IRREGULAR CONVOLUTIONAL AUTO-ENCODER ON POINT CLOUDS

Zhang Yuhui  
Tokyo Institute of Technology  
zhang.y.av@m.titech.ac.jp

Greg Gutmann  
Tokyo Institute of Technology  
gutmann.g.aa@m.titech.ac.jp

Konagaya Akihiko  
Tokyo Institute of Technology  
konagaya.a.aa@m.titech.ac.jp

ABSTRACT

We proposed a novel graph convolutional neural network that could construct a coarse, sparse latent point cloud from a dense, raw point cloud. With a novel non-isotropic convolution operation defined on irregular geometries, the model then can reconstruct the original point cloud from this latent cloud with fine details. Furthermore, we proposed that it is even possible to perform particle simulation using the latent cloud encoded from some simulated particle cloud (e.g. fluids), to accelerate the particle simulation process. Our model has been tested on ShapeNetCore dataset for Auto-Encoding with a limited latent dimension and tested on a synthesis dataset for fluids simulation. We also compare the model with other state-of-the-art models, and several visualizations were done to intuitively understand the model.

1 Introduction

A huge amount of real-life objects could be efficiently represented as particles, or point clouds. General 3D Objects, radar scans, fluids, or even atoms and molecules. Point clouds are an efficient and clean representation of real-world objects, as they directly define a distribution using points as samples. There are also other methods to represent 3D objects, two of the most common usage is polygon meshes and volumetric data arrays. Polygon (typically triangular) meshes are restricted to preserve a topology between vertices, and usually they can only represent the surface of a given object. Volumetric data arrays usually have a regular lattice structure by definition, which is not accurate since objects typically does not follow a regular lattice structure and some noise was introduced when quantizing them to those regular lattices.

Point clouds, however, are the most simple yet accurate way to represent objects among those three representations. Firstly, it is simple since it does not need to have topological relationships between points. Secondly, it is accurate since it does not need to quantize original objects into some grid bins like volumetric representations. Last but not the least, point clouds are very flexible as they could easily represent anything, and they are the easiest to obtain, usually could be directly obtained from data sensors (e.g. LiDAR, satellites) without any pre-processing. Thus, it is very promising and valuable to directly operate on point clouds. However, those point clouds often come with a massive amount of data, introduced a big challenge while processing such particle clouds. The huge amount of data is also a problem for other representation methods as well.

The goal of representation learning is to find powerful, rich and meaningful representations of some complex high-dimensional objects. As inspired by human’s ability of information summarization without a teacher, learning those representations are often preferred in an unsupervised manner. Auto-encoding tasks then become a fundamental task in unsupervised representation learning literature. The origin of learning algorithms with auto-encoders was widely-agreed as the work done by [1]. Auto-encoding typically trains a model by first encoding input data to a smaller latent space (compared to original input space), then reconstructs it as fine as possible. The latent code is often expected as a well-behaved representation, as lots of experiments show that interpolating them provides meaningful results.

Recently, deep auto-encoders have been proven to have a strong capability in finding latent representations of data in an unsupervised manner, such as images [2, 3, 4], 3D volumes [5, 6], polygon meshes [7, 8], videos [9, 10], texts [11], as well as point clouds [12, 13]. In this work, we aim to learn light-weight and information-rich representations of point
Irregular Convolutional Auto-Encoder on Point Clouds

clouds by a graph-based deep auto-encoder, which could benefit in many different ways. For example, we could achieve a better understanding of different shapes by those deep representations, or we could process those deep representations directly rather than the raw point cloud to save computational budget. In the latter case, particularly in this work, we are interested in performing latent-space physics simulation with those deep representations. By doing so, we could achieve similar results compared with raw simulation on particles, with a huge speed boost as our representations are light-weight. Typically, raw particle clouds contain thousands to millions of particles, while our representation only contains tens to hundreds of information-rich particles. In this paper, we will propose a novel point cloud auto-encoder architecture, based on a novel convolution operation defined over irregular point clusters.

1.1 Our contributions

We summarize our contributions in this paper as follows:

- We proposed a novel convolution layer for point clouds. Our convolution layer works without any discretization so that it could directly operate on accurate point clouds. Furthermore, we pointed out and proposed a simple solution to a possible problem that only occurs when the input data distribution is irregular.
- We combined our novel convolution layer with farthest-sampling as pooling methods, which gave us a graph-based deep convolutional neural network that learns representations for point clouds. We extended our encoder by a local-oriented, conditional generator as a decoder, and trained it in an auto-encoding manner. Our novel deep auto-encoder achieved state-of-the-art performance on such tasks. We also provided a detailed analysis of it.
- Besides, we trained a simple yet effective learn-able dynamics estimator to learn the particle dynamics under the encoded latent space. We used fluids (which are based on Navier-Stokes equations) as our dataset for training and evaluation. Our model successfully learned the dynamics under the latent space, shows that our auto-encoder has the ability to produce senseful representations.

1.2 Notations

We now state some notations that will be used throughout this paper. We denote \( p_i \) for the position of \( i \)-th point inside the point cloud, \( N(i) \) as the set of indices of points inside the neighborhood of \( i \)-th point, \( f(\ell) \) as the output feature map of layer \( \ell \) (defined as a vector-valued function over \( \mathbb{R}^3 \)), \( f(\ell)_i \) as the sampled value of the feature map function on the position of \( i \)-th point (which is more widely used in this work), \( P(X) \) the underlying point distribution of the input point cloud, with an intractable probability density function (PDF) \( \mu(x) \) and we sometimes also denote it as \( \|f^{(0)}\| \) since it is the desired input to our model.

2 Related works

Auto-encoders and representation learning  Variational auto-encoders (VAEs) [14] provided a powerful extension to auto-encoders by constructing a variational lower bound to maximize the likelihood, as well as the ability to directly sample from learned data distributions. [4] used a stacked convolutional auto-encoder on images, while [12, 13, 15, 8, 7] operates on 3D shapes, and [16] operates on topological graphs. [2, 3] achieved state-of-the-art results in image synthesis by an auto-regressive variational auto-encoder, showing the power of an auto-encoder and an auto-encoding learning task. [17] trained an auto-encoder to predict how the rotation of a 3D object looks in a 2D image, resulting in useful representations. [18] found that the moments (i.e. mean and variance) of the activation of intermediate layers (i.e. intermediate feature maps) are often corresponded to the style of an image, while the “relative position” of features contains semantical contents, and have achieved arbitrary style transfer between images by simple transformations.

Generative models  Meanwhile, generative models are also attractive for their abilities to estimate generate samples from very complex unknown empirical distributions, e.g. image and video synthesis. As we treat point clouds as point distributions in a 3-D space, we are particularly interested in generating such distributions, conditioned on learned latent representations.

Generative adversarial networks [19] provide a powerful framework with mind-blowing results in diverse generative tasks, such as image synthesis [20, 21, 22, 23, 24, 25, 26], and image-to-image translation [25, 27, 28]. To our interest, such tasks often focused on a conditional distribution where the conditioning methods for the generator could vary across different works [20, 24, 18].

*For consistency, we still write \( f(0) \) as vector form even it is real-valued and the norm operator \( \|\cdot\| \) is unnecessary.
On the other hand, the discriminator acts like a statistical distance between fake generated distribution and real distribution \cite{29}, in particular, they could be KL divergence estimators in vanilla GAN configuration \cite{19}, Wasserstein distance \cite{30,22}, Total Variation \cite{31}, Maximum Mean Discrepancy \cite{32,33,34,35}, f-divergence \cite{36} and so on.

As point clouds are low-dimensional data distributions so that distance estimation could be relatively easy and accurate, a recent work \cite{37} adds a non-parametric upper-bound approximation for Wasserstein distance \cite{38} on top of Wasserstein GANs \cite{30,22} for point cloud generation. Sinkhorn distance \cite{39} are also good estimators and solvers for Wasserstein distance and optimal transport, while it introduced an entropical constraint to the transport plan matrix in the original optimal transport problem, enabled it to use a simple iterative method to solve this constrained problem as an accurate estimator to the original problem.

