# Generative Models as Distributions of Functions

**Emilien Dupont**
University of Oxford
dupont@stats.ox.ac.uk

**Yee Whye Teh**
University of Oxford
y.w.teh@stats.ox.ac.uk

**Arnaud Doucet**
University of Oxford
doucet@stats.ox.ac.uk

## Abstract

Generative models are typically trained on grid-like data such as images. As a result, the size of these models usually scales directly with the underlying grid resolution. In this paper, we abandon discretized grids and instead parameterize individual data points by continuous functions. We then build generative models by learning distributions over such functions. By treating data points as functions, we can abstract away from the specific type of data we train on and construct models that scale independently of signal resolution. To train our model, we use an adversarial approach with a discriminator that acts on continuous signals. Through experiments on both images and 3D shapes, we demonstrate that our model can learn rich distributions of functions independently of data type and resolution.

## 1 Introduction

In generative modeling, data is often represented by discrete arrays. Images are represented by two dimensional grids of RGB values, 3D scenes are represented by three dimensional voxel grids and audio as vectors of discretely sampled waveforms. However, it is often the case that the true underlying signal is continuous. We can therefore also consider representing such signals by continuous functions taking as input grid coordinates and returning features. In the case of images for example, we can define a function $f : \mathbb{R}^2 \to \mathbb{R}^3$ mapping pixel locations to RGB values using a neural network. Such representations, typically referred to as implicit neural representations, coordinate-based neural representations or neural function representations, have the remarkable property that they are independent of signal resolution [46, 37, 9, 60].

In this paper, we build generative models that inherit the attractive properties of implicit representations. By framing generative modeling as learning distributions of functions, we are able to build models that act entirely on continuous spaces, independently of resolution. We achieve this by parameterizing a distribution over neural networks with a hypernetwork [21] and training this distribution with an adversarial approach [16], using a discriminator that acts directly on sets of coordinates (e.g. pixel locations) and features (e.g. RGB values). Crucially, this allows us to train the model irrespective of any underlying discretization or grid and avoid the *curse of discretization* [39].

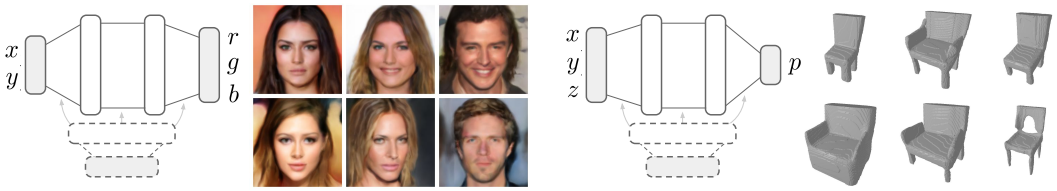


Figure 1: By representing data as continuous functions, we can use the same model to learn distributions of images and 3D shapes, irrespective of any underlying grid or resolution.

Preprint. Under review.

Indeed, standard convolutional generative models act on discretized grids, such as images or voxels, and as a result scale quadratically or cubically with resolution, which quickly becomes intractable at high resolutions, particularly in 3D [46, 38]. In contrast, our model learns distributions on continuous spaces and is agnostic to discretization. While convolutional models currently dominate generative modeling, they are fundamentally limited by the curse of discretization and it is therefore likely that, in order to scale to very large data, we will eventually need to abandon discretization.

To validate our approach, we train generative models on various image and 3D shape datasets. Remarkably, we show that, using our framework, we can learn rich function distributions on both images and 3D shapes using the *same model*. Further, by taking advantage of recent advances in representing high frequency functions with neural networks [40, 64, 60], we also show that, unlike current approaches for generative modeling on continuous spaces [46, 14], we are able to generate sharp and realistic samples. Finally, we explore the continuity of the learned representations and show that our model is often able to generate plausible samples at higher resolutions than it was trained on.

## 2 Representing data as functions

For clarity, we use the case of representing an image by a function as a guiding example, but the methods described here hold more generally and have been used extensively in 3D vision [46, 38, 40].

### 2.1 Representing a single image with a function

Let $I$ be an image such that $I[x, y]$ corresponds to the RGB value at pixel location $(x, y)$. We are interested in representing this image by a function $f : \mathbb{R}^2 \to \mathbb{R}^3$ where $f(x, y) = (r, g, b)$ returns the RGB values at pixel location $(x, y)$. To achieve this, we parameterize a function $f_\theta$ by an MLP with weights $\theta$. We can then learn the function representation by minimizing

$$\min_\theta \sum_{x,y} \|f_\theta(x, y) - I[x, y]\|_2^2,$$

where the sum is over all pixel locations. Remarkably, the representation $f_\theta$ is *independent* of the number of pixels. The representation $f_\theta$ therefore, unlike most image representations, does not depend on the resolution of the image [38, 46, 60].

### 2.2 Representing general data with functions

The above example with images can readily be extended to more general data. Let $\mathbf{x} \in \mathbb{R}^d$ denote coordinates and $\mathbf{y} \in \mathbb{R}^k$ features and assume we are given a data point as a set of coordinate and feature pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$. For an image for example, $\mathbf{x} = (x, y)$ corresponds to pixel locations, $\mathbf{y} = (r, g, b)$ corresponds to RGB values and $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ to the set of all pixel locations and RGB values. Given a set of coordinates and their corresponding features, we can then consider learning a function $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ representing this data point by minimizing

$$\min_\theta \sum_{i=1}^n \|f_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2^2. \tag{1}$$

For function representations, the only assumption on the data is that it can be expressed in terms of coordinates and features. This is very general and includes:

- Images: $f : \mathbb{R}^2 \to \mathbb{R}^3$, $f(x, y) = (r, g, b)$.
- Videos: $f : \mathbb{R}^3 \to \mathbb{R}^3$, $f(x, y, t) = (r, g, b)$.
- Voxels: $f : \mathbb{R}^3 \to \{0, 1\}$, $f(x, y, z) = p$ where $p = 0$ for an empty voxel and $p = 1$ for an occupied voxel.

The core property of these representations is that they scale with signal *complexity* and not with signal resolution [60]. Indeed, the memory required to store data scales quadratically with resolution for images and cubically for voxel grids. In contrast, for function representations, the memory requirements scale directly with signal complexity: to represent a more complex signal, we would need to increase the capacity of the function $f_\theta$, for example by increasing the number of layers in the case of a neural network. This scaling is often much better than the quadratic or cubic scaling required for typical discrete grid representations [46].

### 2.3 Representing high frequency functions

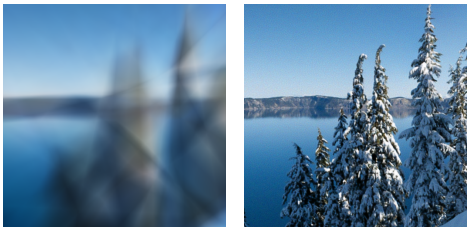

Figure 2: Modeling an image with a function with (right) and without (left) Fourier features.

Recently, it has been shown that learning function representations by minimizing equation (1) is biased towards learning low frequency functions [40, 60, 64]. To alleviate this problem, several approaches have been proposed. Mildenhall et al. [40] transform the coordinates $\mathbf{x}$ with a sinusoidal encoding layer before passing it through the network $f_\theta$. Sitzmann et al. [60] propose learning MLPs with sinusoidal activations and Tancik et al. [64] use random Fourier features [52] on the coordinates. In this paper, we use the random Fourier feature (RFF) encoding as it is not biased towards on axis variation (unlike Mildenhall et al. [40]) and does not require specialized initialization (unlike Sitzmann et al. [60]). Specifically, given a coordinate $\mathbf{x} \in \mathbb{R}^d$, the encoding function $\gamma : \mathbb{R}^d \to \mathbb{R}^{2m}$ is defined as

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi B\mathbf{x}) \\ \sin(2\pi B\mathbf{x}) \end{pmatrix},$$

where $B \in \mathbb{R}^{m \times d}$ is a (potentially learnable) random matrix whose entries are typically sampled from $\mathcal{N}(0, \sigma^2)$. The number of frequencies $m$ and the variance $\sigma^2$ of the entries of $B$ are hyperparameters. To learn high frequency functions, we simply encode $\mathbf{x}$ before passing it through the MLP and minimize

$$\min_\theta \sum_{i=1}^{n} \|f_\theta(\gamma(\mathbf{x}_i)) - \mathbf{y}_i\|_2^2.$$

As can be seen in Figure 2, learning a function representation of an image with a ReLU MLP fails to capture high frequency detail whereas using an RFF encoding followed by a ReLU MLP allows us to faithfully reproduce the image.

