From data to functa: Your data point is a function and you can treat it like one

Emilien Dupont [*1] **Hyunjik Kim** [*2] **S. M. Ali Eslami** [2] **Danilo Rezende** [2] **Dan Rosenbaum** [3 2]

Abstract

It is common practice in deep learning to represent a measurement of the world on a discrete grid, e.g. a 2D grid of pixels. However, the underlying signal represented by these measurements is often continuous, e.g. the scene depicted in an image. A powerful continuous alternative is then to represent these measurements using an *implicit neural representation*, a neural function trained to output the appropriate measurement value for any input spatial location. In this paper, we take this idea to its next level: what would it take to perform deep learning on these functions instead, treating them as data? In this context we refer to the data as *functa*, and propose a framework for deep learning on functa. This view presents a number of challenges around efficient conversion from data to functa, compact representation of functa, and effectively solving downstream tasks on functa. We outline a recipe to overcome these challenges and apply it to a wide range of data modalities including images, 3D shapes, neural radiance fields (NeRF) and data on manifolds. We demonstrate that this approach has various compelling properties across data modalities, in particular on the canonical tasks of generative modeling, data imputation, novel view synthesis and classification.

1. Introduction

In deep learning, data is traditionally represented by arrays. For example, images are represented by their pixel intensities, and 3D shapes by voxel occupancies, both at a discrete set of grid coordinates tied to a particular resolution. However, the underlying signal represented by these arrays is often continuous. It is therefore natural to consider representing such data with continuous quantities.

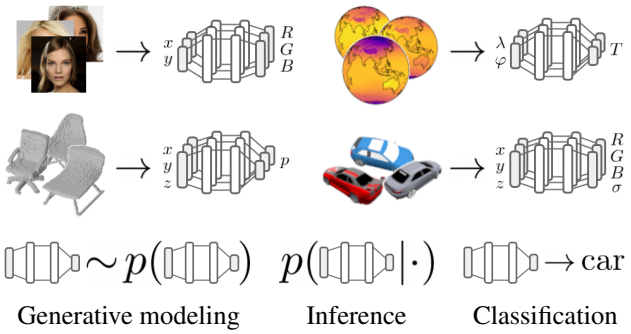

Figure 1. We convert array data into functional data parameterized by neural networks, termed *functa*, and treat these as data points for various downstream machine learning tasks.

Recently, the idea of modelling data with continuous functions has gained popularity. An image, for example, can be represented by a continuous function mapping 2D pixel coordinates to RGB values. When such a function is parameterized by a neural network, it is typically referred to as an *implicit neural representation* (INR). INRs are generally applicable to a wide range of modalities – indeed, various works have demonstrated that INRs can be used to represent images (Stanley, 2007; Ha, 2016), 3D shapes (Mescheder et al., 2019; Chen & Zhang, 2019), signed distance functions (Park et al., 2019), videos (Li et al., 2021), 3D scenes (Mildenhall et al., 2020), audio (Sitzmann et al., 2020b) and data on manifolds (Dupont et al., 2021b). This functional representation offers a number of advantages over array representations. It allows for dealing with data at arbitrary resolutions, as well as data that is difficult to discretize such as neural radiance fields (NeRF) for 3D scene representation (Mildenhall et al., 2020). Parameterizing such functions as neural networks offers additional advantages, in terms of memory-efficiency and as a single architecture that can represent different data modalities.

In light of these advantages, we propose a new framework that 1. converts array data to functional data parameterized by neural networks, and 2. performs deep learning tasks directly on these functions. The first step involves taking

[*]Equal contribution: author order determined by coin flip.
[1]University of Oxford [2]DeepMind [3]University of Haifa.
Correspondence to: Emilien Dupont <dupont@stats.ox.ac.uk>, Hyunjik Kim <hyunjikk@google.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

a dataset of a given modality and fitting an INR to each datapoint. We refer to these functions as *functa*, a concise term for *INRs that are to be thought of as data*. In the second step, we treat functa as data points that we use for deep learning tasks (see Figure 1). A key difference to prior work on multimodal learning of functions (Dupont et al., 2021b; Du et al., 2021) is that we decouple the creation of datasets of functa in the first step and the deep learning task in the second step (e.g. generative modeling, inference, classification). Functions are then treated as data rather than part of the model for solving the task at hand.

While this framework inherits the advantages of INRs, we are also presented with new challenges. Which parameterization of functa do we choose? How do we efficiently create large datasets of functa, or functasets? How can we use functa as inputs to neural networks for downstream deep learning tasks? We are therefore required to lay the groundwork for methodology that is appropriate for working with data points as functions. In this paper we propose an instantiation of this framework, in particular a unified method for efficiently creating large functasets for a wide range of data modalities, including images, voxels, NeRF scenes and data on manifolds. We choose a specific parameterization of functa, called *modulations*, that can be fed into neural architectures to solve various downstream tasks including generative modeling, data imputation, novel view synthesis and classification. While our approach does not yet outperform conventional deep learning on arrays, our results show that the functional view has several desirable properties and is a promising alternative to traditional data representation.

2. Functa: Data points as INRs

We first review INRs then discuss the advantages of using them as data points (functa), as well as the advantages of decoupling the conversion of data to functa and the downstream deep learning task. INRs are functions $f_\theta : \mathcal{X} \to \mathcal{F}$ mapping *coordinates* $\mathbf{x} \in \mathcal{X}$ (e.g. pixel locations) to *features* $\mathbf{f} \in \mathcal{F}$ (e.g. RGB values) with parameters θ. Given a data point as a collection of coordinates $\{\mathbf{x}_i\}_{i\in\mathcal{I}}$ and features $\{\mathbf{f}_i\}_{i\in\mathcal{I}}$ (where $\mathcal{I}$ is an index set corresponding to e.g. all pixel locations in an image), INRs are fitted by minimizing mean squared error over all coordinate locations:

$$\min_\theta \mathcal{L}(f_\theta, \{\mathbf{x}_i, \mathbf{f}_i\}_{i\in\mathcal{I}}) = \min_\theta \sum_{i\in\mathcal{I}} \|f_\theta(\mathbf{x}_i) - \mathbf{f}_i\|_2^2. \quad (1)$$

Hence each f_θ corresponds to e.g. a single image. Typically, f_θ is parameterized by a feedforward neural network (MLP) with positional encodings (Mildenhall et al., 2020; Tancik et al., 2020) or sinusoidal activation functions (Sitzmann et al., 2020b) that allow fitting of high frequency signals.

When considered as elements of a dataset, rather than a model, we refer to INRs as *functa*. Training deep learning models on functasets rather than conventional array datasets has various compelling properties that we describe below.

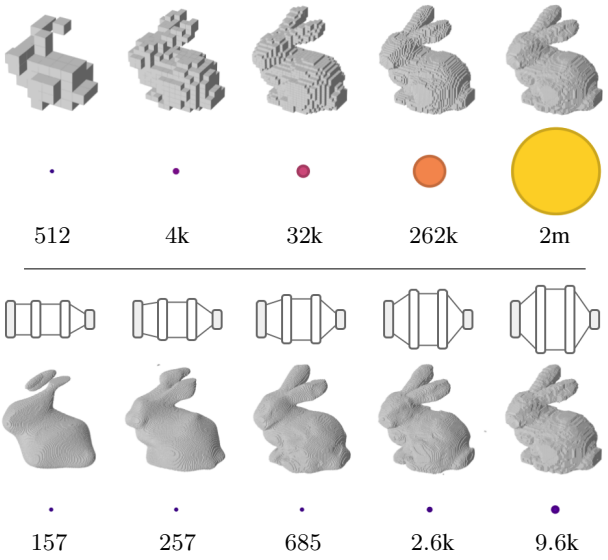

Figure 2. Functa scale much more gracefully with resolution than array representations. Circle area reflects the numerical size of the array (top) / function (bottom). See Appendix A.9 for details.

Scaling. Array based representations typically scale poorly with resolution (e.g. voxel grids scale cubically with resolution). Processing such high resolution data with neural networks is both memory and compute intensive, often becoming a bottleneck (Park et al., 2019; Mescheder et al., 2019). In contrast, functa usually scale much more gracefully with resolution (Figure 2), leading to more efficient training of neural networks for downstream tasks.

Moving away from fixed resolution. Array representations of data are typically stored at a fixed resolution. However, most data in the wild do not follow this form: images come in various shapes and resolutions; data modalities such as lidar are stored as point clouds, 3D shapes as irregular meshes, etc. On the other hand, we can easily convert data at a variety of resolutions to functa, allowing the downstream model to handle data at a range of resolutions.

Signals that are inherently difficult to discretize. While visual signals are continuous, they are easily discretized onto a grid to generate digital images. However, this is not true for many other data modalities which are inherently difficult to discretize. For example, neural radiance fields (NeRF) use volumetric rendering which requires the field to be evaluated at arbitrary spatial locations (Mildenhall et al., 2020) and physical fields often require continuous derivatives to solve differential equations of motion, both of which are incompatible with discrete representations. In addition, it is non-trivial to choose a grid for discretizing data lying on manifolds. Functa provide a natural way to

express data continuously and as such bypass the need for discretization when training models on downstream tasks.

Multimodality. It is standard practice in deep learning to use specialized encoder and decoder architectures for different data modalities. However designing encoders for NeRF scenes, for example, is challenging and requires aggregating a collection of images and poses (Kosiorek et al., 2021). Similarly, data lying on manifolds require highly specialized architectures (Cohen et al., 2018). In contrast, functa can be used to encode and decode a wide variety of data modalities through a generic optimization procedure.

Easing downstream task. The decoupling of 1. creating functa and 2. training models on functa for a downstream task can greatly simplify the task. For example, consider the task of generative modelling on NeRF scenes. Fitting NeRF scenes requires inferring 3D structure from a set of posed 2D images, itself a non-trivial task. Without the above decoupling, learning a generative model of NeRF scenes then requires simultaneously learning a distribution over 3D scenes while inferring 3D structure from 2D images. In our framework, we first infer the 3D scene from 2D images by minimising reconstruction error on views - a relatively easier task - and only after we obtain a functaset of 3D scenes, we model their distribution.

3. Functasets: Datasets of INRs

We now describe how to 1. represent functa suitably so that they can be fed into neural networks for downstream tasks and 2. create large datasets of functa in a scalable manner.

3.1. Functa as MLP modulations

We use SIREN (Sitzmann et al., 2020b) as the base architecture for INRs throughout, since it is known to efficiently represent a wide range of data modalities (*cf.* Appendix A.2 for mathematical formulation of SIREN). The naïve approach for representing functa is to take the parameter vector of the SIREN. However, as SIREN is an MLP, it can have a large number of parameters despite the favourable scaling characteristics compared to array representations, making this representation suboptimal for feeding into neural networks for downstream tasks. Various works have explored *modulations* as an alternative that uses a shared base network across data points to model common structure, with modulations modeling the variation specific to each data point (Perez et al., 2018; Chan et al., 2021b; Mehta et al., 2021). We therefore work with modulations rather than parameters, which are typically much more low dimensional.

Modulations are usually represented as elementwise affine transformations (shift and scale) applied to the activations of the neural network (Perez et al., 2018; Chan et al., 2021b; Mehta et al., 2021). However, we found experimentally

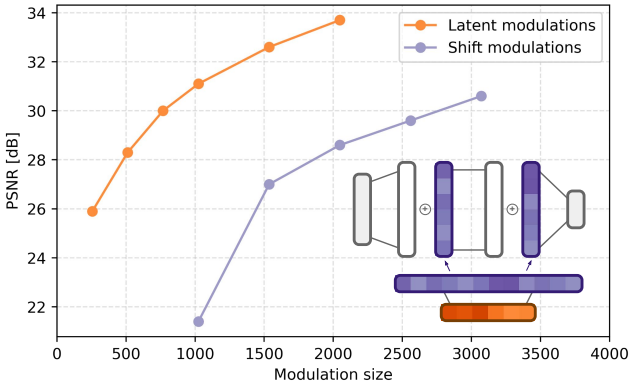

Figure 3. Reconstruction accuracy (in PSNR) vs modulation dimensionality on CelebA-HQ 64×64. Reconstruction accuracy is computed from the MSE between the image array and its INR evaluated at each pixel location. The model architecture is shown on the bottom right, with purple vectors corresponding to shift modulations and orange vectors to latent modulations.

that using shifts only performs just as well as using both shifts & scales, with half the representation size. We provide the mathematical formulation of *shift modulations* in Appendix A.2, along with intuition for the role of shift modulations via perturbation analysis in Appendix B.1.

We can further reduce the number of modulations by using *latent modulations*, a vector that is linearly mapped to the shift modulations that is similar to the generator in Chan et al. (2021b) (Figure 3). Then instead of storing the shift modulations, we can store this latent vector to represent any given data point (with the parameters used for mapping latents to shift modulations being part of the base network shared across data points). Figure 3 shows that this approach results in the best tradeoff between reconstruction accuracy (Equation 1) and the dimensionality of the modulations. We therefore use this architecture for the remainder of the paper and also use the word *modulations* to refer to latent modulations. We typically use modulation dimensions of 256 and 512 as these result in reconstructions that are visually very close to the original data, while being orders of magnitude smaller than array representations. These modulations are then used as inputs for the downstream deep learning tasks that we describe in Section 4. We also experimented with other architectures for representing functa but found that they performed worse (*cf.* Appendix B.2).

3.2. Meta-learning functa

Given the representation of each functa point as a modulation applied to a base network, we are now faced with two problems: 1. How do we learn the weights of the base network such that they efficiently encode the shared characteristics of data points? 2. Fitting INRs can be slow – e.g. fitting a single NeRF scene can take 1 GPU day. If we are to create a dataset of several thousand NeRF scenes (as we

Algorithm 1 Meta-learning functa

Randomly initialize shared base network θ
while not done **do**
  Sample batch $\mathcal{B}$ of data $\{\{\mathbf{x}_i^{(j)}, \mathbf{f}_i^{(j)}\}_{i\in\mathcal{I}}\}_{j\in\mathcal{B}}$
  Set batch modulations to zero $\phi_j \leftarrow 0\ \forall j \in \mathcal{B}$
  for all step $\in \{1, \dots, N_{inner}\}$ and $j \in \mathcal{B}$ **do**
    $\phi_j \leftarrow \phi_j - \epsilon\nabla_\phi \mathcal{L}(f_{\theta,\phi}, \{\mathbf{x}_i^{(j)}, \mathbf{f}_i^{(j)}\}_{i\in\mathcal{I}})|_{\phi=\phi_j}$
  end for
  $\theta \leftarrow \theta - \epsilon' \frac{1}{|\mathcal{B}|} \sum_{j\in\mathcal{B}} \nabla_\theta \mathcal{L}(f_{\theta,\phi}, \{\mathbf{x}_i^{(j)}, \mathbf{f}_i^{(j)}\}_{i\in\mathcal{I}})|_{\phi=\phi_j}$
end while

do in this paper), fitting each of them individually would be prohibitively expensive. How can we efficiently create large datasets of modulations?

