Adversarial Placement Vector Learning

Ayesha Rafique, Tauseef Iftikhar, Nazar Khan
PUCIT, University of the Punjab
Lahore, Pakistan
{mscsf16m020, tauseef.iftikhar, nazarkhan}@pucit.edu.pk

***Abstract*—Automated jigsaw puzzle solving is a challenging problem with numerous scientific applications. We explore whether a Generative Adversarial Network (GAN) can output jigsaw piece placements. State-of-the-art GANs for image-to-image translation cannot solve the jigsaw problem in an exact fashion. Instead of learning image-to-image mappings, we propose a novel piece-to-location mapping problem and present a trainable generative model for producing output that can be interpreted as the placement of jigsaw pieces. This represents a first step in developing a complete learning-based generative model for piece-to-location mappings. We introduce four new evaluation measures for the quality of output locations and show that locations generated by our model perform favorably.**

***Index Terms*—jigsaw puzzle, generative adversarial networks, placement vector, map processing**

I. Introduction

Many scientific problems can be modelled as a jigsaw puzzle. Some examples include protein modelling [11], RNA/DNA modelling [17], analysis of animal behaviour [4], image editing [6], archaeological artifact reconstruction [16] and shredded document reconstruction [22]. The jigsaw puzzle problem is NP-complete if piece-wise compatibility is unreliable [1, 8]. In [3] Bosboom et al. have shown that placing P pieces of an edge-matching puzzle in a single row of dimension $1 \times P$ is NP-hard.

The solution of a jigsaw puzzle involves correct placement of the individual pieces. If every piece is indexed by a number, the solution corresponds to a placement vector representing an appropriate arrangement of the pieces. Different optimization techniques such as probabilistic graphical models, genetic algorithms and linear and quadratic programming have been applied to automatically solve jigsaw puzzles [5, 12, 20, 28]. The output of such approaches can be viewed as a placement vector that is optimal in some sense with respect to the input pieces. An alternative to these approaches is to construct a conditional model that learns the piece-to-location relationship from training data. Once the conditional relationship is learned, any random configuration of new pieces can be directly mapped to the appropriate placement vector without performing an expensive optimization over possible locations. However, before attempting to learn conditional piece-to-placement vector mappings, it is important to explore the effectiveness of learning an unconditional model for placement vectors alone. If an unconditional placement vector generator can not be learned, there is little hope in learning a conditional placement vector generator. Accordingly, this paper deals with learning a novel model for unconditional placement vector generation. We also demonstrate the weaknesses of an end-to-end learning model for direct piece-to-map image transformations.

Fig. 1: **Left**: A Spilsbury jigsaw puzzle from 1776 depicting a dissection of the kingdoms of Europe. The cartographer John Spilsbury is credited as the inventor of such puzzles. **Right**: A dissected map puzzle from 1860 by J.H. Colton.

Jigsaw puzzles and land maps share an interesting history. Such puzzles were invented by the English mapmaker John Spilsbury in the 18-th century [14] by cutting along map boundaries with a saw and creating a puzzle. Two such puzzles representing kingdoms of Europe and dissections of United States are shown in Figure 1.

Constituent pieces of an artifact can be re-positioned by specifying a placement vector representing the mapping between each piece and its placement within a grid of possible locations. The pieces can belong to any artifact such as a toy jigsaw puzzle, a shredded document, biological/chemical molecules or any other kind of problem modeled as a jigsaw puzzle problem. In this work, we describe how adversarial learning can be used to train a model for generating placement vectors.

The paper is organized as follows. After a summary of related approaches in Section II, we describe how unconditional and conditional generative models can be trained through adversarial learning in Section III. Section IV describes how the jigsaw puzzle problem can be modelled as an image-to-image translation task and discusses some weaknesses of that approach. Section V describes how placement vectors can be generated via unconditional adversarial learning. In Section VI, we explain how to evaluate any system that