## 3 Learning distributions of functions

In generative modeling, we are typically given a set of data, such as images, and are interested in approximating the distribution of this data. As we represent data points by functions, we would therefore like to learn a distribution over functions. In the case of images, standard generative models typically sample some noise and feed it through a neural network to output $n$ pixels [16, 30, 54]. In contrast, we sample the weights of a neural network to obtain a function which we can probe at arbitrary coordinates. Such a representation allows us to operate entirely on coordinates and features irrespective of any underlying grid representation that may be available.

To train the function distribution we use an adversarial approach. While it is possible to use other methods, we favored adversarial training as generative adversarial networks (GAN) are typically able to generate sharp and high frequency samples [26, 27].

### 3.1 Neural function distributions

In this section, we describe how to parameterize a distribution over functions. We assume the structure (e.g. the number and width of layers) of the MLP $f_\theta$ representing a single data point is fixed. Learning a distribution over functions $f_\theta$ is then equivalent to learning a distribution over weights $p(\theta)$. The distribution $p(\theta)$ is defined by a latent distribution $p(\mathbf{z})$ and a second function $g_\phi : \mathcal{Z} \to \Theta$, itself with parameters $\phi$, mapping latent variables to the weights $\theta$ of $f_\theta$ (see Figure 3). We can then sample from $p(\theta)$ by sampling $\mathbf{z} \sim p(\mathbf{z})$ and mapping $\mathbf{z}$ through $g_\phi$ to obtain a set of weights $\theta = g_\phi(\mathbf{z})$. We refer to such a distribution over functions as a neural function distribution (NFD). Note that the function $g_\phi$ is itself a neural network parameterizing the weights of another neural network $f_\theta$ and as such is a hypernetwork [21]. Hypernetworks, and special cases thereof [60], have been extensively used to parameterize spaces of 3D scenes [46, 38, 59].

### 3.2 Data representation

While our goal is to learn a distribution over functions, we typically do not have access to the ground truth functions representing the data. Instead, each data point is typically given by some *set* of coordinates and features $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$. For an image for example, we do not have access to a function mapping pixel locations to RGB values but to a collection of $n$ pixels. Such a set of coordinates and features then corresponds to input/output pairs of a function, allowing us to learn function distributions without operating directly on the functions.

A *single* data point then corresponds to a *set* of coordinates and features (e.g. an image is a set of $n$ pixels). We then assume a dataset is given to us as samples $\mathbf{s} \sim p_{\text{data}}(\mathbf{s})$ from a distribution over sets of coordinate and feature pairs. We note that working with sets of coordinate and feature pairs is very flexible - such a representation is agnostic to whether the data originated from a grid and at which resolution it was sampled.

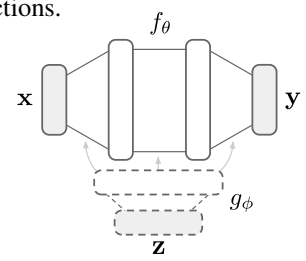


Figure 3: Diagram of neural function distribution architecture. A latent vector $\mathbf{z}$ is mapped through a hypernetwork $g_\phi$ (in dashed lines) to obtain the weights of a function $f_\theta$ (in solid lines) mapping coordinates $\mathbf{x}$ to features $\mathbf{y}$.

Crucially, formulating our problem entirely on sets lets us split individual data points into subsets and train on those. Specifically, given a single data point $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, such as a collection of $n$ pixels, we can randomly subsample $K$ elements, e.g. we can select $K$ pixels among the $n$ pixels in the entire image. Training on such subsets then removes any direct dependence on the resolution of the data. For example, when training on 3D shapes, instead of passing an entire voxel grid to the model, we can train on subsets of the voxel grid, leading to large memory savings. This is not possible with standard convolutional models which are directly tied to the resolution of the grid.

### 3.3 Function generator

Learning distributions of functions with an adversarial approach requires us to define a generator that generates fake functions and a discriminator that distinguishes between real and fake functions. We define the generator with an NFD and use functions sampled from this distribution to generate fake sets of coordinate and feature pairs. The fake data generation follows two steps: 1. sample a function $f_\theta$ from an NFD, 2. evaluate $f_\theta$ at a set of coordinates $\{\mathbf{x}_i\}$ to obtain a set of generated features $\{\mathbf{y}_i\}$. Specifically, given a latent vector $\mathbf{z}$ and a coordinate $\mathbf{x}_i$, we compute a generated feature as $\mathbf{y}_i = f_{g_\phi(\mathbf{z})}(\gamma(\mathbf{x}_i))$ where $\gamma$ is an RFF encoding allowing us to learn high frequency functions.

### 3.4 Point cloud discriminator

In the GAN literature, discriminators are almost always parameterized with convolutional neural networks (CNN). However, the data we consider may not necessarily lie on a grid, in which case it is not possible to use convolutional discriminators. Further, convolutional discriminators scale directly with grid resolution (training a CNN on images at $2\times$ the resolution requires $4\times$ the memory) which partially defeats the purpose of using implicit representations.

As the core idea of our paper is to build generative models that are independent of resolution, we therefore cannot follow the naive approach of using convolutional discriminators. Instead, our discriminator should be able to distinguish between real and fake sets of coordinate and feature pairs. Specifically, we need to define a function $D$ which takes in an *unordered set* $\mathbf{s}$ and returns the probability that this set represents input/output pairs of a real function. We therefore need $D$ to be permutation invariant with respect to the elements $s_i$ of the set $\mathbf{s} = \{s_1, s_2, ..., s_n\}$.

The canonical choice for set functions is the PointNet [49] or DeepSets [71] model family. However, we experimented extensively with such functions and found that they were not adequate for learning complex function distributions (see Section 3.6 for further details). Indeed, while the input to the discriminator is an unordered set $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, there is an underlying notion of distance between points $\mathbf{x}_i$ in the coordinate space. We found that is is crucial to take this into account when training models on large scale datasets. Indeed, we should not consider the coordinate and feature pairs as sets but rather as *point clouds* (i.e. sets with an underlying notion of distance).

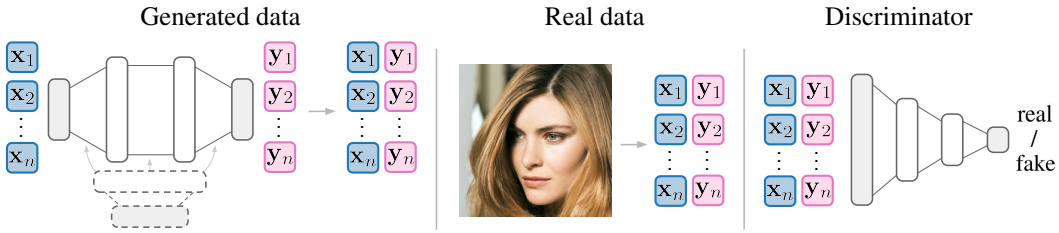


Figure 5: Training procedure for neural function distributions: 1. Sample a function from NFD and evaluate it at various coordinate locations to generate fake point cloud. 2. Convert real data sample to point cloud. 3. Discriminate between real and fake point clouds.

### 3.4.1 PointConv

While several works have tackled the problem of point cloud classification [49, 32, 65], we leverage the PointConv framework introduced by Wu et al. [66] for several reasons. Firstly, PointConv layers are translation equivariant (like regular convolutions) and permutation invariant by construction. Secondly, when sampled on a regular grid, PointConv networks closely match the performance of regular CNNs. Indeed, we can loosely think of PointConv as a continuous equivalent of CNNs and, as such, we can build PointConv architectures that are analogous to typical discriminator architectures.

Specifically, we assume we are given a set of features $\mathbf{f}_i \in \mathbb{R}^{c_{\text{in}}}$ at locations $\mathbf{x}_i$ (we use $\mathbf{f}_i$ to distinguish these hidden features of the network from input features $\mathbf{y}_i$). In contrast to regular convolutions, where the convolution kernels are only defined at certain grid locations, the convolution filters in PointConv are parameterized by an MLP, $W : \mathbb{R}^d \to \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$, mapping coordinates to kernel values. We can therefore evaluate the convolution filters in the entire coordinate space. The PointConv operation at a point $\mathbf{x}$ is then defined as

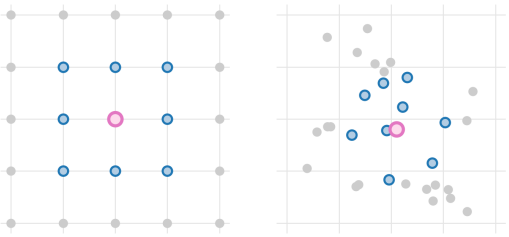

Figure 4: Convolution neighborhood for regular convolutions (left) and PointConv (right).

$$\mathbf{f}_{\text{out}}(\mathbf{x}) = \sum_{\mathbf{x}_i \in N_{\mathbf{x}}} W(\mathbf{x}_i - \mathbf{x})\mathbf{f}_i,$$

where $N_{\mathbf{x}}$ is a set of neighbors of $\mathbf{x}$ over which to perform the convolution (see Figure 4). Interestingly, this neighborhood is found by a nearest neighbor search with respect to some metric on the coordinate space. We therefore have more flexibility in defining the convolution operation as we can choose the most appropriate notion of distance for the space we want to model (our implementation supports fast computation on the GPU for any $\ell_p$ norm). While the neighborhood used for standard convolutions on the grid corresponds to the $\ell_\infty$ norm, we use the $\ell_2$ norm in our experiments.