We solve both problems by meta-learning a network initialization such that each functa point can be fitted in a few gradient steps (*cf.* Algorithm 1). Indeed, Tancik et al. (2021) show that applying MAML (Finn et al., 2017) to INRs can enable fitting of images and NeRF scenes in only a handful of gradient steps. Similar results were also shown previously on signed distance functions (Sitzmann et al., 2020a). However in both works, the initialization of all neural network parameters is meta-learned then fitted to each data point. In our case each functa is stored as a modulation vector, and when creating the functaset we only fit the modulation to each data point with the shared base network fixed. In the inner loop we therefore only update the modulations, whereas in the outer loop we only update the base network weights. This then corresponds to an instance of learning a subset of weights with MAML, also known as CAVIA (Zintgraf et al., 2019).

After meta-learning the base network using the training data, we create a dataset of modulations by running the inner loop on each data point that takes a few gradient steps to fit the modulation. We found that using 3 gradient steps works well for both meta-learning and fitting modulations. This is done for each data point of the training set and test set, to obtain modulation datasets for both training and test. An additional benefit of meta-learning the weights and modulations is that it makes the modulation space smooth, as all modulations are a handful of gradient steps away from each other. We hypothesize that this simplifies downstream tasks, yet also faces limitations. We discuss the limitations of meta-learning and alternatives in Section 7.

4. Deep learning on functa

Given the modulation representation of functa, we train deep learning models directly on the modulations for a selection of downstream tasks. This decoupling of fitting functa and the downstream task is more memory-efficient than joint learning, and modular in the sense that we can use the same functa for different tasks. Also note that modulations are arrays, but are different to the original array data and conventional latent representations in deep learning in that they parameterize a function, hence enjoy the advantages of functa listed in Section 2.

We mainly focus on the task of generative modeling not only because it is challenging, but also because it is a broad task that can be used for inference, which includes applications such as imputation and novel view synthesis as we later describe in this section. We also show results on classification to demonstrate the scope of the framework, covering both generative and discriminative tasks.

Generative modelling. We primarily use normalizing flows (Rezende & Mohamed, 2015; Durkan et al., 2019) and diffusion (Sohl-Dickstein et al., 2015; Ho et al., 2020) for generative modelling, chosen based on ease of optimization. VAEs (Kingma & Welling, 2014; Rezende et al., 2014) and GANs (Goodfellow et al., 2014) are also sensible choices that we do not explore in this paper. We also explored Transformers (Vaswani et al., 2017) as an example of an autoregressive generative model, but found them to underperform (*cf.* Appendix C). This may be because it is difficult to impose a meaningful ordering to the modulation dimensions.

Normalizing flows model the data distribution by mapping a simple base distribution (usually a Gaussian) through a sequence of invertible layers parameterized by neural networks. Diffusions model the data by applying a sequence of fixed Gaussian noise (forward process) to the data, and then training a neural network to denoise the noisy version of the data at each step (backward process). We use Neural Spline Flows (NSF) (Durkan et al., 2019) and DDPM (Ho et al., 2020) with MLP-based architectures as an example of each generative model, providing background and implementation details in Appendix A.4 and Appendix A.5.

Each flow/diffusion layer preserves the dimensionality of data that it is trained on, so the model size typically scales with input dimensionality. For these models, it is therefore much more efficient to train on modulations compared to the original array representation. In the case of NeRF scenes, there is no obvious way to form an array representation of each scene on which we can train flows or diffusion. In contrast, when representing functa as modulations we can directly learn the distribution of modulations, greatly simplifying generative modeling of NeRF scenes.

Inference. Given a generative model over modulations, we can formulate various applications as an inference problem. The learned distribution over modulations can be interpreted as the prior, and the reconstruction loss of the functa corresponding to the modulation can be interpreted as the likelihood. By optimising a weighted average of this prior and likelihood with respect to the modulations ϕ, we can

perform MAP inference:

$$\min_{\phi} -\log p(\phi) + \lambda \sum_{i \in \mathcal{I}} \| f_\phi(\mathbf{x}_i) - \mathbf{f}_i \|_2^2 \tag{2}$$

This can be used for imputation when the likelihood is computed on a partial observation. For example, we can optimise a modulation to achieve high log probability under the prior as well as fitting one half of a voxel grid. We then query the INR represented by the resulting modulation at all grid points to fill in the other half (*cf.* Figure 7). In terms of Equation 2, $\mathcal{I}$ would then correspond to coordinates of one half of the voxel grid. Similarly if the reconstruction error is computed on a (partial) view of a scene, then MAP inference will allow us to infer the scene by fitting the modulations via Equation 2. The scene can then be rendered at arbitrary viewpoints for novel view synthesis. Note that flows are more suitable as the prior for inference than diffusion, as flow densities can be computed exactly and efficiently whereas diffusion densities are usually intractable or expensive to compute.

Classification. We also train classifiers directly on modulation datasets. With only small MLPs, we can reach high test accuracy in a few thousand iterations (minutes of wall clock time on a single GPU).

5. Related Work

Generative models of functions. In early work, Ha (2016) trained GANs and VAEs on functional representations of MNIST. Later, Neural Processes (Garnelo et al., 2018a;b; Kim et al., 2019; Gordon et al., 2019) were introduced to model conditional distributions of 1D functions and images as 2D functions. Skorokhodov et al. (2021); Anokhin et al. (2021) introduce an adversarial approach for learning distributions of INRs for images, generating high quality images. INRs have also been used to learn distributions of 3D shapes using GANs (Chen & Zhang, 2019; Kleineberg et al., 2020), VAEs (Mescheder et al., 2019) and score-based generative models (Cai et al., 2020). While not formally generative models, auto decoders have also been used to parameterize families of 3D shapes (Park et al., 2019; Atzmon & Lipman, 2021). A series of recent works have built generative models for NeRF scenes mostly using GANs (Schwarz et al., 2020; Niemeyer & Geiger, 2021; Chan et al., 2021b;a; DeVries et al., 2021) or VAEs (Kosiorek et al., 2021).

The most closely related works to ours are GASP (Dupont et al., 2021b) and GEM (Du et al., 2021). GASP learns distributions of INRs using a modality agnostic point-cloud discriminator with a GAN for learning distributions of images, 3D shapes and data on manifolds. However, unlike our approach, GASP's GAN-based training is unstable and not applicable to NeRF scenes as it is unclear how to feed in a scene to the discriminator. GEM learns a manifold of INRs in a modality agnostic manner, by embedding latent vectors (mapped to INRs through a hypernetwork) into a space which is regularized to have various desirable properties. GEM is successfully applied across a range of modalities and a variety of tasks, but the learned embeddings are not used to train generative models or classifiers.

Multimodal architectures. The literature on multimodal processing usually relies on modality-specific feature extractors (Kaiser et al., 2017; Chen et al., 2019; Alayrac et al., 2020). However the recently introduced Perceiver (Jaegle et al., 2021b;a) uses a shared achitecture for processing a wide range of data modalities, sharing similarities with our setup. However they are only applied to array representations of data, and it would be an interesting research direction to apply them to functa for downstream tasks.

Diffusion and flows in latent space. There has also been a variety of work on applying diffusion to the latent space of a VAE (Vahdat et al., 2021; Mittal et al., 2021; Wehenkel & Louppe, 2021; Sinha et al., 2021). Similarly there have been various works that use flow priors for VAEs (Chen et al., 2016; Huang et al., 2017; Xiao et al., 2019). These are in contrast to our work that applies diffusion and flows to functional representations.

Deep learning on neural networks. There have been a few works where neural networks are used as inputs to other neural networks. Unterthiner et al. (2020) predict classification accuracies of CNNs directly from their vectorized weights. Schürholt et al. (2021) further apply self-supervised learning to the vectorized weights and use the resulting representations to predict various characteristics of the input classifier. Knyazev et al. (2021); Jaeckle & Kumar (2021); Lu & Kumar (2019) represent the computational graph of neural networks as a GNN that is used to predict optimal parameters, adversarial examples, or branching strategies for neural network verification. These works differ from ours in that their input neural networks do not represent functions, and their goal is to predict quantities about the neural network rather than to perform generative/discriminative tasks.

6. Experiments

We evaluate our framework on four data modalities: *images*, using the CelebA-HQ 64×64 dataset (Karras et al., 2018), *voxels*, using the ShapeNet dataset (Chang et al., 2015), *NeRF scenes*, using the SRN Cars dataset (Sitzmann et al., 2019) and *data on manifolds* using the ERA5 temperature dataset (Hersbach et al., 2019). We implement all models in Jax (Bradbury et al., 2018) using Haiku (Hennigan et al., 2020) and Jaxline (Babuschkin et al., 2020) for training. We discuss negative results in Appendix C.

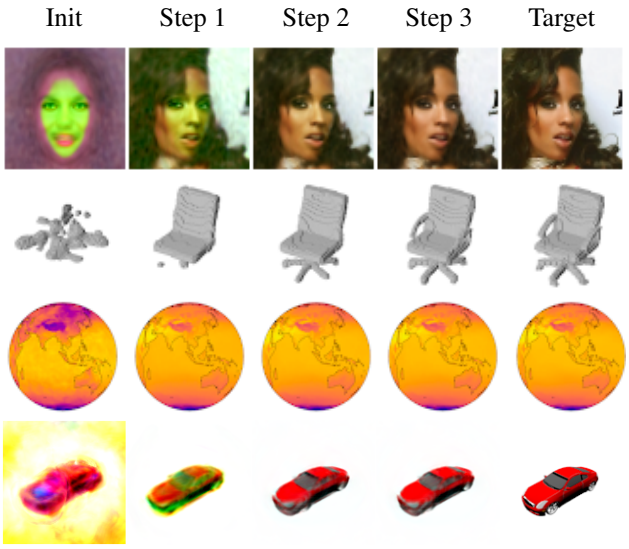

Figure 4. Visualization of the meta-learned initialization, each gradient step of the inner loop and target data point. GIF showing scenes at different poses: bit.ly/3nBjIDO

Dataset, array size	Split	Modulation dimensionality 64	128	256	512	1024
ShapeNet Chairs, 64^3	Train	99.43	99.49	99.49	99.51	99.53
	Test	99.11	99.28	99.38	99.46	99.51
ShapeNet 10 Classes, 64^3	Train	99.36	99.44	99.47	99.52	99.56
	Test	99.30	99.40	99.44	99.50	99.55
CelebA-HQ, 64×64	Train	22.2	24.2	26.6	29.7	32.4
	Test	21.6	23.5	25.6	28.0	30.7
SRN Cars, 128×128	Train	24.3	24.2	24.6	24.6	24.4
	Test	22.4	23.0	23.1	23.2	23.1
ERA5, 181×360	Train	43.2	43.7	43.8	44.0	44.1
	Test	43.2	43.6	43.8	43.9	44.0

Table 1. Mean reconstruction of modulations across each dataset vs modulation size. Metric is voxel accuracy (%) for ShapeNet and PSNR (dB) for the rest. See Appendix A.3 for details on metric.

6.1. Meta-learning

We first create functasets for each data modality by meta-learning an initialization and then fitting modulations to each data point. As shown in Figure 4, we are able to fit modulations to a high degree of accuracy in only 3 gradient steps across a range of data modalities. Table 1 shows that we are able to capture signals accurately using only 64-512 modulations, with accuracy usually increasing with modulation size. The one exception is SRN Cars, where reconstruction quality is similar across all modulation sizes. We suspect this is due to our very basic rendering scheme (*cf.* Section 6.4). Qualitative examples of reconstructions for each data modality are shown in Figure 18 of Appendix D.

6.2. Images

We train DDPM (Ho et al., 2020) on the CelebA-HQ 64×64 functaset with 256 dimensional modulations. As can be seen in Figure 5, the diffusion model is able to produce realistic and convincing samples even though it is trained directly on modulations. Compared to GASP, the samples produced by our model are more coherent, although slightly blurrier. To verify that our model has not memorized the training dataset, we show nearest neighbor samples in Figure 19 in the appendix. Our model achieves an FID score (Heusel et al., 2017) of 40.4, but we found that this was not well correlated with perceptual quality. As noted by Du et al. (2021), this is likely due to FID's property of over-penalizing blurriness (see Appendix A.8 for discussion).

Figure 5. Uncurated samples from GASP and DDPM (diffusion) trained on 256-dim CelebA-HQ 64×64 modulations.

6.3. Voxels

We use two datasets derived from the ShapeNet database (Chang et al., 2015), 64^3 voxels from the chairs category and 64^3 voxels from 10 classes, using data augmentation to ensure each class contains a large number of samples (*cf.* Appendix A.1). We train NSF (Durkan et al., 2019) on a 256 dimensional functaset of the chairs category, with unconditional samples shown in Figure 6 along with comparisons to other baseline generative models using 3D functional representations. As can be seen, our model generates coherent and realistic samples, whereas the baselines tend to be less consistent. Further, our approach stably and consistently produces good results whereas GASP is unstable to train for 3D voxels (Dupont et al., 2021b). Our model can also be used to produce semantically meaningful latent interpolations as shown in Figure 23 in Appendix D. Figure 8 shows samples from a class-conditional NSF trained on 10 classes from ShapeNet, where our model produces realistic and consistent samples for each class. Finally, we show imputation results in Figure 7, where our model is able to infer the shape of the chair from various partial observations, including a simulated lidar scan (last column).

6.4. NeRF scenes

We evaluate our framework on NeRF scenes by using the SRN Cars dataset (Sitzmann et al., 2019), containing 2458

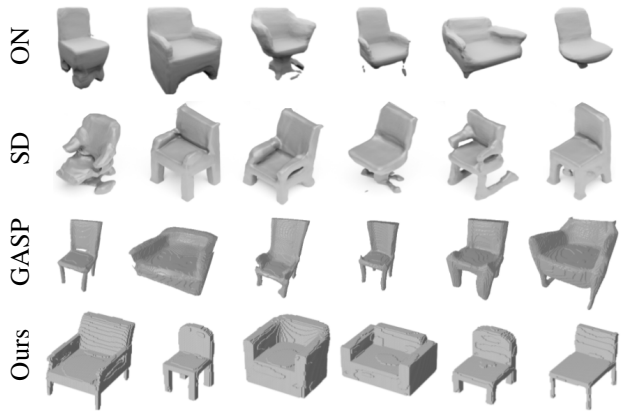

Figure 6. Unconditional samples from our model and three baselines: Occupancy Networks trained as a VAE (ON) from Mescheder et al. (2019), a GAN trained on signed distance functions with a set discriminator (SD) from Kleineberg et al. (2020) and GASP from Dupont et al. (2021b).

