TopoGAN: A Topology-Aware Generative Adversarial Network – Supplementary Material –

Fan Wang[0000−0002−5402−5065], Huidong Liu, Dimitris Samaras, and Chao Chen

Stony Brook University, Stony Brook, NY 11794, USA
{fanwang1,huidliu,samaras}@cs.stonybrook.edu
chao.chen.1@stonybrook.edu

We provide additional discussion of technical details (Section A), experiment details and more qualitative results (Section B). These are not included in the paper due to space constraints.

A Additional Technical Details for the Method

We provide technical details that are not included in the manuscript due to space constraints. In Section A.1, we formally introduce the definitions of topology and persistent homology. In Section A.2, we add details of comparing diagrams and comparing sets of diagrams. In Section A.3, we add detailed derivations of the gradient with regard to the synthetic images.

A.1 Homology, Persistent Homology and Computation

We focus on homology over finite fields GF(2), and leave the more general settings to a more comprehensive text [10, 4]. Given a topological space, e.g., a mask y, each connected component corresponds to a 0-dimensional homology class. Each hole corresponds to a 1-dimensional homology class, i.e., an equivalent class of cycles (loops) such that the difference between any two of them is the boundary of a patch within y. To formalize the definitions, we start with a discretization of the image domain, Ω.

We assume a *cubical complex* consisting of 0-, 1-, and 2-dimensional elements. 0-dimensional elements (vertices) correspond to pixels. 1-dimensional elements are horizontal and vertical edges connecting adjacent pixels. 2-dimensional elements are squares bounded by 2 horizontal edges and 2 vertical edges. A p-chain is a set of p-dimensional elements. The set of all possible p-chains is called the p-dimensional chain group, denoted as C_p. Assume N_p is the number of p-dimensional elements, the cardinality of C_p is 2^{N_p}. We can represent each p-chain as a binary vector of length N_p, using 1's or 0's to indicate whether a particular p-dimensional element does/does not belong to the chain. The sum of any two p-chains is the sum of the two corresponding binary vectors over GF(2) field operations: $1+1=0$, $1+0=1$, and $0+0=0$. This is equivalent to the exclusive-or operation of the two chains treated as two sets of p-dimensional elements.

A boundary operator, ∂_p, maps a p-dimensional element to a $(p-1)$-chain containing its boundary elements. The boundary of an edge includes the two vertices adjacent to it. The boundary of a square includes the 4 edges bounding it. This can be naturally extended to the boundary of a p-chain: $\partial_p(c) = \sum_{\sigma \in c} \partial_p(\sigma)$. The boundary operator is equivalent to multiplying the chain vector with a boundary matrix, $\partial_p(c) \coloneqq D_p \cdot c$. A p-dimensional boundary matrix is an $N_{p-1} \times N_p$ binary matrix. The (i, j)'s entry is 1 if and only if the i-th $(p-1)$-dimensional element belongs to the boundary of the j-th p-dimensional element. The 1-dimensional boundary matrix, D_1, is essentially the incidence matrix of the underlying graph. The boundary operator is a linear operator mapping C_p into C_{p-1}.

A p-cycle is a p-chain whose boundary is empty. A p-boundary is a p-chain which is the boundary of some $(p+1)$-chain. Formally, the set of p-cycles, called the cycle group, denoted as Z_p, is the kernel of ∂_p. The set of p-boundaries, called the boundary group, denoted as B_p, is the image of ∂_{p+1}. It can be verified that a boundary is a cycle, and B_p is a proper subgroup of Z_p. We can now formally define the homology group as the quotient group,

$$\mathsf{H}_p \coloneqq \frac{\mathsf{Z}_p}{\mathsf{B}_p}.$$

Each element of the homology group, called a *homology class*, is an equivalent class of cycles so that their difference is only a boundary. Formally, for any $h \in \mathsf{H}_p$,

$$h = \{z_0 + b \mid b \in \mathsf{B}_p\},$$

for some $z_0 \in \mathsf{Z}_p$. Intuitively, when $p = 0$, a homology class corresponds to a connected component or a set of connected components. When $p = 1$, a homology class corresponds to a hole or a set of holes. The *Betti number of dimension p*, β_p, is the dimension of the homology group, i.e., the maximum number of classes that are linear independent from each other. Essentially, β_0 and β_1 count the number of connected components and the number of holes, respectively.

Persistent homology and its computation. In persistent homology, we use a scalar function to filter the whole image domain Ω. Here we use the distance transform, f_y, as the input scalar function. Recall a sublevel set is part of the domain whose function value is below a given threshold, $\Omega^t_{f_y} = \{x \in \Omega \mid f_y(x) \leq t\}$. The sublevel sets with the monotonically growing thresholds form a nested sequence called the filtration:

$$\emptyset = \Omega^{t_0}_{f_y} \subseteq \Omega^{t_0}_{f_y} \subseteq \cdots \subseteq \Omega^{t_m}_{f_y} = \Omega, \text{ where } -\infty = t_0 < t_1 < \cdots < t_m = +\infty.$$

Since there is an inclusion map from $\Omega^{t_i}_{f_y}$ to $\Omega^{t_{i+1}}_{f_y}$, there exists a well-behaved mapping (homomorphism) between their homology groups.

$$\mathsf{H}_p(\Omega^{t_0}_{f_y}) \to \mathsf{H}_p(\Omega^{t_1}_{f_y}) \to \cdots \to \mathsf{H}_p(\Omega^{t_m}_{f_y}).$$

Denote by $F^{i,j}_p$ the induced mapping of homology group of the i-th sublevel set to the homology group of the j-th sublevel set, $F^{i,j}_p : \mathsf{H}_p(\Omega^{t_i}_{f_y}) \to \mathsf{H}_p(\Omega^{t_j}_{f_y})$. There

exists a homology class created at t_i and killed at t_j if and only if

$$P_p^{i,j} := \frac{\operatorname{im} F_p^{i,j-1} \cap \ker F_p^{j-1,j}}{\operatorname{im} F_p^{i-1,j-1} \cap \ker F_p^{j-1,j}} = 1.$$

This means a point is added to the persistence diagram at (t_i, t_j).

Computation. To compute the persistence diagram from an input function f_y, we first assign a function value to all elements of the cubical complex. The value of each vertex (pixel) is its corresponding f_y value. The function value of an edge is the maximum function value of its two adjacent vertices. The function value of a square is the maximum function value of the four associated vertices. To compute, we first sort the rows and columns of the boundary matrix D_p monotonically according to the function values of the vertices/edges/squares. For the sorted matrix, we run a matrix reduction algorithm from left to right. The algorithm is similar to a Gaussian elimination algorithm, except that it does not permit row or column perturbations.

For column i, we denote by $\text{low}(i)$ the row index of the lowest nonzero entry. We check if any previous column, $i' < i$, has its lowest entry the same as $\text{low}(i)$. If yes, we add column i' to column i. As a result, the lowest nonzero entry of column i is cleared (becomes 0). We recalculate $\text{low}(i)$ and continue. we repeat this procedure until the whole column becomes 0 or $\text{low}(i)$ becomes unique, i.e., no previous column has the same lowest nonzero row index. The resulting lowest nonzero entry, $(\text{low}(i), i)$, corresponds to a persistent homology class that is created when $\sigma_{\text{low}(i)}$ is added to the filtration, and is killed when σ_i is added. We add a point to the persistence diagram with coordinates being the corresponding function values, $(f_y(\sigma_{\text{low}(i)}), f_y(\sigma_i))$.

The computational complexity of the persistence diagram is $O(N^3)$, in which $N = N_0 + N_1 + N_2$ counts the number of elements in the cubical complex. Note N is linear to the number of pixels. Therefore the algorithm is cubic to the image size in the worst case. In practice, it is reasonably efficient. Please find the time cost of the algorithm in Section B.2.

A.2 Distance Between Diagrams and Comparing Sets of Diagrams

The original definition between two persistence diagrams is the p-Wasserstein distance between the two point sets within $\mathbb{R}^2$:

$$\mathcal{W}_p(\text{dgm}_1, \text{dgm}_2) = \min_{\sigma \in \Sigma} \left(\sum_{x \in \text{dgm}_1} \|x - \sigma(x)\|^p \right)^{\frac{1}{p}}.$$

Here σ is a one-to-one correspondence between the two diagrams and Σ is the set of all possible such correspondences. When $p = \infty$, the distance is called the *bottleneck distance*. It has been proven that the bottleneck distance and the more general p-Wasserstein distance are stable, i.e., the difference between two diagrams is uppderbounded by the difference between the two underlying scalar

functions (distance transforms in our case) [2, 3]. It is important to note that each diagram is extended to include all points in the diagonal line, $\Delta \coloneqq \{(b, d) \mid b = d\}$. A nontrivial point in dgm_1 can be matched to either a nontrivial point in dgm_2 or its closest point on the diagonal line.

In our paper, we project all diagrams to their birth time axis and focus on matching w.r.t. birth time only. We also share a similar setting: we match diagram points to nontrivial ones as well as trivial ones. In particular, we match all points in a synthetic diagram to either points of the corresponding real diagram, or to the trivial points ($b = 1$). Since we are generally less interested in removing almost-holes, we only take into consideration the ones matched to the nontrivial points in the real diagram and use their matching distance as the distance $\mathcal{W}_1(\mathrm{dgm}_i^s, \mathrm{dgm}_j^r)$. Computing optimal matching in 1D is not expensive. The complexity is quasilinear to the number of nontrivial points in the diagram. Please note that this is only for the 1-dimensional topology (holes). For 0-dimensional topology (connected components), we will project everything to the death time and carry out similar matching computations.

