Kronecker product obtain slightly better results than the concatenation. However, using the low-rank approximation of [20] yields lower results which means that the full covariances are needed to obtain the best performances. Remark that all of our architecture outperforms the architecture proposed in [4].

Table 2. Accuracy for different fusion strategies, for the 8-classes classification problem on ImageNet validation. *denotes our implementation

Fusion	Accuracy
Doersch et al. [4]*	57.0%
Concatenation	64.6%
Kronecker product	66.4%
Hadamard product	59.2%

We show the results of the sequential classification approach (2 classes, then 8 classes) and the joint classification approach (9 classes) on Table 3. For the binary classification problem, we set the proportion of fragments belonging to the same image to 50% and we obtain 92.5% accuracy. which means that deciding whether two fragments belong to the same image seems to be an easy problem. For the 8 classes problem, we obtain 66.4% accuracy. It is not surprising to reach around 33% error since many fragments are ambiguous with respect to the precise location among three positions. For example, sky fragments are easy to classify on top with respect to the central fragment, but which of the three top positions is often difficult to guess. Finally, the joint classification problem achieves 64.2% (the proportion of fragment belonging to the same image was set to 70%), which indicates that solving the joint problem is not harder than solving the sequence of simpler problems.

Table 3. Classification accuracy for the 2-classes, 8-classes and 9-classes problems on Imagenet, using the Kronecker combination layer

Problem	Accuracy
2-classes neighborhood classifier	92.5%
8-classes position classifier	66.4%
9-classes classifier	64.2%

4.3 Reassembly

In Table 4, we compare various cases of reassembly tasks, using two different accuracy measures. The reconstruction accuracy describe if the puzzle is perfectly solved. The position accuracy counts how many fragments are well placed.

Table 4. Reconstruction accuracies and position accuracies for different reassembly problems

Problem	Reconstruction accuracy		Position accuracy	
	Greedy	Dijkstra	Greedy	Dijkstra
Central known, complete puzzle	41.0	44.4	87.7	89.9
Central unknown, complete puzzle	36.2	39.2	69.5	71.1
Central known, incomplete puzzle	26.5	29.5	80.5	82.4

As we can see, in the case of the complete puzzle where the central fragment is known, we are able to perfectly reassemble the image in 44.4% of the cases using Dijkstra's algorithm, which represent a 3% improvement over the greedy algorithm, which is closer to the optimal solution than one might think. Remark that the position accuracy is around 90%, which is much better than the 66.4% accuracy of the neural network used to solve the task. This shows that solving the reassembly problem can remove some uncertainty the classifier has.

When the central fragment is not know, the reassembly accuracy drops only to 39.2% and the position accuracy drops to 71.1%. This means that reassembling the image without knowing the central fragment is not much more complicated than with the central fragment known, however, if that first step is missed, then all subsequent assignments are likely to be wrong.

We consider adding outsider fragments to the puzzle (Table 5), making the accuracy drop. The increase of computation time triggered by the addition is reasonable as long as the puzzle still contains 9 pieces. Any increment of the number of pieces leads to an factorial increase of the number of solution, making the problem quickly intractable. Nonetheless, any puzzle can be divided into 3×3 puzzles, that would be solved individually and fused.

Table 5. Position and reconstruction accuracies with additional fragments

Number of additional fragments	0	1	2
Reassembly accuracy (Dijkstra)	44.4%	26.3%	14.3%
Position accuracy (Dijkstra)	89.9%	75.3%	64.8%

In Fig. 6, we selected few reconstructions with unknown central fragment. The two first images illustrate a significant part of our dataset in which it is easy to misplace background fragments. Most of our reconstruction errors are due to similar reversals. The type of error illustrated by the right image is rare; but, when the central fragment is misplaced, all the other fragments are shifted.

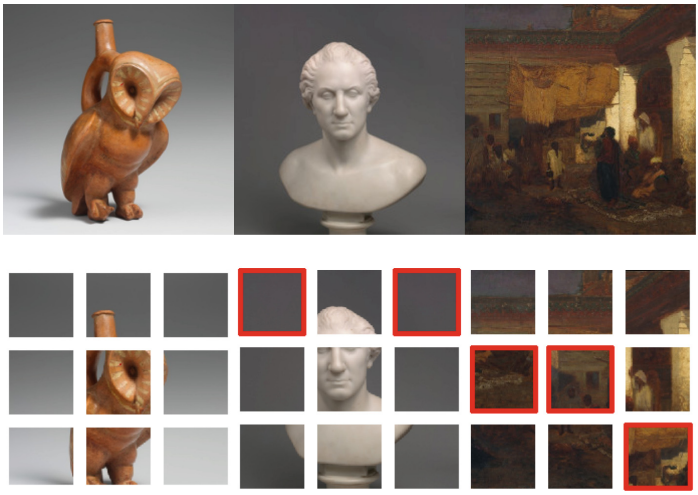


Fig. 6. Examples of reconstructions with unknown central fragment. The red outlined fragments are misplaced (Color figure online)

Finally, we study the case where we have missing fragments (Table 4, last row). In that scenario, only 4 fragment are taken from the image while 8 positions are available. We are still able to predict the position with high accuracy (surprisingly better than in the case where the central fragment is unknown), but perfectly reassembling the image is very difficult. This means that the algorithm tends to drop fragments instead of assigning them to an uncertain location. Figure 7 shows examples of reconstructions in the case of missing fragments.