Figure 7. Imputation from partial test observations using the learned distribution over functa as a prior. GIF showing course of optimization: `bit.ly/3fZnYZG`. Additional chairs: Figure 27

scenes each with 50 posed images of size 128×128. NeRF represents a scene via an INR mapping 3D coordinates to density and RGB values (*cf.* Figure 1). Views of the scene are then generated via volume rendering, and modulations are fitted by minimizing reconstruction loss on the available set of posed images on training scenes (see Appendix A for background and details). While NeRF uses several tricks to achieve good performance, we use a minimal bare bones model for our experiments (*cf.* Appendix A.6), as we are more interested in testing the generative capabilities of our model rather than the visual quality of NeRF. As a baseline, we compare against π-GAN (Chan et al., 2021b) using model weights kindly provided to us by the authors.

We train both DDPM and NSF on NeRF modulations, achieving similar sample quality. As shown in Figure 9, DDPM is able to generate plausible NeRF scenes of cars, giving an FID score of 80.3 (*cf.* Appendix A.1 for details on FID computation) compared to 36.7 for π-GAN. While the samples produced by π-GAN are generally sharper than ours (as reflected by the FID scores), we believe this can be mitigated by using more sophisticated rendering (including e.g. the hierarchical sampling used by π-GAN). Further, we

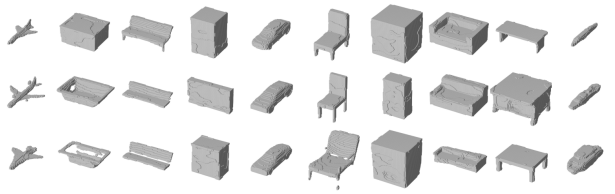

Figure 8. Uncurated samples from a class-conditional flow trained on 256-dim modulations of ShapeNet 10 Classes 64^3.

Figure 9. Uncurated samples from π-GAN and DDPM trained on 64-dim modulations of SRN Cars. GIF showing different poses: `bit.ly/33Ojhzz`

note that training π-GAN requires backpropagating through the volume rendering step, making it very memory intensive. In contrast, we train our model directly on modulations which is much less expensive.

As shown in Figure 10, NSF gives smooth interpolations both in terms of shape and texture. We also perform novel view synthesis experiments in Figure 11. Given a single occluded view of a test scene, the NSF prior is used to infer a scene that is consistent with the occluded view. Without the prior term in the loss, i.e. when only fitting the modulation to the occluded view, the resulting scene is not realistic, highlighting the importance of using the prior for inference.

We highlight that for scenes there is no obvious way to form an array based representation. GAN based models of NeRF scenes (Schwarz et al., 2020; Niemeyer & Geiger, 2021) therefore require complex generators and discriminators that backpropagate through the expensive volume rendering operation. Training a generative model in our case is, in contrast, very simple: we simply train a flow or diffusion directly on the modulations. As the generative model itself is independent of the volume rendering step, we believe this holds promise for scaling generative scene models to larger scales than currently possible. In particular, we used a very simple rendering model (no hierarchical sampling, single shared network for density and RGB, no view dependence)

Figure 10. Latent interpolation between two car scenes with moving pose. GIF: bit.ly/3g41wOY

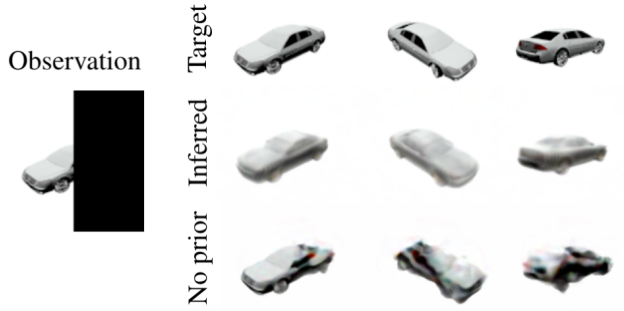

Figure 11. Novel view synthesis from occluded test scene. Without the prior, the model is not able to correctly infer the shape of the car. GIF: bit.ly/3rREHDT. Additional scenes: Figure 28

limiting the perceptual quality of the renderings. Using more sophisticated rendering techniques (Mildenhall et al., 2020; Barron et al., 2021) is likely to significantly improve reconstruction and sample quality.

6.5. Manifold data

To demonstrate the flexibility of our approach, we also train NSF on 256-dimensional modulations of ERA5 temperature data (Hersbach et al., 2019). The dataset contains 10k grids of temperature measurements at equally spaced latitudes λ and longitudes φ. We convert them to Cartesian coordinates $\mathbf{x} = (\cos\lambda\cos\varphi, \cos\lambda\sin\varphi, \sin\lambda)$ for the functa inputs as in Dupont et al. (2021b). As our method is more stable and scalable than GASP, we are able to train on 181×360 grids as opposed to the 46×90 grids used by GASP, yielding higher resolution samples (Figure 12).

6.6. Classification

CLASSIFIER	TEST ACCURACY	n_{PARAMS}
MLP ON FUNCTA	$93.6 \pm 0.1\%$	83K
3D CNN	$93.3 \pm 0.3\%$	550K

Table 2. Classification accuracies and parameter count for MLP on functa vs 3D CNN on array data for ShapeNet 10 Classes, 64^3.

Finally, we evaluate our model for classification using the ShapeNet 10 classes dataset. Given the functaset, the task is to predict the class of the object from the modulation. As a baseline we train a 3D CNN classifier on 64^3 voxels with an architecture based on Maturana & Scherer (2015). As shown in Table 2, our 4 hidden layer MLP of width 128 performs similarly (if not better) than the 8-layer ($6\times$ 3DConv + $2\times$ linear layers) 3D CNN baseline at convergence. Our small MLP can be trained in < 10 minutes on a single GPU with batch size 1024. In contrast, the 3D CNN baseline is memory intensive and thus we run on 8 devices each with batch size 128 (to match batch size) for > 1 hour, highlighting the scalability of performing deep learning tasks on functa. See Appendix A.7 for experimental details and Appendix D for further results and training curves.

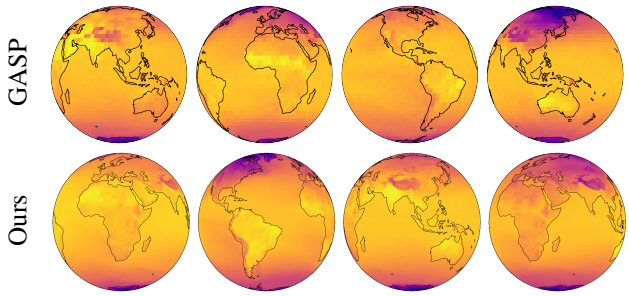

Figure 12. Uncurated samples from GASP and a flow trained on 256-dim modulations of ERA5. GIF: bit.ly/35ya6nr

7. Conclusion, limitations and future work

Conclusion. Motivated by the various compelling properties of functional representations, we propose to view INRs as data points, or functa, and treat these as first class citizens for machine learning tasks. We introduced a method for creating such datasets of functa at scale and showed that it is possible to learn generative models, perform data imputation, novel view synthesis and classification across a wide range of data modalities within this framework.

Limitations and future work. Using a shared framework for different data modalities implies that we cannot employ modality specific inductive biases when designing models trained on functa. Indeed, storing functa as modulations removes spatial structure from the data, forcing us to use general MLP architectures. We see promise in spatial functional representations (e.g. per-patch modulations in Mehta et al. (2021)) on which for example convolutions can be applied, exploiting their locality and translation invariance.

Further, we rely on a variant of MAML to create our functaset, and therefore inherit the limitations of MAML, including a large memory footprint and occasionally unstable training due to the double loop optimization. In addition, MAML constrains the modulations to lie within a few gradient steps of the meta-learned initialization which could limit reconstruction accuracy for more complex datasets. As MAML is a bottleneck in terms of memory and possibly

representational power, it would be interesting to explore alternatives to meta-learning for creating functasets, for example using the autodecoder framework (Park et al., 2019).

When creating the functaset, we must make multiple choices such as the architecture of the base network and the modulation dimensionality. Our metric for this choice was compressibility, i.e. using the smallest number of modulations for a given PSNR. However, this may be suboptimal for the downstream task, and an alternative is to use a task-specific metric and/or jointly learn the INR and the downstream model end-to-end. However this may harm the reconstruction quality of the INR, be memory expensive, and would require retraining separate INRs for separate tasks.

The field of INRs is progressing rapidly and we will likely be able to take advantage of this progress, including hybrid representations (Martel et al., 2021), better activation functions (Ramasinghe & Lucey, 2021) and reduced memory consumption (Huang et al., 2021). There has also been a plethora of work on improving NeRF (Barron et al., 2021; Reiser et al., 2021; Yu et al., 2021; Piala & Clark, 2021), which should be directly applicable to our framework. Further, recent works have shown promise for storing compressed datasets as functions (Dupont et al., 2021a; Chen et al., 2021; Strümpler et al., 2021; Zhang et al., 2021). Using our framework, it may therefore become possible to train deep learning models directly on these compressed datasets, which is challenging for traditional compressed formats such as JPEG (although image-specific exceptions such as Nash et al. (2021) exist). In addition, learning distributions of functa is likely to improve entropy coding and hence compression for these frameworks (Ballé et al., 2016).

Acknowledgments

We thank: Hrushikesh Loya, Milad Alizadeh and Adam Golinski for helpful discussions around meta-learning of modulations; Thu Nguyen-Phuoc for helpful discussions around NeRF; Olivia Wiles for providing DDPM implementation and FID score computation; Conor Durkan and Sander Dieleman for helpful discussions around diffusion; Andy Brock for helpful discussion around implicit neural representations; Andriy Mnih and Yee Whye Teh for general feedback; Eric Ryan Chan for sharing the trained π-GAN model weights for the NeRF baseline. Emilien gratefully acknowledges his PhD funding from Google DeepMind.

References

Alayrac, J.-B., Recasens, A., Schneider, R., Arandjelovic, R., Ramapuram, J., De Fauw, J., Smaira, L., Dieleman, S., and Zisserman, A. Self-supervised multimodal versatile networks. *NeurIPS*, 2(6):7, 2020.

Anokhin, I., Demochkin, K., Khakhulin, T., Sterkin, G., Lempitsky, V., and Korzhenkov, D. Image generators with conditionally-independent pixel synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14278–14287, 2021.

Antoniou, A., Edwards, H., and Storkey, A. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

Atzmon, M. and Lipman, Y. SALD: Sign agnostic learning with derivatives. In *International Conference on Learning Representations*, 2021.

Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Fantacci, C., Godwin, J., Jones, C., Hennigan, T., Hessel, M., Kapturowski, S., Keck, T., Kemaev, I., King, M., Martens, L., Mikulik, V., Norman, T., Quan, J., Papamakarios, G., Ring, R., Ruiz, F., Sanchez, A., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Stokowiec, W., and Viola, F. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Ballé, J., Laparra, V., and Simoncelli, E. P. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

Barron, J. T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., and Srinivasan, P. P. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *arXiv preprint arXiv:2103.13415*, 2021.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Cai, R., Yang, G., Averbuch-Elor, H., Hao, Z., Belongie, S., Snavely, N., and Hariharan, B. Learning gradient fields for shape generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pp. 364–381. Springer, 2020.

Chan, E. R., Lin, C. Z., Chan, M. A., Nagano, K., Pan, B., De Mello, S., Gallo, O., Guibas, L., Tremblay, J., Khamis, S., et al. Efficient geometry-aware 3d generative adversarial networks. *arXiv preprint arXiv:2112.07945*, 2021a.

Chan, E. R., Monteiro, M., Kellnhofer, P., Wu, J., and Wetzstein, G. pi-GAN: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5799–5809, 2021b.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Chen, H., He, B., Wang, H., Ren, Y., Lim, S. N., and Shrivastava, A. Nerv: Neural representations for videos. *Advances in Neural Information Processing Systems*, 34, 2021.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Chen, Y.-C., Li, L., Yu, L., El Kholy, A., Ahmed, F., Gan, Z., Cheng, Y., and Liu, J. Uniter: Learning universal image-text representations. 2019.

Chen, Z. and Zhang, H. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5939–5948, 2019.

Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. Spherical cnns. In *International Conference on Learning Representations*, 2018.

DeVries, T., Bautista, M. A., Srivastava, N., Taylor, G. W., and Susskind, J. M. Unconstrained scene generation with locally conditioned radiance fields. *arXiv preprint arXiv:2104.00670*, 2021.

Du, Y., Collins, M. K., Tenenbaum, B. J., and Sitzmann, V. Learning signal-agnostic manifolds of neural fields. In *Advances in Neural Information Processing Systems*, 2021.

Dupont, E., Goliński, A., Alizadeh, M., Teh, Y. W., and Doucet, A. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021a.

Dupont, E., Teh, Y. W., and Doucet, A. Generative models as distributions of functions. *arXiv preprint arXiv:2102.04776*, 2021b.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Neural spline flows. *Advances in Neural Information Processing Systems*, 32:7511–7522, 2019.

Fathony, R., Sahu, A. K., Willmott, D., and Kolter, J. Z. Multiplicative filter networks. In *International Conference on Learning Representations*, 2020.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. Conditional neural processes. In *International Conference on Machine Learning*, pp. 1704–1713. PMLR, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2014.

Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. Convolutional conditional neural processes. In *International Conference on Learning Representations*, 2019.

Ha, D. Generating large images from latent vectors. *blog.otoro.net*, 2016. URL https://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hennigan, T., Cai, T., Norman, T., and Babuschkin, I. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Hersbach, H., Bell, B., Berrisford, P., Biavati, G., Horányi, A., Muñoz Sabater, J., Nicolas, J., Peubey, C., Radu, R., Rozum, I., Schepers, D., Simmons, A., Soci, C., Dee, D., and Thépaut, J.-N. ERA5 monthly averaged data on single levels from 1979 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels-monthly-means (Accessed 27-09-2021), 2019.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems*, pp. 6626–6637, 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

Huang, C.-W., Touati, A., Dinh, L., Drozdzal, M., Havaei, M., Charlin, L., and Courville, A. Learnable explicit density for continuous latent space and variational inference. *arXiv preprint arXiv:1710.02248*, 2017.