Comparing sets of diagrams. To define a loss, we need to compare different persistence diagram distributions. In the manuscript, we chose to use optimal transport to find matching between synthetic and real diagrams, and define the loss as the total matching distance. We do this with both computation and optimization efficiency in mind. Below we discuss some other possible options and explain why they are not adopted.

The space of persistence diagrams, endowed with the p-Wasserstein distance, is well behaved statistically [9]. One can define and compute the Fréchet mean and variance of a distribution of persistence diagrams given sufficient sample diagrams [13]. However, we are not using this method as the computation is very expensive and it is not likely to be the best choice. It involves starting with a mean diagram and iteratively evaluate and optimize its distance with all sample diagrams. Evaluating the distance of the mean diagram with each sample diagram involves one round of the Hungarian method, which is cubic to the diagram size, but can be slightly faster as the points are embedded in 2D [7]. Also the result is only a locally optimal choice (the Fréchet mean of persistence diagrams is not necessarily unique).

An alternative idea is to consider the two diagram distributions in the reproducing kernel Hilbert space and compare them via Maximum Mean Discrepancy (MMD) [5]. Various persistent diagram kernels [1, 12, 8] can be used. Whatever kernel is used, MMD will be evaluated on all pairs of diagrams. Its gradient will try to match each synthetic diagram to all real diagrams. The different matching directions will cancel each other out, resulting in optimization inefficiency.

Meanwhile, we use MMD to evaluate GAN performance in terms of topology (Section 3.4 in the manuscript). When choosing an evaluation metric, we are less concerned about the computational and optimization efficiency. Thus, We

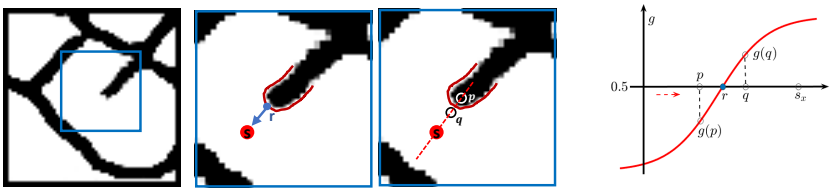

Fig. 1. p is the pixel that is closest to s_x and $g(p) < 0.5$. Find a neighbor pixel of p, say q, such that $g(q) > 0.5$ and p, q and s_x are along a straight line. We fit a sigmoid function crossing $(p, g(p))$ and $(q, g(q))$. We need to find the coordinate of r such that its sigmoid function value is 0.5.

choose the unbiased MMD:

$$\mathrm{MMD}_u(\mathcal{D}_{syn}, \mathcal{D}_{real})^2 = \frac{1}{n(n-1)} \sum_{i,j=1, j\neq i}^{n} \langle \Phi(\mathrm{dgm}_i^s), \Phi(\mathrm{dgm}_j^s) \rangle_{\mathcal{H}}$$
$$+ \frac{1}{n(n-1)} \sum_{i,j=1, i\neq j}^{n} \langle \Phi(\mathrm{dgm}_i^r), \Phi(\mathrm{dgm}_j^r) \rangle_{\mathcal{H}} - \frac{2}{n^2} \sum_{i,j=1}^{n} \langle \Phi(\mathrm{dgm}_i^s), \Phi(\mathrm{dgm}_j^r) \rangle_{\mathcal{H}} \quad (1)$$

Here $\langle \Phi(\mathrm{dgm}_i^*), \Phi(\mathrm{dgm}_j^*) \rangle$ are the kernel distance between the corresponding synthetic/real diagrams.

A.3 Calculation of the gradient w.r.t. $G(z_i)$

We add additional details about the gradient of the loss w.r.t. the i-th synthetic image, whose mask is y_i. For simplicity we denote $g = G(z_i)$. Recall the birth time of an almost-hole (point x in the diagram) is the distance transform function value at the saddle point s_x, $b_x = f_{y_i}(s_x)$. It suffices to focus on the function value at s_x, $\frac{\partial f_{y_i}(s_x)}{\partial g}$. Note the synthetic image g is different from the synthetic mask y_i, which is the sublevel set of g thresholded at value 0.5. When we take a gradient step to decrease the distance transform function value at a saddle point s_x, we are essentially shrinking/growing the mask y_i at its closest boundary point, r, along the normal direction of the boundary of y_i (see Fig. 1). To achieve the goal, we need to change the function values of g near this point. In particular, we find two pixels p and q whose function values can uniquely determine the location of r. This gives us the opportunity to rewrite the distance between s_x and r as a function of $g(p)$ and $g(q)$. Then the gradient can be calculated.

Denote by p the pixel within y_i that is closest to s_x. Let q be the neighbor pixel of p that is closest to s_x. Obviously $q \notin y_i$. Their function values $g(p) < 0.5 < g(q)$. We approximate r by the intersection of the line (p, q) and the boundary of y_i. Since g is only sampled at pixels, we interpolate it between p and q using a sigmoid function (red curve in Fig. 1). The location of r (which sits in between q and p) can be exactly solved as:

$$r = p + \frac{q-p}{\|q-p\|} \ln \left[\frac{g(q) - g(p)}{g(q) \cdot e^{-\|q-p\|} - g(p)} \right].$$

We then take the partial derivative of $f_{y_i}(s_x) = \|s_x - r\|$ w.r.t. $g(q)$ and $g(p)$. This gives us the derivative $\frac{\partial f_{y_i}(s_x)}{\partial g}$. In practice, the calculation will be significantly simplified if we assume p, q, r and s_x are colinear.

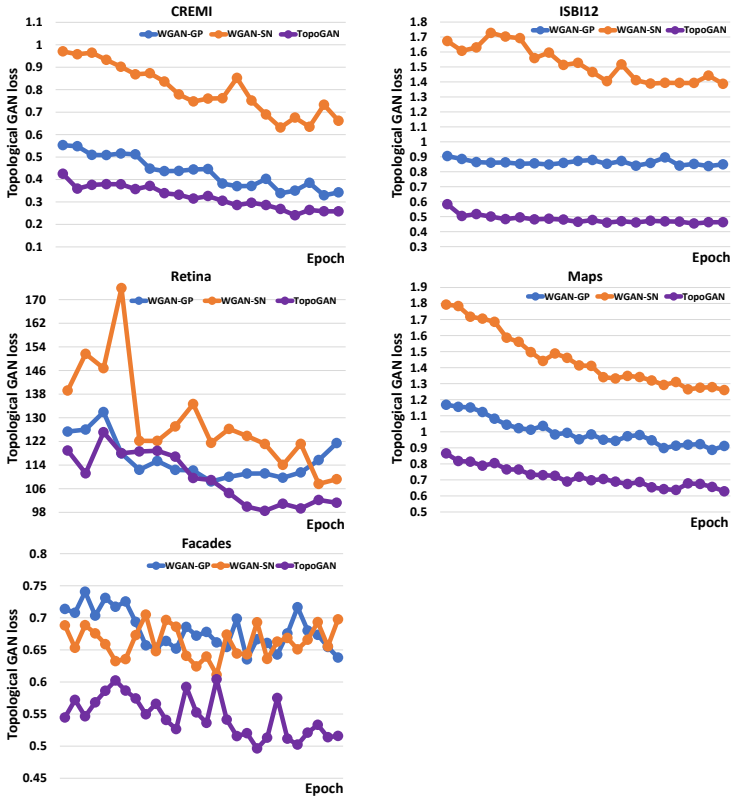

Fig. 2. We report the averaged topological GAN loss during training on different datasets. Blue curve: WGAN-GP; orange curve: WGAN-SN; purple curve: TopoGAN.

B Additional Experiment Details

We first provide details of how TopoGAN is implemented and trained in Section B.1. In Section B.2, we explore how TopoGAN behaves when the weight of topological GAN loss changes and the effects of GAN architecture on TopoGAN. TopoGAN proves to be architecture agnostic when we replace the backbone networks with ResNets. At the end of Section B.2, we share details about the time consumption of topological GAN loss computation. In the manuscript, we have shown how TopoGAN can help in a downstream segmentation task. In

Fig. 3. Evaluation of TopoGANs trained with different loss weights on the **CREMI** dataset. From left to right, the loss weights are: 0.001, $0.001/2$, $0.001/2^2$, $0.001/2^3$, $0.001/2^4$, and 0. We report mean of FID, unbiased MMD, and Betti score out of a 3-fold cross validation.

Section B.3, we provide detailed descriptions of our segmentation pipeline and its training strategy. We also discuss the segmentation results. This section is concluded with more qualitative results from **CREMI**, **ISBI12**, and **Maps** for reference.

B.1 TopoGAN implementation and training

TopoGAN is built on top of WGAN-GP. We follow the standard training procedure of WGAN-GP. We use $\alpha = 0.0002$, $\beta_1 = 0$, and $\beta_2 = 0.99$ as the Adam hyperparameters and 10 as gradient penalty coefficient. Deep convolutional generative adversarial networks [11] (DCGANs) are adopted as the backbone network architectures for TopoGAN. For each dataset, all networks are trained from scratch with a batch size of 32 to an equal number of epochs. We use a CPU implementation of linear programming (Python Optimal Transport LP solver) to solve for optimal transport. The complexity of optimal transport is $O(n^{2.5})$ where n is the number of images.