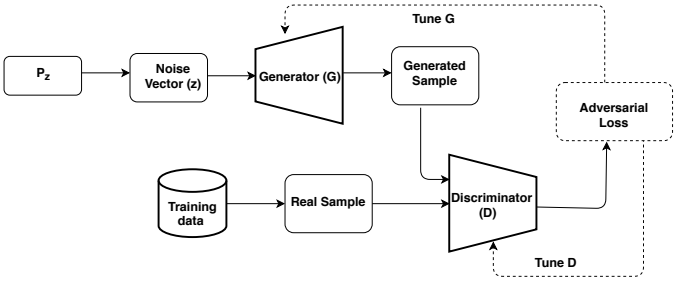

Fig. 2: GAN training framework. The discriminator D tries to classify generated samples as fake and real samples as real. Both networks D and G are trained through feedback from D as shown by dotted arrows.

claims to generate placement vectors and perform the proposed evaluation on placement vectors generated by our method. Finally, we present some concluding thoughts in Section VII.

II. Related Work

The jigsaw puzzle problem initially attracted researchers working on boundary and shape matching techniques [4, 1, 27]. For them, it as a good tool to evaluate the effectiveness of their algorithms. Due to limited computational resources and the combinatorial nature of the problem, initial attempts involved very small jigsaw puzzles.

Jigsaw puzzle problems can be of many types depending on the measures used to compute affinity between pieces. In *apictorial* jigsaw problems, only shape information is available [9, 1, 12]. In *pictorial* jigsaw problems, the contents of individual pieces are also used [20, 5, 19]. Some methods use both shape and content [7, 29].

Given affinities between pieces, an optimization algorithm is employed to find the best arrangement of pieces. Previous automated approaches have used greedy algorithms [20, 10], global solutions using probabilistic [5] and genetic [26, 24, 23] algorithms, particle filtering [28], loop constraint [25] and linear [32] and quadratic [2] programming.

However, all of the above-mentioned techniques are focused on finding the optimized solution of a given jigsaw problem. This can be a time consuming process. In this paper, we present a novel perspective of the jigsaw problem. Instead of optimizing for each given jigsaw problem, a parameter-based machine learning model can be trained to output placement vectors corresponding to input pieces. Once the model is learned from training data, new jigsaw problems can be solved simply and quickly via forward propagation.

III. Generative Adversarial Networks

Generative adversarial networks (GANs) [13] have heralded the latest revolution in machine learning. Training a GAN involves playing a minimax game between two players. One player is called the *generator* G and the other is called the *discriminator* D. Both are usually taken to be some form of deep neural networks. The goal of G is to generate random samples $G(z, \theta_g)$ from a distribution where z is a random noise vector and θ_g are trainable parameters of G. Given sample vectors, real (x) or fake ($G(z, \theta_g)$), the discriminator D outputs the probability, $D(x, \theta_d)$ and $D(G(z, \theta_g), \theta_d)$ respectively, of the input sample coming from the real distribution. The parameters θ_d of the discriminator are updated so as to enable D to assign high probability $D(x, \theta_d)$ to real samples and low probability $D(G(z, \theta_g), \theta_d)$ to anything generated by G. Simultaneously, the parameters θ_g of the generator are updated so as to generate fake samples that obtain a high probability $D(G(z, \theta_g), \theta_d)$ from the discriminator. These competing goals turn D and G into adversaries with the goal that eventually both will become effective, especially G since it will start generating samples that appear to be real. The overall training objective of GANs is as follows

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}}[\log(D(x))] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{1}$$

$$D^* = \arg\max_{\theta_d} V(D, G) \tag{2}$$

$$G^* = \arg\min_{\theta_g} V(D^*, G) \tag{3}$$

An illustration of the training framework for GANs is shown in Figure 2.

Conditional GANs (cGANs) [18] are an extension of GANs which take additional information y as input. Any information such as class labels, images or text can be given as y. Both generator and discriminator use this information and are therefore conditioned on it. The objective function of cGANs is as follows

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log(D(x|y))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)))] \tag{4}$$

where the only difference is that inputs to the discriminator and generator now include the conditioning variable y as well. In the next section, we describe how a conditional GAN framework can be used to solve the jigsaw puzzle problem.