## 3.5 Training

We use the traditional (non saturating) GAN loss [16] for training and illustrate the entire procedure for a single training step in Figure 5. To stabilize training, we define an equivalent for point clouds of the $R_1$ penalty from Mescheder et al. [37]. For images, $R_1$ regularization corresponds to penalizing the gradient norm of the discriminator with respect to the input image. For a set $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, we define the penalty as

$$R_1(\mathbf{s}) = \frac{1}{2}\|\nabla_{\mathbf{y}_1,\ldots,\mathbf{y}_n} D(\mathbf{s})\|^2 = \frac{1}{2}\sum_{\mathbf{y}_i} \|\nabla_{\mathbf{y}_i} D(\mathbf{s})\|^2,$$

that is we penalize the gradient norm of the discriminator with respect to the features. Using $R_1$ regularization was not strictly necessary for our models to converge but we found that it gave slight but consistent improvements in performance. Crucially, our entire modeling procedure is then independent of the resolution. Indeed, the generator, discriminator and loss all act on point clouds which can be subsampled independently of resolution.

### 3.6 How not to learn distributions of functions

In developing our model, we found that several approaches which intuitively seem appropriate for learning distributions of functions do not work in the context of generative modeling. We briefly describe these here and provide further details and proofs in the appendix.

**Set discriminators**. As described in Section 3.4, the canonical choice for set functions is the PointNet/DeepSet model family [49, 71]. However, we found both theoretically and experimentally that these functions were not suitable as discriminators for complex function distributions. Indeed, these models do not directly take advantage of the metric on the space of coordinates, which we conjecture is crucial for learning rich function distributions. In addition, we show in the appendix that the Lipschitz constant of set functions can be very large, leading to unstable GAN training [2, 56, 37]. We provide further theoretical and experimental insights on set discriminators in the appendix.

**Auto-decoders**. A common method for embedding functions into a latent space is the auto-decoder framework [46]. This framework and variants of it have been extensively used in 3D computer vision [46, 59]. While auto-decoders excel at a variety of tasks, we show in the appendix that the objective used to train these models is not appropriate for generative modeling. We provide further analysis and experimental results on auto-decoders in the appendix.

While none of the above models were able to learn function distributions on complex datasets such as CelebAHQ, all of them worked well on MNIST. We therefore believe that MNIST is not a meaningful benchmark for generative modeling of functions and encourage future research in this area to include experiments on more complex datasets.

## 4 Related work

**Implicit representations**. Implicit representations for 3D geometry were initially (and concurrently) proposed by [46, 38, 9]. A large body of work has since taken advantage of these representations for inverse rendering [59, 40, 45, 70], modeling dynamic scenes [44, 48], modeling 3D scenes [3, 24, 18] and superresolution [8]. As implicit representations are biased towards low frequency functions, Mildenhall et al. [40] proposed using a positional encoding to learn high frequency representations. This has since been extended to MLPs with sinusoidal activations [60] and Fourier features [64].

**Distributions of functions**. A large portion of work on implicit representations focuses on representing single instances but some of these have also been applied to distributions of representations [46, 38, 59, 60]. While these models are able to embed data into a space of implicit representations, they typically do not learn distributions from which we can sample and as such are not suitable for generative modeling. Neural processes [14, 15] are another family of models that learn distributions of functions. However, the focus of these is on uncertainty quantification and meta-learning rather than generative modeling. Further, these models do not scale to large datasets, although adding attention [29] and translation equivariance [17] helps alleviate this. In concurrent work, [61, 1, 6, 57] have also used an adversarial approach to learn distributions of high frequency implicit representations. Crucially, *these all use standard convolutional grid discriminators* and as such do not inherit several advantages of implicit representations: they are restricted to data lying on a grid and suffer from the curse of discretization. In contrast, our model is entirely continuous and independent of resolution and, as a result, we are able to train on a variety of datasets with limited computational resources. A more detailed discussion of related work in this area is included in the appendix.

**Point clouds**. Since the introduction of PointNet [49], several works have used pointwise MLPs with an aggregation function to extract features from point clouds [50, 72, 69]. Other methods are based on building equivalents of grid convolutions for point clouds, including parameterizing convolution filters by MLPs [35, 34, 66], using learnable kernel points or products of simple functions [68, 65] or applying continuous transformations before performing convolutions [32]. For a complete survey of point cloud methods in deep learning we refer the reader to Guo et al. [20].

## 5 Experiments

We evaluate our model on CelebAHQ [26] at $64 \times 64$ and $128 \times 128$ resolution as well as on 3D shapes from the ShapeNet [7] chairs category. Note that for both images and 3D shapes we use the *exact same model* except for the input and output dimensions of the function representation and the

Figure 6: Samples from our model trained on CelebAHQ $64 \times 64$ (left) and $128 \times 128$ (right). Each image corresponds to a function which was sampled from our model and then evaluated on the grid. To produce this figure we sampled 5 batches and chose the best batch by visual inspection.

hyperparameters of the Fourier features. For all datasets, we use an MLP with 3 hidden layers of size 128 for the function representation and an MLP with 2 hidden layers of size 256 and 512 for the hypernetwork parameterizing the NFD. Remarkably, we find that such a simple architecture is sufficient for learning rich distributions of images and 3D shapes.

The point cloud discriminator is loosely based on the the DCGAN architecture [51]. Specifically, we use $3^d$ neighbors for each PointConv layer and downsample points by a factor of $2^d$ at every pooling layer while doubling the number of channels. We implemented our model in PyTorch [47] and performed all training on a single 2080Ti GPU with 11GB of RAM. The code can be found at `https://github.com/EmilienDupont/neural-function-distributions`.

### 5.1 Generative modeling

We first evaluate our model on the task of image generation. To generate images, we sample a function from the learned NFD and evaluate it on a grid. As can be seen in in Figure 6, our model produces sharp and realistic images both at $64 \times 64$ and $128 \times 128$ resolution. While there are still artifacts and occasionally poor samples (particularly at $128 \times 128$ resolution), the images are generally convincing and show that the model has learned a meaningful distribution of functions representing the data. To the best of our knowledge, this is the first time data of this complexity has been modeled in an entirely continuous fashion.

### 5.2 Superresolution

As the representations we learn are independent of resolution, we can use our model to generate images at higher resolutions than the data on which it was trained. We show examples of this in Figure 7 by first sampling a function from our model, evaluating it at the resolution on which it was trained and then evaluating it at a $4\times$ higher resolution. Surprisingly, our model is able to generate convincing $256 \times 256$ images even though it has only ever seen $64 \times 64$ images during training.

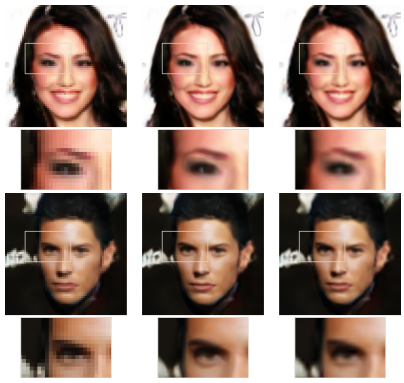

Figure 7: Superresolution. The first column corresponds to the original resolution, the second column to $4\times$ the resolution and the third column to bicubic upsampling.

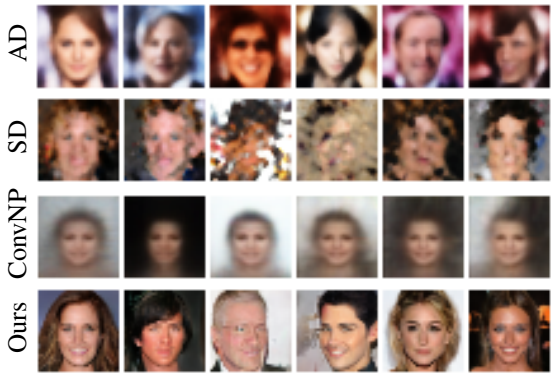


Figure 8: Baseline comparisons on CelebAHQ $32 \times 32$. Note that the ConvNP model was trained on CelebA (not CelebAHQ) and as such has a different crop than the other models.