Huang, Z., Bai, S., and Kolter, J. Z. Implicit^2: Implicit layers for implicit representations. *Advances in Neural Information Processing Systems*, 34, 2021.

Jaeckle, F. and Kumar, M. P. Generating adversarial examples with graph neural networks. *arXiv preprint arXiv:2105.14644*, 2021.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021a.

Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver: General perception with iterative attention. *arXiv preprint arXiv:2103.03206*, 2021b.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119, 2020.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Kleineberg, M., Fey, M., and Weichert, F. Adversarial generation of continuous implicit shape representations. In *Eurographics - Short Papers*, 2020.

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero Soriano, A. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 34, 2021.

Kosiorek, A. R., Strathmann, H., Zoran, D., Moreno, P., Schneider, R., Mokrá, S., and Rezende, D. J. Nerf-vae: A geometry aware 3d scene generative model. *arXiv preprint arXiv:2104.00587*, 2021.

Li, Z., Zhou, F., Chen, F., and Li, H. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

Li, Z., Niklaus, S., Snavely, N., and Wang, O. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6498–6508, 2021.

Lu, J. and Kumar, M. P. Neural network branching for neural network verification. *arXiv preprint arXiv:1912.01329*, 2019.

Martel, J. N., Lindell, D. B., Lin, C. Z., Chan, E. R., Monteiro, M., and Wetzstein, G. Acorn: Adaptive coordinate networks for neural scene representation. *arXiv preprint arXiv:2105.02788*, 2021.

Maturana, D. and Scherer, S. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928. IEEE, 2015.

Mehta, I., Gharbi, M., Barnes, C., Shechtman, E., Ramamoorthi, R., and Chandraker, M. Modulated periodic activations for generalizable local functional representations, 2021.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pp. 405–421, 2020.

Mittal, G., Engel, J., Hawthorne, C., and Simon, I. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

Nash, C., Menick, J., Dieleman, S., and Battaglia, P. W. Generating images with sparse representations. *arXiv preprint arXiv:2103.03841*, 2021.

Niemeyer, M. and Geiger, A. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11453–11464, 2021.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Piala, M. and Clark, R. Terminerf: Ray termination prediction for efficient neural rendering. In *2021 International Conference on 3D Vision (3DV)*, pp. 1106–1114. IEEE, 2021.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Ramasinghe, S. and Lucey, S. Beyond periodicity: Towards a unifying framework for activations in coordinate-mlps. *arXiv preprint arXiv:2111.15135*, 2021.

Razavi, A., van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pp. 14866–14876, 2019.

Reiser, C., Peng, S., Liao, Y., and Geiger, A. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *arXiv preprint arXiv:2103.13744*, 2021.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pp. 1278–1286. PMLR, 2014.

Schürholt, K., Kostadinov, D., and Borth, D. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34, 2021.

Schwarz, K., Liao, Y., Niemeyer, M., and Geiger, A. GRAF: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33, 2020.

Sinha, A., Song, J., Meng, C., and Ermon, S. D2c: Diffusion-denoising models for few-shot conditional generation. *arXiv preprint arXiv:2106.06819*, 2021.

Sitzmann, V., Zollhöfer, M., and Wetzstein, G. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, pp. 1121–1132, 2019.

Sitzmann, V., Chan, E. R., Tucker, R., Snavely, N., and Wetzstein, G. Metasdf: Meta-learning signed distance functions. In *Proc. NeurIPS*, 2020a.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020b.

Skorokhodov, I., Ignatyev, S., and Elhoseiny, M. Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10753–10764, 2021.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Stanley, K. O. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

Strümpler, Y., Postels, J., Yang, R., Van Gool, L., and Tombari, F. Implicit neural representations for image compression. *arXiv preprint arXiv:2112.04267*, 2021.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, 2020.

Tancik, M., Mildenhall, B., Wang, T., Schmidt, D., Srinivasan, P. P., Barron, J. T., and Ng, R. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2846–2855, 2021.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

Vahdat, A., Kreis, K., and Kautz, J. Score-based generative modeling in latent space. *arXiv preprint arXiv:2106.05931*, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wehenkel, A. and Louppe, G. Diffusion priors in variational autoencoders. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021.

Xiao, Z., Yan, Q., and Amit, Y. Generative latent flow. *arXiv preprint arXiv:1905.10485*, 2019.

Yu, A., Li, R., Tancik, M., Li, H., Ng, R., and Kanazawa, A. Plenoctrees for real-time rendering of neural radiance fields. *arXiv preprint arXiv:2103.14024*, 2021.

Zhang, Y., van Rozendaal, T., Brehmer, J., Nagel, M., and Cohen, T. Implicit neural video compression. *arXiv preprint arXiv:2112.11312*, 2021.

Zintgraf, L., Shiarli, K., Kurin, V., Hofmann, K., and Whiteson, S. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pp. 7693–7702. PMLR, 2019.

A. Experimental details

A.1. Datasets

CelebA-HQ 64 $\times$ 64. We use a train/test split of 27,000/3,000. The pixel coordinates $\mathbf{x}_i$ (inputs to SIREN) are normalized to lie in $[0, 1]^2$ and pixel intensities $\mathbf{f}_i$ are also normalized to lie in $[0, 1]$. More precisely the $\mathbf{x}_i$ are set to be the coordinates of the pixel centers when the image corners are at $\{(0, 0), (1, 0), (0, 1), (1, 1)\}$, the vertices of the unit square.

ShapeNet. We take the original 128^3 voxel dataset and downscale to 64^3 using `scipy.ndimage.zoom` with threshold=0.05. We found this value of the threshold to be a suitable tradeoff between preserving the structure of the original shapes and having a smooth shape. We also found it important to augment the dataset for preventing overfitting for downstream generative modelling. We apply a 50-fold augmentation by independently rescaling the shape in each of the 3 axes (x, y, z) by a randomly sampled scale in $U[0.75, 1.25]$. The resulting Chairs dataset has a train/test split of 304,850/33,900 and the 10 class dataset has a train/test split of 1,516,750/168,850. The voxel grid coordinates $\mathbf{x}_i$ lie in $[0, 1]^3$, and the voxel occupancies $\mathbf{f}_i$ are binary, one of $\{0, 1\}$. Similarly to images the $\mathbf{x}_i$ are set to be the coordinates of the voxel centers when the full 64^3 voxel's corners are at the vertices of the unit cube.

ERA5 temperature. This dataset comes with several decades (1979-2020) worth of temperature observations of grids at equally spaced latitudes λ and longitudes φ across the globe, which we downsample to a 181×360 grid (from the original 721×1440 grid). Note that we treat different time steps as different data points, as modelling the data as a function of time and latitude/longitude would reduce the dataset to a single data point. The $\mathbf{x}_i = (\cos\lambda\cos\varphi, \cos\lambda\sin\varphi, \sin\lambda)$ are 3D Cartesian coordinates obtained from latitudes λ that are equally spaced between $\pi/2$ and -$\pi/2$ and longitudes φ that are equally spaced between 0 and $\frac{2\pi(n-1)}{n}$ where n is the number of distinct values of longitude (360). We use a train/test split of 9676/2420.

SRN Cars. This dataset has a train/test split of 2458/703 car scenes where each train scene has 50 views (randomly sampled from sphere centered at car) and test scene has 251 views (uniformly distributed on upper hemisphere centered at car). Each view consists of:

1. RGB image of size 128×128×3
2. Pose information of size 4×4 that contains a 3×3 orthogonal matrix mapping the camera's frame of reference to world coordinates and 3×1 coordinates of the camera's position in the world
3. Scalar focal length of camera

The pose information and focal value are used to calculate the origin and direction of the 128×128 rays from the camera position to each pixel of the view. Then `num_points_per_ray` points are sampled along each ray at regular intervals, starting from a distance `near` to `far` from the camera. This dataset uses values (`near`, `far`)=$(1.25, 2.75)$. The 3D coordinates of these sampled points on the ray are the $\mathbf{x}_i$ that are fed into a SIREN to produce the corresponding RGB and density value corresponding to that point in 3D space. These values along the ray are combined via volumetric rendering (see details in Appendix A.6) to compute the prediction of the pixel intensity corresponding to that ray, and the SIREN's parameters are optimized to minimise the reconstruction error at all rays for `num_points` subsampled pixel locations and `num_views` subsampled views (subsampling is necessary to fit the optimization in device memory). These are hyperparameters that are sweeped over for meta-learning (see details below). Regarding FID computation, note that FID is typically calculated with respect to the train set. However for SRN Cars, the train set only contains random views, half of which are looking into the bottom of the car. Thus it is unsuitable to use the train set for sample quality quantification via FID, where we are much more interested in views from the upper hemisphere. Hence we calculate FID score by computing InceptionV3 (Szegedy et al., 2016) features on all views of the test set, and comparing against the same views for n sampled scenes where n is the number of test scenes.

A.2. SIREN and modulations

Each layer of SIREN (Sitzmann et al., 2020b) is parameterised as follows:

$$\mathbf{x} \mapsto \sin(\omega_0(\mathbf{W}\mathbf{x} + \mathbf{b})) \tag{3}$$

where $\mathbf{W}, \mathbf{b}$ are trainable parameters and ω_0 is a fixed hyperparameter. We fix $\omega_0 = 30$ for all experiments (except for SRN Cars where we use $\omega_0 = 5$), as we observed that performance was similar for $\omega_0 = \{30, 50, 70\}$. Following the original

SIREN implementation, the weights $\mathbf{W}$ are randomly initialized from $U[-\frac{1}{n_{in}}, \frac{1}{n_{in}}]$ for the first layer and $U[-\frac{1}{\omega_0}\sqrt{\frac{6}{n_{in}}}, \frac{1}{\omega_0}\sqrt{\frac{6}{n_{in}}}]$ for subsequent layers, where n_{in} is the input dimensionality of the layer. Note that initialization is different than usual to account for the ω_0 multiplicative term in each SIREN layer. Biases $\mathbf{b}$ are initialized to zero as usual.

ModulatedSIREN is a variant of SIREN that contains a shift modulation $\mathbf{s}$. In our context the SIREN is treated as the base network shared across data points and the shift modulations model the variation across the dataset. For an n-layer ModulatedSIREN with weights and biases $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1:n}$, $\mathbf{s} = [\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)}]$ corresponding to the shift modulation $\mathbf{s}$, the ith layer is parameterised as:

$$\mathbf{x} \mapsto \sin(\omega_0(\mathbf{W}^{(i)}\mathbf{x} + \mathbf{b}^{(i)} + \mathbf{s}^{(i)})) \tag{4}$$

LatentModulatedSIREN is the variant that we use throughout the paper, which uses a latent modulation vector ϕ that is linearly mapped to the shift modulation. So each layer of LatentModulatedSIREN is the same as ModulatedSIREN (Equation 4), but the shift modulation $\mathbf{s}$ is parameterised by $\mathbf{s} = \mathbf{W}'\phi + \mathbf{b}'$ for learnable weights $\mathbf{W}'$ and biases $\mathbf{b}'$, both initialized by the Haiku default.

For all SIREN variants, the final layer is simply a linear layer (with no ω_0 scaling nor sine non-linearity), with 0.5 added to the output since the targets are preprocessed to lie in $[0, 1]$. For both ModulatedSIREN and LatentModulatedSIREN, the modulations are always initialized to 0 and fitted by a few gradient descent steps of the inner loop, instead of being randomly initialized.

For a given size for the latent modulation for each dataset, we sweep over the LatentModulatedSIREN depth from $\{10, 15, 20\}$ and width from $\{256, 384, 512\}$ and choose the architecture with the best test PSNR/voxel accuracy after meta-learning on the training set.

A.3. Meta-learning Functa

DATASET	BATCH SIZE PER DEVICE	NUM DEVICES	NUM ITERATIONS
SHAPENET CHAIRS	1	8	1E6
SHAPENET 10 CLASSES	1	8	1E6
CELEBA-HQ	16	16	5E5
SRN CARS	1	8	5E5
ERA5 TEMPERATURE	2	2	1E5

Table 3. Selected hyperparameter values for meta-learning functa for each dataset.

For the outer loop, we use Adam (Kingma & Ba, 2014) with a fixed base learning rate of 3e-6. For the inner loop, we use SGD with learning rate 1e-2. For each modality we use the maximum batch size per device that fits in memory for the biggest LatentModulatedSIREN model we sweep over. In Table 3 we provide hyperparameter values for each dataset.

For SRN Cars, we provide further details about subsampling views and pixels in Appendix A.6.

Tips for meta-learning:

- Training can be unstable for larger SIRENs, in which case it helps to reduce the outer loop learning rate - lowering inner loop learning rate had little effect.
- Raising batch size always helps.
- Narrow and deep SIREN architectures work better than wide and shallow ones.
- Training for many iterations helps.

Note that the PSNR values across whole datasets in Table 1 are computed by first taking the average MSE across whole dataset then converting to PSNR by the formula: $\text{PSNR} = -10\log_{10}(\text{MSE})$. The voxel accuracy is simply the ratio of correctly reconstructed voxels (after rounding to $\{0,1\}$) to all voxels, averaged across the dataset.

As an additional detail, we apply meta-SGD (Li et al., 2017) for our meta-learning i.e. instead of using a fixed learning rate in the inner loop, we learn a learning rate for each parameter (i.e. each modulation). Note that meta-SGD only uses one step

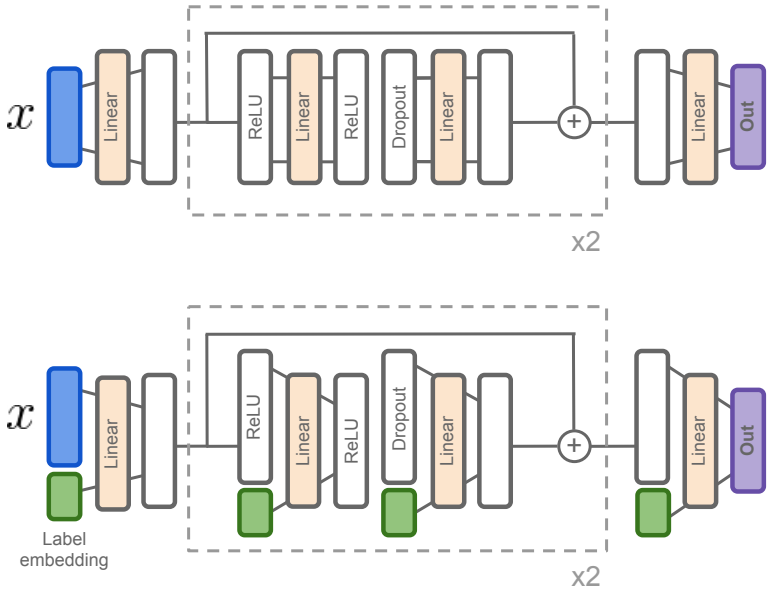

Figure 13. Architecture for neural spline flow conditioner g_θ for the unconditional flow (top) and the class-conditional flow (bottom).

to fit their model, whereas we use several steps. Other approaches have also used learnable learning rates for multiple steps, e.g. MAML++ (Antoniou et al., 2018) uses per-layer (as opposed to per-parameter) and per-step learning rates. The learning rates are initialised to $U[0.005, 0.1]$ and clipped at $(0, 1)$. We found that meta-SGD noticeably improves performance for shift modulations, although didn't make much difference for latent modulations.