During TopoGAN training, the topological GAN loss is not applied to the generator until training of WGAN-GP stabilizes. This is to improve efficiency and ensure that the topology loss is applied to almost-complete images. We discard topological structures with lifespan under 1.0 with a filter on persistence diagrams. At an iteration of generator training, a batch of synthetic images are matched to the randomly selected half of the whole training set with optimal transport for topology loss computation with Eq. (4) in the manuscript. We match half of the training set for efficiency purposes. We use 0.0005 as the the weight of the topological GAN loss throughout the entire experiment.

B.2 Ablation study of the topological GAN loss

Fig. 2 depicts how the topological GAN loss changes during training. We report the loss averaged over all batches for each epoch. With explicit distance minimization in topological space, TopoGAN consistently shows smaller topology loss than baseline GANs.

Table 1. Evaluation of TopoGANs trained with different weights of topological GAN loss on the **CREMI** dataset. Mean and standard deviation of a 3-fold cross validation is reported for FID, unbiased MMD, and Betti score.

	FID	unbiased MMD	Betti score
0.001	206.990±0.787	0.146±0.005	6.498±0.173
0.001/2	20.967±0.195	0.134±0.019	0.015±0.001
$0.001/2^2$	21.385±0.115	0.138±0.002	0.068±0.004
$0.001/2^3$	21.008±0.666	0.140±0.017	0.128±0.007
$0.001/2^4$	21.403±0.500	0.136±0.002	0.160±0.005
0	21.646±0.138	0.142±0.014	0.236±0.003

Table 2. Comparisons of TopoGAN against baseline GANs w.r.t. FID, unbiased MMD, and Betti score on the **CREMI** dataset. We use ResNets as backbone architectures for all GANs.

	FID	unbiased MMD	Betti score
WGAN-GP	17.085±0.113	0.091±0.006	0.141±0.007
WGAN-SN	88.496±0.259	0.498±0.021	0.740±0.007
TopoGAN	**14.560**±0.045	**0.049**±0.009	**0.062**±0.003

The weight of topological GAN loss. We use different weights of topological GAN loss to study how it affects GAN training. We test weights of 0.001, 0.001/2, $0.001/2^2$, $0.001/2^3$, $0.001/2^4$ and 0 in our experiment. TopoGANs trained with different loss weights on **CREMI** dataset are finally evaluated with three metrics: FID, unbiased MMD, and Betti score. For all metrics, lower number indicates better performance. The results are summarized in Table 1 and visualized in Fig. 3. Note that when the weight is 0 the model degrades to WGAN-GP.

We empirically set the weight of topological GAN loss as 0.001/2 and reported experiment results in the manuscript. When the weight is too large (0.001), the generator is heavily biased and the discriminator fails to pull the generator back afterwards. Therefore we have large numbers for all metrics (FID, unbiased MMD, and Betti score). As the weight gradually decreases from 0.001, all three metrics first decrease to the minimum at 0.001/2 (the chosen loss weight in the manuscript) and then increase as expected and achieve their maximum when the loss weight reaches 0.

TopoGAN architecture. To explore the effects of GAN architectures on TopoGAN, we replace DCGANs used as backbone architectures in the manuscript with ResNets for all GANs and report performance of TopoGAN, WGAN-GP, WGAN-SN on **CREMI** in Table. 2. All GANs are trained for equal number of epochs. TopoGAN still outperforms WGAN-GP and WGAN-SN on all three metrics with ResNets as architecture which suggests architecture agnostic nature of TopoGAN.

Topological GAN loss computation cost. The time performance in this section is reported from a computer with Intel Core i7-9700K and 8GB RAM. The loss computation is performed on a CPU. The persistence diagrams of the training data can be pre-computed and pre-loaded during training. Therefore, we do not account for this time when calculating the topological GAN loss. Computing persistence diagrams (either dimension 0 or dimension 1) for a 64×64 grayscale image takes roughly 0.022 seconds. For a 128×128 grayscale image, it takes about 0.055 seconds. Note the time taken to compute persistence diagrams depends heavily on the images. Images containing a significantly larger number of topological structures may take longer. Calculating 1-dimension $\mathcal{W}_1$ distance for a pair of persistence diagrams consumes approximately 0.00012 seconds. Loss computation for a pair of diagrams including $\mathcal{W}_1$ distance, optimal transport, and other routines takes about 0.0002 seconds for 0-dimension topological structures and 0.00035 seconds for 1-dimension structures. As an example, computing topological GAN loss for a batch of 32 **CREMI** images with a database containing 600 images, takes roughly 7.424 seconds ($0.022 \times 32 + 32 \times 600 \times 0.00035 = 7.424$).

B.3 Application to a downstream task – segmentation

We provide additional experiment details of the downstream segmentation task (results reported in Section 4 of the manuscript). Fig. 4 illustrates our segmentation pipeline. A texture network (pix2pix) is first trained with real training data. This network is then applied to paint texture on masks generated by TopoGAN and other baseline GANs. A mask and its corresponding texture image are referred to as a synthetic data pair. We compare the segmentation performances of UNets trained with synthetic data pairs from TopoGAN, WGAN-GP, WGAN-SN and with real training data. We also train segmentation networks with real training data augmented with synthetic data from each GAN method. The segmentation networks are evaluated on real test data w.r.t. three metrics: (1) pixel accuracy, (2) Dice score, and (3) Adapted Rand Index (ARI). We report mean and standard deviation of the above metrics from a 3-fold cross validation in Table 3. A total of 21 segmentation networks are trained for each dataset (3 trained with real data, 9 trained with synthetic data, and 9 trained with real plus synthetic data for each GAN method).

From Table 3, segmentation networks trained with real data augmented with synthetic data from TopoGAN achieve best results across all three datasets and metrics. However, TopoGAN outperforms the baseline GANs by only a small margin. In terms of qualitative results, continuous boundaries generated by TopoGAN are destroyed in the textured masks output from pix2pix (see Figs. 6, 7, and 8). The texturing process weakens the advantage TopoGAN possesses to generate continuous edges and complete loops which potentially explains the marginal improvement.

Segmentation network training. As **Maps** and **Facade** do not have ground truth segmentation data, we apply our segmentation pipeline to **CREMI**, **ISBI12**,

and **Retina**. A pix2pix is trained for each dataset with paired data (texture images + segmentation masks) and we use ResNet [6] with 9 blocks as architecture for pix2pix. The default training parameters are kept intact with batch size of 8. Each UNet is trained for 40 epochs with batch size as 32 and learning rate as 0.005 without image resizing on synthetic or real data. For data augmentation (GP+real data, SN+real data, Topo+real data in Table 3), UNets are first pretrained for 40 epochs with the synthetic data. The pretrained UNets are then fine-tuned on real training data for another 20 epochs. All networks are finally evaluated on the real test data.

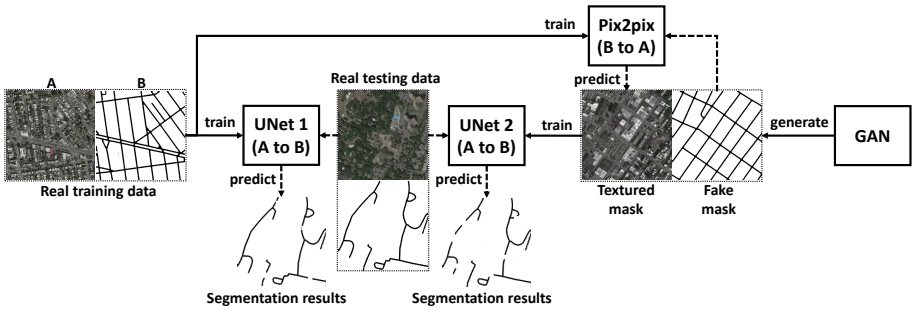

Fig. 4. Pipeline of our segmentation pipeline. We train a texture network (pix2pix) and a segmentation network (UNet) with paired real training data. The trained pix2pix takes as inputs the generated masks and outputs textured masks. Masks and textured masks are then used to train UNets. All segmentation networks are evaluated on real test data w.r.t three metrics: Pixel accuracy, Dice score, and Adapted Rand Index (ARI).