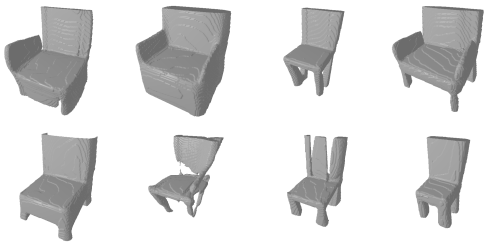

Figure 9: Samples from our model trained on ShapeNet chairs. While the model was trained on $32^3$ data, the generated samples are at $128^3$ resolution. To generate the plot, we sampled 5 batches and chose the best batch.

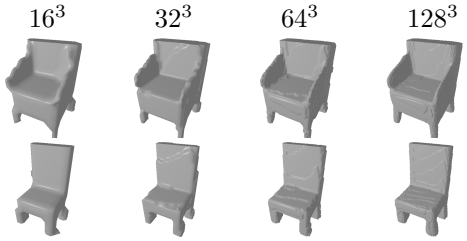


Figure 10: Evaluating the same function at different resolutions. As samples from our model can be probed at arbitrary coordinates, we can increase the resolution to render smoother meshes.

### 5.3 Baselines

In this section we compare our model against three baselines: a model trained using the auto-decoder (AD) framework [46], a model trained with a set discriminator (SD) and a convolutional neural process (ConvNP) [17]. To the best of our knowledge, these are the only other model families that can learn generative models in a continuous manner, without relying on a grid representation (which is required for regular CNNs). Results comparing all three models on CelebAHQ $32 \times 32$ are shown in Figure 8. As can be seen, the baselines generate blurry and incoherent samples, while our model is able to generate sharp, diverse and plausible samples. Quantitatively, our model (Table 1) outperforms all baselines, although it lags behind state of the art convolutional GANs specialized to images [33].

| | CelebAHQ64 | CelebAHQ128 |
|---|---|---|
| SD | 236.82 | - |
| AD | 117.80 | - |
| Ours | 7.42 | 19.16 |
| Conv [33] | 4.00 | 5.74 |

Table 1: FID scores (lower is better) for various models on CelebAHQ datasets.

### 5.4 3D scenes

To test the versatility and scalability of our model, we also train it on a different task: generating 3D shapes. To achieve this, we let the function representation $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ map $x, y, z$ coordinates to an occupancy probability $p$ (which is 0 if the location is empty and 1 if it is part of an object). To generate data, we follow the setup from Mescheder et al. [38]. Specifically, we use the voxel grids from Choy et al. [10] representing the chairs category from the ShapeNet [7] dataset. The dataset contains 6778 chairs each of dimension $32^3$. As each 3D model is large (a set of $32^3 = 32,768$ points), we uniformly subsample $K = 4096$ points from each object during training, which leads to large memory savings (Figure 11) and allows us to train with large batch sizes even on limited hardware. Crucially, *this is not possible with convolutional discriminators* and is a key property of our model: we can scale the model independently of the resolution of the data.

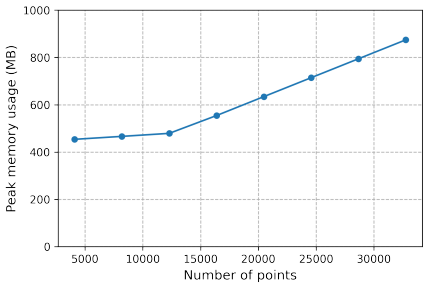


Figure 11: GPU memory consumption as a function of the number of points $K$ in voxel grid.

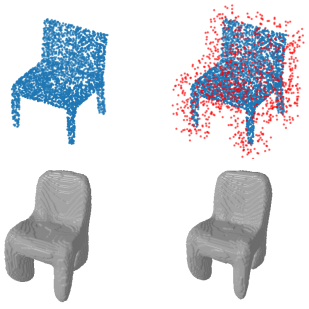

Figure 12: Top row: Point cloud without (left) and with (right) empty space samples. Bottom row: Samples from trained model.

In order to visualize results, we convert the continuous functions sampled from our model to meshes we can render. To achieve this, we first sample a function from our model and evaluate it on a high resolution grid (usually $128^3$). We then threshold the values of this grid at $0.5$ (we found the model was robust to choices of threshold) so voxels with values above the threshold are occupied while the rest are empty. Finally, we use the marching cubes algorithm [36] to convert the grid to a 3D mesh which we render with PyTorch3D [53]. As shown in Figure 9, our model is able to generate mostly coherent and realistic samples. The continuous nature of the data representation also allows us to sample our model at high resolutions to produce clean and smooth surfaces (see Figure 10). Remarkably, we were able to use the same model we used for images with little tuning.

Finally, we consider a task where using convolutional discriminators is impossible even in the limit of infinite memory: we train our model on shapenet point clouds from [42]. These point clouds do not align with a grid and as such cannot be processed by a convolutional discriminator. We note that running our model on such a dataset is straightforward as our method is agnostic to whether the data comes from a discrete voxel grid or a continuous point cloud. As shown in Figure 12, each point cloud consists of points within the object and points outside the object. While we found that training was less stable for this task, our model is still able to generate reasonable samples (Figure 12).

## 6 Scope, limitations and future work

**Limitations**. One of the main weaknesses of our method is that it does not match the sample quality of state of the art convolutional generative models [27, 28]. While the goal of this paper was not to create state of the art samples but rather to demonstrate that we can learn rich data distributions in function space, it would still be desirable to narrow the gap in performance. We strived for simplicity when designing our model but hypothesize that standard GAN tricks [26, 27, 2, 5] could help improve sample quality. In addition, we found that training could be unstable, particularly for the 3D experiments: the model typically collapses to generating simple shapes (e.g. four legged chairs) even if the data contains complex shapes (e.g. office chairs). This is especially true for the point cloud experiments where the model often collapses to generating a single shape. In addition, we found that subsampling the points typically leads to less stable training. For example, on CelebAHQ, decreasing the number of points per example also decreases the quality of the generated images (see appendix for samples and failure examples). We conjecture that this is due to the nearest neighbor search in the discriminator: when subsampling points, a nearest neighbor may lie very far from a query point, potentially leading to unstable training. More refined sampling methods as well as radius neighborhood searches (which are more robust than nearest neighbor searches [65]) should help improve stability. Finally, determining the neighborhood for the point cloud convolution can be expensive when a large number of points is used. Finding alternative architectures or incorporating methods for fast neighbor search is therefore an important direction of future work [25].

**Future work**. We believe there are many interesting avenues for future work. Firstly, as our model only depends on coordinates and features and a notion of distance on the coordinate space, it would be interesting to extend our model to more exotic spaces and manifolds. In addition, it would be interesting to apply our model to geospatial [23], geological [11], meteorological [62] or molecular [67] data which typically do not lie on a regular grid. In computer vision, we hope that our approach will help scale generative models to larger datasets. While our model in its current form could not scale to truly large datasets (such as room scale 3D scenes or very high resolution images), framing generative models entirely in terms of coordinates and features could be a first step towards this [60].

## 7 Conclusion

In this paper, we introduced a method for learning generative models that act entirely on continuous spaces and as such are independent of signal resolution. We achieved this by learning distributions over functions representing the data instead of learning distributions over the data directly. Through experiments on both images and 3D shapes, we showed that our model learns rich function distributions that scale independently of the underlying data resolution. We therefore hope the strengths of our method, combined with further research into stabilizing training with point cloud discriminators, could lead to algorithms operating on scales larger than are possible with grid discriminators. Indeed, while grid-based generative models currently outperform continuous models, we believe scaling to very large data (particularly in 3D where densely discretizing becomes intractable at high resolutions), will eventually require abandoning discrete grids, and hope our work provides a step in this direction.

## 8 Acknowledgements

We thank William Zhang, Yuyang Shi, Jin Xu, Valentin De Bortoli, Jean-Francois Ton and Kaspar Märtens for providing feedback on an early version of the paper. We also thank Charline Le Lan, Jean-Francois Ton and Bobby He for helpful discussions. We thank Yann Dubois for providing the ConvNP samples as well as helpful discussions around neural processes. Emilien gratefully acknowledges his PhD funding from Google DeepMind.

## References

[1] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. *arXiv preprint arXiv:2011.13775*, 2020.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

[3] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2565–2574, 2020.

[4] Sam Bond-Taylor and Chris G Willcocks. Gradient origin networks. *arXiv preprint arXiv:2007.02798*, 2020.

[5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[6] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. *arXiv preprint arXiv:2012.00926*, 2020.

[7] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[8] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020.

[9] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019.