**Graph convolutional networks (GCNs)** Graphs are very powerful mathematical representations for relationships. Especially, in point clouds, we could define such relationships by the spatial relationships of points, namely a k-Nearest Neighbor Graph (k-NNG). Thus we could focus on some local neighbors of a point to extract features hierarchically. Many machine learning methods were developed for such graph representations.

Spectral methods were applied as the convolution in the spatial domain represented as multiplication in the spectral (Fourier) domain. This could be well-approximated via Chebyshev polynomials up to a given order as suggested in \cite{40}, and was used to build GCNs in \cite{41}. A first-order approximation was given by \cite{42}, hence the convolution was defined over the 1-neighbors (i.e. 1 step away) of a given node in the graph. Those convolution operations are isotropic (i.e. weights are the same for different neighborhoods with the same order), as they are all equals to $W$ in equation 1. In contrast, conventional convolution kernels often have different weights for different input nodes. To tackle this problem, attention mechanism has been used as it could let the model focus on different parts in the output for different neighbors \cite{43}. The layer-wise convolution operations are typically defined as \cite{42}:

$$X_{\ell+1} = \sigma \left( \bar{D}^{-\frac{1}{2}} \bar{A} \bar{D}^{-\frac{1}{2}} X_{\ell} W_{\ell} \right) \tag{1}$$

where $W_{\ell}$ is the weight matrix, $\bar{A} = A + I$ is the graph adjacency matrix $A$ with self-loops added, $\bar{D}$ is the diagonal node degree matrix, used to normalize $\bar{A}$. Such spectral methods often require the graph topology to be fixed along training, therefore they do not fit in our conditions.

Other methods follow a message-passing scheme between vertices \cite{44,45,46,47}. In those works, convolution is often defined as a combination of functions over graph edges. For example, in Interaction Networks \cite{44}, layer-wise convolution operations for a single vertex are defined as equation 13 in section 4.1.

Besides convolution, pooling operations are also important since it could reduce computational cost and let the convolution operation operates on a larger receptive field. Non-parametric methods, such as binary tree indexing \cite{41} and graph clustering \cite{48,49}, were used to coarsen the graph, while some works used a learnable pooling operation \cite{50,51} for graph coarsening. As suggested in \cite{52,53}, farthest sampling could be very efficient in pooling for graphs with spatial information (e.g. k-NNG of point clouds) as they could coarsen the graph uniformly in space.

**Geometric convolution operations** As 3D geometries (e.g. point clouds, polygon meshes) have spatial features besides topological connections, many works are done in this field focused on spatial-sensitive convolution operations. Lots of approach aims to use regular 3d convolution kernels as they firstly pre-process point cloud or geometric data into voxels. Per-point networks followed by a collector network is also a common approach \cite{54,52,55}. Besides, \cite{56} developed a similar approach that first map irregular point cloud into a regular lattice, and map them back to point cloud after regular convolution. \cite{57} used spherical (angular and radial) bins to discretize the original input point cloud to regular convolution operations with different weight matrices defined on different bins. \cite{58} constructs a convolution method based on spectral methods. \cite{59} used a permutation matrix to do the convolution on a local neighborhood while keeping the permutation invariance and also enables the network to efficiently collect spatial features. \cite{60} presents a novel method that parameterizes each convolution kernel to small-sized point clouds (e.g. 4 points), and their convolution operation works as the computation of kernel correlation between the local point cloud and convolution kernel. However, none of those methods have considered the density of input point cloud distribution, which could be crucial for further development and deep models.

**Deep learning with point clouds and 3D geometries** Traditional point cloud processing algorithms were often based on feature histograms. Recently, many learning-based methods were studied \cite{54,52,12,61,55}. Several approaches used per-point feature extractor multi-layer perceptrons (MLPs) along with a global combination operation \cite{54,55}, which were often chosen as max-pooling or summation. PointNet++ \cite{52} extended \cite{54} by hierarchically stacking PointNet \cite{54} with farthest sampling as pooling operation.
Figure 1: Over-all architecture of our model. As we described in section 3, our encoder was built with hierarchical blocks, where the size of each block $N \times (C + 3)$ are as $N$ points with $C$ channels and 3-d positions. Our decoder was composed of local conditional generators, as we highlighted specific local patches in the figure with different colors. All the visualizations about point clouds in this work are done with CloudCompare [71].

[62] proposed a general graph network framework for point cloud machine learning methods rather than some novel convolution operations or point out important problems as we did in our paper. [63] mainly focused on pooling layers occurred in point cloud processing networks, as they tried to maintain the overall structure of the original point cloud while pooling. [64] defined super-points in the similar manner of super-pixels in image processing field, using some simple geometrical structures. [65] was an extension of voxel-based methods, and they designed their method to preserve as much information as possible during discretization from raw points to voxels. In contrast, our purpose method was different from above approaches, as our model operates directly on raw points without discretization, and we defined a novel convolution operation and we pointed out an important problem in irregular data processing along with two different approaches purposed by us in order to solve that problem.

Meanwhile, there are also many works aims for generating and auto-encoding point clouds and 3D shapes [66, 12, 13, 67, 37, 68, 69, 8, 7, 15]. [66, 13] used a deterministic approach to generate point clouds as a whole object, while [12, 37, 68] used conditional generators to learn a transformation from a trivial distribution to the desired point cloud. While generating point clouds, statistical distances are widely used as the objective function for generative models. Chamfer pseudo-distance is widely used [66, 12, 13]. However [13] found that Chamfer pseudo-distance is blind to point densities and suggests Wasserstein distance estimators. Those estimators were also widely used [66, 13, 37] in this field.

Physics simulation with deep learning Recently there are some works aimed to perform physics and quantum motion dynamics with learning methods [44, 45, 49, 46], where fluids simulation are commonly used as a showcase for those works. Some of them target on 3-D spaces directly, aimed for a differentiable simulator [44, 45, 49], while others targets on an encoded latent space representation [5, 6, 70] for fast simulations. Learnable methods are nearly the only choice for latent dynamics since they are typically intractable.

Almost all of latent space dynamics estimators targets to a Eulerian point of view [6, 70, 5], where they use vector and scalar fields as a representation instead of particles or points. To the best of our knowledge, few prior works were done as a latent dynamics estimator for massive amount of particles under a Lagrangian point of view, possibly caused by the obstacles while dealing with irregular points data. Lagrangian methods could have arbitrary large simulation space, and more efficient as the computational cost is not sensitive to the size of simulation space, compared to Eulerian methods.

3 The proposed point cloud Auto-Encoder

Our model is built up by k-Nearest Neighbor Graphs, followed by novel convolutional layers combined with farthest-sampling pooling layers [52] which are connected iteratively with residual connections, ends up with a generator conditioned on latent features by Adaptive Instance Normalization (AdaIN) [18] as the decoder, shown by figure 1. Section 3.1 and section 3.5 states our convolution operation, while section 3.2 introduces the encoder architecture and 3.3 discussed about the decoder. Finally, 3.4 shows different choices for the loss function and the training process.
3.1 Convolution

Typically, convolution over a regular lattice (e.g. word embeddings, pixels and voxels) are defined as (in 1-D):

\[ f_i^{(\ell+1)} = \sum_{j=-k}^{k} w_j f_{i+j} \] (2)

This definition and parameterization are highly relied on a regular grid layout of data points, which does not hold in point cloud processing scenarios. Recall the definition of convolution:

\[ (w * f)(x) = \int_{-\infty}^{\infty} w(\tau) f(x-\tau)d\tau \] (3)

It is clear that equation 2 is a discrete and truncated version of equation 3, as the weights of conventional convolution operations are discrete functions of local positions. Such parameterization was only possible with a regular data lattice. Consider point distribution as the input, we could then rewrite equation 3 for point distributions as

\[ f_i^{(\ell+1)} = \int_{\mathcal{N}(i)} W^{(\ell)}(\tau)^T f^{(\ell)}(p_i + \tau)d\tau \approx \sum_{j \in \mathcal{N}(i)} W_{\theta}(f_j(p_j) - f_j(p_i))^T f^{(\ell)}(p_j) \] (4)

Where \( W_{\theta}(\cdot) \in \mathbb{R}^3 \rightarrow \mathbb{R}^{d \times c^{\ell+1}} \) is a parametric function as the convolutional kernel weights with learnable parameters \( \theta \), and \( \mathcal{N}(i) \) is the 1-neighborhood of point \( i \) in the k-NNG of input point cloud.