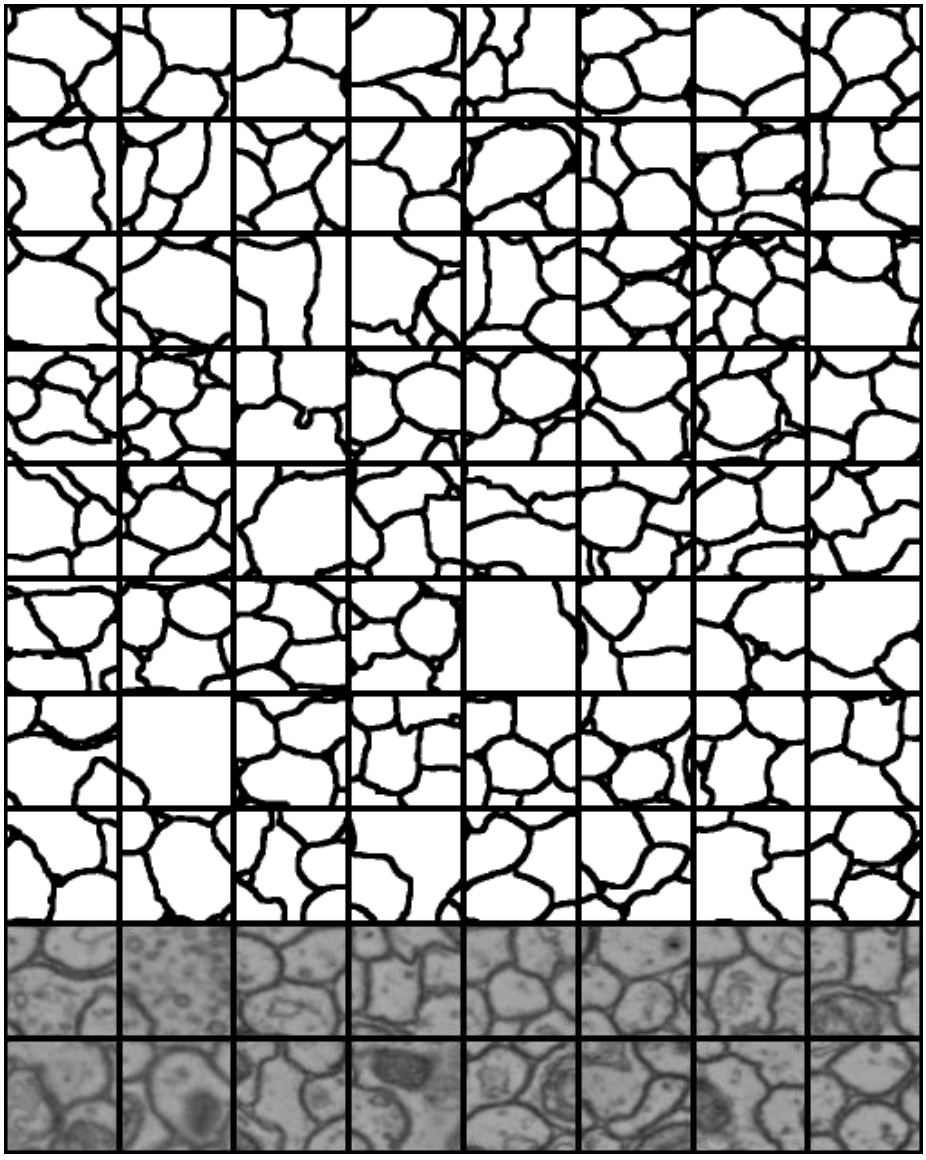

Fig. 5. Real masks and textures of **CREMI** for reference. Textures correspond to the last two rows of masks. In the real textures, boundaries are clear and loops are complete.

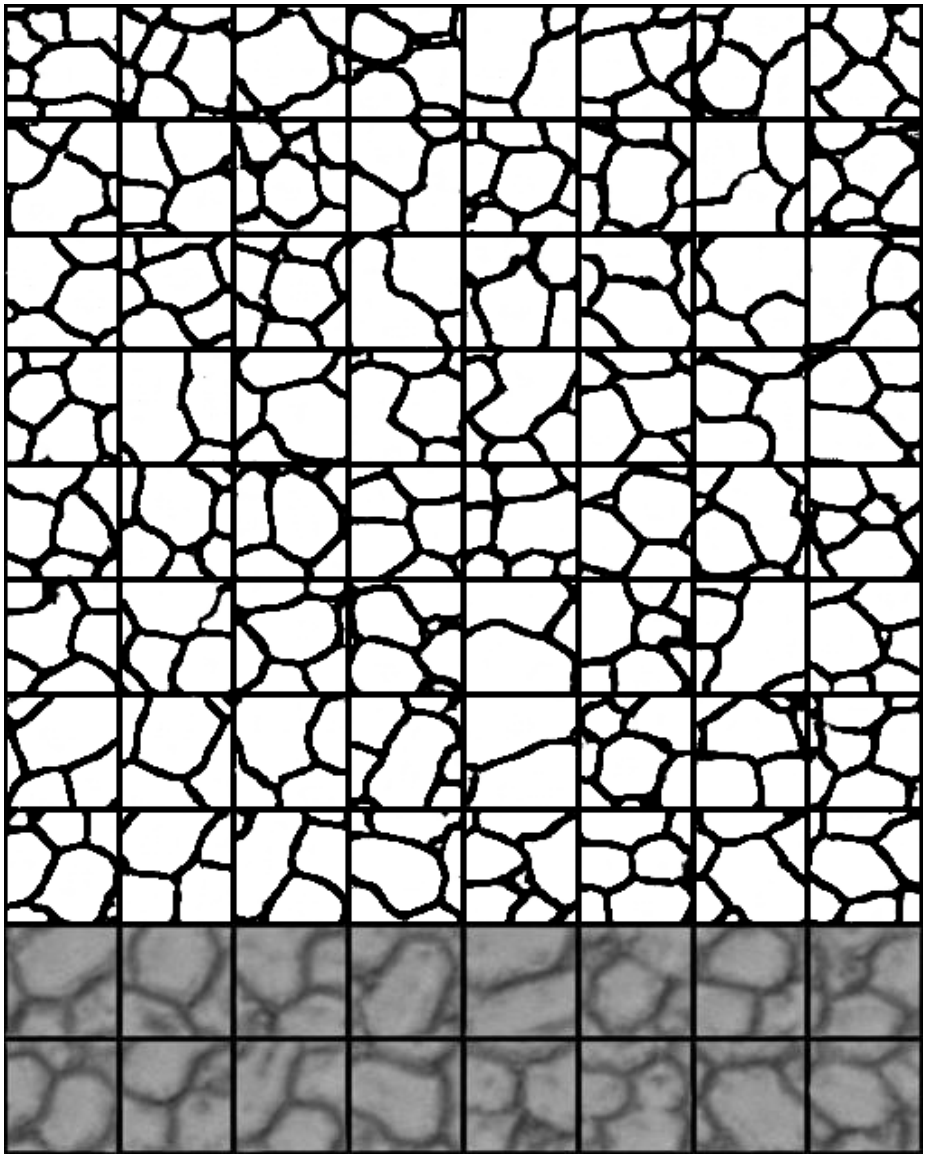

Fig. 6. Fake masks and textures of **CREMI** from TopoGAN. Textures correspond to the last two rows of masks. Note in the textured masks by pix2pix, continuous boundaries in original masks are destroyed and disconnected. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

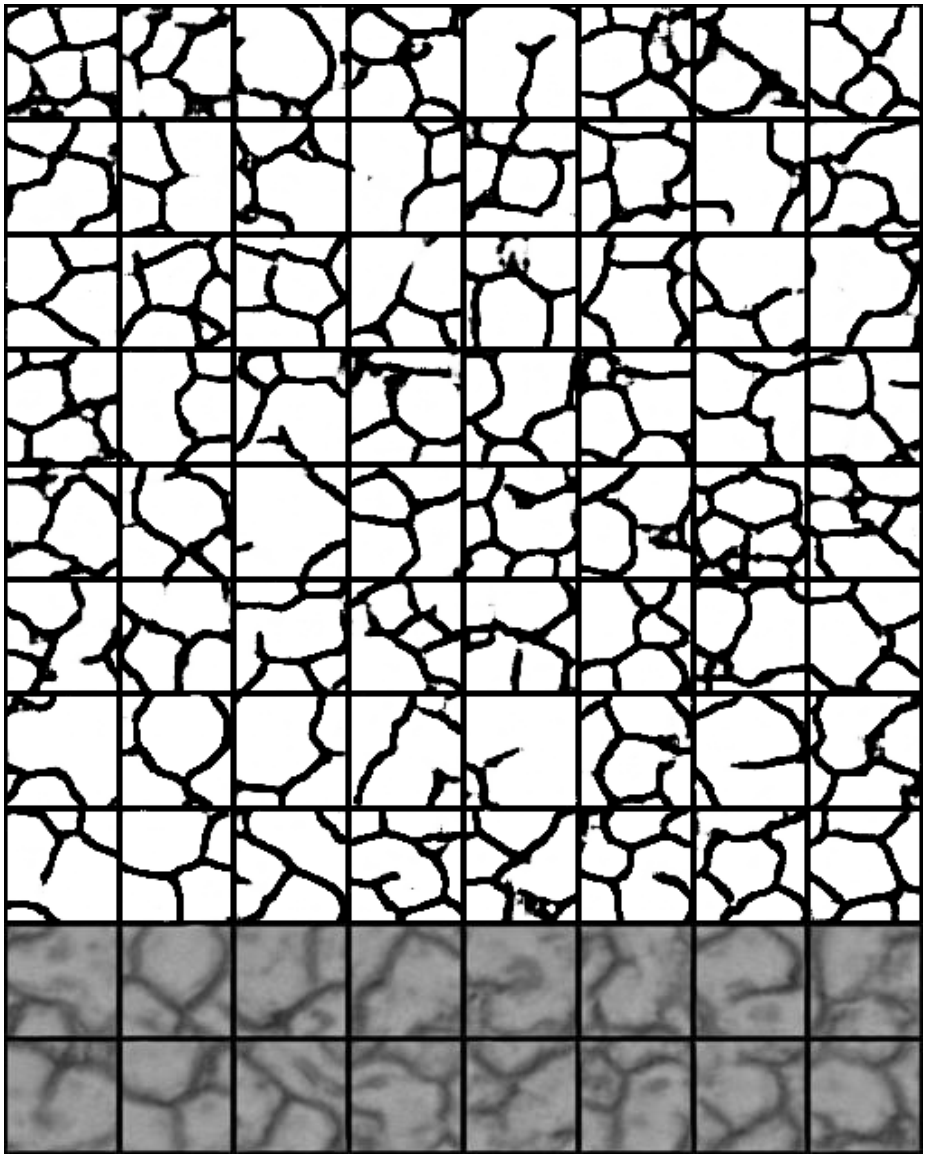

Fig. 7. Fake masks and textures of **CREMI** from WGAN-GP. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