[10] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision*, pages 628–644. Springer, 2016.

[11] Emilien Dupont, Tuanfeng Zhang, Peter Tilke, Lin Liang, and William Bailey. Generating realistic geology conditioned on physical measurements with generative adversarial networks. *arXiv preprint arXiv:1802.03065*, 2018.

[12] Emilien Dupont, Miguel Bautista Martin, Alex Colburn, Aditya Sankar, Josh Susskind, and Qi Shan. Equivariant neural rendering. In *International Conference on Machine Learning*, pages 2761–2770. PMLR, 2020.

[13] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *International Conference on Machine Learning*, pages 3165–3176. PMLR, 2020.

[14] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2018.

[15] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[16] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[17] Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

[18] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

[19] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[20] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[21] David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. *International Conference on Learning Representations*, 2017.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

[23] Neal Jean, Marshall Burke, Michael Xie, W Matthew Davis, David B Lobell, and Stefano Ermon. Combining satellite imagery and machine learning to predict poverty. *Science*, 353 (6301):790–794, 2016.

[24] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019.

[26] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[27] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[28] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[29] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[31] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[32] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in Neural Information Processing Systems*, 31: 820–830, 2018.

[33] Chieh Hubert Lin, Chia-Che Chang, Yu-Sheng Chen, Da-Cheng Juan, Wei Wei, and Hwann-Tzong Chen. Coco-gan: Generation by parts via conditional coordinating. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4512–4521, 2019.

[34] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5239–5248, 2019.

[35] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[36] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Siggraph Computer Graphics*, 21(4):163–169, 1987.

[37] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine learning*, pages 3481–3490. PMLR, 2018.

[38] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[39] Lars Morten Mescheder. *Stability and Expressiveness of Deep Generative Models*. PhD thesis, Universität Tübingen, 2020.

[40] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020.

[41] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[42] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019.

[43] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. *arXiv preprint arXiv:2011.12100*, 2020.

[44] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5379–5389, 2019.

[45] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[46] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[48] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.

[49] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Eecognition*, pages 652–660, 2017.

[50] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[51] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[52] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, 2008.

[53] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

[54] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286. PMLR, 2014.

[55] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021.

[56] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems*, pages 2018–2028, 2017.

[57] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020.

[58] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.1.1.

[59] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, pages 1121–1132, 2019.

[60] Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.

[61] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. *arXiv preprint arXiv:2011.12026*, 2020.

[62] Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. Metnet: A neural weather model for precipitation forecasting. *arXiv preprint arXiv:2003.12140*, 2020.

[63] Karl Stelzner, Kristian Kersting, and Adam R Kosiorek. Generative adversarial set transformers. In *Workshop on Object-Oriented Learning at ICML*, volume 2020, 2020.

[64] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

[65] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

[66] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[67] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

[68] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018.

[69] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5589–5598, 2020.

[70] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. *arXiv preprint arXiv:2012.02190*, 2020.

[71] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[72] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5565–5573, 2019.

# A Experimental details

In this section we provide experimental details necessary to reproduce all results in the paper. All the models were implemented in PyTorch [47] and trained on a single 2080Ti GPU with 11GB of RAM.

## A.1 Single image experiment

To produce Figure 2, we trained a ReLU MLP with 2 hidden layers each with 256 units, using tanh as the final non-linearity. We trained for 1000 iterations with Adam using a learning rate of 1e-3. For the RFF encoding we set $m = 256$ and $\sigma = 10$.

## A.2 Distribution experiments

For all experiments (both images and 3D shapes), we parameterized $f_\theta$ by an MLP with 3 hidden layers, each with 128 units. We used a latent dimension of 64 and an MLP with 2 hidden layers of dimension 256 and 512 for the hypernetwork $g_\phi$. We normalized all coordinates to lie in $[-1, 1]^d$ and all features to lie in $[-1, 1]^k$. We used LeakyReLU non-linearities both in the NFD and discriminator. The final output of the function representation was followed by a tanh non-linearity.

For the point cloud discriminator, we used $3^d$ neighbors in each convolution layer and followed every convolution by an average pooling layer reducing the number of points by $2^d$. We applied a sigmoid as the final non-linearity. We used an MLP with 4 hidden layers each of size 16 to parameterize all weight MLPs. Unless stated otherwise, we use Adam with a learning rate of 1e-4 for the hypernetwork weights and 4e-4 for the discriminator weights with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ as is standard for GAN training. For each dataset, we trained for a large number of epochs and chose the best model by visual inspection.

### A.2.1 MNIST

- Dimensions: $d = 2$, $k = 1$
- Fourier features: $m = 128$, $\sigma = 1$
- Discriminator channels: $64, 128, 256$
- Batch size: 128
- Epochs: 150

### A.2.2 CelebAHQ 64x64

- Dimensions: $d = 2$, $k = 3$
- Fourier features: $m = 128$, $\sigma = 2$
- Discriminator channels: $64, 128, 256, 512$
- Batch size: 64
- Epochs: 300

### A.2.3 CelebAHQ 128x128

- Dimensions: $d = 2$, $k = 3$
- Fourier features: $m = 128$, $\sigma = 3$
- Discriminator channels: $64, 128, 256, 512, 1024$
- Batch size: 22
- Epochs: 70

### A.2.4 ShapeNet voxels

- Dimensions: $d = 3$, $k = 1$
- Fourier features: None
- Discriminator channels: $32, 64, 128, 256$

- Batch size: 24
- Learning rates: Generator 2e-5, Discriminator 8e-5
- Epochs: 200

#### A.2.5 ShapeNet point clouds

- Dimensions: $d = 3$, $k = 1$
- Fourier features: None
- Discriminator channels: $32, 64, 128, 256$
- Batch size: 24
- Learning rates: Generator 2e-5, Discriminator 8e-5
- Epochs: 200

### A.3 Things we tried that didn't work

- We initially let the function representation $f_\theta$ have 2 hidden layers of size 256, instead of 3 layers of size 128. However, we found that this did not work well, particularly for more complex datasets. We hypothesize that this is because the number of weights in a single $256 \to 256$ linear layer is $4\times$ the number of weights in a single $128 \to 128$ layer. As such, the number of weights in four $128 \to 128$ layers is the same as a single $256 \to 256$, even though such a 4-layer network would be much more expressive. Since the hypernetwork needs to output all the weights of the function representation, the final layer of the hypernetwork will be extremely large if the number of function weights is large. It is therefore important to make the network as expressive as possible with as few weights as possible, i.e. by making the network thinner and deeper.
- As the paper introducing the $R_1$ penalty [37] does not use batchnorm [22] in the discriminator, we initially ran experiments without using batchnorm. However, we found that using batchnorm both in the weight MLPs and between PointConv layers was crucial for stable training. We hypothesize that this is because using standard initializations for the weights of PointConv layers would result in PointConv outputs (which correspond to the weights in regular convolutions) that are large. Adding batchnorm fixed this initialization issue and resulted in stable training.
- In the PointConv paper, it was shown that the number of hidden layers in the weight MLPs does not significantly affect classification performance [66]. We therefore initially experimented with single hidden layer MLPs for the weights. However, we found that it is crucial to use deep networks for the weight MLPs in order to build discriminators that are expressive enough for the datasets we consider.
- We experimented with learning the frequencies of the Fourier features (i.e. learning $B$) but found that this did not significantly boost performance and generally resulted in slower training.

### A.4 Shapenet point clouds data

We used the point clouds from the ShapeNet chairs category produced by [42] to train our model for the point cloud experiments. These point clouds are given as lists of $(x, y, z)$ coordinates lying in $[-0.5, 0.5]^3$. However, in order to train our model we require both samples from within the point cloud and samples from outside the point cloud. The function representation is then trained to map $(x, y, z)$ to 1 for a point within the object and 0 for a point outside the object. To generate samples outside the point cloud, we used the following simple algorithm:

1. Uniformly sample points within $[-0.5, 0.5]^3$.
2. For each sampled point, find its nearest neighbor in point cloud.
3. Let $\delta$ be the distance to the nearest neighbor, if $a < \delta < b$ for some constants $a, b$ accept point, otherwise reject it.

The constant $a$ should then be large enough so that we don't sample points within the point cloud and $b$ should be small enough that we don't sample points too far away from the point cloud. Intuitively, this allows us to sample points around the boundary of the object, which we can then use as empty space samples to train our model. We set $a = 0.04$ and $b = 0.12$ in our experiments. Each point cloud then consists of 3000 interior points (taken from [42]) and 3000 exterior points (sampled using our method). There are a total of 3746 point clouds in the dataset.