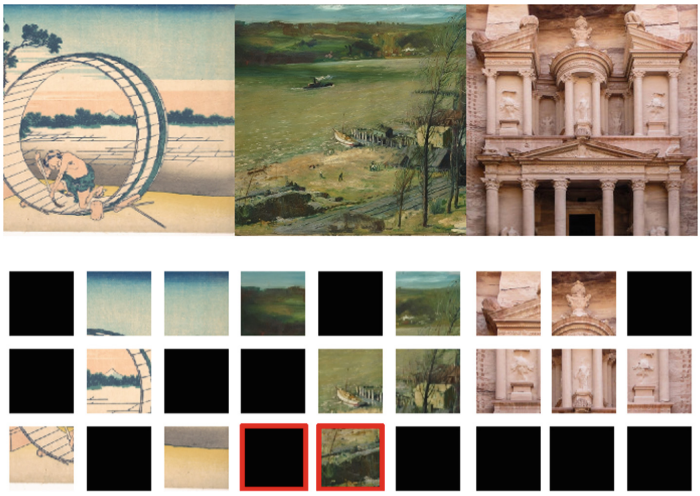


Fig. 7. Examples of reconstructions with 4 missing fragments. The red outlined fragments are misplaced (Color figure online)

5 Conclusion

In this paper, we tackled the image reassembly problem where given a unordered list of image fragments, we want to recover the original image. To that end, we proposed a deep neural network architecture that predicts the relative position of a given pair of fragments. Then, we cast the reassembly problem into a shortest path in a graph algorithm for which we propose several construction algorithms depending on whether the puzzle is complete or if there are missing pieces. We propose a new dataset containing 14,000 images to test several reassembly tasks and we show that we are able to perfectly reassemble the image 44.4% of the time in the simpler case and 29.5% of the time if there are missing pieces.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. NIPS **1**, 1097–1105 (2012)
2. Johnson, J., Karpathy, A., Fei-Fei, L.: DenseCap: fully convolutional localization networks for dense captioning. In: CVPR (2016)
3. Gordo, A., Almazán, J., Revaud, J., Larlus, D.: Deep image retrieval: learning global representations for image search. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9910, pp. 241–257. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46466-4_15
4. Doersch, C., Gupta, A., Efros, A.: Unsupervised visual representation learning by context prediction. In: ICCV (2015)
5. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. IJCV **115**(3), 211–252 (2015)

6. Rasheed, N., Nordin, M.J.: A survey of classification and reconstruction methods for the 2D archaeological objects. In: ISTMET, pp. 142–147, August 2015
7. Rasheed, N., Nordin, M.J.: A survey of computer methods in reconstruction of 3D archaeological pottery objects. Int. J. Adv. Res. **3**, 712–714 (2015)
8. McBride, J., Kimia, B.: Archaeological fragment reconstruction using curve-matching. In: CVPRW (2003)
9. Jampy, F., Hostein, A., Fauvet, E., Laligant, O., Truchetet, F.: 3D puzzle reconstruction for archeological fragments. In: 3DIPM (2015)
10. Zhu, L., Zhou, Z., Zhang, J., Hu, D.: A partial curve matching method for automatic reassembly of 2D fragments. In: Huang, D.S., Li, K., Irwin, G.W. (eds.) Intelligent Computing in Signal Processing and Pattern Recognition. LNCIS, vol. 345, pp. 645–650. Springer, Berlin (2006). https://doi.org/10.1007/978-3-540-37258-5_70
11. Hammoudeh, Z., Pollett, C.: Clustering-based, fully automated mixed-bag jigsaw puzzle solving. In: Felsberg, M., Heyden, A., Krüger, N. (eds.) CAIP 2017. LNCS, vol. 10425, pp. 205–217. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64698-5_18
12. Andaló, F., Taubin, G., Goldenstein, S.: PSQP: puzzle solving by quadratic programming. IEEE TPAMI **39**, 385–396 (2017)
13. Lifang, C., Cao, D., Liu, Y.: A new intelligent jigsaw puzzle algorithm base on mixed similarity and symbol matrix. IJPRAI **32**, 1859001 (2018)
14. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS, pp. 3111–3119 (2013)
15. Kiros, R., et al.: Skip-thought vectors. In: NIPS, pp. 3294–3302 (2015)
16. Noroozi, M., Favaro, P.: Unsupervised learning of visual representations by solving jigsaw puzzles (2015)
17. Gur, S., Ben-Shahar, O.: From square pieces to brick walls: the next challenge in solving jigsaw puzzles. In: ICCV (2017)
18. Lin, T.Y., RoyChowdhury, A., Maji, S.: Bilinear CNN models for fine-grained visual recognition. In: ICCV, pp. 1449–1457 (2015)
19. Gao, Y., Beijbom, O., Zhang, N., Darrell, T.: Compact bilinear pooling. In: IEEE CVPR, pp. 317–326 (2016)
20. Kim, J.H., On, K.W., Lim, W., Ha, J., Zhang, B.-T.: Hadamard product for low-rank bilinear pooling. In: ICLR (2017)
21. Ben-younes, H., Cadene, R., Cord, M., Thome, N.: MUTAN: multimodal tucker fusion for visual question answering, pp. 2612–2620 (2017)
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ILSVRC (2014)
23. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
24. Nair, V., Hinton, G.: Rectified linear units improve restricted Boltzmann machines. In: ICML (2010)
25. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE CVPR, pp. 770–778 (2016)
26. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
27. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)