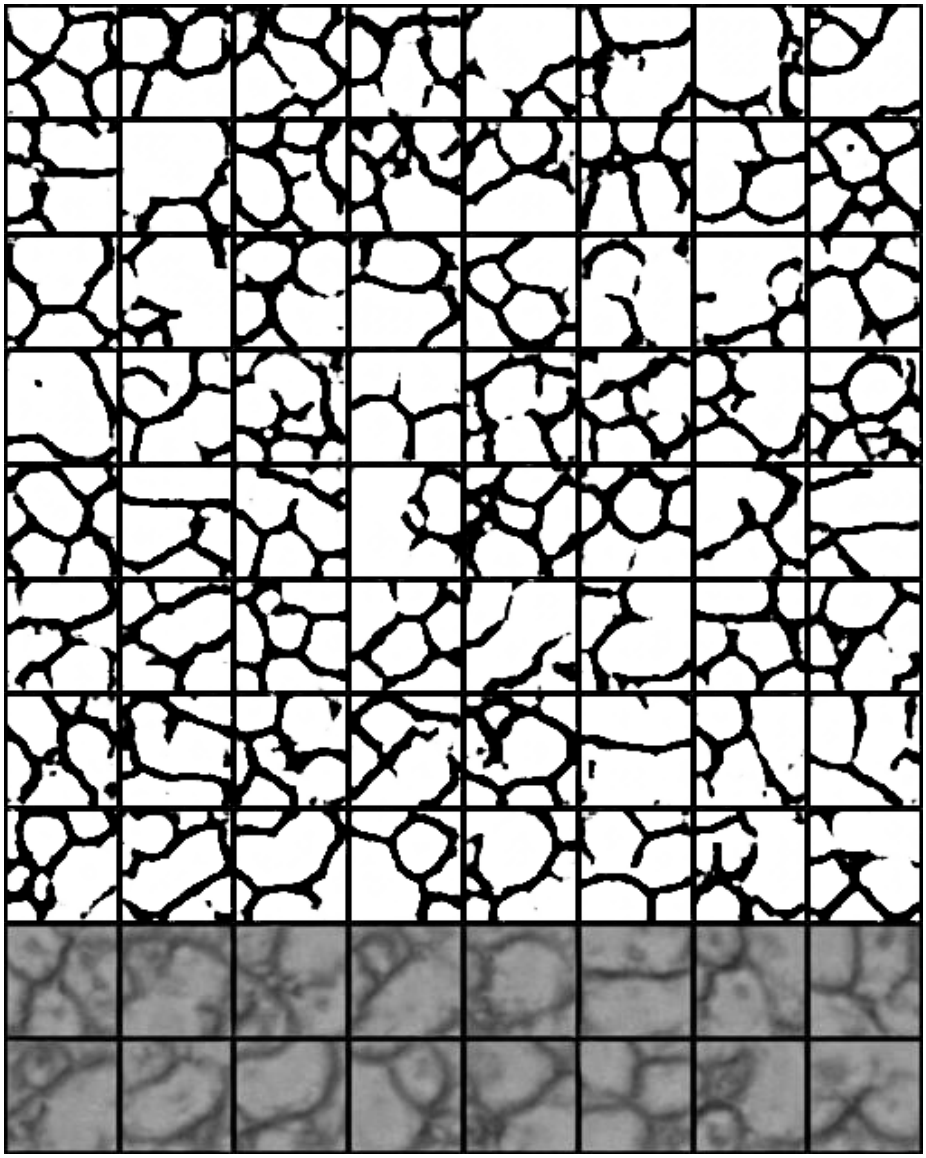

Fig. 8. Fake masks and textures of **CREMI** from WGAN-SN. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

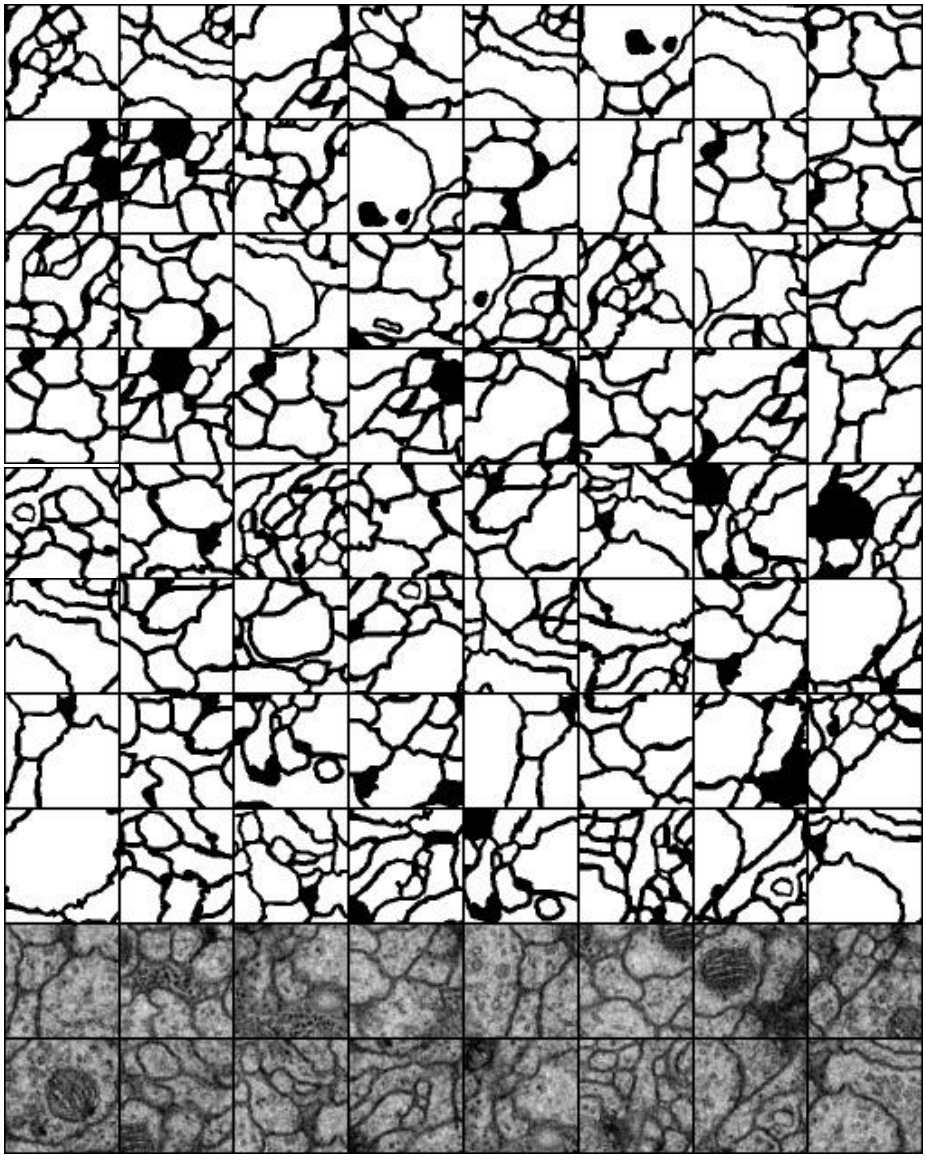

Fig. 9. Real masks and textures of **ISBI12** for reference. Textures correspond to the last two rows of masks. In the real textures, boundaries are clear and loops are complete.