IV. Solving the Jigsaw via Image-to-Image Translation

In the jigsaw puzzle problem, we have to arrange constituent pieces in a meaningful way. By arranging the input pieces in the form of an unordered image, we can treat the jigsaw puzzle problem as an image-to-image translation task. In the image-to-image translation problem, the objective is to learn a mapping from one category of images to another category. For example, day images to night images. For our problem, one can attempt to learn the mapping from images of unordered pieces to images of ordered pieces. The training data can be constructed using Algorithm 1.

Image-to-image translation can be effectively achieved via the Pix2Pix framework [15] which is a variation of a cGAN. The objective of Pix2Pix is written as

$$D^* = \arg\max_D V(D, G) \tag{5}$$

$$G^* = \arg\min_G V(D^*, G) + \lambda \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1] \tag{6}$$

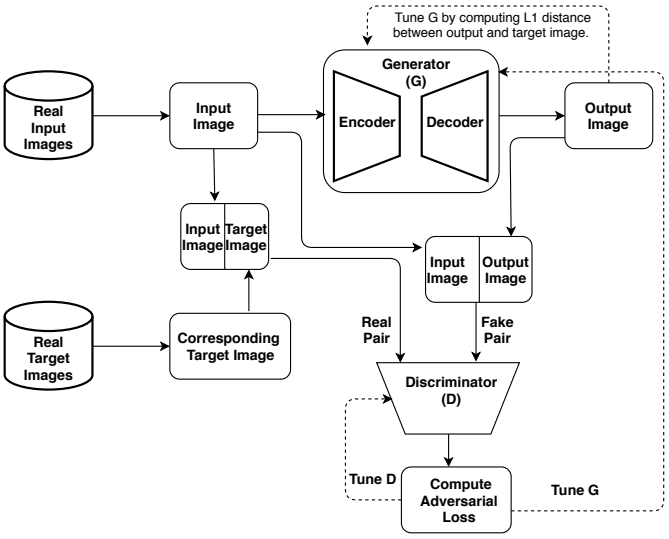

Fig. 3: Illustration of the Pix2Pix framework.

Algorithm 1 Image patch scrambling for construction of data for training a jigsaw solver via image-to-image translation.

1: **for** every training image I **do**
2: Divide I into P non-overlapping patches
3: **for** K times **do**
4: Rearrange the pieces randomly to form an unordered image I' of the same size as I.
5: Form training pair with input I' and corresponding target I.
6: **end for**
7: **end for**

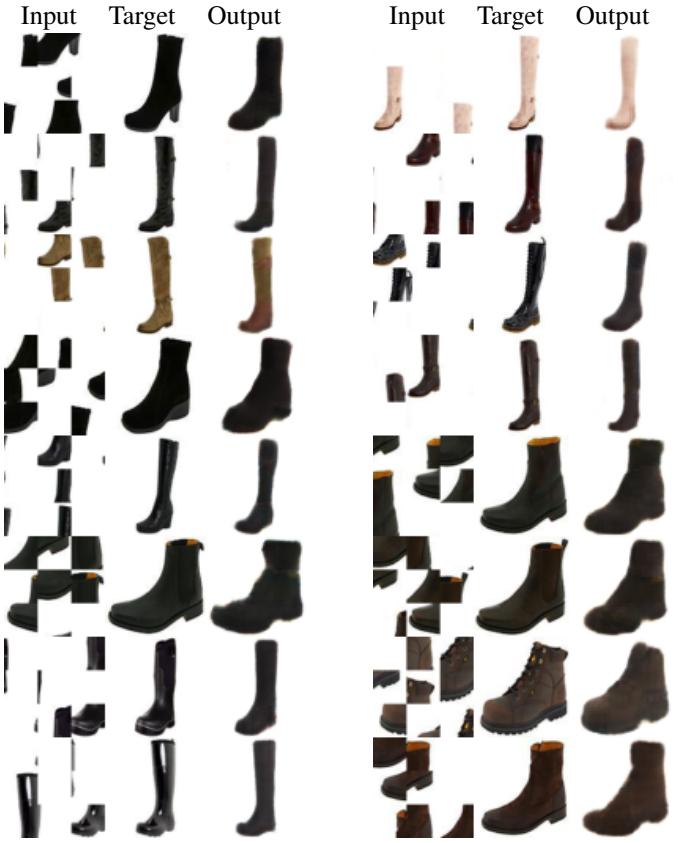

Fig. 4: Shoe images generated from shuffled input patches. While results seem satisfactory, the input patches are not reconstructed exactly.

where the $L1$ loss encourages the generator to produce a sample that resembles the conditioning variable y.