Density elimination The term \( \mu(p_j) \) in equation 4 is introduced to eliminate the influence of input point density. If we remove the eliminating term, the convolution operation will then produce \( f_i^{(\ell+1)}(x) = \int W(\tau)^T f^{(\ell)}(x+\tau)\mu(x+\tau)d\tau \), as it actually computes the expected value \( \mathbb{E}[W(\tau)^T f^{(\ell)}(x+\tau)] \) by definition and the domain of integration has changed. So that the term \( \mu(x) \) will keep being multiplied again and again through layers unless we eliminate it. This will not be a problem in conventional convolutional layers since \( \mu(x) \) represents a uniform distribution. Note that this treatment is still far from the best solution as it still limits the feature maps to have the same "support" as input point distribution \( P(X) \), and the estimation of \( \mu(x) \) from an empirical distribution is nontrivial. We could see the product of this convolution layer \( \ell \), a point cloud with features \( f_i^{(\ell)} \) as a sample from the true feature map \( f^{(\ell)} \) using the sampling points provided in the input point cloud. Since the feature map could also have non-zero values outside the point cloud, where we have omitted, we are still far from the best solution. However, in practice we found that simply removing this term \( \mu(p_j) \) still works fine, therefore we will do so in the following descriptions for simplicity. We will further discuss about this issue in section 3.5.

Parameterization We parameterize the function \( W_{\theta}(\cdot) \) by a multi-layer perceptron (MLP) with 2 hidden layers and leaky ReLU \([72, 73]\), training jointly with other part of our model. Since the space \( \mathbb{R}^{d^{\ell+1}} \) could be too big to compute the MLP and its gradients for training, we firstly simplified the resulting matrix by a simple decomposition

\[ W = A \Phi, \] with \( A \in \mathbb{R}^{c^{\ell+1} \times d^{\ell+1}} \) and \( \Phi \in \mathbb{R}^{d^{\ell+1} \times c^{\ell+1}} \) and \( d \ll c^{\ell+1} \). Following an intuition that \( A \) collects input feature information, producing \( d \)-dimensional proxy-features, and \( \Phi \) collects spatial information, we therefore use \( c^{\ell+1} \) different \( A \)’s for different output channels, and \( \Phi \) are then parameterized by MLP as \( \Phi_{\theta}(\cdot) \in \mathbb{R}^{d^{\ell+1}} \rightarrow \mathbb{R}^{d \times c^{\ell+1}}. \) Hence only \( \Phi \) is spatial-sensitive and \( A \) remains isotropic w.r.t. local positions. Finally we have:

\[ f_i^{(\ell+1)} = b^{(\ell)} + \sum_{j \in \mathcal{N}(i)} 1^{T} \left( \text{reshape} \left( A^{T} f_j^{(\ell)} \right) \otimes \Phi_{\theta}(p_j - p_i) \right) \] (5)

where \( A \in \mathbb{R}^{c^{\ell} \times (d^{\ell+1})} \) is \( c^{\ell+1} \) of \( A \)’s stacked in columns, \( \text{reshape}(\cdot) \) reshapes a vector from \( d^{\ell+1} \times 1 \) to a \( d \times c^{\ell+1} \) matrix, \( \otimes \) denote the element-wise multiplication, \( 1 \in \mathbb{R}^d \) denotes an one-valued vector for summation. The trainable parameters within a layer \( \ell \) are \( b^{(\ell)} \in \mathbb{R}^{c^{\ell}} \) as the bias vector, \( c^{\ell} \) of different \( A \)’s as \( A \in \mathbb{R}^{c^{\ell} \times d^{\ell+1}} \) as the channel-wise weights, and \( \theta \) of the spatial weight MLP \( \Phi_{\theta}(\cdot) \in \mathbb{R}^{d^{\ell+1}} \rightarrow \mathbb{R}^{d \times c^{\ell+1}}. \) In most of our experiments, we empirically choose \( d = 2 \) as a good trade-off between computational cost and model performance. We practically found that overall performance could be improved by using batch-renormalization \([74]\) inside the spatial weight MLP \( \Phi_{\theta} \), therefore we used them in our work.

\[ ^1 \text{We omitted} \ell \text{ for simplicity.} \]
Figure 2: Non-uniform distribution of point clouds. Densities were estimated with kernel density estimation, with the same hyper-parameters described in section 3.5. As for the color scale, red stands for dense areas, while blue stands for sparse areas. The brightness of a point is caused by shadows calculated from screen space ambient obscuration.

3.2 Graph-based point cloud encoder

As observed from figure 1, the encoder was composed of $K$ fine-to-coarse blocks connected with the farthest-sampling layers as pooling layers as suggested in [52]. Each block builds a k-Nearest Neighbor Graph (k-NNG) of the input point cloud in the very beginning. Then the proposed convolutional layers were used in each block iteratively in order to effectively extract local features around the convolution centroid. After each convolution, we apply a leaky ReLU as a non-linear activation function. We will also apply any normalization layers right before the non-linear activation function, as we will do so in experiments.

We also use additive residual connections [75] in our encoders to avoid vanishing gradients and accelerates the convergence of our model. Those connections are shown as black arrows in figure 1, while a connection from layer $a$ to layer $b$ is a simple addition:

$$f_{res}^{(b)} = f^{(b)} + f^{(a)}$$

Note that this operation is only possible when layer $a$ and layer $b$ are supported by the same point cloud with same number of points.

Vectors instead of clouds as the latent code

This part is optional. In order to obtain a latent vector, which is more widely used in representation learning literature and prior works, we simply add a global read-out layer over the final output of the encoder (the most coarse graph). This layer is equivalent to a pooled cloud with a single node located in the origin point, and we do convolution from the last coarsened cloud to this 1-node cloud. We also appended 2 extra fully-connected layers after the 1-node convolution to better mix the features.

3.3 Conditional decoder

As we have informative-rich latent point cloud as a latent code representation, it is straightforward to let the final product of decoding (reconstruction) to be a mixture of the decoding results of every latent point, within the local space of that point. In precise, as indicated in figure 1, we consider a uniform mixture model and use a conditional generator to populate points around a given latent point. Inspired by [24], we use Adaptive Instance Normalization (AdaIN) [18] to condition the generator. The AdaIN operation is defined as:

$$\text{AdaIN}(X, y_\mu, y_\sigma) = y_\sigma \frac{X - \mu(X)}{\sigma(X)} + y_\mu$$

Where $X \in \mathbb{R}^{N \times c}$ is the feature matrix with $N$ points and $c$ feature channels, $y_\mu, y_\sigma \in \mathbb{R}^c$ are vectors used for conditioning, $\mu(X), \sigma(X) \in \mathbb{R}^c$ are the mean and standard deviation for each channels of $X$, respectively. This operation first normalizes $X$, and then apply a translation and scaling by $y_\mu, y_\sigma$ as the conditioning step.

As illustrated in figure 1, our decoder was composed by a point-wise MLP (with leaky ReLU) with 5 hidden layers, while taking a uniformly distributed random noise as an input. AdaIN is applied to each decoding layers while $y_\mu, y_\sigma$ are calculated layer-dependently via different networks, transforming corresponding latent point feature to $y_\mu$ and $y_\sigma$. The decoding network now learns a transformation that maps a trivial distribution to the desired distribution, while conditioning on latent features. We observed a significant performance boost by applying AdaIN as the conditioner instead of concatenation, e.g. $y = \Phi_0(\text{concat}(z, l))$ where $z, l$ are random noise vector and latent features respectively.

After we get the local point cloud of each latent point, we simply translate them to the global space (without rotation) and take the union of them as the final reconstruction, which would be used to calculate a distance with the ground-truth point cloud as a learning objective.
Irregular Convolutional Auto-Encoder on Point Clouds

Figure 3: Different distance metric under toy datasets (EMD: exact Earth Mover’s distance, Sinkhorn: Sinkhorn distance, a-EMD: auction based EMD, CpD: Chamfer pseudo-distance). Blue lines indicate the weight in the transport matrix used for over-all distance calculation. Note how Chamfer pseudo-distance failed to capture a consistent distance between distributions by assign lots of points in one distribution to only one point in another distribution. Meanwhile, although Sinkhorn distance seems better than auction-based EMD (a-EMD) in this figure, as it is more similar to exact EMD, it is relatively hard to use such distance in practice due to the numerical obstacles of Sinkhorn iteration when the number of points becomes bigger.