A.4. Neural Spline Flows (NSF)

Background. The Neural Spline Flow (NSF) (Durkan et al., 2019) is an example of a normalizing flow (Rezende & Mohamed, 2015) that models data $\mathbf{x}$ as the output of a differentiable and invertible transformation f_θ of Gaussian noise $\mathbf{z}$:

$$\mathbf{x} = f_\theta(\mathbf{z}), \quad \mathbf{z} \sim p(\mathbf{z}) \tag{5}$$

where f_θ is parameterized with a neural network. Note that in our context, the data $\mathbf{x}$ are modulations. The inverse transformation f_θ^{-1} (that takes in the modulation data) is parameterized by a neural network whose parameters are optimized by maximising the log probability of the data under the flow, that is computed by the change of variable formula:

$$\log p(\mathbf{x}) = \log p(f_\theta^{-1}(\mathbf{x})) + \log \left| \frac{\partial f_\theta^{-1}}{\partial \mathbf{x}} \right|. \tag{6}$$

The invertible transformation f_θ consists of multiple flow layers (each with a distinct set of parameters) where each flow layer is the composition of a coupling transform and a PLU invertible linear transform.

A *coupling transform* maps an input $\mathbf{x} \in \mathbb{R}^{2d}$ to output $\mathbf{y} \in \mathbb{R}^{2d}$ as follows:

$$[\mathbf{y}_{1:d}, \mathbf{y}_{d+1:2d}] = [\mathbf{x}_{1:d}, b(\mathbf{x}_{d+1:2d}; g_\theta(\mathbf{x}_{1:d}))] \tag{7}$$

where $b : \mathbb{R}^d \to \mathbb{R}^d$ is an elementwise bijection based on neural splines, whose parameters are given by a conditioner $g_\theta(\mathbf{x}_{1:d})$. It is easy to check that for any invertible b, the coupling transform is also invertible. This conditioner $g_\theta : \mathbb{R}^d \to \mathbb{R}^{pd}$ is an MLP that outputs the p parameters per input dimension for the neural spline bijector.

A *PLU linear transform* is a linear layer that is composed of three linear maps $\mathbf{W} = \mathbf{PLU}$, where each linear map is a $2d \times 2d$ matrix. $\mathbf{P}$ is a random permutation matrix that shuffles the $2d$ input dimensions, and $\mathbf{L}$ is a lower triangular matrix with ones on the diagonal. Similarly $\mathbf{U}$ is an upper triangular matrix. This guarantees that the composition $\mathbf{W}$ is invertible, and the inverse can be computed efficiently by solving two triangular systems, one for $\mathbf{U}$ and one for $\mathbf{L}$, then taking the

inverse permutation $\mathbf{P}^{-1}$. We found that using PLU linear layers led to a noticeable improvement in both training and test likelihood.

Once the NSF is trained, sampling is performed by sampling $\mathbf{z} \sim p(\mathbf{z})$ then passing it through the flow f_θ to obtain a sample $\mathbf{x} = f_\theta(\mathbf{z})$.

Training details. Modulations are centered and normalized by elementwise mean & standard deviation across the training modulation dataset, then scaled by a hyperparameter `norm_factor` before they are fed into the NSF. For the base distribution of the flow, we use a Gaussian with 0 mean and standard deviation 0.25 (to match the inputs normalized by `norm_factor`=4). We fix $p = 3n_{bins} + 1$ where $n_{bins} = 8$ is the number of bins for the rational quadratic spline bijector (*cf.* Durkan et al. (2019) for a detailed formulation of rational quadratic spline bijectors). See Figure 13 for the architectures of g that we use for the unconditional flow and the class-conditional flow. We used Adam with a custom learning rate schedule that warms up linearly from 0 to 3e-4 for the first 4000 warmup iterations, then decays proportional to the square root of the iteration count. The linear layers in the conditioner all have output dimensionalities 128, and the label embeddings are 32-dimensional learnable parameters.

Model selection. The hyperparameters that we tuned are the number of flow layers, sweeped over $\{4, 8, 16, 32, 64, 128, 256, 320\}$ and the dropout probability, sweeped over $\{0., 0.1, 0.2, 0.3\}$. The best model and checkpoint used for showing samples and performing inference was chosen by using test likelihood as the metric. See Table 4 for selected hyperparameter values for each modulation dataset.

Tips. We found that for ShapeNet, data augmentation was key to prevent the NSF from overfitting.

DATASET	BATCH SIZE	NUM FLOW LAYERS	DROPOUT PROBABILITY	NUM ITERATIONS
SHAPENET CHAIRS (256 MOD-DIM)	64	320	0.2	1E6
SHAPENET 10 CLASSES (256 MOD-DIM)	64	128	0.3	1E6
ERA5 TEMPERATURE (256 MOD-DIM)	256	32	0.2	2E5
SRN CARS (512 MOD-DIM)	64	8	0.3	1E5

Table 4. Selected hyperparameter values for Neural Spline flows for each dataset.

A.5. Denoising Diffusion Probabilistic Models (DDPM)

Background. Diffusion models (Sohl-Dickstein et al., 2015) model the joint distribution of the data $\mathbf{x}_0$ (in our case the modulations) and noisy versions of the data $\mathbf{x}_1, \ldots, \mathbf{x}_T$ where $\mathbf{x}_t$ is obtained by linearly scaling $\mathbf{x}_{t-1}$ then adding Gaussian noise. This can be thought of as sequentially adding Gaussian noise to the data $\mathbf{x}_0$ to create variables $\mathbf{x}_1, \ldots, \mathbf{x}_T$, with suitably scaled noise such that the marginal distribution of $\mathbf{x}_T$ is a standard Gaussian. This is called the *forward (diffusion) process* that is formulated as

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) \coloneqq \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{8}$$

for a fixed sequence of variances $\beta_{1:T}$. A desirable property of these Gaussian conditionals is that it admits sampling $\mathbf{x}_t$ directly from $\mathbf{x}_0$ without having to sample the intermediate steps since $q(\mathbf{x}_t|\mathbf{x}_0)$ is a Gaussian that can be written in closed form

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{9}$$

where $\alpha_t \coloneqq 1 - \beta_t$ and $\bar{\alpha}_t \coloneqq \prod_{s=1}^{t} \alpha_s$. Note that this forward process is fixed and has no learnable parameters.

The aim of diffusion models is to learn the *reverse (diffusion) process* that is formulated as:

$$p(\mathbf{x}_T) \coloneqq \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}), \quad p_\theta(\mathbf{x}_{0:T}) \coloneqq p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{10}$$

The mean and variance of the reverse conditionals are parameterized by neural networks, whose parameters are optimized by minimizing

$$\mathbb{E}_q \left[\mathrm{KL}[q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)]\right] \tag{11}$$

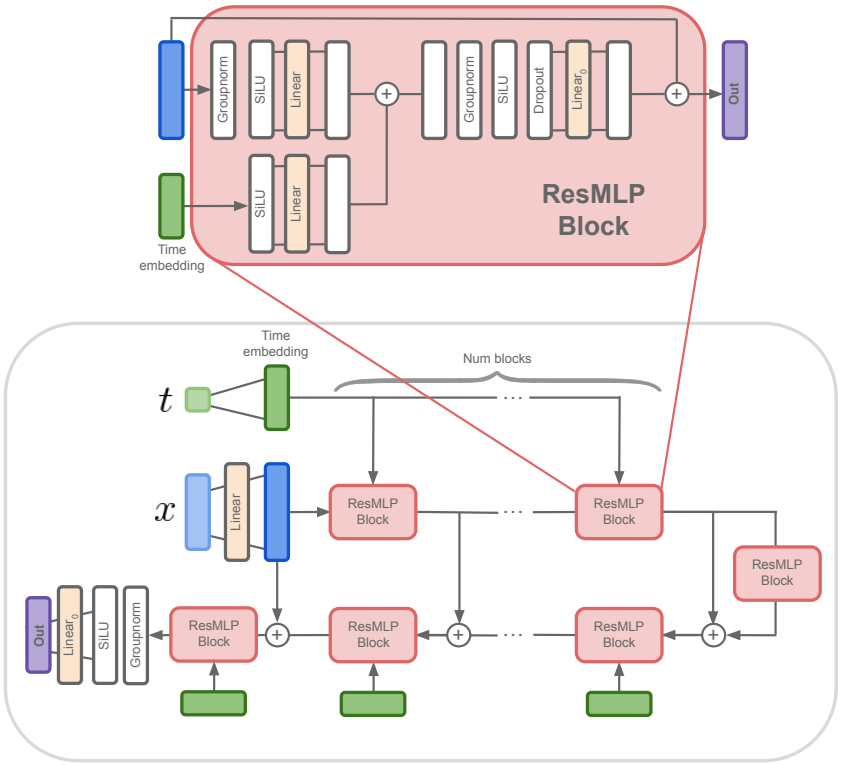

Figure 14. Architecture for $\boldsymbol{\epsilon}_\theta$ in our implementation of DDPM. Linear_0 stands for a linear layer with zero initialization.

for each value of t, where a value of $t \in [2, 1000]$ is randomly subsampled at each training iteration. It turns out that the term $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is also Gaussian and can be written in closed form, hence the above KL is tractable.

DDPM (Ho et al., 2020) simplifies the above loss to the expression (see paper for derivation):

$$\|\boldsymbol{\epsilon_t} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2 \tag{12}$$

where $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$ is reparameterized as $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon}_t$ for $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Here $\boldsymbol{\epsilon}_\theta$ is the neural network predicting $\boldsymbol{\epsilon}_t$ from $\mathbf{x}_t$, which is a more convenient parameterization for optimizing the loss than explicitly parameterizing $\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta$.

Once trained, sampling is performed by sampling $\mathbf{x}_T \sim p(\mathbf{x}_T)$ then taking the reverse process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to sample $\mathbf{x}_{T-1}, \ldots, \mathbf{x}_0$ in sequence. The resulting $\mathbf{x}_0$ is a sample from the model.

Training details. As with NSF, modulations are centered and normalized by elementwise mean & standard deviation across the training modulation dataset, and Adam is used with a custom learning rate schedule that warms up linearly from 0 to 3e-4 for the first 4000 warmup iterations, then decays proportional to the square root of the iteration count. We use $T = 1000$ and $\beta_{1:T} = \texttt{np.linspace}(10^{-4}, 0.02, T)$ following Ho et al. (2020), and parameterize $\boldsymbol{\epsilon}_\theta$ as in Figure 14, which again closely follows the architecture used in Ho et al. (2020) except using ResidualMLPs in place of CNNs.

Model selection. The hyperparameters that we tuned are the width of the ResidualMLPs, sweeped over $\{512, 1024, 2048, 3072\}$, the number of blocks, sweeped over $\{4, 8\}$ and the dropout probability in Reisudal MLPs, sweeped over $\{0., 0.1, 0.2, 0.3\}$. We take the model at the end of training to show samples, as we found that FID had limited correlation with perceptual quality (*cf.* Section 6.2) and sample quality seemed to have converged by the end of training. For DDPM on 256-dimensional modulations of CelebA-HQ, we use a batch size of 128 with width 3072, 4 blocks and 0.3 dropout probability, training for 1e6 iterations. For DDPM on 64-dimensional modulations of SRN Cars, we use a batch size of 256 with width 512, 4 blocks and 0. dropout, training for 1e5 iterations.

Tips. We found that width was the single most important hyperparameter, where wider models gave better samples. Also dropout was important to prevent the model from memorizing the training set.

A.6. Neural Radiance Fields (NeRF)

Background. In NeRF (Mildenhall et al., 2020), a 3D scene is expressed by a (scene) function that maps a 3D point's coordinates and viewing direction of the camera looking at that point to the RGB colour of the point and its density. In practice this function is an MLP (LatentModulatedSIREN in our case), that is trained on a dataset of multiple views of the same scene, using each view's pose information and its focal length, as described in Appendix A.1. Given this function, the model uses a volumetric rendering formula to construct views of the scene, by shooting rays from the camera to each pixel and aggregating the function values at points along each ray in a differentiable manner. The MLP parameters of the scene function are optimized by minimizing the reconstruction loss on all views. See Mildenhall et al. (2020) for details.

Training details. For simplicity, we use a minimal volumetric rendering algorithm (see Listing 1) that closely follows the implementation of Tancik et al. (2021), a simplified version of the rendering used in Mildenhall et al. (2020). We do not use hierarchical volume sampling (coarse/fine points on ray), use a single SIREN for outputting the RGB & density (as opposed to the standard practice of having separate networks/heads for each), no view dependence (usually the viewing direction is fed into the scene function as extra information). The (near, far) values that determine the start and end points for sampling on the ray are fixed to (0.8, 1.8).

Model selection. Note that the array containing all training views is large (50 views each with $128\times128\times3$ pixel values), hence we can only use a single scene for each training iteration per device (batch size=1 per device, so must use multiple devices for larger batch sizes). Further we must subsample views and pixels over which we optimize the reconstruction error, in order to fit the meta-learning in memory. Hence for meta-learning we tune the hyperparameters such as number of points per ray, sweeped over (16, 32, 64), the number of subsampled views, sweeped over (4, 8, 16), and the number of subsampled pixels, sweeped over (512, 1024, 2048). The optimal hyperparameters for the 512-dimensional modulation dataset is: 32 points per ray, 16 views and 512 pixels per view. Note that for the modulation dataset creation, we can use more views and pixels as we only need to run the inner loop of the meta-learning (rather than having to backpropagating through it, which is memory costly). Hence we used 32 views and 2048 pixels per view for the modulation dataset creation.