### A.5 Quantitative experiments

We computed FID scores using the `pytorch-fid` library [58]. We generated 30k samples for both CelebAHQ $64 \times 64$ and $128 \times 128$ and used default settings for all hyperparameters. We note that the FID scores for the convolutional baselines in the main paper were computed on CelebA (not the HQ version) and are therefore not directly comparable with our model. However, convolutional GANs would also outperform our model on CelebAHQ.

## B Additional related work

Due to space constraints, we provide more detailed descriptions of related work in this section, focusing on distributions of implicit representations. Gradient Origin Networks [4] model distributions of implicit representations using an encoder free model (similarly to auto-decoders), instead using gradients of the latents as an encoder. In addition, several recent (and independent) works have tackled the problem of learning distributions of NeRF [40] scenes, which are special cases of implicit representations. This includes GRAF [57] which concatenates a latent vector to an implicit representation and trains the model adversarially using a patch-based convolutional discriminator, GIRAFFE [43] which adds compositionality to the generator and pi-GAN [6] which models the generator using modulations to the hidden layer activations. We note that GRAF in particular introduces an interesting multi-scale patch-based discriminator that does not need to align with the pixel grid. However, the patches are, unlike our method, still restricted to be evenly sampled in a square and are implemented only in 2D. While it would be possible to generalize this to 3D, this would still suffer from poor scaling as the entire 3D patch would need to be densely sampled. Concurrently to our work, [1] and [61] learn distributions of implicit representations for the special case of images, using standard convolutional discriminators. All these works employ standard convolutional grid discriminators that act on images and as such scale directly with the number of pixels. While the focus of most of these works is computer vision, we note that implicit representations have also been used beyond vision tasks, for example for analysing sequential data [55] or modeling audio [60].

Finally, we note that there is an extensive literature on improving training of GANs. While we strived for simplicity when designing our model, we hypothesize that approaches such as progressive growing [26], using specialized generator architectures [27] and losses [2], truncating the prior distribution [5] and using residual connections in the discriminator could help improve sample quality. It would also be interesting to incorporate symmetries into our model [13], particularly for the 3D scene experiments [12]. Indeed, the realism of a point cloud does not depend on the pose in which it is observed and as such the discriminator should be rotation and translation invariant.

## C Models that are not suitable for learning function distributions

### C.1 Auto-decoders

We briefly introduce auto-decoder models following the setup in [46] and describe why they are not suitable as generative models. As in the NFD case, we assume we are given a dataset of $N$ samples $\{\mathbf{s}^{(i)}\}_{i=1}^N$ (where each sample $\mathbf{s}^{(i)}$ is a set). We then associate a latent vector $\mathbf{z}^{(i)}$ with each sample $\mathbf{s}^{(i)}$. We further parameterize a probabilistic model $p_\theta(\mathbf{s}^{(i)}|\mathbf{z}^{(i)})$ (similar to the decoder in variational autoencoders) by a neural network with learnable parameters $\theta$ (typically returning the mean of a Gaussian with fixed variance). The optimal parameters are then estimated as

$$\arg\max_{\theta, \{\mathbf{z}^{(i)}\}} \sum_{i=1}^{N} \log p_\theta(\mathbf{s}^{(i)}|\mathbf{z}^{(i)}) + \log p(\mathbf{z}^{(i)}),$$

Figure 13: Left: Samples from an auto-decoder model trained on MNIST. Right: Samples from an auto-decoder model trained on CelebAHQ $32 \times 32$.

where $p(\mathbf{z})$ is a (typically Gaussian) prior over the $\mathbf{z}^{(i)}$'s. Crucially the latents vectors $\mathbf{z}^{(i)}$ are themselves learnable and optimized. However, maximizing $\log p(\mathbf{z}^{(i)}) \propto -\|\mathbf{z}^{(i)}\|^2$ does not encourage the $\mathbf{z}^{(i)}$'s to be distributed according to the prior, but only encourages them to have a small norm. Note that this is because we are optimizing the *samples* and not the *parameters* of the Gaussian prior. As such, after training, the $\mathbf{z}^{(i)}$'s are unlikely to be distributed according to the prior. Sampling from the prior to generate new samples from the model will therefore not work.

We hypothesize that this is why the prior is required to have very low variance for the auto-decoder model to work well [46]. Indeed, if the norm of the $\mathbf{z}^{(i)}$'s is so small that they are barely changed during training, they will remain close to their initial Gaussian distribution. While this trick is sufficient to learn distributions of simple datasets such as MNIST, we were unable to obtain good results on more complex and high frequency datasets such as CelebAHQ. Results of our best models are shown in Figure 13.

We also note that auto-decoders were not necessarily built to act as generative models. Auto-decoders have for example excelled at embedding 3D shape data into a latent space [46] and learning distributions over 3D scenes for inverse rendering [59]. Our analysis therefore does not detract from the usefulness of auto-decoders, but instead shows that auto-decoders may not be suitable for the task of generative modeling.

### C.2 Set discriminators

In this section, we analyse the use of set discriminators for learning NFDs. Given a datapoint $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ represented as a set, we build a permutation invariant set discriminator as a PointNet/DeepSet [49, 71] function

$$D(\mathbf{s}) = \rho\left(\frac{1}{\sqrt{n}} \sum_{i=1}^{n} \varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i))\right),$$

where $\rho$ and $\varphi$ are both MLPs and $\gamma_x$ and $\gamma_y$ are RFF encodings for the coordinates and features respectively. Recall that the RFF encoding function $\gamma$ is defined as

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi B\mathbf{x}) \\ \sin(2\pi B\mathbf{x}) \end{pmatrix},$$

where $B$ is a (potentially learnable) random matrix of frequencies. While the RFF encodings are not strictly necessary, we were unable to learn high frequency functions without them. Note also that we normalize the sum over set elements by $\sqrt{n}$ instead of $n$ as is typical - as shown in Section C.3.1 this is to make the Lipschitz constant of the set discriminator independent of $n$.

We experimented extensively with such models, varying architectures and encoding hyperparameters (including not using an encoding) but were unable to get satisfactory results on CelebAHQ, even at a resolution of $32 \times 32$. Our best results are shown in Figure 14. As can be seen, the model is able to generate plausible samples for MNIST but fails on CelebAHQ.

While PointNet/DeepSet functions are universal approximators of set functions [71], they do not explicitly model set element interactions. As such, we also experimented with Set Transformers [31] which model interactions using self-attention. However, we found that using such architectures did

Figure 14: Left: Samples from a set discriminator model trained on MNIST. Right: Samples from a set discriminator model trained on CelebAHQ $32 \times 32$.

not improve performance. As mentioned in the main paper, we therefore conjecture that explicitly taking into account the metric on the coordinate space (as is done in PointConv) is crucial for learning complex neural distributions. We note that Set Transformers have also been used as a discriminator to model sets [63], although this was only done for small scale datasets.

In addition to our experimental results, we also provide some theoretical evidence that set discriminators may be ill-suited for generative modeling of functions. Specifically, we show that the Lipschitz constant of set discriminators and RFF encodings are typically very large.

### C.3 The Lipschitz constant of set discriminators

Several works have shown that limiting the Lipschitz constant (or equivalently the largest gradient norm) of the discriminator is important for stable GAN training [2, 19, 56, 41, 37]. This is typically achieved either by penalizing the gradient norm or by explicitly constraining the Lipschitz constant of each layer in the discriminator. Intuitively, this ensures that the gradients of the discriminator with respect to its input do not grow too large and hence that gradients with respect to the weights of the generator do not grow too large either (which can lead to unstable training). In the following subsections, we show that the Lipschitz constant of set discriminators and specifically the Lipschitz constant of RFF encodings are large in most realistic settings.

#### C.3.1 Architecture

**Proposition 1.** *The Lipschitz constant of the set discriminator $D$ is bounded by*

$$Lip(D) \leq Lip(\rho)Lip(\varphi)\sqrt{Lip(\gamma_x)^2 + Lip(\gamma_y)^2}$$

See Section D for a proof. In the case where the RFF encoding is fixed, imposing gradient penalties on $D$ would therefore reduce the Lipschitz constant of $\rho$ and $\varphi$ but not of $\gamma_x$ and $\gamma_y$. If the RFF encoding is learned, its Lipschitz constant could also be penalized. However, as shown in [64], learning high frequency functions typically requires large frequencies in the matrix $B$. We show in the following section that the Lipschitz constant of $\gamma$ is directly proportional to the spectral norm of $B$.

#### C.3.2 Lipschitz constant of random Fourier features

**Proposition 2.** *The Lipschitz constant of $\gamma(\mathbf{x})$ is bounded by*

$$Lip(\gamma) \leq \sqrt{8}\pi\|B\|$$

See Section D for a proof. There is therefore a fundamental tradeoff between how much high frequency detail the discriminator can learn (requiring a large Lipschitz constant) and its training stability (requiring a low Lipschitz constant). In practice, for the settings we used in this paper, the spectral norm of $B$ is on the order of 100s, which is too large for stable GAN training.