3.4 Training the Auto-Encoder

We used the ADAM [76] optimizer to minimize a statistical-distance-based loss function between reconstructed clouds and the reference cloud. All parameters within our auto-encoder model were trained jointly via back-propagation.

3.4.1 Loss functions

As point clouds are point distributions, statistical distances become a rational choice for the loss function, or the objective function, that we want to minimize between the ground-truth cloud and the reconstructed cloud. As our decoder is a conditional generator, the loss function acts as a non-parametric discriminator here as in GAN literature. Common statistical distance such as the Kullback–Leibler divergence, Total variation distance, etc. are nontrivial to compute for empirical distributions, though there are prior works aims to estimate such distance for empirical distributions [77]. Some of them (e.g. Kullback–Leibler divergence) are known as they do not provide gradients (constant) or even no proper definition on non-overlay part of the supports of two distributions [29]. Therefore we consider following statistical distances as they have been studied for point clouds in prior works [66, 12, 13].

Notations We denote a N dimensional point cloud as a set of points \( S = \{x : x \in \mathbb{R}^N\} \) which follows a N dimensional distribution \( P(X) \) with PDF \( \mu(x) \). We also denote the euclidean distance between point \( x \in S \) and point \( y \in S \) as \( \|x - y\| \), and the reference and candidate point clouds \( S_g \) and \( S_r \) respectively.

Chamfer pseudo-distance It used to be a popular choice as it is easy to compute in parallel and works well in practice. Chamfer pseudo-distance\(^1\) is defined as:

\[
\text{d}_{\text{CD}}(S_r, S_g) = \sum_{x \in S_r} \min_{y \in S_g} \|x - y\| + \sum_{y \in S_g} \min_{x \in S_r} \|x - y\|
\]

However, as [13] points out, Chamfer pseudo-distances are "blind" to the density of two distributions, often resulting a non-uniform reconstruction result. As illustrated in figure 3, some well-placed reconstructed points could "block" the loss function of a point cluster in ground-truth, even if they have a dramatic difference in quantities. Thus, we do not use Chamfer pseudo-distance as our loss function.

Earth Mover’s distance and Optimal Transport Earth Mover’s Distance (EMD) is an ideal choice for point cloud auto-encoders as it assumes a strict “1-to-1 mapping” of mass between source and target distributions. EMD is defined

\(^1\)Since the triangle inequality does not hold for Chamfer pseudo-distance, we used the term "pseudo".
as (a special case of Wasserstein distance):
\[
d_{EM}(\mu_r, \mu_g) = \inf_{\gamma \in \Gamma(\mu_r, \mu_g)} \int_{\mathbb{R}^N \times \mathbb{R}^N} \|x - y\| d\gamma(x, y)
\]
(9)
Where \(\Gamma(\mu_r, \mu_g)\) is the set of all possible joint distributions with marginals \(\mu_r\) and \(\mu_g\).

Or as in its discrete form (with a slight abuse of notation):
\[
d_{EM}(S_r, S_g) = \min_{\gamma} \langle \gamma, M \rangle_F
\]
where
\[
M_{xy} = \|x - y\|, \quad x \in S_r, \quad y \in S_g
\]
\[
s.t. \quad \gamma 1 = r,
\gamma^T 1 = g,
\gamma \geq 0
\]
(10)
Where \(|S_r| = |S_g| = m\), while \(r, g \in \mathbb{R}^m\), \(r_i = g_i = \frac{1}{m}\) is an uniform vector represents sample weights in both point clouds, and \(\langle \cdot, \cdot \rangle_F\) denotes the Frobenius inner product of matrices. \(\gamma\) is a doubly stochastic matrix, represents the transport plan between cloud \(S_r\) and cloud \(S_g\) as it should satisfy the constraints \(\gamma 1 = r, \gamma^T 1 = g\) (i.e. plan an exact match of mass from source to destination). One big drawback of EMD is they need a relatively long time to compute than Chamfer pseudo-distance and hard to parallelize. Fortunately, EMD is well studied by prior works of Optimal Transport (OT) and there exist many fast approximations with controllable error-rate for EMD. We briefly state two of them as follows \([38, 39]\):

The first one \([38]\) was an auction-based approach with good parallelizability, as this approach was widely used in prior works \([66, 37, 13]\) of learning based point cloud processing. We refer to this approach as auction-EMD or a-EMD in the following descriptions. We used an implementation from \([66]\) in our work. As stated in \([66]\), this approach gives highly accurate results with approximation error on the magnitude of 1% with 3-D point clouds and 1024 points each.

The second approach, namely the Sinkhorn distance \([39]\) was developed recently in OT. It is done by using an additional entropy-based regularization term on the original OT problem (equation 10):
\[
\min_{\gamma} \langle \gamma, M \rangle_F - \lambda h(\gamma)
\]
\[
s.t. \gamma 1 = r, \quad \gamma^T 1 = g, \quad \gamma \geq 0
\]
(11)
Where \(h(\gamma) = -\sum_{i,j} \gamma_{ij} \log \gamma_{ij}\) is the entropy of \(\gamma\), as \(\gamma\) is a doubly stochastic matrix. According to \([39]\), adding an entropy regularization term enforces a simple structure on the optimal regularized transport matrix \(\gamma\), which could be solved iteratively using Sinkhorn iteration. We refer the readers to \([39]\) for further details. This approach is also easy for parallelization with great accuracy. Sinkhorn distance requires a great numerical capability to compute accurately, which could be time-consuming and hard for implementation. We have implemented it using TensorFlow \([78]\), with different numerical capabilities, namely FP32 for single-precision floating-point numbers float32 and FP64 for double-precision. We observed a performance boost while using precise FP64-based calculation of Sinkhorn distance as the loss function compared to a-EMD.

### 3.5 Extension to the point cloud convolution kernel

As we have shown in section 3.1, feature maps in each layer were influenced by the distribution of point clouds. A simple approach is already shown in section 3.1, as we refer to density elimination. We use kernel density estimation methods (truncated to the 1-neighborhood within the k-NNG) on each point to estimate the density \(\mu(x)\). Radial basis function (RBF) kernels \(K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)\) are a common choice for such task, and we practically choose kernel size \(\sigma = \mathbb{E}_{x,y \in S} [\|x-y\|]\) as given point cloud \(S\) and \(x, y\) are distinct neighbors in the k-NNG. We apply the density elimination by divide the feature maps (as point cloud with features) \(f_i^{(\ell)}\) in each intermediate layer \(\ell \geq 1\) by the estimated input point cloud density.

Being inspired by the widely used max-pooling operation for combining features in point clouds \([54, 55, 12, 52, 13]\), we also construct another well-working alternative approach, by substituting summation with maximum in equation 5:
\[
f_i^{(\ell+1)} = \mathbf{b}^{(\ell)} + \max_{j \in \mathcal{N}(i)} 1^T \left( \text{reshape} \left( A^{T} f_j^{(\ell)} \right) \otimes \Phi_{\theta} (\langle p_j - p_i \rangle) \right)
\]
(12)
Where the maximum is applied channel-wise. We call this variation a "max-evidence" process, as we intuitively state as follows. Firstly, each point contributes to some evidence, where the confidence of that evidence becomes higher as...
We have implemented our models and all extensions with python library TensorFlow \[78\]. We used 2 different datasets in our experiments. For the point cloud auto-encoder, we used the ShapeNetCore dataset \[79\] as it is a large-scale dataset contains various 3-D models with annotations, and it is available in the public domain. ShapeNetCore \[79\] contains 51,300 unique 3-D objects under 55 different categories, such as motorbike, table, watercraft, sofa, provided

the point stays in better positions. Then a max-pooling over all neighbors was applied to collect all evidence without letting evidence from different points overlap, that is, different points are not allowed to contribute to the same evidence multiple times. Finally, by a linear transformation, we collect all the evidence to calculate the features, as features could be supported by multiple pieces of evidence.

The above 2 processes significantly benefits the model for auto-encoding tasks in our experiments.