Tips. We found it best to max out memory by trying different combinations of the 3 hyperparamters (number of points per ray, number of subsampled views, number of subsampled pixels)

```python
import jax
import jax.numpy as jnp

def render_pose(model, params, height, width, focal, pose,
                num_points_per_ray, near, far, white_background):
  """Render view of a single scene.

  Args:
    model: MLP with input_size = 3, output_size = 4 (3 for RGB and 1 for density.)
    params: Model params.
    height: Height of image.
    width: Width of image.
    focal: Focal length.
    pose: Pose (camera to world matrix) of shape (4, 4).
    num_points_per_ray: Number of points per ray. Splits rays
        into equally spaced points.
    near: Point nearest to the camera where ray starts.
    far: Point furthest from the camera where ray ends.
    white_background: If True sets default RGB value to be 1,
        otherwise will be set to 0 (black).
  Returns:
    rgb_map: Rendered view. Array of shape (..., 3).
  """
  # Compute rays and split into ray origins and ray directions
  i, j = jnp.meshgrid(jnp.arange(width), jnp.arange(height), indexing='xy')
  dirs = jnp.stack([(i - width * .5) / focal,
                    -(j - height * .5) / focal,
                    -jnp.ones_like(i)], -1)
  rays_d = jnp.sum(dirs[..., jnp.newaxis, :] * pose[:3, :3], -1) # (..., 3)
  rays_o = jnp.broadcast_to(pose[:3, -1], rays_d.shape)  # (..., 3)
  # Compute 3D query coordinates
  z_vals = jnp.linspace(near, far, num_points_per_ray)
  # The below line uses (a lot of) broadcasting. In terms of shapes:
  # (...,1,3) + (...,1,3) * (num_points_per_ray,1) = (...,num_points_per_ray,3)
  coords = rays_o[..., None, :] + rays_d[..., None, :] * z_vals[..., :, None]
  out = model(params, coords)  # (..., num_points_per_ray, 4)
  # Compute colors and volume densities
  rgb, density = out[..., :3], out[
      ..., 3]  # (..., num_points_per_ray, 3), (..., num_points_per_ray)
  density = jax.nn.elu(density, alpha=0.1) + 0.1 # Ensure density is positive
  density = jnp.clip(density, 0., 10.)  # upper bound density at 10
  # Calculate distance between consecutive points along ray.
  distance_between_points = z_vals[..., 1:] - z_vals[..., :-1]
  distances = jnp.concatenate([distance_between_points,
                               1e-3 * jnp.ones(1)])  # (num_points_per_ray,)
  # Alpha will have a value between 0 and 1
  alpha = 1. - jnp.exp(-density * distances)  # (..., num_points_per_ray)
  # Ensure transmittance is <= 1 (and greater than 1e-10)
  trans = jnp.minimum(1., 1. - alpha + 1e-10)
  # Make the first transmittance value along the ray equal to 1 for every ray
  trans = jnp.concatenate([jnp.ones_like(trans[..., :1]), trans[..., :-1]],
                          -1)  # (..., num_points_per_ray)
  cum_trans = jnp.cumprod(trans, -1)  # T_i in Equation (3) of NeRF paper.
  weights = alpha * cum_trans  # (..., num_points_per_ray)
  # Sum RGB values along the ray
  rgb_map = jnp.sum(weights[..., None] * rgb, -2)  # (..., 3)
  # Optionally make background white
  if white_background:
    acc_map = jnp.sum(weights, -1)  # Accumulate weights   (...)
    rgb_map = rgb_map + (1. - acc_map[..., None])  # Add white background
  return rgb_map
```

Listing 1. Minimal volumetric rendering

A.7. Classification

We use the default MLP in Haiku (Hennigan et al., 2020) with SiLU/swish activations (Hendrycks & Gimpel, 2016; Ramachandran et al., 2017) and dropout for classifying modulations. The width is sweeped over (128, 256, 512) and the depth over (2,3,4).

For the 3D CNN baseline we closely follow the architecture in (Maturana & Scherer, 2015), which applies a sequence of 3D Conv layers with stride 2, followed by a flattening, then 2 linear layers, the first of which has width 128 and the last of which outputs logits for the classification. Each 3D Conv/linear layer is followed by a SiLU activation then dropout. The 3D Conv layers all have the same number of output channels sweeped over (16, 32, 64), and the number of 3D Conv layers is also sweeped over (4,5,6).

Both classifiers are optimized for 3e4 iterations with Adam at a fixed learning rate of 1e-4, and a total batch size of 1024. For the 3D CNN, this is spread over 8 devices to fit the batch in memory. In Section 6.6, we show results for the best-performing MLP with 128 width and the best-performing 3D CNN with 16 output channels, with error bars calculated over 3 different random seeds. In Appendix D we show results for larger models.

A.8. FID scores

For CelebA-HQ, we evaluated our model in terms of FID (Heusel et al., 2017) but found that it was not a meaningful perceptual metric for our model. FID scores at the latter stages of training were sometimes worse than early in training even though samples were perceptually of much higher quality. Further, the FID score of the functaset itself (fitted directly to the dataset) is 28.4 for 256-dimensional modulations and 17.2 for 512-dimensional modulations, despite both being perceptually very close to the original dataset (*cf.* Figure 18). We believe this is due to FID's property of over-penalizing blurriness, with very similar effects observed by Du et al. (2021) and Razavi et al. (2019). Nonetheless, our model obtains an FID score of 40.4, compared to 13.5 for GASP, 5.9 for StyleGANv2 (Karras et al., 2020) and 175.3 for a VAE (Du et al., 2021).

A.9. Figure 2 details

The original 128^3 voxel grid was downscaled using the `scipy.ndimage.zoom` function to each of the resolutions 8^3, 16^3, 32^3, 64^3, 128^3. For each grid, we performed a manual search over SIREN (*cf.* Appendix A.2) architectures (width, depth and ω_0) to find the smallest architecture that could fit the voxel grid to 99.9% accuracy. The settings for each architecture are summarized in Table 5.

RESOLUTION	DEPTH	WIDTH	ω_0	n_{PARAMS}
8^3	4	6	8	157
16^3	4	8	12	257
32^3	5	12	12	685
64^3	7	20	14	2621
128^3	10	32	50	9665

Table 5. Hyperparameter values used for Figure 2.

B. Modulation analysis

B.1. Perturbation analysis

We provide intuition for the role of each shift modulation dimension by perturbation analysis of individual shift modulation dimensions in Figure 15, for functa with 4-layer SIREN modulations with width 256 trained on the 32×32 CelebA-HQ training set. The left column shows a plot of modulation index (x-axis) against RMSE in the perturbed reconstruction (y-axis) after applying a perturbation of $\delta = -0.01$ to the scalar shift modulation at that index. We colour code modulation units by layer. For each figure row we take 4 different shift modulation units from a given layer with similar RMSEs (sampled from the blue box) and visualize reconstructions after perturbing each unit on 4 different training images (middle column), along with the L1 error of the reconstruction for each spatial dimension (right column).

The first row shows perturbations of 4 units of the SIREN's initial hidden layer (layer 0) that have high RMSE (> 5). We see that these units are responsible for features that are roughly periodic along the diagonal, with different periodicity and

direction - the troughs show zero error along the diagonal lines.

The second row shows 4 units of medium RMSE ($\in [2, 3]$), where the periodicity remains but the frequency is lower than the high RMSE units. Hence it seems that the initial hidden layer contains a range of units that account for features of different frequencies in pixel space.

The third row shows 4 units in the next layer (layer 1), where the errors in pixel space are no longer periodic, and seem to be aligned with the edges of the image. Also the magnitudes of the perturbation in the image space are smaller than the previous layer, consistent with their RMSE values.

The final row shows 4 units in the subsesquent layer (layer 2) again showing errors that seem to be aligned with the edges, but are a bit smoother and the magnitudes of the perturbation in the image space are yet smaller than previous layers.

Overall RMSE due to perturbation tends to decrease with depth of the SIREN, indicating that the modulations of earlier layers have a greater role in modelling the variations across the dataset compared to later layers.

One other interesting observation from the right column is that the perturbations for a given modulation unit lead to very similar error patterns across all images. This suggests that we can think of each modulation dim as representing the coefficient of a basis function, since changing the value of this coefficient leads to similar changes for all images. However note from the figure that these basis functions are non-local. This might pose challenges for downstream neural networks to extract high level information from such modulations, given that locality is an important inductive bias for many architectures such as CNNs and Transformers. Hence we may wish to explore other decoder architectures such as Gaussian activations (Ramasinghe & Lucey, 2021) or Multiplicative Gabor Networks (Fathony et al., 2020) whose modulations may correspond to more localised information.

B.2. Alternative Modulation architectures

We explored alternative forms of modulations, which we found to be similar if not worse than latent modulations introduced in Section 3.1:

Scale+Shift, Scale-only. We only used shift modulations which are added to the SIREN activations. We found that when using both scale and shift, the scale values are almost always optimized to 1, hence redundant, giving similar performance to shift-only modulations. Scale-only modulations gave noticeably worse performance, resulting in around 5dB drop in reconstruction PSNR on images. We conjecture that this is due to gradients with respect to shift being independent to activation magnitude, whereas gradients with respect to scale is linearly dependent, resulting in higher magnitude gradients for shift modulations that allow faster optimization.

SubsetModulatedSiren. As shown in Appendix B.1, reconstructions are more sensitive to earlier modulation layers than later ones. Hence we can reduce the number of shift modulations by only using them for the first few layers of the MLP corresponding to the INR. While also competitive, we found that this approach does slightly worse than using latent modulations. Although we haven't yet tried, we expect better performance when these two approaches are combined, so that the latent code is only mapped to a subset of modulations.

ModSine. We experimented with the architecture in Mehta et al. (2021). Instead of mapping a latent vector to the modulations, we map a latent vector through a network that has the same shape as the base network and treat the inner layers of the mapping network as modulations. However, we found that this approach gave lower reconstruction accuracy than latent modulations for the same modulation dimension.

Concatenating latent to input of SIREN or ReluMLP+pos-enc We have also tried meta-learning with the architecture in Sitzmann et al. (2020a) that concatenates the latent vector as input to the MLP decoder, and results were worse than latent modulations: we tried both SIREN and ReluMLP+pos-enc, sweeping over optimization hyperparameters and controlling for latent & model size. The best was a SIREN with concatenated latent, giving test PSNR 23.0dB for latent-dim=256 on CelebA, notably worse than latent modulation's 25.6dB test PSNR in Table 1.

C. Negative Results

Autoregressive Transformers. As a third type of generative model, we tried training Transformers (Vaswani et al., 2017) on modulations (after discretizing each modulation to 5 bits after which we saw little loss in reconstruction accuracy). Yet the resulting samples shown in Figure 16 were clearly perceptually worse than diffusion. We suspect this is because it is

difficult to impose a meaningful ordering on the modulations for autoregressive modelling - the ordering is something we haven't played around with too much.

Attention within diffusion and flow. For the NSF, we tried using Tranfsormers for the conditioner but this did not outperform the MLP. For diffusion, we also tried using Transformers in place of residual MLPs, but this also performed worse.

First order MAML. We also tried first order meta-learning that does not backpropagate through the inner loop, to see if we could save memory with little sacrifice in performance. However this led to a drastic reduction in reconstruction PSNR, on the order of 10dB for images.

Using more than 3 gradient steps for fitting modulations. Figure 17 shows reconstructions with up to 4 steps. Note that in most cases the difference between the 3rd and 4th steps is imperceptible, and for chairs the performance actually deteriorates (likely due to the fixed learning rate for the inner loop being too high). Using a greater number of steps, e.g. 50, we can obtain perceptibly better reconstructions for images (around 2dB better) but the modulation space is now less smooth (more gradient steps away from the initial modulation), so samples from generative modelling showed worse sample quality.

Using more than 3 inner loop steps for meta-learning. For scene data, we tried meta-learning with 4 or 5 inner loop steps. This led to a larger memory consumption, and hence we had to subsample fewer views and pixels per scene, yielding no noticeable improvement in terms of PSNR. For scenes, it seems very likely that the minimal rendering is the bottleneck for our setup.

Flow on 256 mod-dim for images performed noticeably worse than DDPM in terms of sample quality for images.

Flow on 512-dimensional modulations for voxels gave much worse sample quality than 256-dimensional modulations. We conjecture that using fewer modulations helps obtain a smoother modulation space, despite having worse PSNR/voxel accuracy.

Diffusion on 256 mod-dim for voxels was also noticeably worse than flows. Overall it appears as though flows are more suited for ShapeNet and diffusion for CelebA-HQ, and we conjecture that this is a property of the dataset rather than the modality.

Bounding box subsampling for SRN Cars. For scenes, when subsampling pixels from views for meta-learning, we tried subsampling pixels within the bounding box that contains the car, to avoid sampling white background pixels that are less informative than the foreground car pixels. However this didn't work better than subsampling at random, and we found that subsample white background pixels helps to achieve higher PSNR to accurately recover the white background.

Auto-decoder approach for learning per-datapoint modulations. Encouraged by the success of Park et al. (2019) on learning signed-distance functions, we have tried the auto-decoder approach for learning per-datapoint modulations on images. In particular we store a randomly initialized set of (latent) modulations per image (unshared params), along with a set of shared parameters that we jointly optimize by minimizing reconstruction error on each batch during training. This is attractive because it does not require memory-expensive double loop optimizations like meta-learning. However for auto-decoding, training was unstable and PSNR was notably worse than for meta-learning, despite carefully tuning separate learning rates for the shared and unshared parameters with a varying number of gradient updates for each set of parameters. It is unclear as to why this approach seems to work for signed-distance functions yet does not seem to work for other data modalities.

D. Further Results

Original vs Reconstructions from modulations. In Figure 18 we compare the original array dataset (top row) against the reconstruction from 256-dimensional modulations obtained from meta-learning for each of the data modalities (used to compute the middle column for Table 1). We see that the reconstructions are reasonably close to the original, although some fine detail can be missed for complex shapes and scenes. As mentioned in Appendix C, using more inner loop gradient steps to fit the modulations can result in better reconstructions but at the sacrifice of performance for the downstream task e.g. generative modelling.

Image samples and nearest neighbours in training set. In Figure 19 we show the nearest neighbours of samples from the diffusion model trained on 256-dimensional CelebA-HQ modulations. As can be seen, the samples are reasonably different to the training data, indicating that the diffusion hasn't simply memorised the training set. We show similar results

for diffusion models trained on 512-dimensional modulations.