In most image-to-image translation problems, input and output share some common features but differ in style. Therefore, for preserving these common features, the deep neural network used in the Pix2Pix generator G is the U-Net [21] which is an encoder-decoder network that uses skip connections between corresponding layers of the encoder and decoder. The encoder takes an input image and maps it to an intermediate representation. The skip connections allow this intermediate representation to carry over features from the input. The decoder then uses the intermediate representation to generate the corresponding output image. The Pix2Pix framework is illustrated in Figure 3.

To see the effectiveness of Pix2Pix for solving jigsaw puzzle problems, we perform experiments on two datasets. In the first experiment, we randomly select 500 images of shoes from the UT Zappos50K dataset [30, 31] and in the second experiment, we use 1400 images of summer scenes from the winter2summer dataset used in [33]. We resize the images to 64×64 pixels. Training examples are constructed using Algorithm 1 with $P = 9$ and $K = 1$.

We train Pix2Pix on the resulting training examples for 65 epochs on the shoes dataset and 100 epochs on the summer scenes dataset. The goal of training is to encourage the Pix2Pix architecture to set its weights so as to reconstruct ordered images from scrambled patches, i.e., the jigsaw problem.

Some results are shown in Figures 4 and 5. From these results, we can see that Pix2Pix successfully solves the jigsaw puzzle problem when the puzzle pieces are not too complex as in the shoe images. However, for patches with complex content such as in the summer scenes, Pix2Pix was less successful in reconstructing meaningful images. This is because Pix2Pix does not simply rearrange the input patches. Instead, it generates each pixel of the output image and therefore has no incentive to transfer a patch unchanged from input to another location in the output. This will be true for any image-to-image translation method that produces the output image in a pixel-by-pixel fashion. Therefore, learning a jigsaw puzzle solver requires novel rethinking of the problem. The next section lays the foundations of one such perspective.

V. A Generative Model for Placement Vectors

Definition 1. Placement Vector: A placement vector for P pieces must have four qualities:

1) It must contain integer-valued entries only.
2) Range of values should be from 1 till P.