However, as we stated in section 3.1, we are still limited by the constrictions of the "support" of feature maps, where they are forced to have the same "support" with the input point cloud. Perhaps, a resampling step could solve this problem, as in \[56\] which resamples the feature maps to a regular hexagonal lattice. However a regular lattice may not be the optimal solution, as it can neither extend to higher-dimensional cases, e.g. "point cloud" where each point is actually an \( H \times W \times 3 \)-D image instead of 3-D points, nor accurate since resampling everything to regular grids could lose more information than an optimal resampling method. As solving this problem will be a far step beyond our current work, we leave it as further research directions.

4 Latent space simulation: A proof of concept

4.1 Interaction Networks

We adopted the structure of Interaction Network (IN) \[44\] as a learnable latent space dynamics estimator. IN is a graph network based on the message-passing principal \[46\]. For a given relation graph with vertices \( o_i \) and edges \( e_i \) between vertices \( u_i \) and \( v_i \), all in time step \( \ell \), we could briefly state the definition of an IN as follows:

\[
\begin{align*}
    e_i^{\ell+1} &= \Phi_E((u_i^\ell; v_i^\ell); e_i^\ell) \\
    o_i^{\ell+1} &= \sum_{e_i \in \mathcal{N}(o_i)} \Phi_O(e_i^{\ell+1})
\end{align*}
\]

(13)

Where \((\cdot; \cdot)\) denotes channel-wise concatenation, \(\Phi_E\) and \(\Phi_O\) are parametric functions which stands for edge and vertex networks, \(\mathcal{N}(o_i)\) denotes the incoming edges for vertex \(i\). In practice, \(\Phi_E\) and \(\Phi_O\) are often implemented as per-vertex or per-edge MLPs. A single layer in IN is a combination of an edge-step and a vertex-step, as shown in equation 13.

4.2 Latent space simulator

Model We implemented a simple IN with 1 layer per time step, with tanh function \(\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}\) as activation functions. By \(n\) layer per time step, we mean that we do \(n\) iterations as in equation 13 within a simulation time step to get the results. Vertex features \(o_i\) are obtained from our point cloud auto-encoder as shown in section 3, concatenated with global positions of latent points. Edges are constructed as a k-NNG at each time step, and edge features \(e_i\) are a concatenation of relative positions and distance between vertices. Note we discarded the edge features every step. After a time step, we use a single linear transformation to obtain the movement of a latent point \(\Delta o \in \mathbb{R}^3\) and apply it to the point as \(p_i = p_i + \Delta p_i\). We also used residual connections inside this IN as \(o_i^{\ell+1} = o_i^{\ell} + \sum_{e_i \in \mathcal{N}(o_i)} \Phi_O(e_i^{\ell+1})\).

Loss function In order to define the loss function, we first compute a transport plan \(\gamma\) as in section 3.4, only using the positions \(\in \mathbb{R}^3\). Then we calculate an L2 cost matrix \(M\) using both position and latent features. Finally our loss function is defined as \(L_{sim} = \langle \gamma, M \rangle_F\).

Training As suggested in \[6\], to improve the consistency of our model during long-term simulations, we train the model iteratively for \(T = 50\) time steps and calculate the loss in each time step, where the only input is the initial state of the simulated system. We combined losses in each time step by a weighted average with exponential weights \(\alpha^{\Delta t}\), where we choose \(\alpha = 0.95\) in our experiments. We trained the simulator separately with our auto-encoder for an acceptable computation budget.

5 Experimental Results

5.1 Experiment set-up

We have implemented our models and all extensions with python library TensorFlow \[78\]. We used 2 different datasets in our experiments. For the point cloud auto-encoder, we used the ShapeNetCore dataset \[79\] as it is a large-scale dataset contains various 3-D models with annotations, and it is available in the public domain. ShapeNetCore \[79\] contains 51,300 unique 3-D objects under 55 different categories, such as motorbike, table, watercraft, sofa, provided
Irregular Convolutional Auto-Encoder on Point Clouds

| Approach                  | Latent dimension | M-EMD ($\times 10^{-2}$) |
|---------------------------|------------------|---------------------------|
| Random gaussian           |                  | 17.9160                   |
| Achlioptas et al. (AE-EMD) [13] | 512              | 3.8563                    |
| FoldingNet [12]           | 512              | 3.7528                    |
| Ours (latent vector)      | 512              | 2.9290                    |
| **Ours (latent cloud)**   | 512              | **2.3486**                |
| Optimal (GT)              |                  | 2.1548                    |

Table 1: Main quantitative results. Random gaussian stands for a random gaussian with the same mean and variance compared to the training set. The M-EMD for row "Optimal (GT)" was obtained via different point-cloud samples from the same ground-truth polygon mesh, hence it states a lower-bound of error that we could ever achieve.

as polygon meshes. We sampled points from mesh surfaces by CloudCompare [71]. We use the official training, validation and testing splits of [79], as a result, there are 35,708 samples in the training set for training, 5,158 samples in the validation set for hyper-parameter tuning and 10,261 samples in the testing set for evaluation. We use Sinkhorn distance-based loss as the loss function by default, as if we don’t specify a choice for loss functions in the following descriptions. In particular, we choose $\lambda = 0.002$ as in equation 11, which is only possible under double precision floating numbers (FP64) in our experiments.

We trained our model along with other models from prior works till convergence. In order to compare with the same latent representations as prior works, we also trained a latent vector version using the method we have described in section 3.2. For our cloud-based model, we used the architecture as shown in figure 1 with density-elimination (section 3.5) and layer normalization [80], and trained it for 30 epochs with batch size 8, learning rate $5 \times 10^{-5}$, $\beta_1 0.9$ and $\beta_2 0.999$ with ADAM optimizer. For our vector-based model, we used the same hyper-parameter as the cloud-based model, but with a global read-out convolution as described in section 3.2 with 2 512-d fully-connected layers appended afterward. For FoldingNet [12] we trained with a-EMD instead of Chamfer pseudo-distance with same hyper-parameters, but for 60 epochs with batch size 16 and without weight decay. For AE-EMD [13] we trained it for 80 epochs with batch size 16.

We report the mean of exact EMD (we referred to as M-EMD) between reconstructed cloud and reference cloud over the official test set, calculated via the python package pot [81].

For latent space dynamics estimation as well as point cloud auto-encoders under particle dynamics scenarios, we generated two synthesis dataset, namely "FLUIDS" for fluid dynamics and "LJP" for a simple dynamics between neutral particles. FLUIDS was collected using mantaflow [82] using FLIP [83] method by a modified example scene "breaking dam" with 760 steps total, down-sampled to 380 steps, time step 0.5 unit, grid resolution 16, 5120 particles for simulation and down-sampled to 2048 particles without re-sampling. We modified the example scene configuration so that we could have a diverse set of initial states. LJP was collected using our simulation program, where it only simulates the dynamics of neutral particles under Lennard Jones Potential with a numerical cutoff. We will show more details about our dynamics dataset in the following descriptions. We trained our latent space estimation network as described in section 4.2 with $T = 50, \alpha = 0.95$, 7 epochs on our entire training dataset (397K iterations), batchsize 5 and ADAM optimizer configured as same as above.

5.2 Results and quantitative metrics

We summarized our results as shown in table 1, as well as in figure 4. As stated in table 1, our approach is significantly better than prior works, such as FoldingNet [12] or AE-EMD as described in [13]. The improvement of our model could be easily observed from figure 4. We picked some complex objects in the ShapeNetCore [79] dataset in order to better illustrate the improvements, such as shelf and piano. We also picked some uncommon objects, such as headphone and vase, as well as common objects such as airplane and bench.

Latent vector-based models such as FoldingNet [12] works better in common objects, as they could reconstruct the over-all shape of those objects, even some detailed parts. For example, they could reconstruct the engines under the wings of a airplane, or the supporting legs of a bench. As those objects appear many times among the dataset, the network is more likely to learn the shape of those objects. And for uncommon object categories, like piano and vase, baseline models could only reconstruct a rough and inaccurate over-all shape of those objects. They work even worse for the object headphone, where baseline models failed to reconstruct nearly anything senseful. However, our model works well in either of those objects as shown in figure 4.
Figure 4: Reconstruction results of our models with cloud and vector as latent code. We fairly selected some complex and uncommon objects in the ShapeNetCore [79] testing-set for better comparison. Baseline models are FoldingNet [12] and AE-EMD as in [13]. Note how our latent cloud model could reconstruct complex objects such as shelf, lamp, and piano, and how our latent vector model could still produce good results by preserving a hollow structure for object post, preserving the over-all, sharp structure as well as all 3 legs for object piano, and a clean reconstruction for table, while other baseline models struggled on those objects.