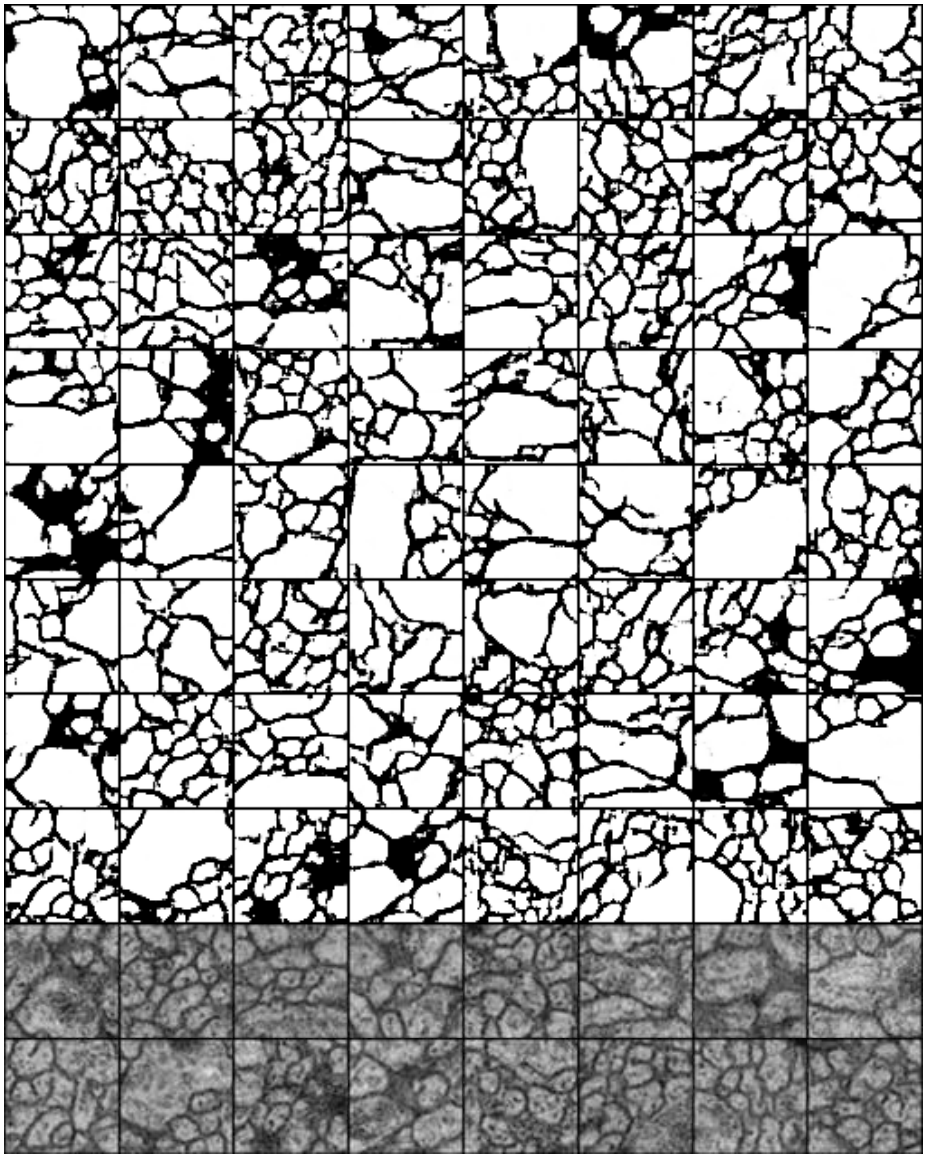

Fig. 10. Fake masks and textures of **ISBI12** from TopoGAN. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

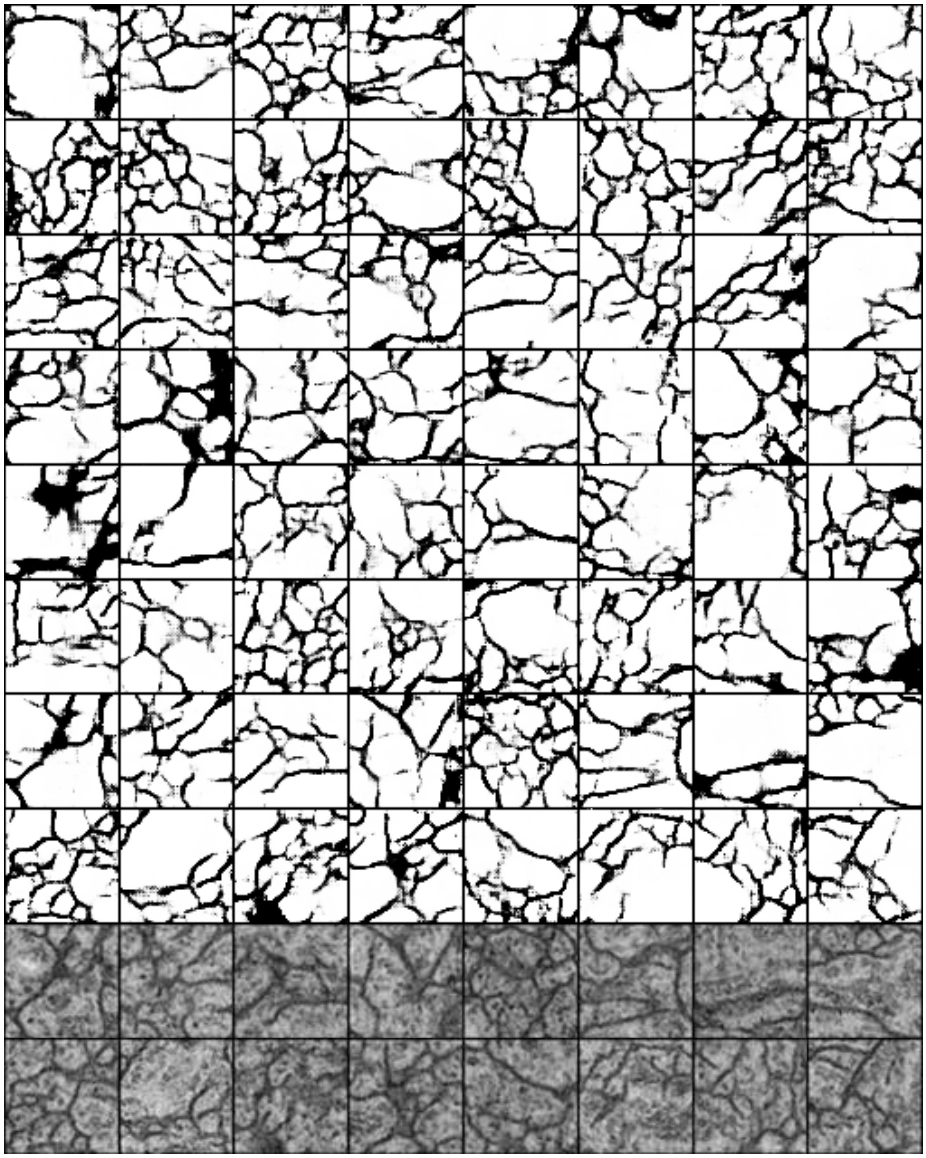

Fig. 11. Fake masks and textures of **ISBI12** from WGAN-GP. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

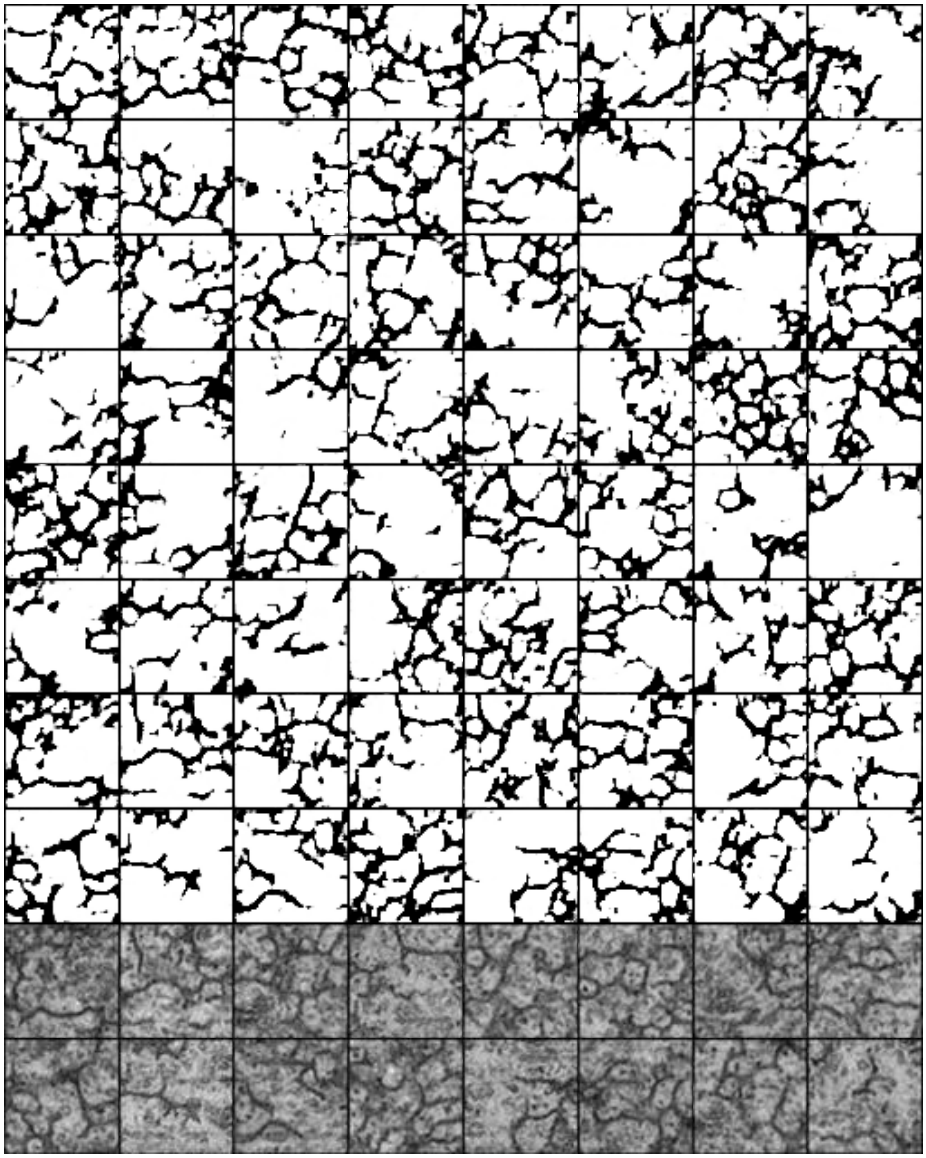

Fig. 12. Fake masks and textures of **ISBI12** from WGAN-SN. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