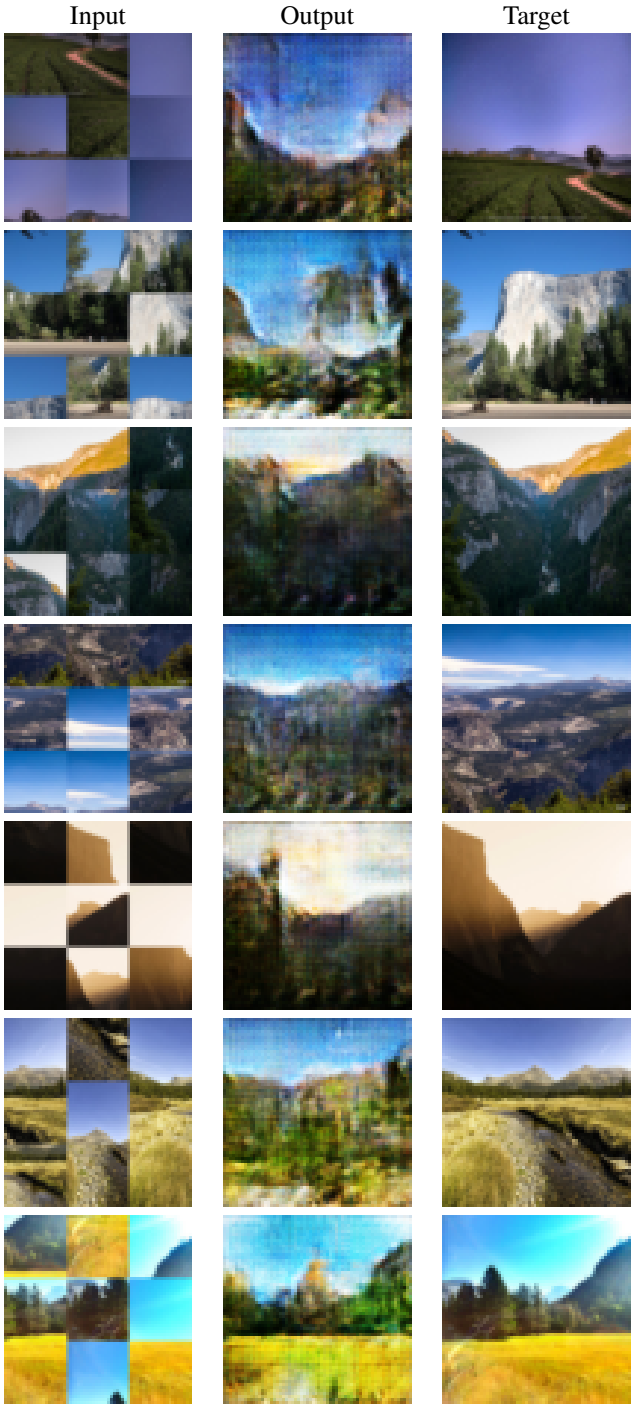

Fig. 5: Outdoor summer scenes regenerated from shuffled input pieces. For such complex scenes, the generated images do not contain exact copies of the input pieces.

3) There should be no missing numbers. Every integer from 1 till P must appear in the placement vector.
4) There should be no duplicates. An integer must appear only once.

A placement vector $\mathbf{v} \in \mathcal{Z}_+^P$, indicates the correct positions

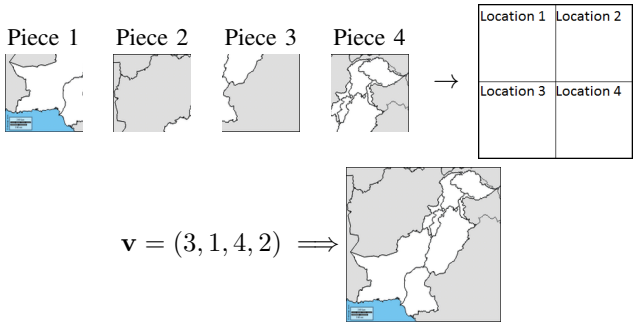

Fig. 6: Illustration of placement vectors. **Top row**: Four pieces to be placed in 2×2 grid. **Bottom row**: Placement vector $\mathbf{v}$ corresponding to correct placement of pieces.

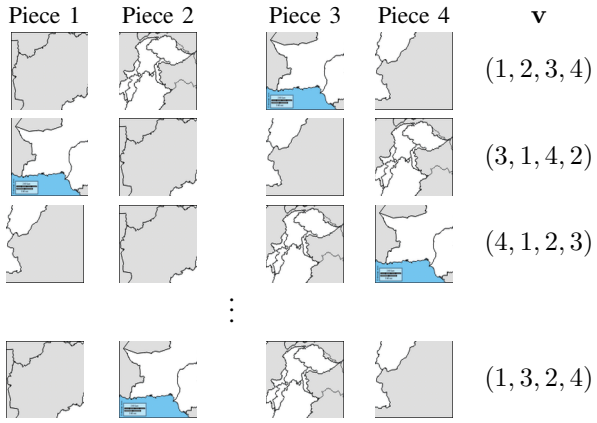

Fig. 7: Some random training examples for learning piece-to-location relationships. For every random ordering of input pieces, the correct placement vector $\mathbf{v}$ is used as the target.

of P pieces on a rectangular grid with P locations where each element $v_i \in \{1, 2, \ldots, n\}$ is the location of piece i on a grid. If $i \neq j$ then each element $v_i \neq v_j$. That is, elements in the placement vector $\mathbf{v}$ should not be duplicates of each other. Figure 6 illustrates how a placement vector captures the piece-to-location mapping. A placement vector $\mathbf{v} = (3, 1, 4, 2)$ for four pieces indicates that first piece should be placed at location 3 on a grid while traversing the grid from left to right and top to bottom, second piece should be placed at location 1, third piece should be placed at location 4 and fourth piece should be placed at location 2 on the grid.