Even stylized objects under common categories are relatively hard to reconstruct, as the row chair shows in figure 4, baseline models failed to reconstruct that stylized chair object, and they reconstructed the special chair to a common chair. In the same row, we could observe that our latent cloud-based model is significantly better than other models, where it reconstructs mostly of the parts in that uncommon object, including those thin armrests. Meanwhile, the object shelf is particularly difficult to reconstruct as it has many hollow holes inside it. But our model still successfully reconstructed those details, and baseline models failed to reconstruct those holes.

Intuitively, we could state that our model, especially the model that used point clouds as latent representations, were strongly benefited by preserving a coarsened shape of the input point cloud in the latent space. Our model also benefits from a layered, fine-to-coarse structure as they could handle small, non-convex details properly during the process. Baseline models, especially [13], mainly used a per-point MLP along with global max-pooling operation to combine features of every point, where [54] have shown that the final max-pooling combination operation will filter out many details, as the max-pooling roughly picks a convex hull of the original object. FoldingNet [12] added a graph-based local max-pooling operation as a feature combination step in a local neighborhood, but it still uses a global max-pooling operation without any hierarchical process. So that neither FoldingNet [12] nor AE-EMD [13] will get a significantly improved performance for objects like the shelf in figure 4.

5.3 Latent space dynamics

We trained our latent space estimator model on top of a pre-trained auto-encoder model. Our dataset consists of 1,000 simulations with a different initial state and randomly positioned particles. The auto-encoder used in this experiment
Figure 5: Latent space dynamics simulation. Upper figure represents FLUIDS dataset, while the figure in the middle represents LJP dataset. Particles are colored by their current velocity, as our auto-encoder for latent space simulation is designed to encode and decode both position and velocity data. Note that the only input to our system is the first frame where velocity is zero everywhere, and our model iteratively predicts all 379 frames (for FLUIDS) or 259 frames (for LJP) in chronological order on its own. Our model successfully catches the dynamics under latent space in this simple experiment. The lower figure indicates the mean loss (Sinkhorn distance) w.r.t. time steps during evaluation.

has a 64-point latent cloud with 64 feature channels, resulting in a 4,288-d latent space. In this experiment, we also introduced velocity along with position to our point cloud, so that each 3-d point also carry 3-d velocity information as the input feature map with 3 channels. Our results could be seen in figure 5, which shows that we got senseful results in this simple experimental set-up. Since we only used a simple, single-layer Interaction Networks model without any extensions, we may expect that better results could be achieved in the near future.

5.4 What does the model learned? A visualization of convolution kernel and activations

In order to obtain a better understanding of our model, we visualized the convolution kernel in each layer as well as the activations (i.e. feature maps produced) in each layer, as shown in figure 6. For convolution kernels, we rendered two isosurfaces of the entire kernel for a specific output channel (equation 5) w.r.t. position $\in \mathbb{R}^3$ and input $x^{(0)} = 1$, where light-colored isosurface locates on the 50%th of kernel value range, and blue-solid isosurface locates on the 70%th of kernel value range. Only the convolution kernel of the first convolution layer was visualized, as it is the only layer that
Irregular Convolutional Auto-Encoder on Point Clouds

Figure 6: kernel and activation visualizations, where we have used the same color scale as above. Note that selected channels are no longer consistent after (contains) 2nd block, as those layers would have many channels. In addition, farthest sampling preserves a uniformly-distributed structure in the pooled clouds, which greatly benefits our model.

have only 1 input channel, which is convenient for visualization. We also visualized the per-channel activation (i.e. feature maps) of each layer and colored them using the same color scale as previous sections. We have selected the first convolution layer, the last convolution layer of each hierarchical block (without the final fully-connected layer) for visualization.

From figure 6 we could observe that our convolution kernel can extract local features of a given point cloud. For example, kernel (a) catches "x-y walls" (i.e. local facelets that are orthogonal to the z-axis), which could be clearly seen in both the kernel visualization and activations on several inputs. On the other hand, kernel (b) catches both "x-z ceilings" and "x-y walls", as seen in visualizations. Kernel (b) is also easier to get activated compared to the kernel (a). As the input point cloud was processed to the next hierarchical level, we could observe that some special part of it was activated, such as the handle and wheels of a motorbike. Then finally we could get a rich latent representation of the input cloud, where we could use it to reconstruct the cloud as a union of local patches, as figure 6 shown with varying colors for different patches.

5.5 Ablation studies

As for the ablation study, we trained our model with different parameters and extensions, with a baseline trained as our latent cloud model stated in our main experiments, without density elimination and layer normalization, and uses a-EMD as the loss function instead of Sinkhorn-FP64. We evaluate the model performance after substituting or adding different parts to our model as an (inverted) ablation study. We trained all the models in our ablation experiments in a shorter time, for 10 epochs with batch size 8, and a larger learning rate $3 \times 10^{-4}$ and all other parameters are the same as above.
Figure 7: Results for our ablation study, better viewed with proper magnification. Please pay attention to two small but important zoomed images for column "Chamfer", as they illustrates collapsing points and the reason of why column "Chamfer" seems to have less points than others. They actually have same number of points, but many of the points collapsed into a single "point" in column "Chamfer", probably caused by the bad properties of Chamfer pseudo-distance as we stated in figure 3. Note how concat-convolution and concat-conditioning harms the model and producing distorted reconstructions. Layer normalization, density elimination and max-evidence all regularizes the model, resulting fine and sharp results. Results for Chamfer pseudo-distance is more accurate due to an accurate point-wise distance, but it fails to preserve an uniformly distributed point cloud.

Latent cloud size As observed from table 2, more latent points with fewer feature channels per point will lead to better results. However, since the local patches would be so small that the model may not learn useful and macroscopic representations, we kept using 32 latent points in our final model.

The effectiveness of our proposed convolution operation From table 2 we could find out enough evidence for the benefits of our convolution layer, compared with conventional concatenation-based convolutions. Meanwhile, the generalization gap between training and inference will become huge if we introduce batch-renormalization [74] to concatenation-based convolution layers, and in contrast, it actually could benefit our convolution layers as we observed a significant boost in training and converging speed. As normalization becomes more important as the network goes deeper, where we have used a deep model here, will lead concatenation-based convolution layers to bad results. Nevertheless, concatenation-based convolution layers themselves are not efficient and not designed for point clouds, where our convolution layers could efficiently extract local spatial features easily.

By changing $d$ from 2 to 6 does improve the network performance, but only by a slight amount. Since the computation complexity grows nearly linearly w.r.t. $d$, this modification triples the computational cost for the convolution layers. Although our main computational overhead comes from the calculation of EMD and building the k-NNG and the calculation of regular matrix products in modern GPUs was well-optimized, increasing $d$ to 6 still remains un-efficient for computation in both time and memory, compares to the slightly increased performance. Thus we choose $d = 2$ as a sweet balance point of the trade-off shown above.