High resolution samples. In addition to unconditional generation, we also use our model to generate high resolution samples (512×512) as shown in Figure 20. The left is obtained by querying the function represented by a 256 modulation sampled from the flow on a 512×512 grid, and the right is obtained by querying the same function on a 64×64 grid then applying bicubic interpolation to upscale to 512×512 resolution. We can see that the left is more crisp than the right, especially around the jawline and the teeth, indicating that the flow has learned a good internal representation of faces from various images in the CelebA-HQ dataset.

Uncurated samples from flow on ShapeNet. In Figure 21 we show uncurated samples from the NSF trained on 256-dimensional modulations of ShapeNet Chairs. We also show samples at various temperatures of the base distribution in Figure 22, where samples become more globally coherent with lower temperature but show less diversity. Additionally in Figure 25 we show uncurated samples from the class-conditional flow trained on 256-dimensional modulations of ShapeNet 10 classes.

ShapeNet 10 classification results for (V)AE latent representations. As a baseline, we have fit Autoencoders (AE) and Variational Autoencoders (VAE) (Kingma & Welling, 2014; Rezende et al., 2014) on ShapeNet with 3DCNN enc/decoders and 256-dim latents. Sweeping over hyperparams such as model size gave (V)AEs with reconstruction voxel-acc $\in [98, 99.5]\%$. Classification on resulting (V)AE latents, with the same MLP hyperparam sweep for functa, gave best test accuracy of $93.2 \pm 0.2\%$ (AE), $91.9 \pm 0.3\%$ (VAE) - similar if not worse than 256-dim functa. Note that functa provide a unified architectural framework for various modalities, whereas VAEs need modality specific architectures e.g. designing a VAE for NeRF scenes is tricky (c.f. NeRF-VAE). Also it is unclear how to do inference (imputation, novel view synthesis) from partial observations for VAEs, yet this can be done with functa as shown in the paper.

ShapeNet 10 classification on functa for bigger classifiers. In Table 6 we show test accuracies and parameter counts for bigger models compared to those shown in Section 6.6. Note that bigger models perform better for both classes of models, and bigger 3D CNNs perform better than bigger MLPs on functa, although we need much bigger 3D CNNs to do so. In Figure 24 we also show train/test accuracy and per-class test-accuracy throughout training for both the MLP classifier on modulations and the 3D CNN classifier. The classes that each classifier struggles with are fairly similar.

	MLP ON FUNCTA			3D CNN ON ARRAY		
TEST ACCURACY	$93.6 \pm 0.1\%$	$93.9 \pm 0.1\%$	$94.0 \pm 0.1\%$	$93.3 \pm 0.3\%$	$94.5 \pm 0.2\%$	$94.8 \pm 0.0\%$
n_{PARAMS}	83K	212K	924K	550K	2.0M	7.9M

Table 6. Same as Table 2 but for a wider range of model sizes

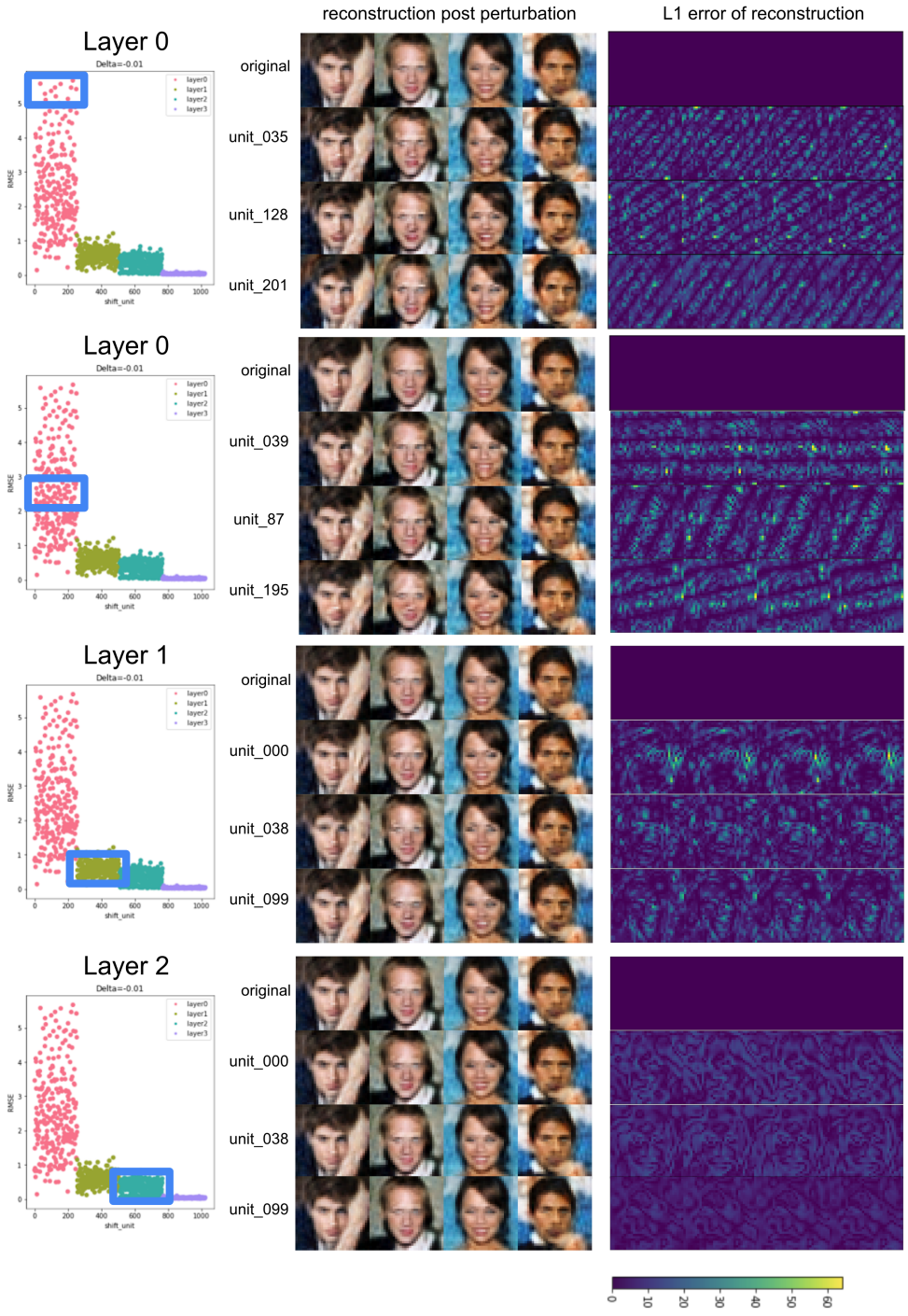

Figure 15. Visualization of errors in pixel space when perturbing individual modulation dimensions of each layer by 0.01.

Figure 16. Uncurated samples from autoregressive Transformer trained on 256-dim modulations for CelebA-HQ 64×64.

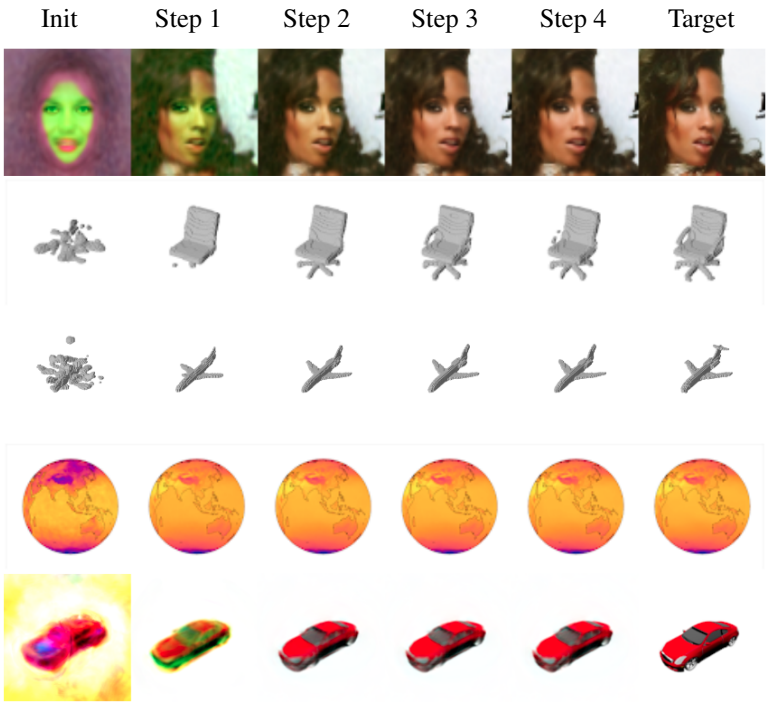

Figure 17. Same as Figure 4 but with an extra inner loop step, and extra row for ShapeNet 10 classes.

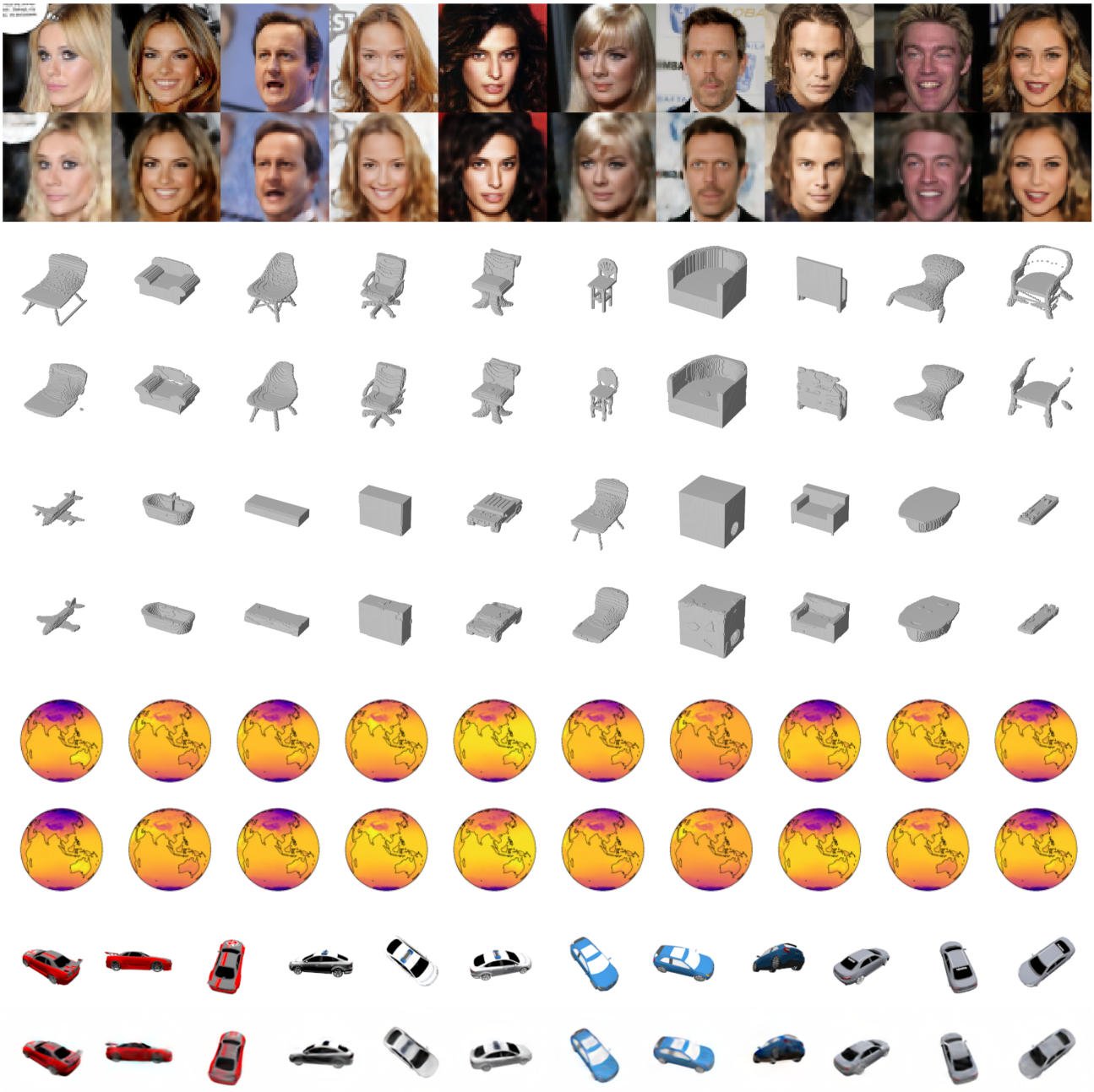

Figure 18. Original vs reconstruction for 256-dimensional modulations on the first few data points of each training dataset.

Figure 19. Nearest neighbours of samples from diffusion model trained on 256-dim CelebA-HQ 64×64 modulations. Leftmost column are samples, and subsequent columns are the closest neighbours in L2 distance in modulation space.

Figure 20. Comparison of rendering sample at 512 resolution vs rendering at 64 resolution then upsampling to 512 by bicubic interpolation

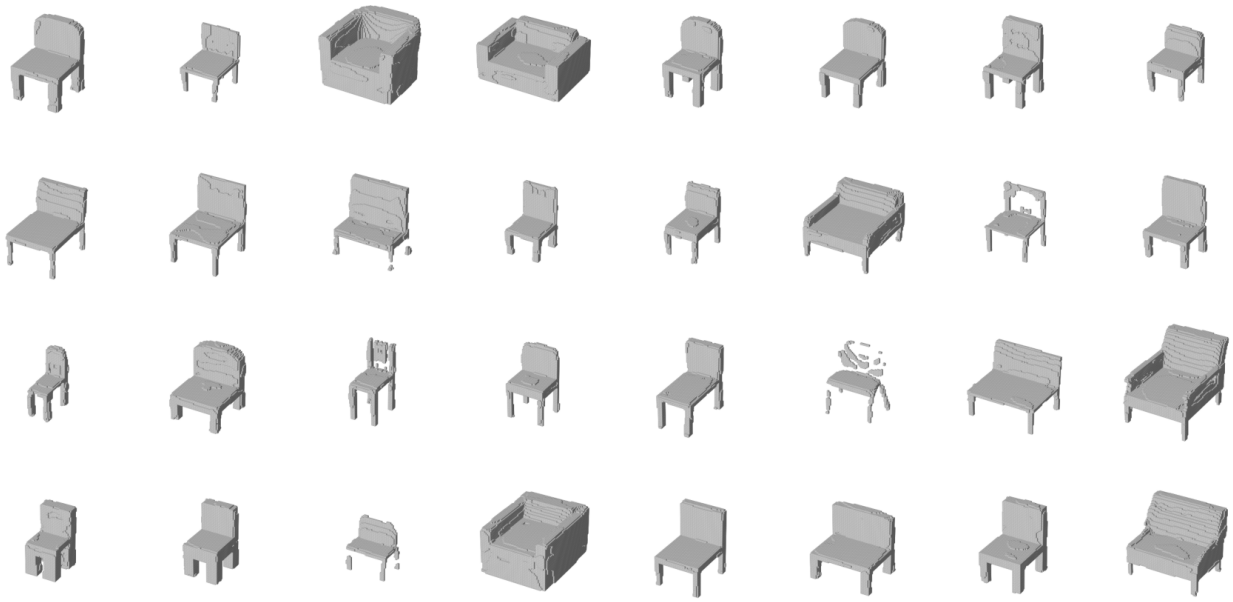

Figure 21. Uncurated samples from flow trained on 256-dim ShapeNet Chairs 64^3 modulations, temperature 0.95.