To model the jigsaw problem as a placement vector learning problem, we must first address the following question: "Can a GAN generate samples from the distribution of placement vectors?" Surely, if an unconditional GAN cannot generate vectors resembling placement vectors then there is little hope of a cGAN conditioned on input pieces to generate a vector representing their placement. To answer this question, we generate training data consisting of placement vectors and train a GAN to model the distribution of these vectors.

The training data is generated as follows. For a given number P of locations, we generate random permutations of the set $1, 2, \ldots, P$. There can be $P!$ different permutations of

this set. For our experiment, we randomly select $200,000$ such permutations for $P = 9$ as our training set. This corresponds to the last column of Figure 7. For training a complete piece-to-location mapping model, one would need the random image pieces as well.

Now our goal is to check whether an adversarially trained generator is able to learn to produce placement vectors with the properties discussed in Section VI. The GAN model that we use consists of two simple multilayer perceptrons (MLP) with one hidden layer as the generator and discrimnator networks. Input to the generator is a 50-dimensional random noise vector sampled from a uniform distribution and its output is a P-dimensional vector representing a placement vector. The discriminator takes an P-dimensional vector as input and outputs the probability of that vector being a real placement vector. We train the GAN on our $200,000$ sample training set for $10,000$ epochs. We then use the trained generator to randomly sample $10,000$ vectors. Ideally, these vectors should possess all the qualities of placement vectors. The next section describes how to evaluate the generated set of vectors against the properties possessed by placement vectors according to Definition 1.

VI. Evaluation of placement vector generation

In this section, we explain how to evaluate a system that generates placement vectors. Any system that claims to generate placement vectors must be evaluated for the four qualities mentioned in Definition 1. Accordingly, we compute the following four measures for any placement vector $\mathbf{v}$:

1) Floatingness
2) Out-of-range ratio
3) Missing ratio
4) Duplicate ratio

Standard machine learning modules such as linear regression or neural networks do not necessarily produce integer-valued outputs. But their outputs can be evaluated for their floatingness. Let $\hat{\mathbf{v}} = \text{round}(\mathbf{v})$ be the closest integer-valued placement vector to $\mathbf{v}$. Then floatingness can be computed by the formula given in Equation 7.

$$\text{Floatingness}(\mathbf{v}) = \frac{2}{P}\sum_{i=1}^{P}|v_i - \hat{v}_i| \tag{7}$$

For example, floatingness([3, 6, 2]) is 0 and floatingness([3.5, 6.5, 2.5]) is 1. Out-of-range ratio can be computed by counting the number of entries in $\hat{\mathbf{v}}$ that are less than 1 or greater than P and dividing this count by P. Missing ratio can be computed by counting the integers from 1 to P that do not appear in $\hat{\mathbf{v}}$ and then dividing by P. Duplicate ratio is the number of duplicate entries in $\hat{\mathbf{v}}$ divided by $P - 1$ since the maximum number of duplicate entries can be $P - 1$. Note that all four evaluation measures produce values between 0 and 1. Lower values indicate better placements vectors.

Figure 8 shows two generated placement vectors along with values of their evaluation measures. The first vector has no out-of-range values, includes all numbers from 1 to $P = 9$, has no duplicates and has a floatingness value of 0.21. The second generated vector has no out-of-range values, has two missing numbers (2 and 4), has two duplicated entries and has a floatingness value of 0.34.

A well-known problem with GANs is the mode collapse problem which is the situation when a GAN maps all or most inputs to the same output. Mode collapse implies that generated samples will suffer from low diversity. To evaluate the performance of our GAN which generates placement vectors, we also compute a measure of similarity among the generated samples. Let $(\mathbf{v}_1, \ldots, \mathbf{v}_N)$ denote a set of N placement vectors generated by a GAN. We can compute a measure of similarity amongst the generated vectors as

$$\text{Similarity}(\mathbf{v}_1, \ldots, \mathbf{v}_N) = \frac{2}{N(N-1)}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\mathbb{I}(L0(\mathbf{v}_i, \mathbf{v}_j), 0) \tag{8}$$

where $\mathbb{I}(a, b) = 1$ if a and b are equal and 0 otherwise. The $L0$-norm computes the Hamming distance between two placement vectors. If any two vectors are identical, the term inside the summations will become 1, otherwise it will be 0. So we count only those vectors which are exactly identical to any other sample. Similarity measures the lack of variation in the output of the GAN. The value of similarity will be in range 0 to 1 with 0 implying that all generated samples are distinct and 1 implying that all generated samples are exactly identical to each other.