From the table, we could also observe that both density-elimination and max-evidence brings a significant improvement to the overall performance. As we indicated in section 3.1 and 3.5, those methods are introduced to eliminate the influence of input point cloud density. Since density elimination is more intuitively and theoretically supported, and it produces visually finer results compared with max-evidence as shown in figure 4 with nearly the same quantitative
Irregular Convolutional Auto-Encoder on Point Clouds

| Modifications                  | M-EMD ($\times 10^{-2}$) | ΔM-EMD ($\times 10^{-4}$) |
|-------------------------------|---------------------------|---------------------------|
| Baseline                      | 2.8597                    |                           |
| Random Gaussian               | 17.9160                   | +1505.63                  |
| Convolution                   |                           |                           |
| $d = 6$                       | 2.7674                    | - 9.23                    |
| Concat convolution            | 4.1985                    | + 133.88                  |
| Max-evidence (ME)             | 2.6465                    | - 21.32                   |
| Density-elimination (DE)      | 2.6651                    | - 19.46                   |
| Latent code                   |                           |                           |
| Cloud, $16 \times 32$         | 2.9584                    | + 9.87                    |
| Cloud, $64 \times 8$         | 2.6940                    | - 16.57                   |
| Decoder                       |                           |                           |
| Concat conditioner            | 3.4564                    | + 59.67                   |
| Normalization                 |                           |                           |
| Batch renorm [74]             | 2.7102                    | - 14.95                   |
| Layer norm (LN) [80]          | 2.6027                    | - 25.70                   |
| Instance norm [84]            | 2.7714                    | - 8.83                    |
| Loss function                 |                           |                           |
| Chamfer pseudo-dist.          | 6.6061                    | + 374.64                  |
| Sinkhorn distance FP32        | 2.6775                    | - 18.22                   |
| Sinkhorn distance FP64        | 2.6203                    | - 23.94                   |
| Combined                      |                           |                           |
| Density-elimination + LN      | 2.5917                    | - 26.80                   |
| Sinkhorn FP64 + DE + LN, 18 epochs | 2.3486                  | - 51.11                   |
| Optimal (GT)                  | 2.1548                    | - 70.49                   |

Table 2: Quantitative results for our ablation studies. $d$ is as we stated in equation 5, concat convolution as $x_{i}^{(\ell+1)} = \sum_{j \in N(i)} \Phi \left( x_{j}^{(\ell)} ; x_{i}^{(\ell)} ; (p_{j} - p_{i}) \right)$, concat conditioner as in section 3.3, normalization as we applied to the output of each layer before activation functions, and loss functions as in section 3.4.

results, we considered using density-elimination in our final model. However, as we stated in section 3.5, this approach might be still far from an optimal one.

Choice of loss functions As stated in table 2, Chamfer pseudo-distance behaves dramatically worse than a-EMD as well as Sinkhorn distance. From figure 7 we could observe that the results produced by Chamfer pseudo-distance mainly suffer from an unbalanced density in different parts of the reconstructed point cloud, which was already shown by [13].

As for the Sinkhorn distance, it requires a great numerical capability to avoid numerical overflows during the Sinkhorn iteration when using lower entropy constraints, which is crucial in order to produce better results than a-EMD. Such constraints often require the capability of hardware to efficiently compute double-precision floating-point numbers float64, which is not efficient in older GPUs. We found that under regular float32 settings, the lowest possible regularization term $\frac{1}{\lambda}$ is $1.3 \times 10^{-2}$, which we refer to "Sinkhorn distance FP32", and for float64 settings the lowest possible regularization term is $2 \times 10^{-3}$ as we refer to "Sinkhorn distance FP64". Note that we first normalize the pair-wise cost matrix by dividing it element-wise by the largest entry over the entire mini-batch, only for computing the transport matrix. This could make the computation more stable and allows lower regularization terms.

Although we have above limitations, we still use Sinkhorn distance while high-speed FP64 calculation is available. Sinkhorn distance is definitely a better choice than a-EMD in such a task, and also dramatically boosts the training process, as we stated in table 2. By using FP32 we got nearly the same (about 8% longer) training time per epoch than a-EMD, and by using FP64 we are doubled the time than FP32 in modern GPUs, while 32 times slower on NVIDIA Maxwell GPUs. Note how Sinkhorn distance-based loss leads the model to a more accurate reconstruction.

Conditioning methods of the decoding generator AdaIN [18] and the methods introduced in [24] is significantly better than concatenation-based methods for conditioning the decoder, as shown in table 2. A fundamental difference of those methods is that style-based conditioner [24] fuses the conditioning vector (style) to each layer in the decoder, while concatenation-based method only appends the conditioning vector to the input of the entire decoder. Recent developments in GANs and image synthesis [24, 21, 85, 86] also support that concatenation-based conditioning is not optimal for such tasks.

$^{1}$Since the cost matrix is non-negative, it is equivalent to the entry has the largest absolute value.
Normalization methods

As indicated in table 2, layer normalization [80] works best with our model, among 3 candidate methods. Since our model needs a relatively large memory, where only a small-sized minibatch is allowed, batch-based normalization [87] e.g. batch re-normalization [74] still suffers from small batch-sizes. As normalization could be crucial for deep models as they reduced the “internal covariate shift” as stated in the original work of batch normalization [87], we choose layer normalization in our model.

6 Future works and conclusion

In this paper, we proposed a novel convolutional layer for irregular point clouds. We also proposed a deep auto-encoder model based on our convolutional layer, which outperforms other baseline models and shows the benefits of our novel convolution operation, as well as a learnable latent dynamics estimator, working well under the encoded latent space. Our experiments have successfully shown the efficiency of our convolutional layer as well as our deep auto-encoder model.

However, as we stated in section 3.5, we are still far from the best solution for irregular convolution operations. Those irregular convolution operations are particularly useful since that kind of convolution could work with arbitrary data, not only constrained to 3-D geometries, but also for high-dimensional "point" distributions such as a bunch of images, videos, text, etc. In that high-dimensional case, we could use advanced networks (e.g. convolutional neural networks, transformers [88], etc.) to replace the spatial MLP $\Phi_\theta$ as in equation 5. This leads to an important future research direction, as efforts will be needed to extend convolution to high-dimensional, irregular points. GANs [19] could be useful in such cases as they provided a well-behaved statistical distance estimator in high-dimensional spaces.

On the other hand, as we proposed a novel general framework for point cloud processing, further researches may also focus on latent space simulation, point cloud compressing, classification, segmentation, etc. As our model still has many parts to be improved (e.g. variational auto-encoders [14], neighbors selection [52], pooling methods), we are also interested in improving our model in those directions.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1”. In: ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362.
[2] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural Discrete Representation Learning”. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. 2017, pp. 6309–6318.
[3] Ali Razavi, Aäron van den Oord, and Oriol Vinyals. “Generating Diverse High-Fidelity Images with VQ-VAE-2”. In: CoRR abs/1906.00446 (2019). arXiv: 1906.00446.
[4] Jonathan Masci et al. “Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction”. In: Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I. 2011, pp. 52–59.
[5] Steffen Wiewel, Moritz Becher, and Nils Thürey. “Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow”. In: Comput. Graph. Forum 38.2 (2019), pp. 71–82.
[6] Byungsoo Kim et al. “Deep Fluids: A Generative Network for Parameterized Fluid Simulations”. In: Comput. Graph. Forum 38.2 (2019), pp. 59–70.
[7] Qingyang Tan et al. “Mesh-based autoencoders for localized deformation component analysis”. In: Thirty-Second AAAI Conference on Artificial Intelligence. 2018.
[8] Qingyang Tan et al. “Variational autoencoders for deforming 3d mesh models”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 5841–5850.
[9] Viorica Patraucean, Ankur Handa, and Roberto Cipolla. “Spatio-temporal video autoencoder with differentiable memory”. In: CoRR abs/1511.06309 (2015). arXiv: 1511.06309.
[10] Yong Shean Chong and Yong Haur Tay. “Abnormal Event Detection in Videos Using Spatiotemporal Autoencoder”. In: Advances in Neural Networks - ISNN 2017 - 14th International Symposium, ISNN 2017, Sapporo, Hokkaido, and Muroran, Hokkaido, Japan, June 21-26, 2017, Proceedings, Part II. 2017, pp. 189–196.
[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. 2014, pp. 3104–3112.
[12] Yaoqing Yang et al. “FoldingNet: Point Cloud Auto-Encoder via Deep Grid Deformation”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. 2018, pp. 206–215.

[13] Panos Achlioptas et al. “Learning Representations and Generative Models for 3D Point Clouds”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. 2018.

[14] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014.

[15] Charlie Nash and Christopher K. I. Williams. “The shape variational autoencoder: A deep generative model of part-segmented 3D objects”. In: *Comput. Graph. Forum* 36.5 (2017), pp. 1–12.

[16] Shirui Pan et al. “Adversarially Regularized Graph Autoencoder for Graph Embedding”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. 2018, pp. 2609–2615.

[17] William Lotter, Gabriel Kreiman, and David Cox. “Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[18] Xun Huang and Serge J. Belongie. “Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization”. In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, pp. 1510–1519.