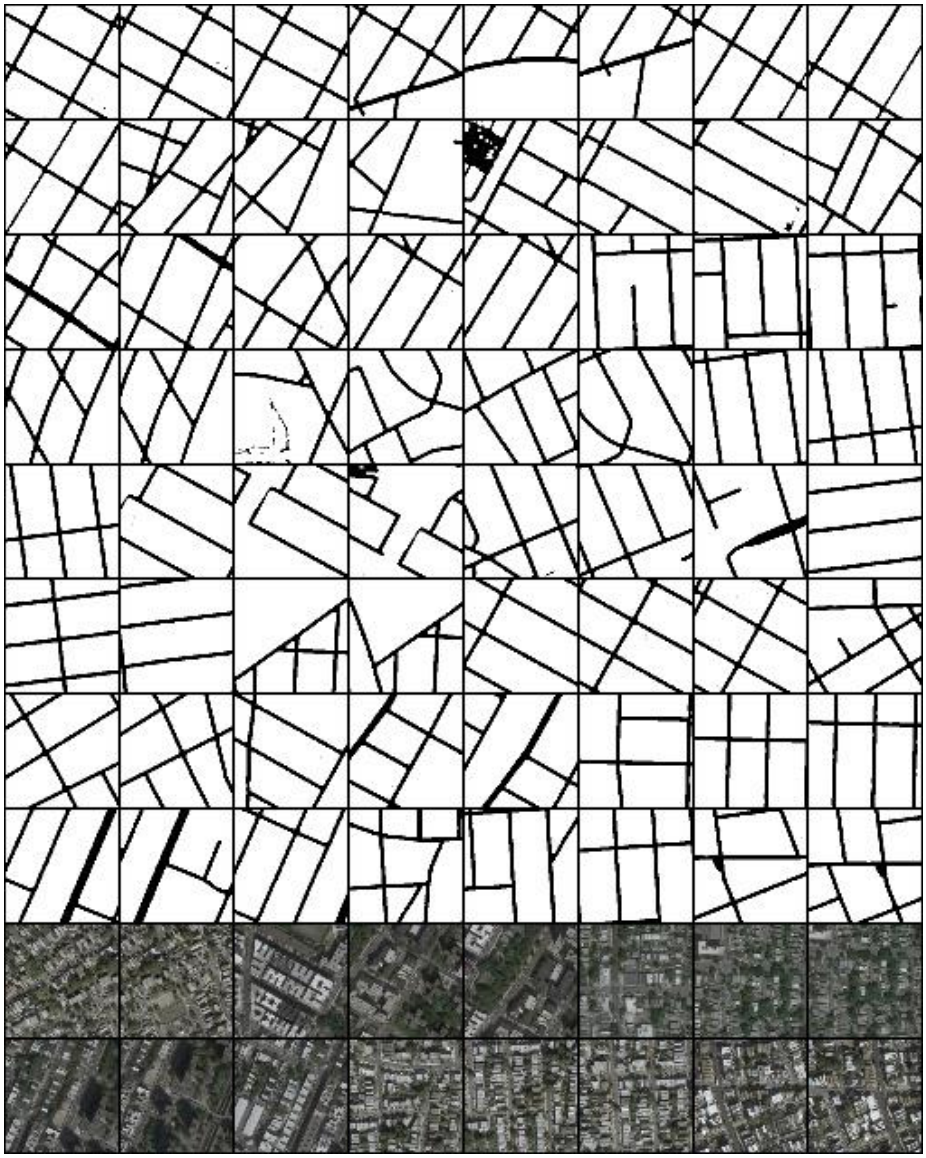

Fig. 13. Real masks and textures of **Maps** for reference. Textures correspond to the last two rows of masks.

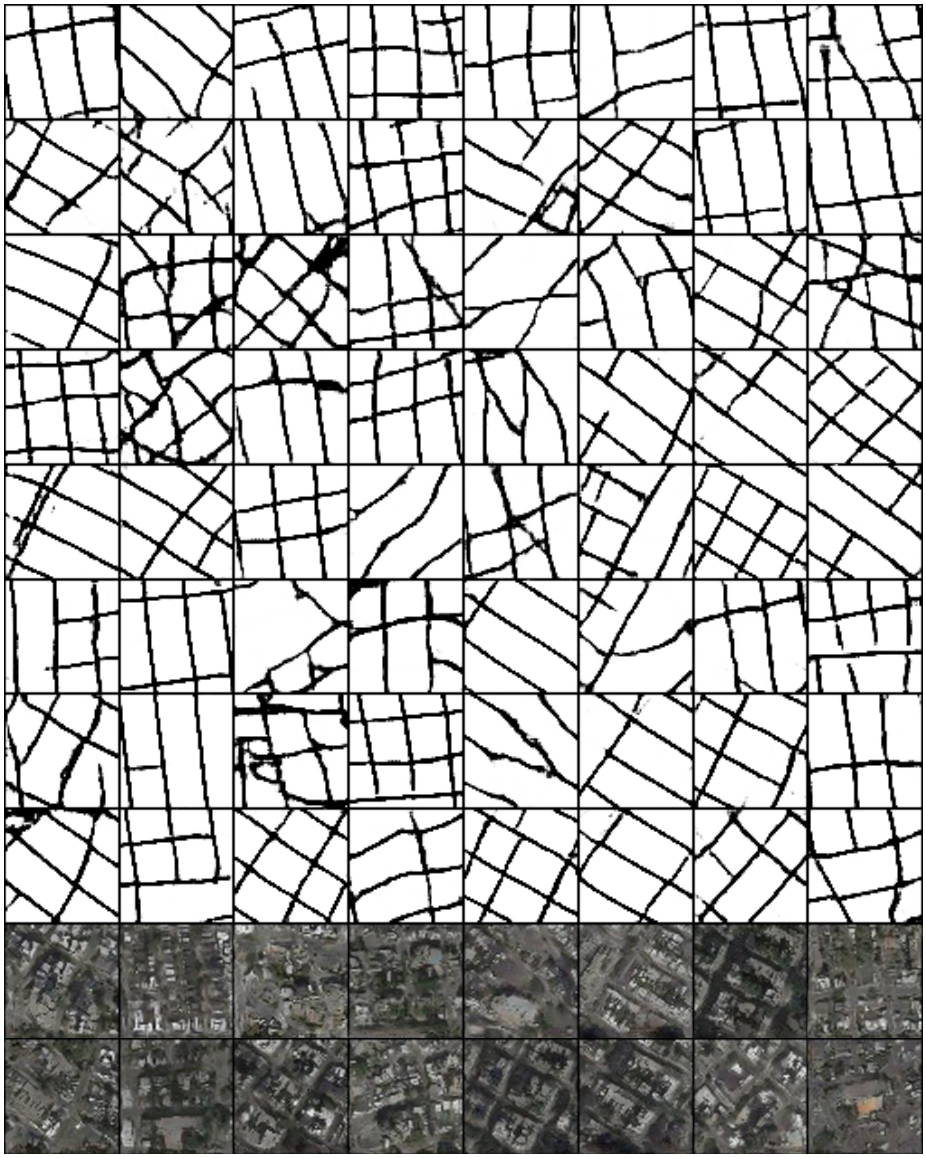

Fig. 14. Fake masks and textures of **Maps** from TopoGAN. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