For $10,000$ placement vectors generated by our trained GAN, the five evaluation measures are shown in Table I. All five measures have values in the range 0 to 1 with lower values indicating better placement vectors. We can see that our trained GAN performed well with respect to all five evaluation measures and successfully learned to generate vectors that can be interpreted as placement vectors.

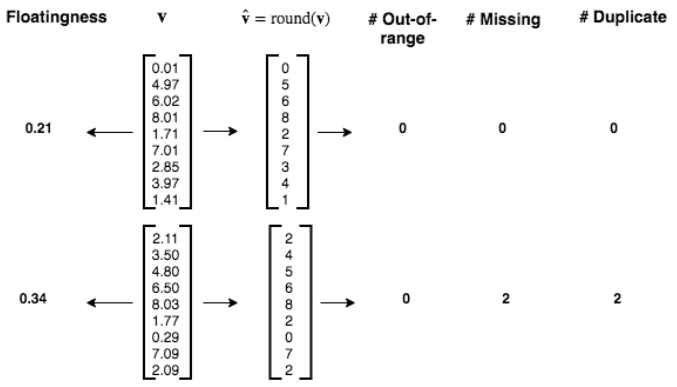

Fig. 8: Sample placement vectors generated by our trained GAN and their evaluation by our evaluation measures. See text for details.

VII. Conclusion

We have presented a generative model of placement vectors. Such vectors can be used to represent ordering of unordered

TABLE I: Evaluation of placement vectors generated by our trained GAN. Range of each measure is from 0 to 1 with lower values indicating better placement vectors.

Floatingness Ratio	0.1930
Out-of-Range Ratio	0.0001
Missing Ratio	0.0533
Duplicate Ratio	0.0596
Similarity	0.0012

data such as dissected image pieces of a jigsaw puzzle problem or reconstruction of a larger map from its constituent pieces. Image-to-image translation models have been shown to be insufficient for this problem since they do not reconstruct the original pieces exactly. This motivates a new perspective of the problem which involves learning piece-to-location mappings. We have focused on whether a GAN has the representational power to generate location or placement vectors. We have shown that an unconditioned GAN can be trained to generate vectors with properties resembling those for placement vectors. We have also presented evaluation measures for any system claiming to generate placement vectors and have shown that our trained GAN has favorable values for these evaluation measures.

The natural question that remains to be answered is whether a conditional GAN can learn the complete piece-to-location mapping task. In other words, can a GAN learn to solve jigsaw puzzle problems. The current work represents the first step towards answering this question in the sense that we have shown that an unconditioned GAN is able to generate placement vectors representing locations.

VIII. Acknowledgements

Figures 1, 6 and 7 used content from digital collections of the British Library and from davidrumsey.com and d-maps.com.

References

[1] Tom Altman. "Solving the jigsaw puzzle problem in linear time". In: *Applied Artificial Intelligence an International Journal* 3.4 (1989), pp. 453–462.

[2] Fernanda A Andalo, Gabriel Taubin, and Sione Goldenstein. "Solving image puzzles with a simple quadratic programming formulation". In: *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE. 2012, pp. 63–70.

[3] Jeffrey Bosboom et al. "Even 1× n Edge-Matching and Jigsaw Puzzles are Really Hard". In: *Journal of Information Processing* 25 (2017), pp. 682–694.

[4] Ivan D Chase. "The sequential analysis of aggressive acts during hierarchy formation: an application of the jigsaw puzzle approach". In: *Animal Behaviour* 33.1 (1985), pp. 86–100.

[5] Taeg Sang Cho, Shai Avidan, and William T Freeman. "A probabilistic image jigsaw puzzle solver". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 183–190.