[19] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *Corr abs/1406.2661* (2014). arXiv: 1406.2661.

[20] Mehdi Mirza and Simon Osindero. “Conditional Generative Adversarial Nets”. In: *Corr abs/1411.1784* (2014). arXiv: 1411.1784.

[21] Han Zhang et al. “Self-Attention Generative Adversarial Networks”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019, pp. 7354–7363.

[22] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pp. 5769–5779.

[23] Andrew Brock, Jeff Donahue, and Karen Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: *Corr abs/1809.11096* (2018). arXiv: 1809.11096.

[24] Tero Karras, Samuli Laine, and Timo Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *Corr abs/1812.04948* (2018). arXiv: 1812.04948.

[25] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks”. In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, pp. 2242–2251.

[26] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018.

[27] Casey Chu, Andrey Zhmoginov, and Mark Sandler. “CycleGAN, a Master of Steganography”. In: *Corr abs/1712.02950* (2017). arXiv: 1712.02950.

[28] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pp. 5967–5976.

[29] Martin Arjovsky and Léon Bottou. “Towards Principled Methods for Training Generative Adversarial Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[30] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein GAN”. In: *Corr abs/1701.07875* (2017). arXiv: 1701.07875.

[31] Junbo Jake Zhao, Michaël Mathieu, and Yann LeCun. “Energy-based Generative Adversarial Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[32] Yujia Li, Kevin Swersky, and Richard S. Zemel. “Generative Moment Matching Networks”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 2015, pp. 1718–1727.
[33] Yong Ren et al. “Conditional Generative Moment-Matching Networks”. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. 2016, pp. 2928–2936.

[34] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. “Training generative neural networks via Maximum Mean Discrepancy optimization”. In: Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands. 2015, pp. 258–267.

[35] Dougal J. Sutherland et al. “Generative Models and Model Criticism via Optimized Maximum Mean Discrepancy”. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. 2017.

[36] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. “f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization”. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. 2016, pp. 271–279.

[37] Chun-Liang Li et al. “Point cloud gan”. In: arXiv preprint arXiv:1810.05795 (2018).

[38] Dimitri P Bertsekas. “A distributed asynchronous relaxation algorithm for the assignment problem”. In: 1985 24th IEEE Conference on Decision and Control. IEEE. 1985, pp. 1703–1704.

[39] Marco Cuturi. “Sinkhorn Distances: Lightspeed Computation of Optimal Transport”. In: Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013, Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. 2013, pp. 2292–2300.

[40] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. “Wavelets on graphs via spectral graph theory”. In: Applied and Computational Harmonic Analysis 30.2 (2011), pp. 129–150.

[41] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. 2016, pp. 3837–3845.

[42] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. 2017.

[43] Petar Velickovic et al. “Graph Attention Networks”. In: 6th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. 2017.

[44] Peter W. Battaglia et al. “Interaction Networks for Learning about Objects, Relations and Physics”. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. 2016, pp. 4502–4510.

[45] Damian Mrowca et al. “Flexible neural representation for physics prediction”. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. 2018, pp. 8813–8824.

[46] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. 2017, pp. 1263–1272.

[47] Peter W. Battaglia et al. “Relational inductive biases, deep learning, and graph networks”. In: CoRR abs/1806.01261 (2018). arXiv: 1806.01261.

[48] Martin Simonovsky and Nikos Komodakis. “Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs”. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017. 2017, pp. 29–38.

[49] Yunzhu Li et al. “Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids”. In: CoRR abs/1810.01566 (2018). arXiv: 1810.01566.

[50] Zhitao Ying et al. “Hierarchical Graph Representation Learning with Differentiable Pooling”. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. 2018, pp. 4805–4815.

[51] Hongyang Gao and Shuiwang Ji. Graph U-Net. 2019.

[52] Charles Ruizhongtai Qi et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: Advances in Neural Information Processing Systems 30. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5099–5108.

[53] Chu Wang, Babak Samari, and Kaleem Siddiqi. “Local Spectral Graph Convolution for Point Set Feature Learning”. In: Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IV. 2018, pp. 56–71.
Irregular Convolutional Auto-Encoder on Point Clouds

[54] R. Q. Charles et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). July 2017, pp. 77–85.

[55] Manzil Zaheer et al. “Deep Sets”. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. 2017, pp. 3394–3404.

[56] Hang Su et al. “SPLATNet: Sparse Lattice Networks for Point Cloud Processing”. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pp. 2530–2539.

[57] Huan Lei, Naveed Akhtar, and Ajmal Mian. “Spherical Convolutional Neural Network for 3D Point Clouds”. In: CoRR abs/1805.07872 (2018). arXiv: 1805.07872.

[58] Chu Wang, Babak Samari, and Kaleem Siddiqi. “Local Spectral Graph Convolution for Point Set Feature Learning”. In: Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IV. 2018, pp. 56–71.

[59] Yangyan Li et al. “PointCNN: Convolution On X-Transformed Points”. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. 2018, pp. 828–838.

[60] Yiru Shen et al. “Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling”. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pp. 4548–4557.

[61] L. Tchapmi et al. “SEGCloud: Semantic Segmentation of 3D Point Clouds”. In: 2017 International Conference on 3D Vision (3DV). Oct. 2017, pp. 537–547.

[62] Yue Wang et al. “Dynamic Graph CNN for Learning on Point Clouds”. In: CoRR abs/1801.07829 (2018). arXiv: 1801.07829.

[63] Jiaxin Li, Ben M. Chen, and Gim Hee Lee. “SO-Net: Self-Organizing Network for Point Cloud Analysis”. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pp. 9397–9406.

[64] Loic Landrieu and Martin Simonovsky. “Large-Scale Point Cloud Semantic Segmentation With Superpoint Graphs”. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pp. 4558–4567.

[65] Matan Atzmon, Haggai Maron, and Yaron Lipman. “Point convolutional neural networks by extension operators”. In: ACM Trans. Graph. 37.4 (2018), 71:1–71:12.

[66] H. Fan, H. Su, and L. Guibas. “A Point Set Generation Network for 3D Object Reconstruction from a Single Image”. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). July 2017, pp. 2463–2471.

[67] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. “Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction”. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pp. 7114–7121.

[68] Mathieu Gadelha, Subhransu Maji, and Rui Wang. “Shape Generation using Spatially Partitioned Point Clouds”. In: (July 2017).

[69] Nanyang Wang et al. “Pixel2mesh: Generating 3d mesh models from single rgb images”. In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, pp. 52–67.

[70] Jonathan Tompson et al. “Accelerating Eulerian Fluid Simulation With Convolutional Networks”. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. 2017, pp. 3424–3433.

[71] CloudCompare (version 2.11) [GPL software]. http://www.cloudcompare.org/. 2019.

[72] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel. 2010, pp. 807–814.

[73] Christian Szegedy et al. “Going deeper with convolutions”. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. 2015, pp. 1–9.

[74] Sergey Ioffe. “Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models”. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. 2017, pp. 1945–1953.
[75] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pp. 770–778.

[76] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015.

[77] Barnabás Póczos, Liang Xiong, and Jeff G. Schneider. “Nonparametric Divergence Estimation with Applications to Machine Learning on Distributions”. In: UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011. 2011, pp. 599–608.

[78] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[79] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[80] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer Normalization”. In: CoRR abs/1607.06450 (2016). arXiv: 1607.06450.

[81] R’emi Flamary and Nicolas Courty. *POT Python Optimal Transport library*. https://github.com/rflamary/POT. 2017.

[82] Nils Thuerey and Tobias Pfaff. *MantaFlow*: http://mantaflow.com. 2018.

[83] Jeremiah U Brackbill, Douglas B Kothe, and Hans M Ruppel. “FLIP: a low-dissipation, particle-in-cell method for fluid flow”. In: Computer Physics Communications 48.1 (1988), pp. 25–38.

[84] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: CoRR abs/1607.08022 (2016). arXiv: 1607.08022.

[85] Harm de Vries et al. “Modulating early visual processing by language”. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. 2017, pp. 6594–6604.

[86] Takeru Miyato and Masanori Koyama. “cGANs with Projection Discriminator”. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. 2018.

[87] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. 2015, pp. 448–456.

[88] Ashish Vaswani et al. “Attention is All you Need”. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. 2017, pp. 5998–6008.