# D Proofs

## D.1 Prerequisites

We denote by $\|\cdot\|_2$ the $\ell_2$ norm for vectors and by $\|\cdot\|$ the spectral norm for matrices (i.e. the matrix norm induced by the $\ell_2$ norm). The spectral norm is defined as

$$\|A\| = \sup_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2 = \sigma_{\max}(A) = \sqrt{\lambda_{\max}(A^T A)}$$

where $\sigma_{\max}$ denotes the largest singular value and $\lambda_{\max}$ the largest eigenvalue.

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Lipschitz constant $\text{Lip}(f)$ (if it exists) is defined as the largest value $L$ such that

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\|_2 \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|_2$$

for all $\mathbf{x}_1, \mathbf{x}_2$. The Lipschitz constant is equivalently defined for differentiable functions as

$$\text{Lip}(f) = \sup_{\mathbf{x}} \|\nabla f(\mathbf{x})\|.$$

Note that when composing two functions $f$ and $g$ we have

$$\text{Lip}(f \circ g) \leq \text{Lip}(f)\text{Lip}(g).$$

We will also make use of the following lemmas.

### D.1.1 Spectral norm of concatenation

**Lemma 1.** *Let $A \in \mathbb{R}^{n\times d}$ and $B \in \mathbb{R}^{m\times d}$ be two matrices and denote by $\begin{pmatrix} A \\ B \end{pmatrix}$ their rowwise concatenation. Then we have the following inequality in the spectral norm*

$$\left\| \begin{pmatrix} A \\ B \end{pmatrix} \right\| \leq \sqrt{\|A\|^2 + \|B\|^2}.$$

*Proof.*[1]

$$\begin{aligned} \left\| \begin{pmatrix} A \\ B \end{pmatrix} \right\|^2 &= \lambda_{\max}\left( \begin{pmatrix} A \\ B \end{pmatrix}^T \begin{pmatrix} A \\ B \end{pmatrix} \right) \\ &= \lambda_{\max}(A^T A + B^T B) \\ &\leq \lambda_{\max}(A^T A) + \lambda_{\max}(B^T B) \\ &= \|A\|^2 + \|B\|^2, \end{aligned}$$

where we used the definition of the spectral norm in the first line and Weyl's inequality for symmetric matrices in the third line.

### D.1.2 Inequality for $\ell_1$ and $\ell_2$ norm

**Lemma 2.** *Let $\mathbf{x}_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$. Then*

$$\sum_{i=1}^n \|\mathbf{x}_i\|_2 \leq \sqrt{n}\|(\mathbf{x}_1, \ldots, \mathbf{x}_n)\|_2.$$

*Proof.*

$$\begin{aligned} \sum_{i=1}^n \|\mathbf{x}_i\|_2 &= \sum_{i=1}^n \|\mathbf{x}_i\|_2 \cdot 1 \\ &\leq \left( \sum_{i=1}^n \|\mathbf{x}_i\|_2^2 \right)^{\frac{1}{2}} \left( \sum_{i=1}^n 1^2 \right)^{\frac{1}{2}} \\ &= \sqrt{n}\|(\mathbf{x}_1, \ldots, \mathbf{x}_n)\|_2, \end{aligned}$$

[1] This proof was inspired by `https://math.stackexchange.com/questions/2006773/spectral-norm-of-concatenation-of-two-matrices`

where we used Cauchy-Schwarz in the second line. Note that this is an extension of the well-known inequality $\|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2$ to the case where each component of the vector $\mathbf{x}$ is the $\ell_2$ norm of another vector.

#### D.1.3 Lipschitz constant of sum of identical functions

**Lemma 3.** *Let $\mathbf{x}_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$ and let $f$ be a function with Lipschitz constant Lip$(f)$. Define $g(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{i=1}^n f(\mathbf{x}_i)$. Then*

$$\textit{Lip}(g) \leq \sqrt{n}\textit{Lip}(f).$$

*Proof.*

$$\begin{aligned}
\|g(\mathbf{x}_1, \ldots, \mathbf{x}_n) - g(\mathbf{y}_1, ..., \mathbf{y}_n)\|_2 &= \left\| \sum_{i=1}^n (f(\mathbf{x}_i) - f(\mathbf{y}_i)) \right\|_2 \\
&\leq \sum_{i=1}^n \|f(\mathbf{x}_i) - f(\mathbf{y}_i)\|_2 \\
&\leq \text{Lip}(f) \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{y}_i\|_2 \\
&\leq \sqrt{n}\text{Lip}(f) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} - \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_n \end{pmatrix} \right\|_2 .
\end{aligned}$$

Where we used the triangle inequality for norms in the second line, the definition of Lipschitz constants in the second line and Lemma 2 in the third line.

#### D.1.4 Lipschitz constant of concatenation

**Lemma 4.** *Let $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^p \to \mathbb{R}^q$ be functions with Lipschitz constant Lip$(g)$ and Lip$(h)$ respectively. Define $f : \mathbb{R}^{n+p} \to \mathbb{R}^{m+q}$ as the concatenation of $g$ and $h$, that is $f(\mathbf{x}, \mathbf{y}) = (g(\mathbf{x}), h(\mathbf{y}))$. Then*

$$\textit{Lip}(f) \leq \sqrt{\textit{Lip}(g)^2 + \textit{Lip}(h)^2}.$$

*Proof.*

$$\begin{aligned}
\|f(\mathbf{x}_1, \mathbf{y}_1) - f(\mathbf{x}_2, \mathbf{y}_2)\|_2^2 &= \left\| \begin{pmatrix} g(\mathbf{x}_1) - g(\mathbf{x}_2) \\ h(\mathbf{y}_1) - h(\mathbf{y}_2) \end{pmatrix} \right\|_2^2 \\
&= \|g(\mathbf{x}_1) - g(\mathbf{x}_2)\|_2^2 + \|h(\mathbf{y}_1) - h(\mathbf{y}_2)\|_2^2 \\
&\leq \text{Lip}(g)^2 \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 + \text{Lip}(h)^2 \|\mathbf{y}_1 - \mathbf{y}_2\|_2^2 \\
&\leq \text{Lip}(g)^2 (\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 + \|\mathbf{y}_1 - \mathbf{y}_2\|_2^2) + \text{Lip}(h)^2 (\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 + \|\mathbf{y}_1 - \mathbf{y}_2\|_2^2) \\
&= (\text{Lip}(g)^2 + \text{Lip}(h)^2) \left\| \begin{pmatrix} \mathbf{x}_1 - \mathbf{x}_2 \\ \mathbf{y}_1 - \mathbf{y}_2 \end{pmatrix} \right\|_2^2
\end{aligned}$$

where we used the definition of the $\ell_2$ norm in the second and last line.

### D.2 Lipschitz constant of Fourier feature encoding

We define the random Fourier feature encoding $\gamma : \mathbb{R}^d \to \mathbb{R}^{2m}$ as

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi B\mathbf{x}) \\ \sin(2\pi B\mathbf{x}) \end{pmatrix}$$

where $B \in \mathbb{R}^{m \times d}$.

**Proposition 3.** *The Lipschitz constant of $\gamma(\mathbf{x})$ is bounded by*

$$\mathit{Lip}(\gamma) \leq \sqrt{8}\pi\|B\|.$$

*Proof.* Define $\mathbf{u}(\mathbf{x}) = \cos(2\pi B\mathbf{x})$ and $\mathbf{v}(\mathbf{x}) = \sin(2\pi B\mathbf{x})$. By definition of the Lipschitz constant and applying Lemma 1 we have

$$\begin{aligned}
\text{Lip}(\gamma) &= \sup_{\mathbf{x}} \|\nabla\gamma(\mathbf{x})\| \\
&= \sup_{\mathbf{x}} \left\| \begin{pmatrix} \nabla\cos(2\pi B\mathbf{x}) \\ \nabla\sin(2\pi B\mathbf{x}) \end{pmatrix} \right\| \\
&= \sup_{\mathbf{x}} \left\| \begin{pmatrix} \nabla\mathbf{u}(\mathbf{x}) \\ \nabla\mathbf{v}(\mathbf{x}) \end{pmatrix} \right\| \\
&\leq \sup_{\mathbf{x}} \sqrt{\|\nabla\mathbf{u}(\mathbf{x})\|^2 + \|\nabla\mathbf{v}(\mathbf{x})\|^2} \\
&\leq \sqrt{\sup_{\mathbf{x}} \|\nabla\mathbf{u}(\mathbf{x})\|^2 + \sup_{\mathbf{x}} \|\nabla\mathbf{v}(\mathbf{x})\|^2}.
\end{aligned}$$

The derivative of $\mathbf{u}$ is given by

$$\begin{aligned}
(\nabla\mathbf{u}(\mathbf{x}))_{ij} &= \frac{\partial u_i(\mathbf{x})}{\partial x_j} \\
&= \frac{\partial}{\partial x_j}\cos(2\pi\mathbf{b}_i^T\mathbf{x}) \\
&= -2\pi b_{ij}\sin(2\pi\mathbf{b}_i^T\mathbf{x}) \\
&= -2\pi b_{ij} v_i(\mathbf{x}),
\end{aligned}$$

where $\mathbf{b}_i$ corresponds to the ith row of $B$. We can write this more compactly as $\nabla\mathbf{u}(\mathbf{x}) = -2\pi\text{diag}(\mathbf{v}(\mathbf{x}))B$. A similar calculation for $\mathbf{v}(\mathbf{x})$ shows that $\nabla\mathbf{v}(\mathbf{x}) = 2\pi\text{diag}(\mathbf{u}(\mathbf{x}))B$.