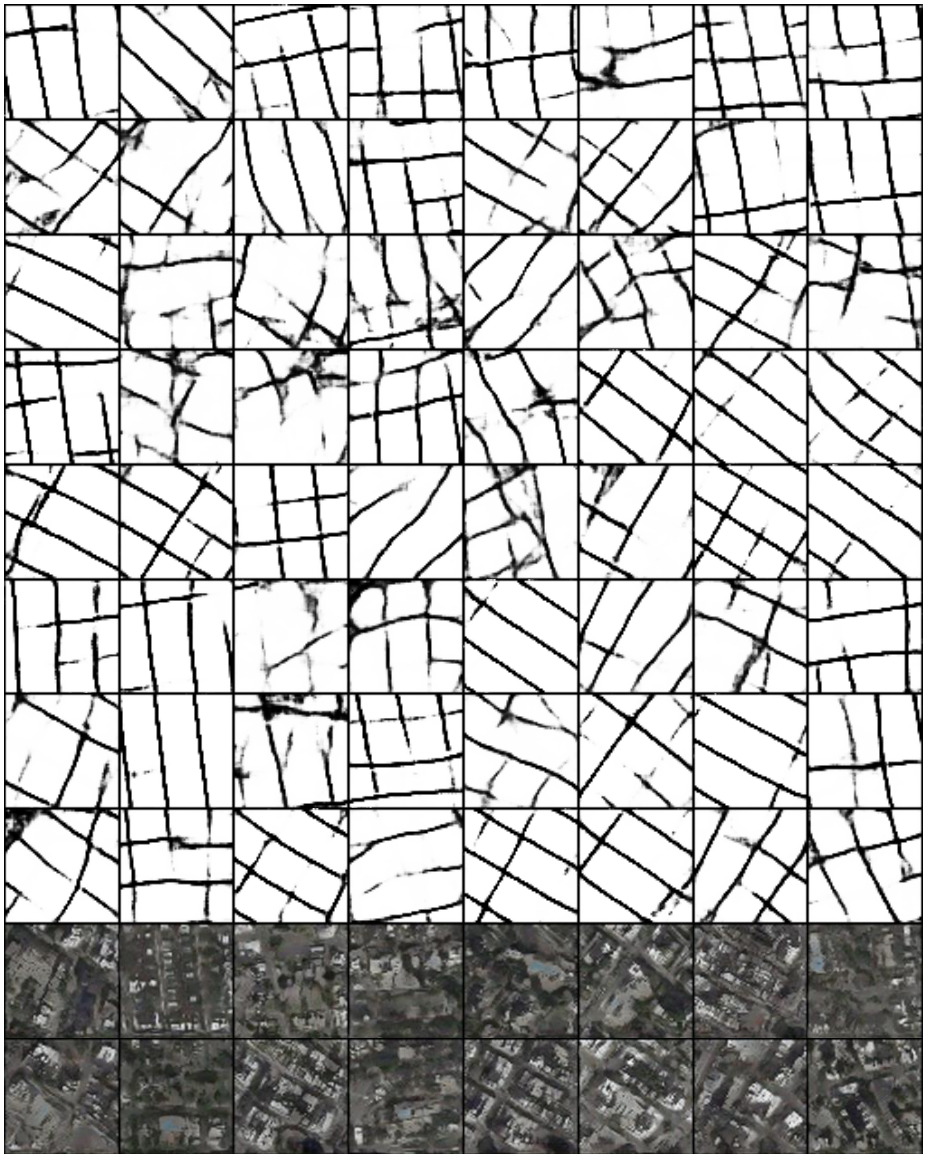

Fig. 15. Fake masks and textures of **Maps** from WGAN-GP. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

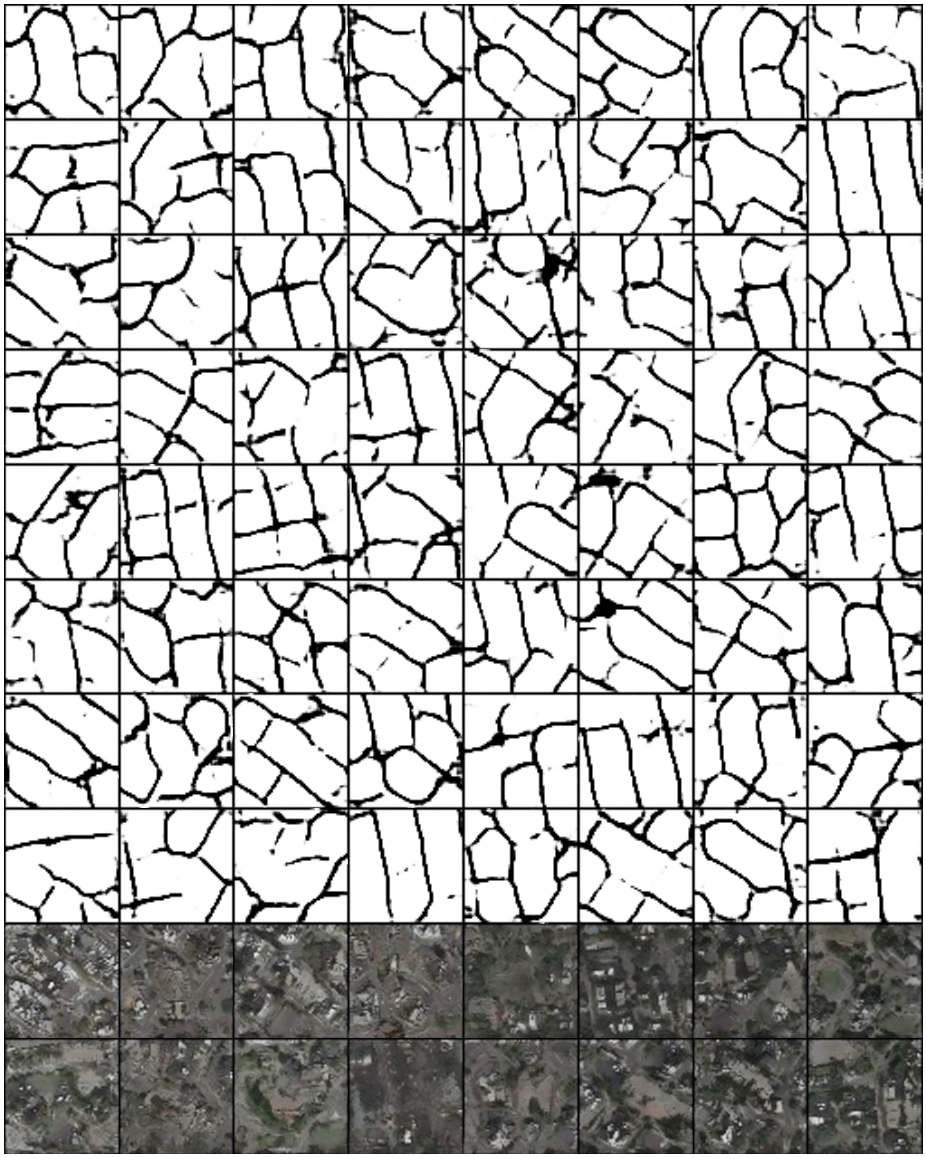

Fig. 16. Fake masks and textures of **Maps** from WGAN-SN. Textures correspond to the last two rows of masks. The same set of noise is used to generate results for TopoGAN, WGAN-GP, and WGAN-SN.

Table 3. Pixel accuracy, dice score, and adapted rand index (ARI) of segmentation networks on real test data. For each dataset, we train a total of 21 segmentation networks with real training data, synthetic pairs from TopoGAN and two baselines, and synthetic pairs augmented with real data. We report mean and standard deviation from a 3-fold cross validation.

		Accuracy	Dice score	ARI
CREMI	Real data	0.857±0.007	0.896 ±0.004	0.514±0.019
	WGAN-GP	0.757±0.020	0.820±0.018	0.269±0.038
	WGAN-SN	0.766±0.022	0.827±0.019	0.286±0.044
	TopoGAN	0.791±0.010	0.851±0.011	0.330±0.017
	GP+real data	0.856±0.010	0.897±0.008	0.512±0.029
	SN+real data	0.859±0.009	0.900±0.004	0.517±0.029
	Topo+real data	**0.864**±0.008	**0.902**±0.006	**0.532**±0.024
ISBI12	Real data	0.900±0.015	0.932±0.011	0.625 ±0.039
	WGAN-GP	0.893±0.005	0.927±0.005	0.597±0.011
	WGAN-SN	0.848±0.010	0.902±0.008	0.435±0.027
	TopoGAN	0.903±0.006	0.933±0.006	0.633±0.014
	GP+real data	0.918±0.007	0.943±0.007	0.688±0.018
	SN+real data	0.913±0.008	0.905±0.054	0.673±0.019
	Topo+real data	**0.921**±0.009	**0.944**±0.008	**0.695**±0.024
Retina	Real data	0.810±0.013	0.883±0.010	0.272±0.012
	WGAN-GP	0.827±0.017	0.891±0.012	0.346±0.055
	TopoGAN	0.830±0.017	0.892±0.013	0.357±0.049
	GP+real data	0.842±0.017	0.899±0.010	0.409±0.065
	Topo+real data	**0.852**±0.022	**0.906**±0.014	**0.427**±0.084

References

1. Carriere, M., Cuturi, M., Oudot, S.: Sliced wasserstein kernel for persistence diagrams. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 664–673. JMLR. org (2017)
2. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Stability of persistence diagrams. Discrete & Computational Geometry **37**(1), 103–120 (2007)
3. Cohen-Steiner, D., Edelsbrunner, H., Harer, J., Mileyko, Y.: Lipschitz functions have l p-stable persistence. Foundations of computational mathematics **10**(2), 127–139 (2010)
4. Edelsbrunner, H., Harer, J.: Computational topology: an introduction. American Mathematical Soc. (2010)
5. Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., Smola, A.: A kernel two-sample test. Journal of Machine Learning Research **13**, 723–773 (Mar 2012)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
7. Kerber, M., Morozov, D., Nigmetov, A.: Geometry helps to compare persistence diagrams. Journal of Experimental Algorithmics (JEA) **22**, 1–4 (2017)
8. Kusano, G., Hiraoka, Y., Fukumizu, K.: Persistence weighted gaussian kernel for topological data analysis. In: International Conference on Machine Learning. pp. 2004–2013 (2016)
9. Mileyko, Y., Mukherjee, S., Harer, J.: Probability measures on the space of persistence diagrams. Inverse Problems **27**(12), 124007 (2011)
10. Munkres, J.R.: Elements of algebraic topology, addinson (1984)
11. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks (2015), http://arxiv.org/abs/1511.06434, cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016
12. Reininghaus, J., Huber, S., Bauer, U., Kwitt, R.: A stable multi-scale kernel for topological machine learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4741–4748 (2015)
13. Turner, K., Mileyko, Y., Mukherjee, S., Harer, J.: Fréchet means for distributions of persistence diagrams. Discrete & Computational Geometry **52**(1), 44–70 (2014)