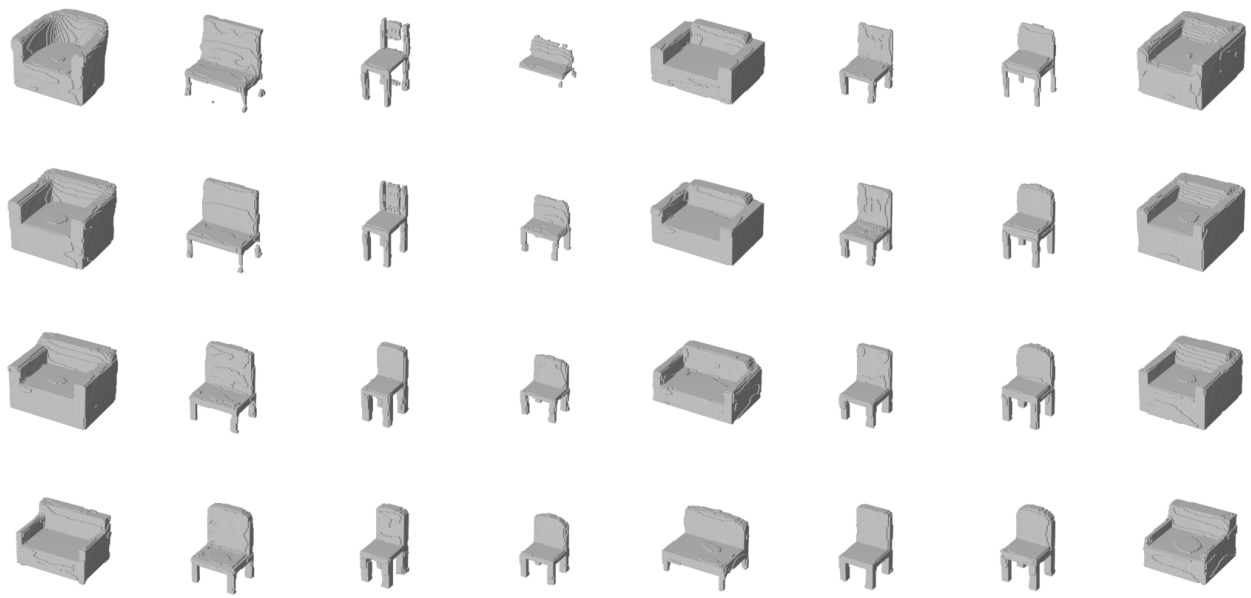

Figure 22. Uncurated samples from flow trained on 256-dim modulations of ShapeNet Chairs 64^3, at varying temperatures. Temperatures for each row: 1.0, 0.9, 0.8, 0.7.

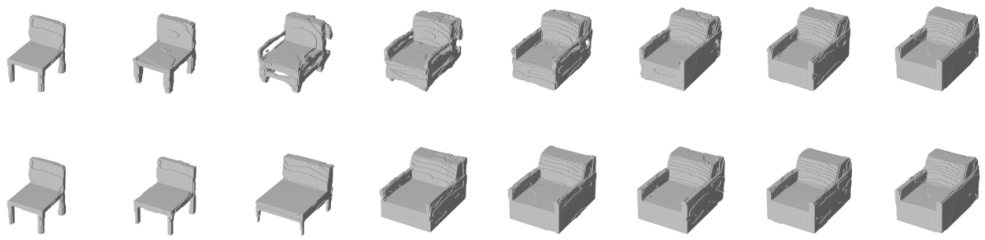

Figure 23. Comparison of interpolation in modulation space (top) and flow latent space (bottom) for 256-dim modulations of ShapeNet 10 Classes 64^3.

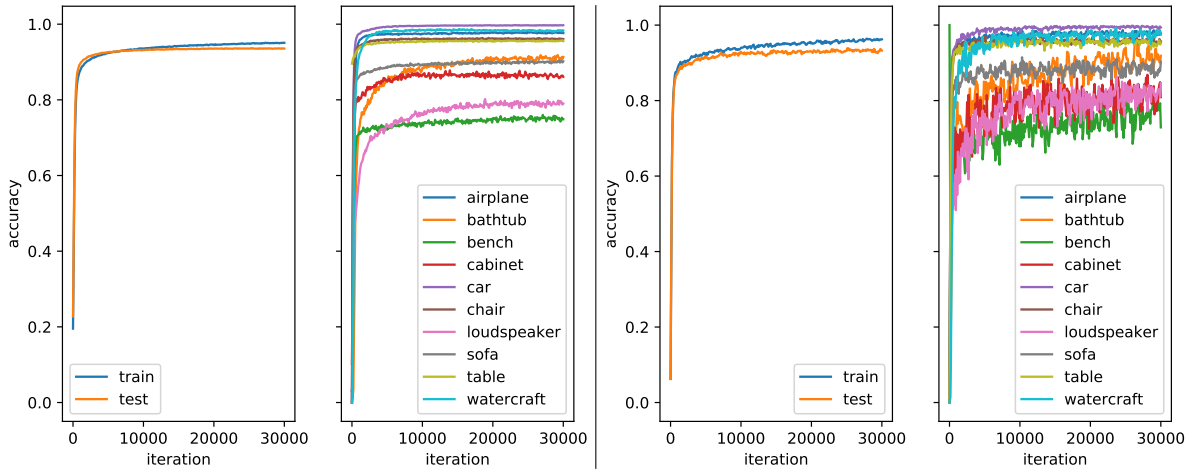

Figure 24. Left: Train/test (left) & per-class test (right) classification accuracy throughout training on 256-dim modulations of ShapeNet 10 Classes 64^3, using batch size 1024. Right: Same for 3D CNN, but using batch size 64 (GPU memory only allows up to 8 per device). Both Using exponential moving average with decay=0.99 and bs=1024.

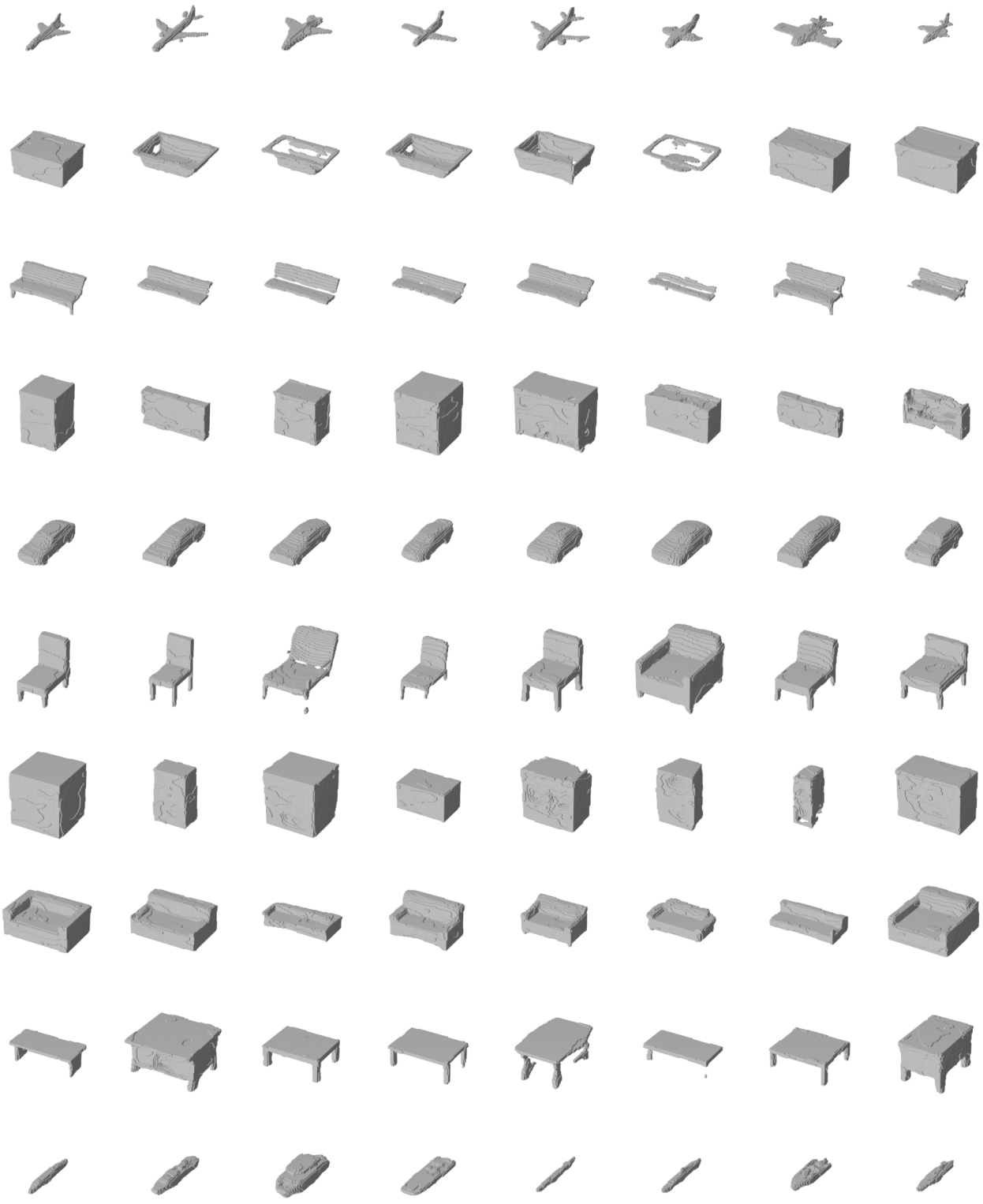

Figure 25. Uncurated samples from class-conditional flow trained on 256-dim modulations of ShapeNet 10 classes 64^3.

Figure 26. Uncurated samples from DDPM trained on 64-dim modulations of SRN Cars.

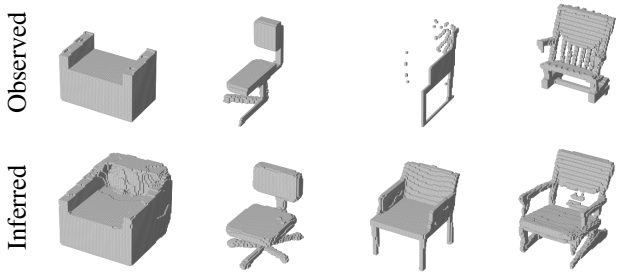

Figure 27. Additional results to Figure 7 for imputation of different chairs from the test set. GIF showing course of optimization: bit.ly/3G7KDh8

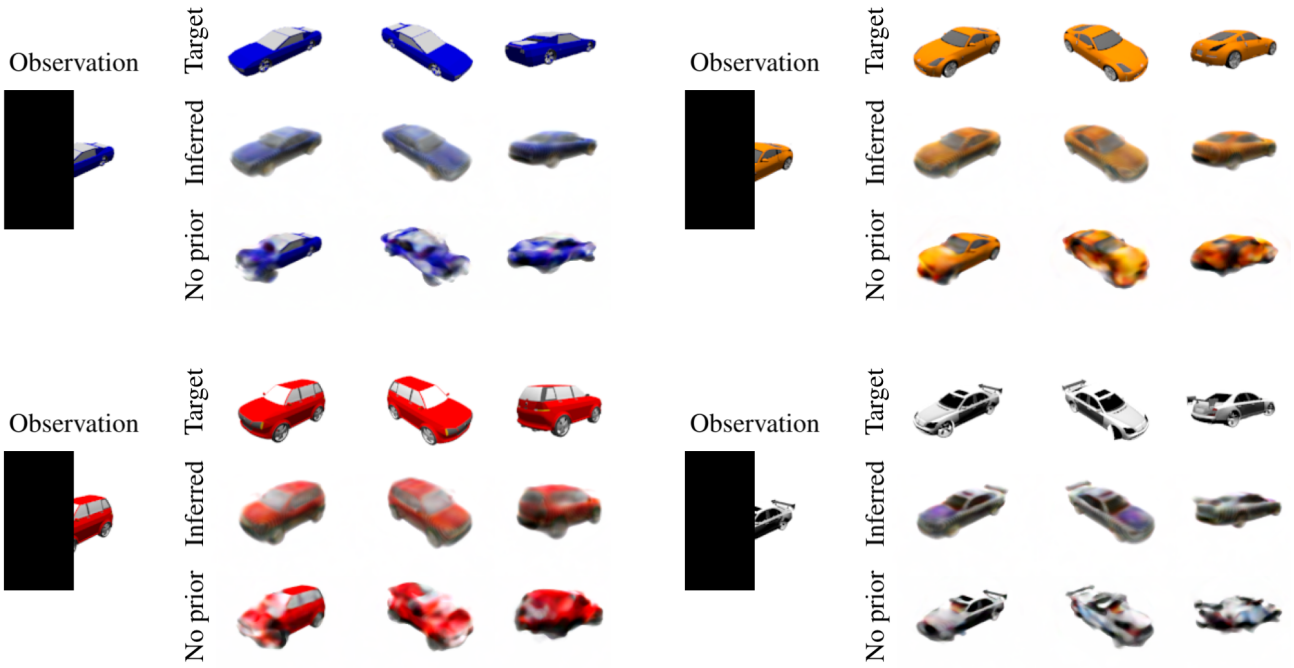

Figure 28. Additional results to Figure 11 for novel view synthesis from additional occluded test scenes. GIF: bit.ly/3x842vO