All that remains is then to calculate the norms $\|\nabla\mathbf{u}(\mathbf{x})\|$ and $\|\nabla\mathbf{v}(\mathbf{x})\|$. Using submultiplicativity of the spectral norm we have

$$\begin{aligned}
\sup_{\mathbf{x}} \|\nabla\mathbf{u}(\mathbf{x})\| &= \sup_{\mathbf{x}} 2\pi\|\text{diag}(\mathbf{v}(\mathbf{x}))B\| \\
&\leq \sup_{\mathbf{x}} 2\pi\|\text{diag}(\mathbf{v}(\mathbf{x}))\|\|B\| \\
&= 2\pi\|B\|,
\end{aligned}$$

where we used the fact that the spectral norm of diagonal matrix is equal to its largest entry and that $|v_i(\mathbf{x})| \leq 1$ for all $i$. Similar reasoning gives $\sup_{\mathbf{x}} \|\nabla\mathbf{u}(\mathbf{x})\| = 2\pi\|B\|$. Finally we obtain

$$\begin{aligned}
\text{Lip}(\gamma) &\leq \sqrt{\sup_{\mathbf{x}} \|\nabla\mathbf{u}(\mathbf{x})\|^2 + \sup_{\mathbf{x}} \|\nabla\mathbf{v}(\mathbf{x})\|^2} \\
&\leq \sqrt{(2\pi\|B\|)^2 + (2\pi\|B\|)^2} \\
&= \sqrt{8}\pi\|B\|.
\end{aligned}$$

### D.3 Lipschitz constant of set discriminator

The set discriminator $D : \mathbb{R}^{n\times(d+k)} \to [0,1]$ is defined by

$$D(\mathbf{s}) = \rho\left(\frac{1}{\sqrt{n}}\sum_{i=1}^{n}\varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i))\right),$$

where $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n \in \mathbb{R}^{n\times(d+k)}$ is treated as a fixed vector and each $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in \mathbb{R}^k$. The Fourier feature encodings for $\mathbf{x}_i$ and $\mathbf{y}_i$ are given by functions $\gamma_x : \mathbb{R}^d \to \mathbb{R}^{2m_x}$ and $\gamma_y : \mathbb{R}^k \to \mathbb{R}^{2m_y}$ respectively. The function $\varphi : \mathbb{R}^{2(m_x+m_y)} \to \mathbb{R}^p$ maps coordinates and features to an encoding of dimension $p$. Finally $\rho : \mathbb{R}^p \to [0, 1]$ maps the encoding to the probability of the sample being real.

**Proposition 4.** *The Lipschitz constant of the set discriminator $D$ is bounded by*

$$\mathit{Lip}(D) \leq \mathit{Lip}(\rho)\mathit{Lip}(\varphi)\sqrt{\mathit{Lip}(\gamma_x)^2 + \mathit{Lip}(\gamma_y)^2}.$$

*Proof.* Write

$$\begin{aligned} D(\mathbf{s}) &= \rho\left(\frac{1}{\sqrt{n}}\sum_{i=1}^n \varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i))\right) \\ &= \rho(\eta(\mathbf{s})) \end{aligned}$$

where $\eta(\mathbf{s}) = \frac{1}{\sqrt{n}}\sum_{i=1}^n \varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i))$. Then we have

$$\mathrm{Lip}(D) \leq \mathrm{Lip}(\rho)\mathrm{Lip}(\eta).$$

We can further write

$$\begin{aligned} \eta(\mathbf{s}) &= \frac{1}{\sqrt{n}}\sum_{i=1}^n \varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i)) \\ &= \frac{1}{\sqrt{n}}\sum_{i=1}^n \theta(\mathbf{s}_i), \end{aligned}$$

where $\mathbf{s}_i = (\mathbf{x}_i, \mathbf{y}_i)$ and $\theta(\mathbf{s}_i) = \varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i))$. By Lemma 3 we have

$$\mathrm{Lip}(\eta) \leq \frac{1}{\sqrt{n}}\sqrt{n}\mathrm{Lip}(\theta) = \mathrm{Lip}(\theta).$$

We can then write

$$\begin{aligned} \theta(\mathbf{s}_i) &= \varphi(\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i)) \\ &= \varphi(\psi(\mathbf{s}_i)) \end{aligned}$$

where $\psi(\mathbf{s}_i) = (\gamma_x(\mathbf{x}_i), \gamma_y(\mathbf{y}_i))$. We then have, using Lemma 4

$$\mathrm{Lip}(\theta) \leq \mathrm{Lip}(\varphi)\mathrm{Lip}(\psi) \leq \mathrm{Lip}(\varphi)\sqrt{\mathrm{Lip}(\gamma_x)^2 + \mathrm{Lip}(\gamma_y)^2}.$$

Putting everything together we finally obtain

$$\mathrm{Lip}(D) \leq \mathrm{Lip}(\rho)\mathrm{Lip}(\varphi)\sqrt{\mathrm{Lip}(\gamma_x) + \mathrm{Lip}(\gamma_y)}.$$

## E Failure examples

Figure 15: Left: Samples from model trained on CelebAHQ $64 \times 64$ using $K = 2048$ pixels (50%). Right: Samples from model trained using $K = 3072$ pixels (75%).

Figure 16: Selected samples highlighting failure modes of our model, including generation of unrealistic and incoherent samples.

## F Additional results

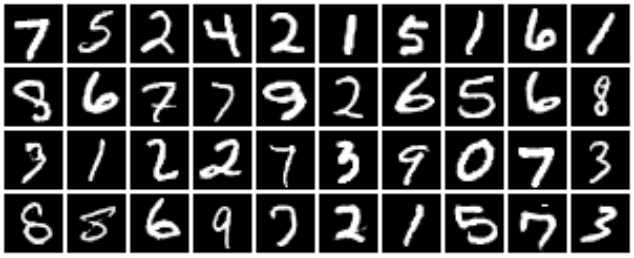

Figure 17: Additional MNIST samples.

Figure 18: Additional CelebAHQ $64 \times 64$ samples.

Figure 19: Additional CelebAHQ $128 \times 128$ samples.

Figure 20: Additional superresolution samples. Left column shows superresolution from $64 \times 64 \rightarrow 256 \times 256$ and right column shows superresolution from $64 \times 64 \rightarrow 512 \times 512$

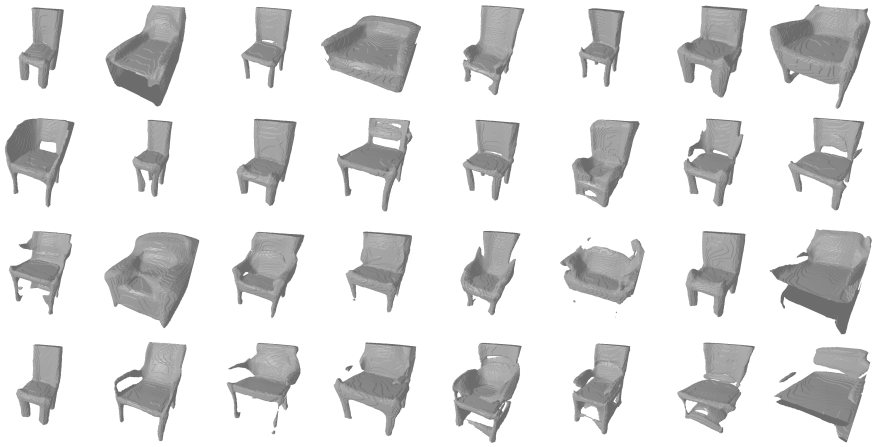

Figure 21: Additional Shapenet chairs samples.