[6] Taeg Sang Cho et al. "The patch transform and its applications to image editing". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[7] Min Gyo Chung, Margaret M Fleck, and David A Forsyth. "Jigsaw puzzle solver using shape and color". In: *Signal Processing Proceedings, 1998. ICSP'98. 1998 Fourth International Conference on*. Vol. 2. IEEE. 1998, pp. 877–880.

[8] Erik D Demaine and Martin L Demaine. "Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity". In: *Graphs and Combinatorics* 23.1 (2007), pp. 195–208.

[9] Herbert Freeman and L Garder. "Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition". In: *IEEE Transactions on Electronic Computers* 2 (1964), pp. 118–127.

[10] Andrew 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.

[11] Nadine C Gassner, Walter A Baase, and Brian W Matthews. "A test of the" jigsaw puzzle" model for protein folding by multiple methionine substitutions within the core of T4 lysozyme". In: *Proceedings of the National Academy of Sciences* 93.22 (1996), pp. 12155–12158.

[12] David Goldberg, Christopher Malon, and Marshall Bern. "A global approach to automatic solution of jigsaw puzzles". In: *Proceedings of the eighteenth annual symposium on Computational geometry*. ACM. 2002, pp. 82–87.

[13] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680.

[14] L. Hannas. *The English jigsaw puzzle, 1760-1890: with a descriptive check-list of puzzles in the museums of Great Britain and the author's collection*. Wayland, 1972.

[15] Phillip Isola et al. "Image-to-image translation with conditional adversarial networks". In: *arXiv preprint* (2017).

[16] Florian Kleber and Robert Sablatnig. "A survey of techniques for document and archaeology artefact reconstruction". In: *2009 10th International Conference on Document Analysis and Recognition*. IEEE. 2009, pp. 1061–1065.

[17] William Marande and Gertraud Burger. "Mitochondrial DNA as a genomic jigsaw puzzle". In: *Science* 318.5849 (2007), pp. 415–415.

[18] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[19] Ture R Nielsen, Peter Drewsen, and Klaus Hansen. "Solving jigsaw puzzles using image features". In: *Pattern Recognition Letters* 29.14 (2008), pp. 1924–1933.

[20] Dolev Pomeranz, Michal Shemesh, and Ohad 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.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[22] Ankush Roy and Utpal Garain. "A Probabilistic Model for Reconstruction of Torn Forensic Documents". In: *2013 12th International Conference on Document Analysis and Recognition*. IEEE. 2013, pp. 494–498.

[23] Dror Sholomon, Omid E David, and Nathan S Netanyahu. "A Generalized Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles of Complex Types." In: *AAAI*. 2014, pp. 2839–2845.

[24] Dror Sholomon, Omid David, and Nathan S. Netanyahu. "A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013.

[25] Kilho Son, James Hays, and David B Cooper. "Solving square jigsaw puzzles with loop constraints". In: *European Conference on Computer Vision*. Springer. 2014, pp. 32–46.

[26] Fubito Toyama et al. "Assembly of puzzles using a genetic algorithm". In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. Vol. 4. IEEE. 2002, pp. 389–392.

[27] Roger W Webster, Paul S LaFollette, and Robert L Stafford. "Isthmus critical points for solving jigsaw puzzles in computer vision". In: *IEEE transactions on systems, man, and cybernetics* 21.5 (1991), pp. 1271–1278.

[28] Xingwei Yang, Nagesh Adluru, and Longin Jan Latecki. "Particle filter with state permutations for solving image jigsaw puzzles". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 2873–2880.

[29] Feng-Hui Yao and Gui-Feng Shao. "A shape and image merging technique to solve jigsaw puzzles". In: *Pattern Recognition Letters* 24.12 (2003), pp. 1819–1835.

[30] A. Yu and K. Grauman. "Fine-Grained Visual Comparisons with Local Learning". In: *Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[31] A. Yu and K. Grauman. "Semantic Jitter: Dense Supervision for Visual Comparisons via Synthetic Images". In: *International Conference on Computer Vision (ICCV)*. Oct. 2017.

[32] Rui Yu, Chris Russell, and Lourdes Agapito. "Solving Jigsaw Puzzles with Linear Programming". In: *arXiv preprint arXiv:1511.04472* (2015).

[33] Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *arXiv preprint arXiv:1703.10593* (2017).